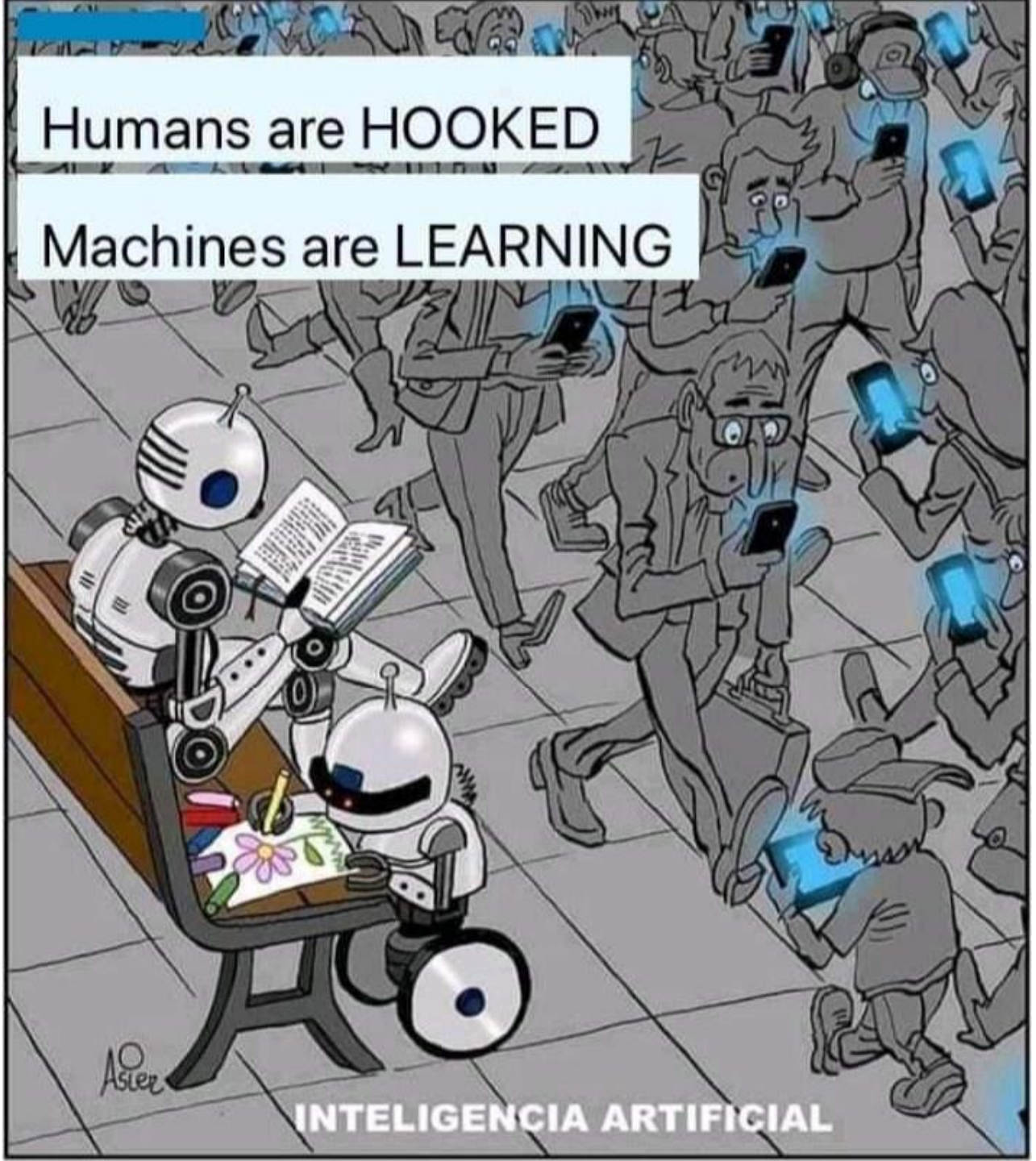
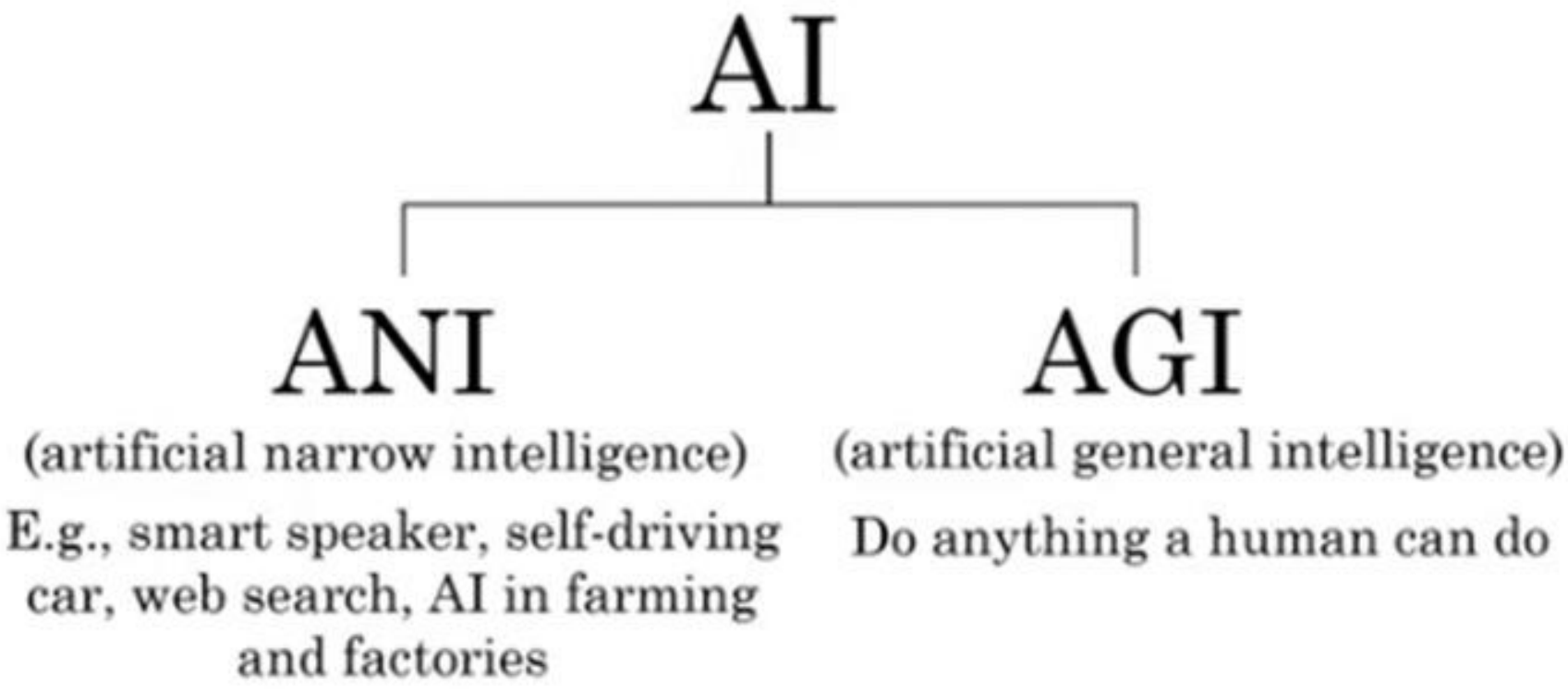


Agentic AI – Copilot to Autopilot

Naveen Kumar Bhansali
Co-Founder BlitzAI | Adjunct Faculty @ IIM Bangalore

AGI Vs. ANI


AI
ANI
(artificial narrow intelligence)
E.g., smart speaker, self-driving car, web search, AI in farming and factories
AGI
(artificial general intelligence)
Do anything a human can do

AGI Levels

- **AGI tier list**
- **-- Google DeepMind**

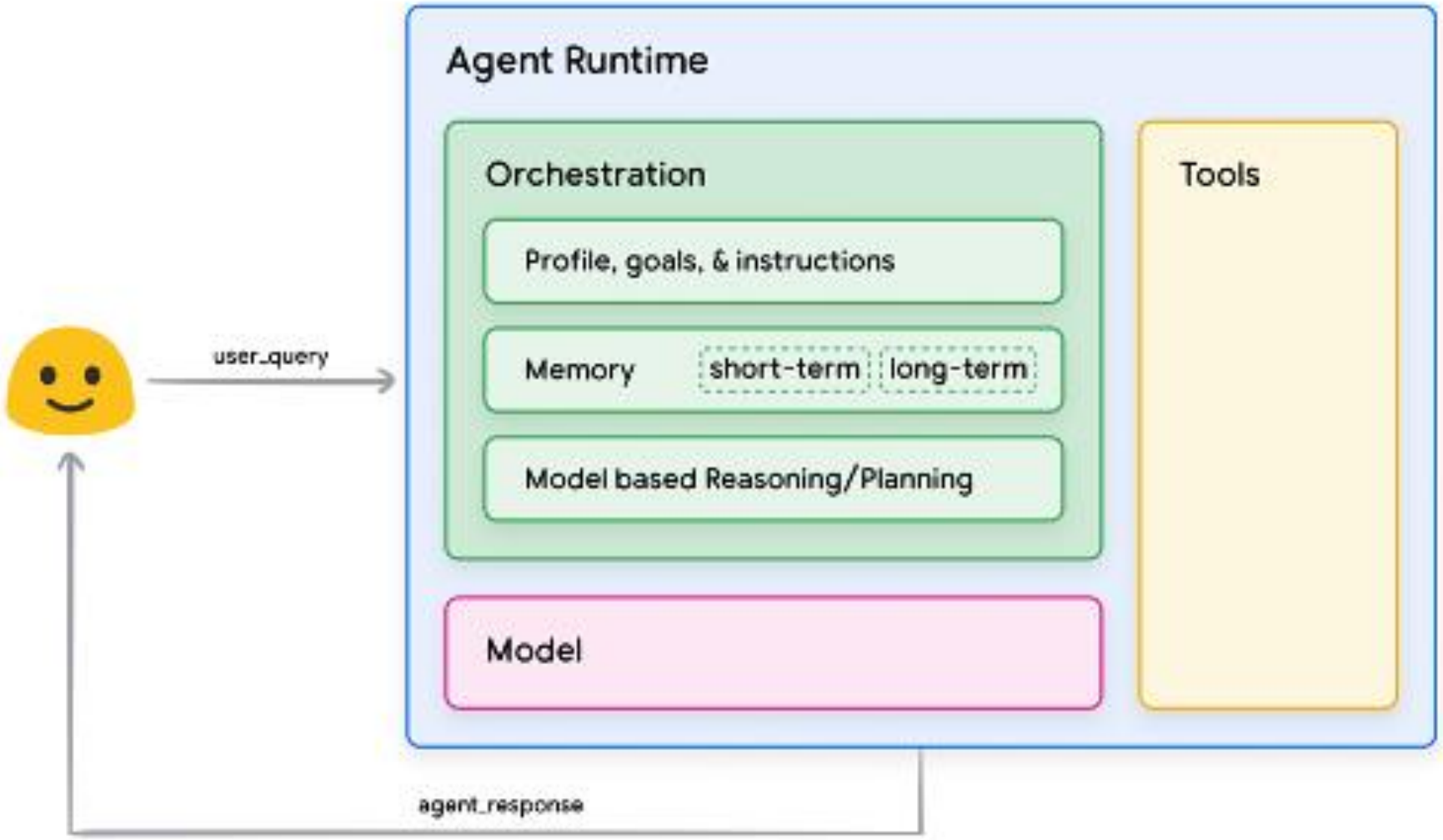
Levels of AGI

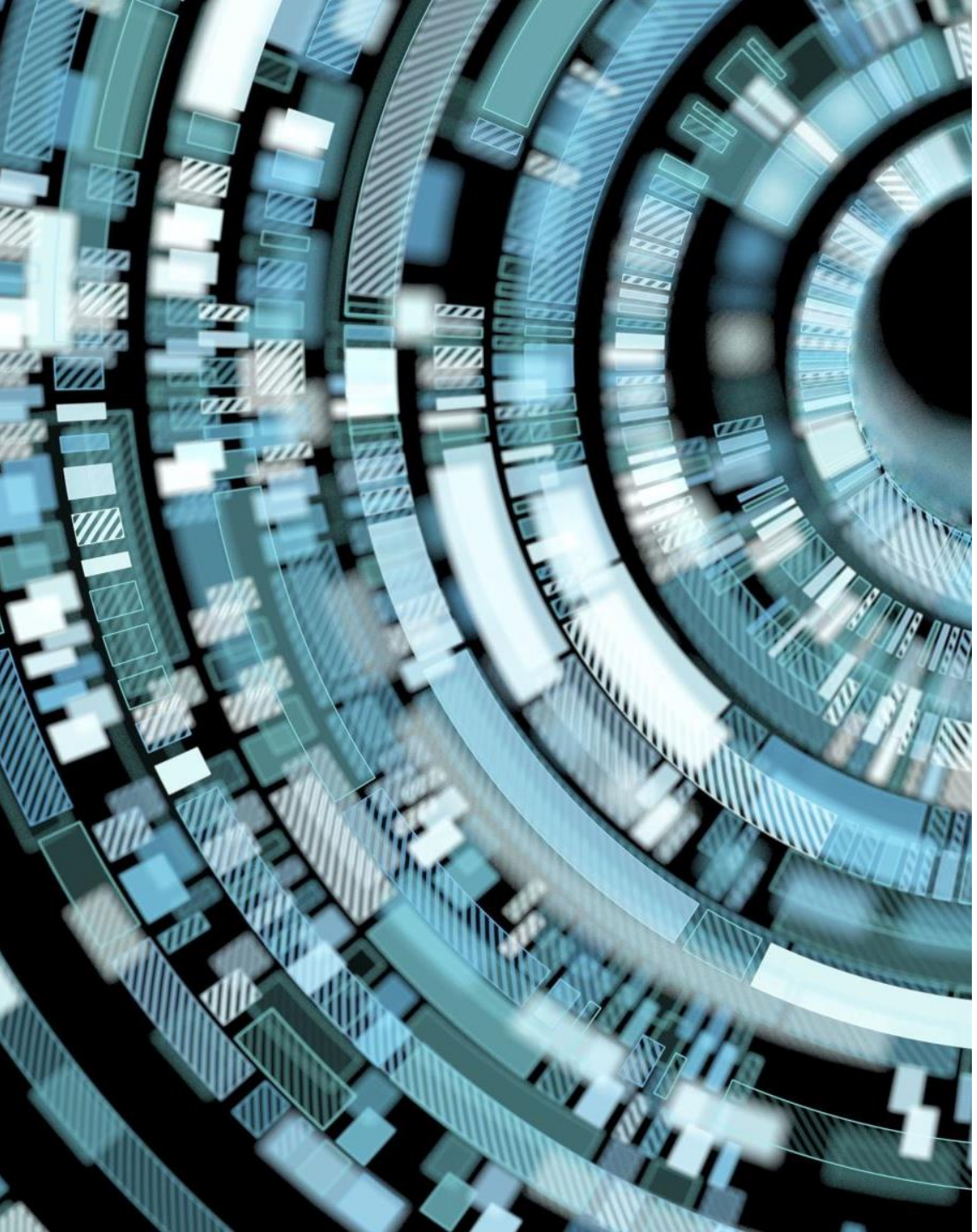
Performance (rows) x **Generality** (columns)	**Narrow** *clearly scoped task or set of tasks*	**General** *wide range of non-physical tasks, including metacognitive abilities like learning new skills*
Level 0: No AI	**Narrow Non-AI** calculator software; compiler	**General Non-AI** human-in-the-loop computing, e.g., Amazon Mechanical Turk
Level 1: Emerging *equal to or somewhat better than an unskilled human*	**Emerging Narrow AI** GOFAI[4]; simple rule-based systems, e.g., SHRDLU (Winograd, 1971)	**Emerging AGI** ChatGPT (OpenAI, 2023), Bard (Anil et al., 2023), Llama 2 (Touvron et al., 2023)
Level 2: Competent *at least 50th percentile of skilled adults*	**Competent Narrow AI** toxicity detectors such as Jigsaw (Das et al., 2022); Smart Speakers such as Siri (Apple), Alexa (Amazon), or Google Assistant (Google); VQA systems such as PaLI (Chen et al., 2023); Watson (IBM); SOTA LLMs for a subset of tasks (e.g., short essay writing, simple coding)	**Competent AGI** not yet achieved
Level 3: Expert *at least 90th percentile of skilled adults*	**Expert Narrow AI** spelling & grammar checkers such as Grammarly (Grammarly, 2023); generative image models such as Imagen (Saharia et al., 2022) or Dall-E 2 (Ramesh et al., 2022)	**Expert AGI** not yet achieved
Level 4: Virtuoso *at least 99th percentile of skilled adults*	**Virtuoso Narrow AI** Deep Blue (Campbell et al., 2002), AlphaGo (Silver et al., 2016, 2017)	**Virtuoso AGI** not yet achieved
Level 5: Superhuman *outperforms 100% of humans*	**Superhuman Narrow AI** AlphaFold (Jumper et al., 2021; Varadi et al., 2021), AlphaZero (Silver et al., 2018), StockFish (Stockfish, 2023)	**Artificial Superintelligence (ASI)** not yet achieved

Generative AI Vs. Agentic AI

	Generative AI	Agentic AI
Main Purpose	Content creation based on training data in response to user prompts	Autonomous action and decision-making
Autonomy	Low; reacts to user input and cannot set its own goals	High; acts independently to set and pursue goals
Adaptability	Shows some adaptability, but cannot independently adapt to fully new or unstructured environments	Can adjust its behavior in response to changing conditions of real-world or virtual environments
Goal Setting	No independent goal setting; operates within predefined constraints	Capable of setting its own goals
Human Oversight	Necessary; operates based on user-provided prompts	Minimal; able to function with little to no human intervention

General agent architecture and components





LLMs

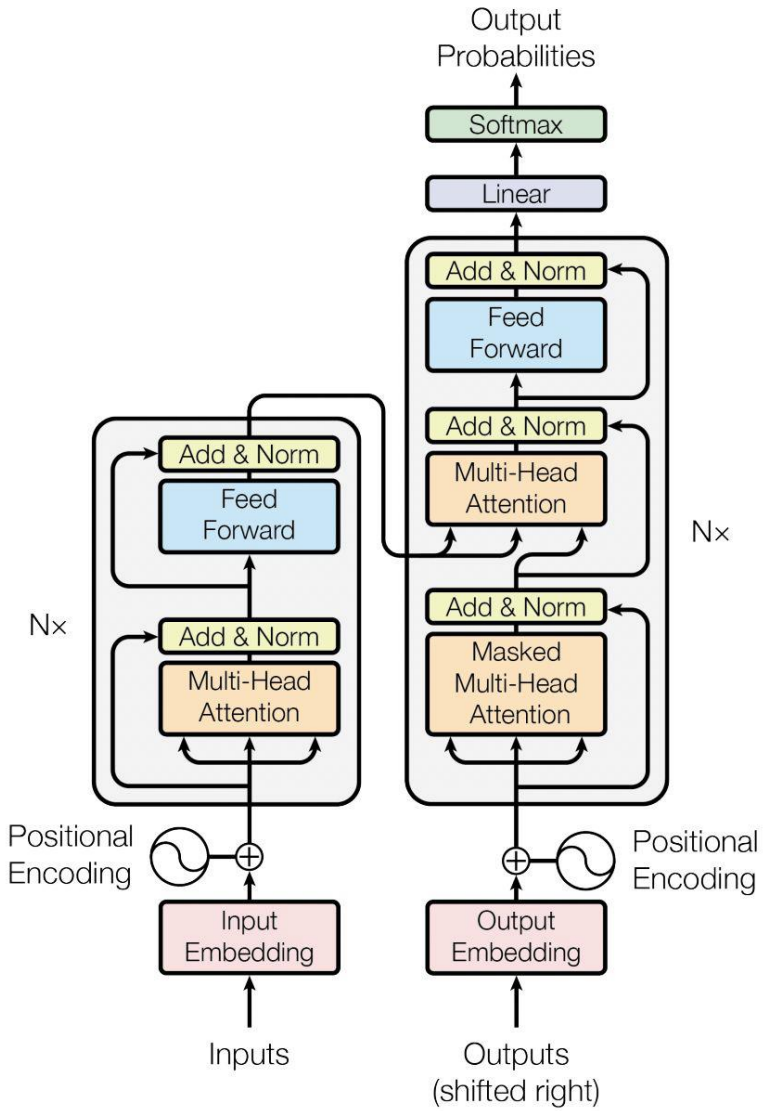
- A Language Model
- Consisting of Neural network with many parameters
  - (typically, billions to trillions of weights or more)
- Trained on large quantities (trillions of words) of unlabelled text.
  - Wikipedia, GitHub, Common Crawl, The Pile etc.
- Using self-supervised learning

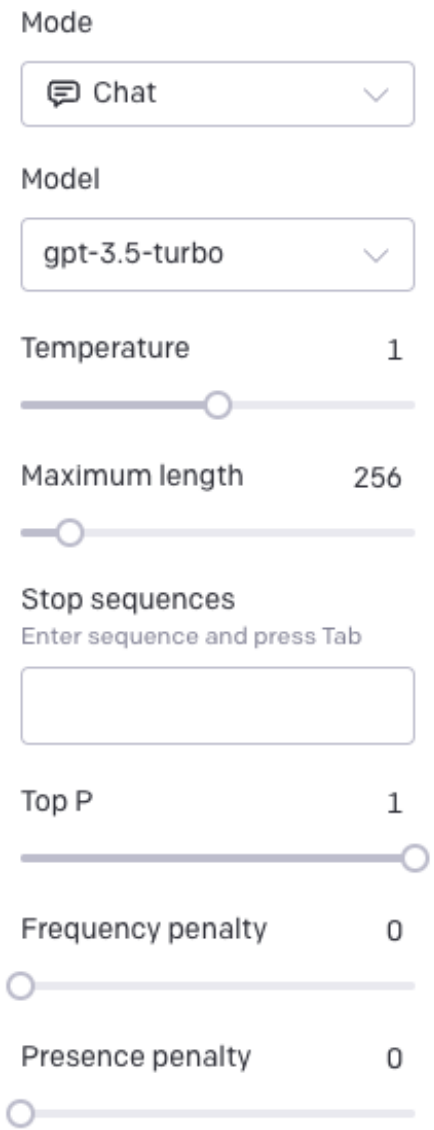
LLMs – Based on?

Attention is all you need

https://ig.ft.com/generative-ai/

Transformers architecture

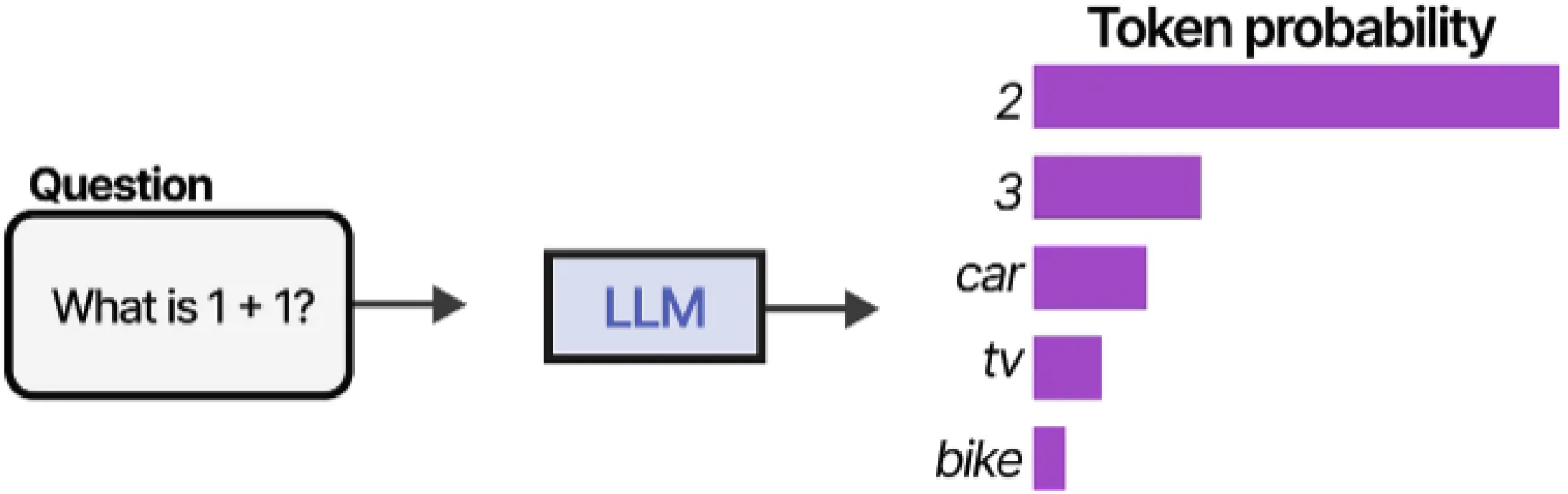




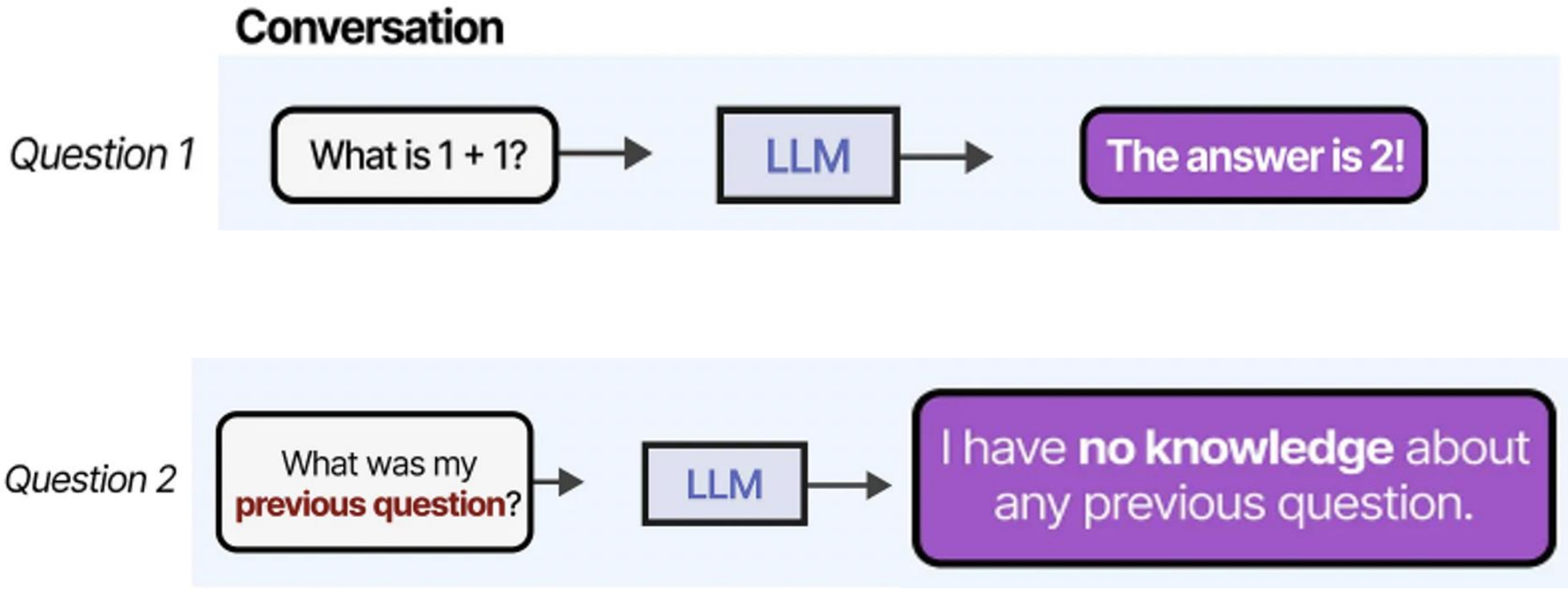
OpenAI Playground

https://platform.openai.com/playground

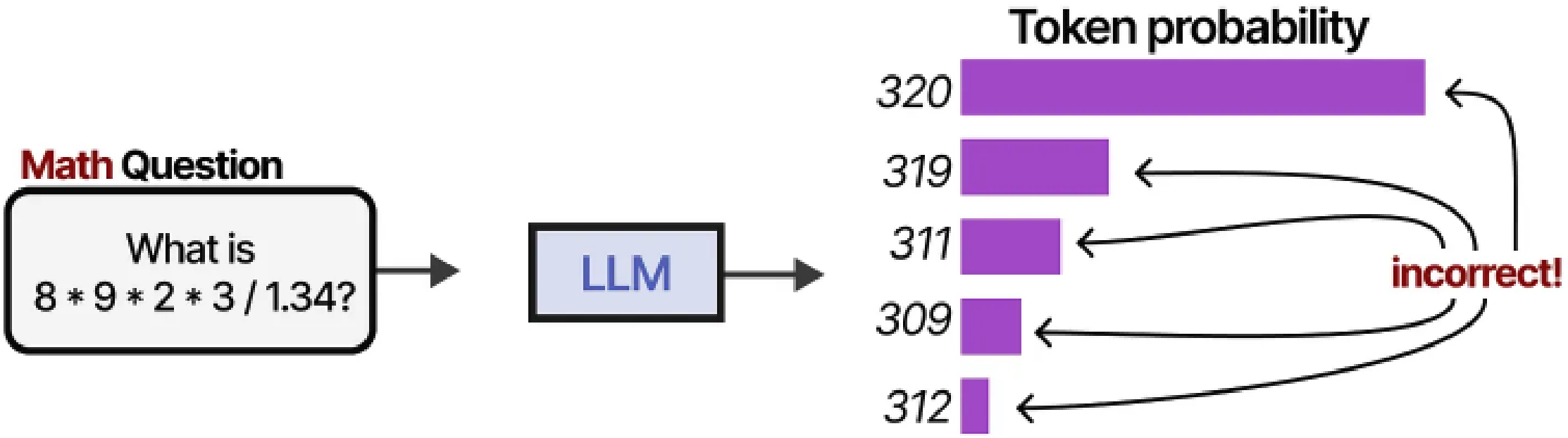
Model - LLM



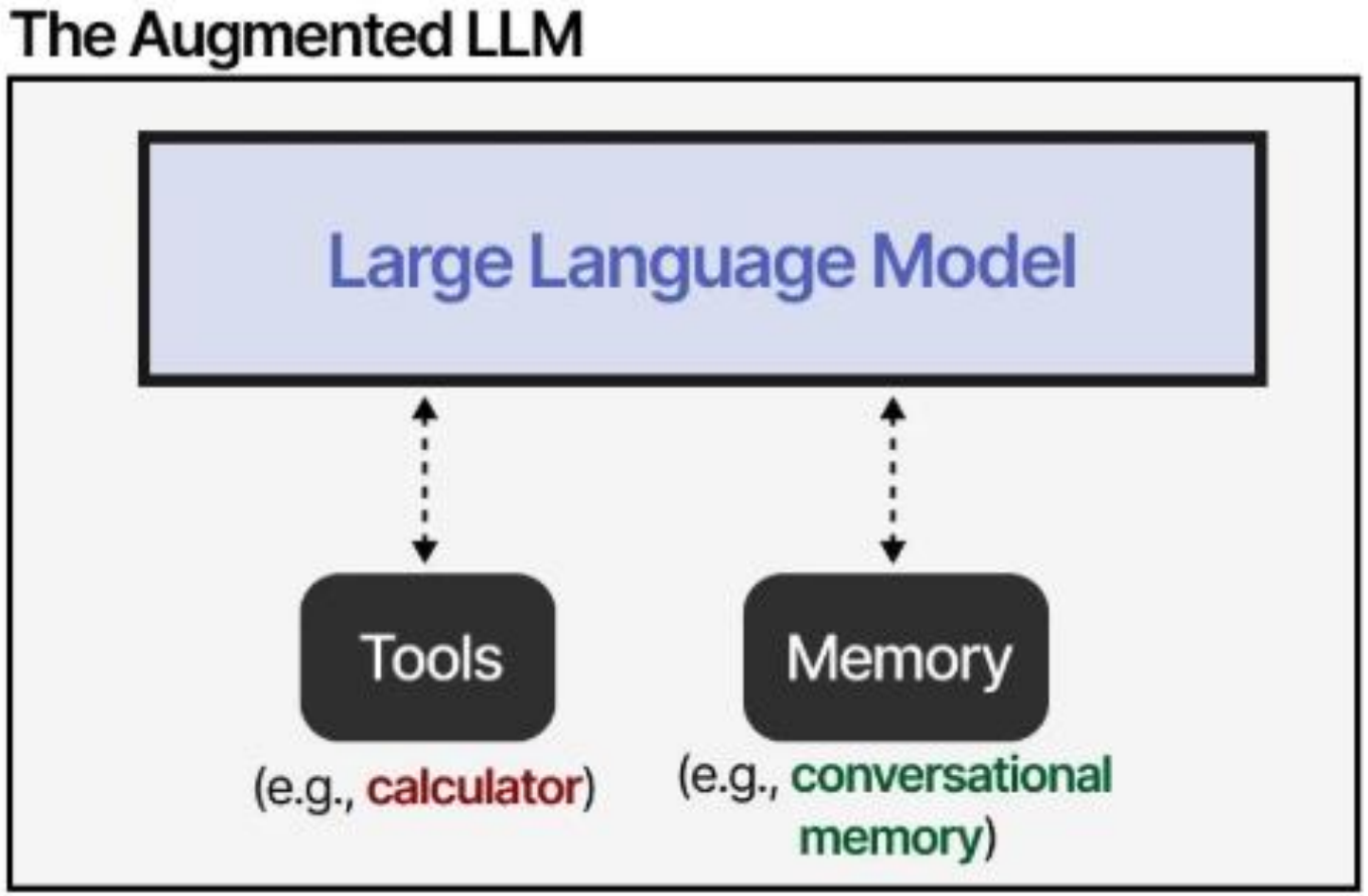
Model - LLM


Conversation
Question 1
What is 1 + 1?
LLM
The answer is 2!
Question 2
What was my previous question?
LLM
I have no knowledge about any previous question.

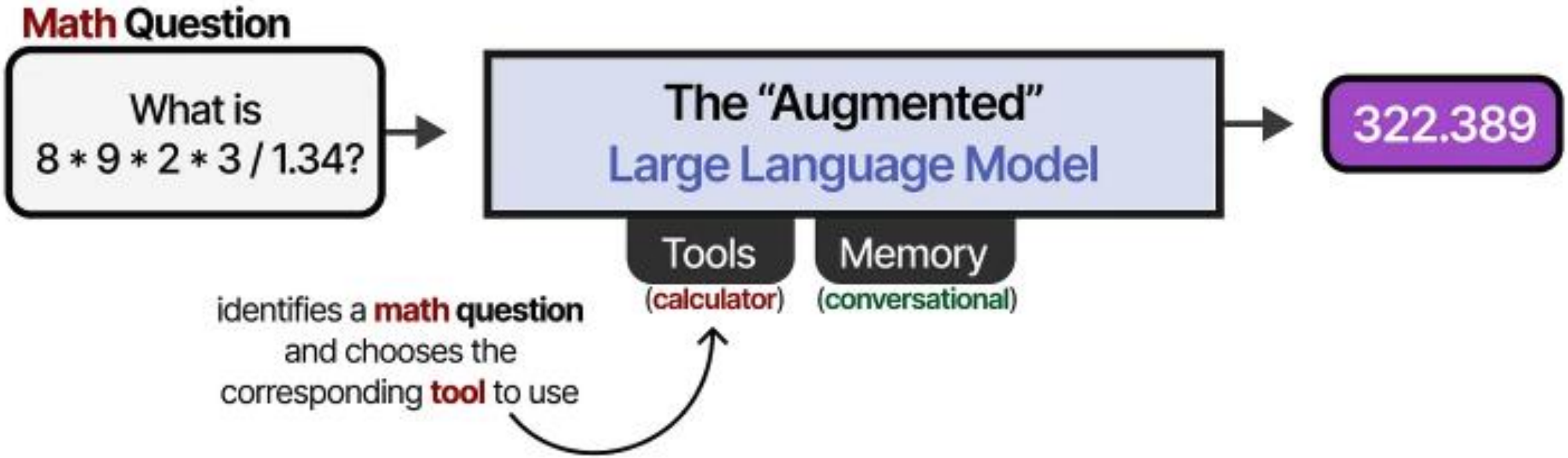
Model - LLM


Math Question
What is
8 * 9 * 2 * 3 / 1.34?
LLM
Token probability
320
319
311
309
312
incorrect!

Augmented LLM

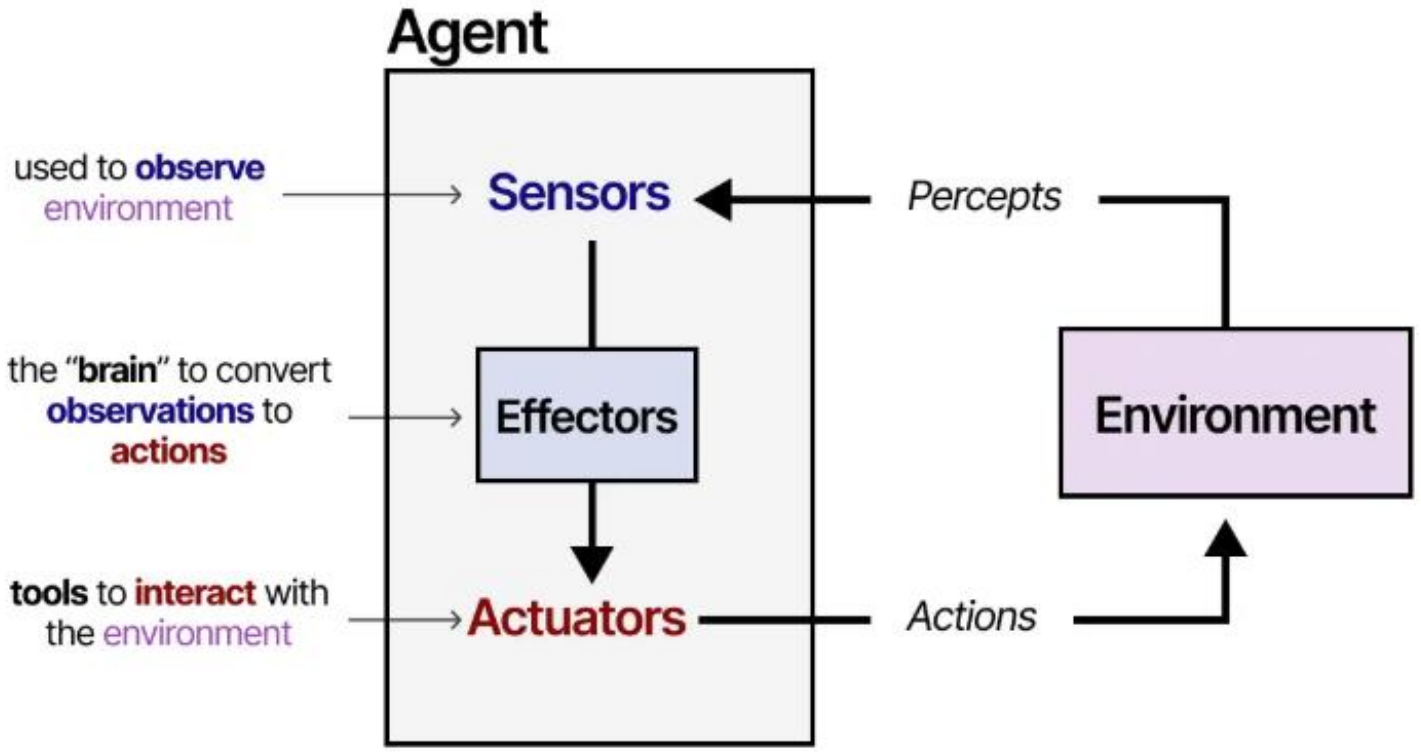


Augmented LLM



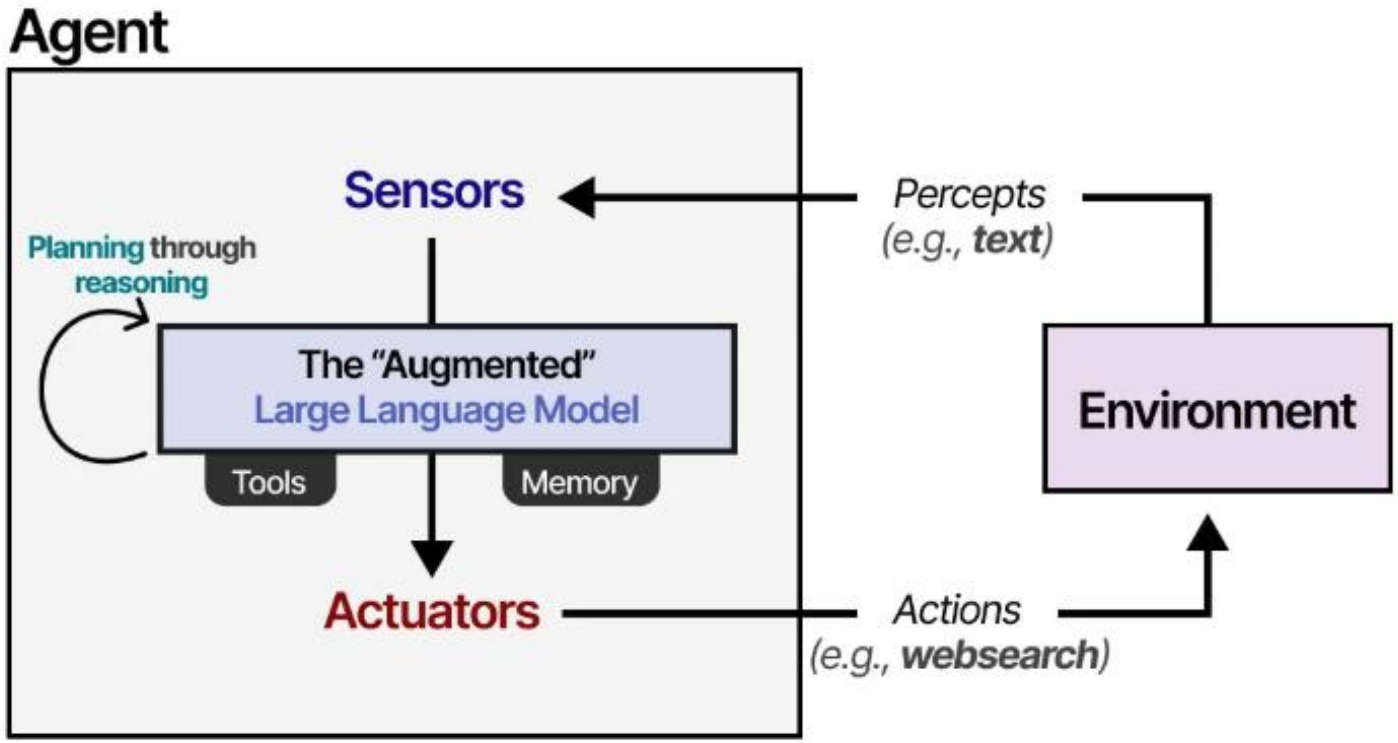
Agent

- Agents interact with their environment and typically consist of several important components:
- **Environments** — The world the agent interacts with.
- **Sensors** — Used to observe the environment.
- **Actuators** — Tools used to interact with the environment.
- **Effectors** — The “brain” or rules deciding how to go from observations to actions.



Agent

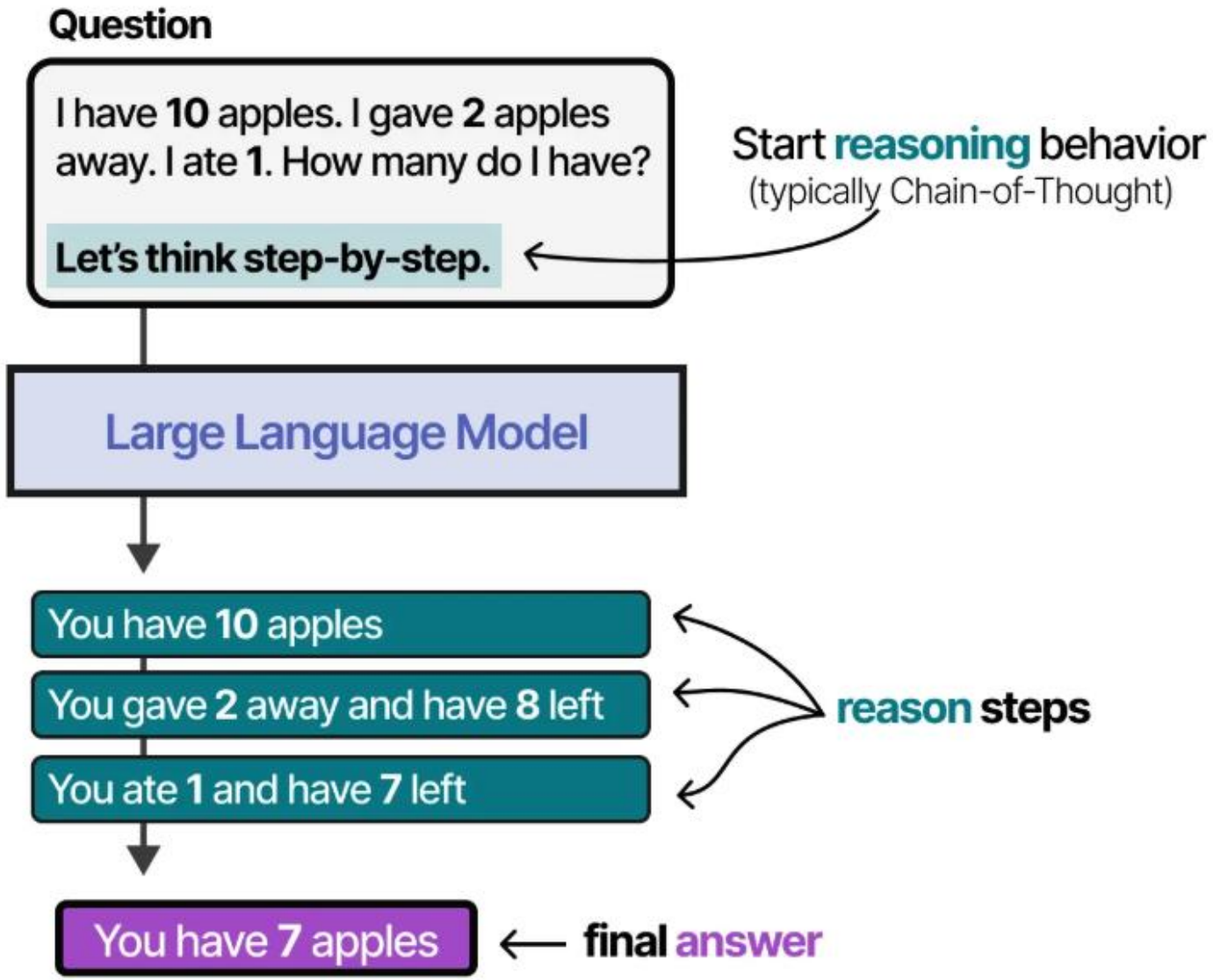
Using the “Augmented” LLM, the Agent can observe the environment through textual input (as LLMs are generally **textual models**) and perform certain actions through its use of tools (like **searching the web**).



Agent

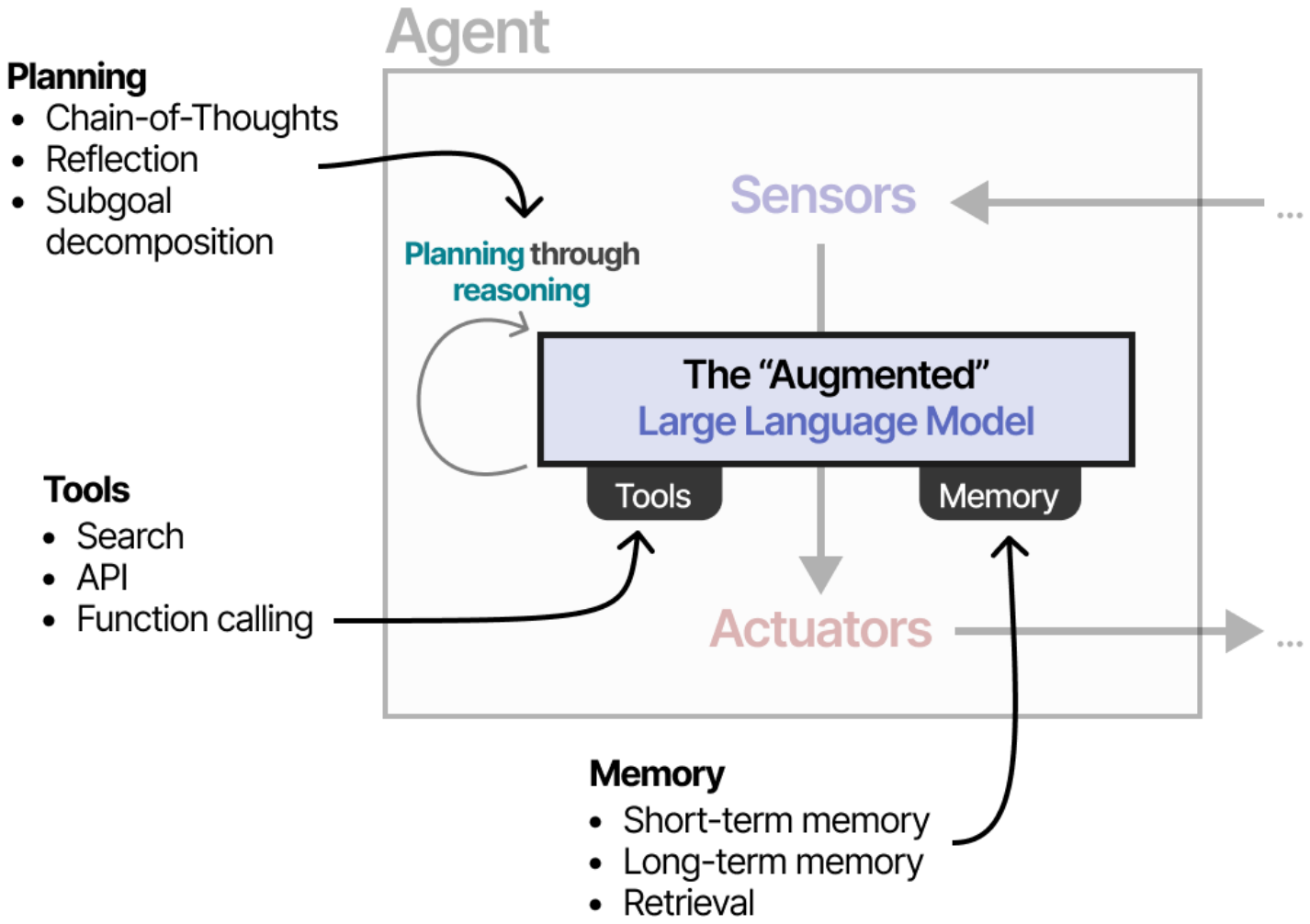
To select which actions to take, the LLM Agent has a vital component: its ability to plan.

For this, LLMs need to be able to “reason” and “think” through methods like **chain-of-thought.**

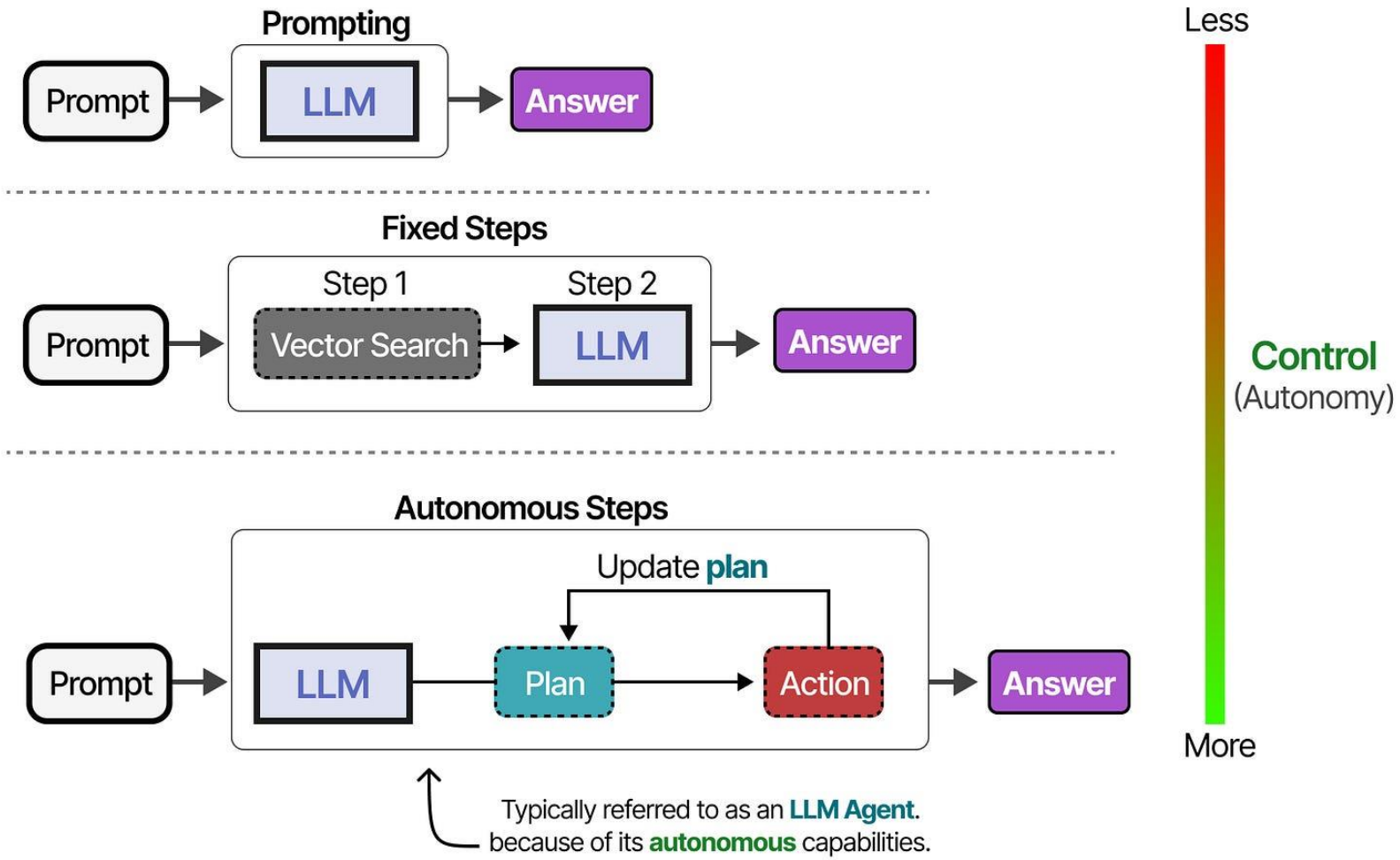


Agent

- This planning behavior allows the Agent to understand the situation (**LLM**), plan next steps (**planning**), take actions (**tools**), and keep track of the taken actions (**memory**).



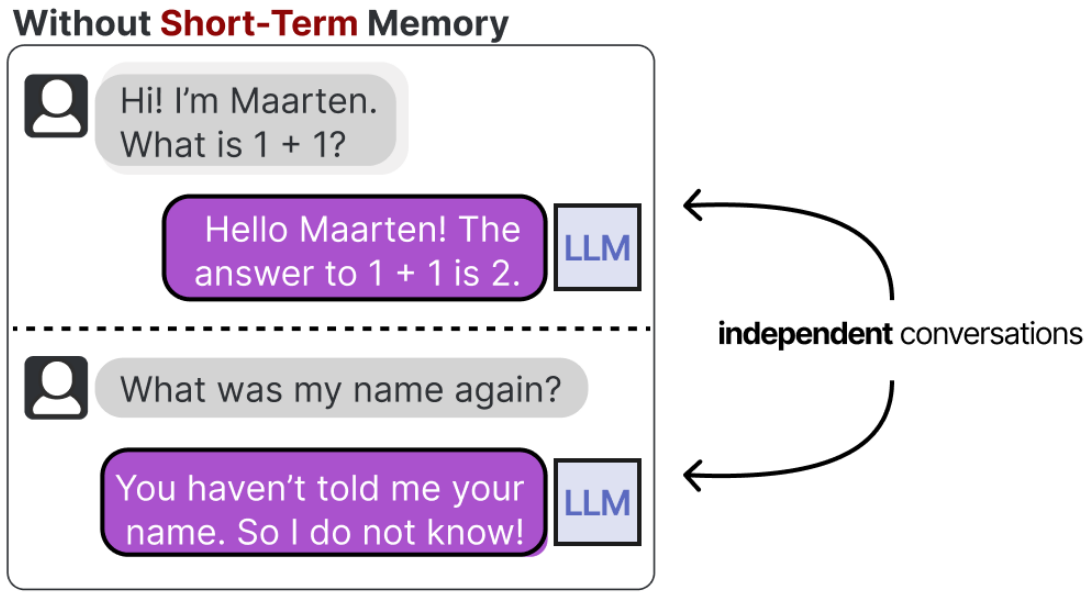
Agent



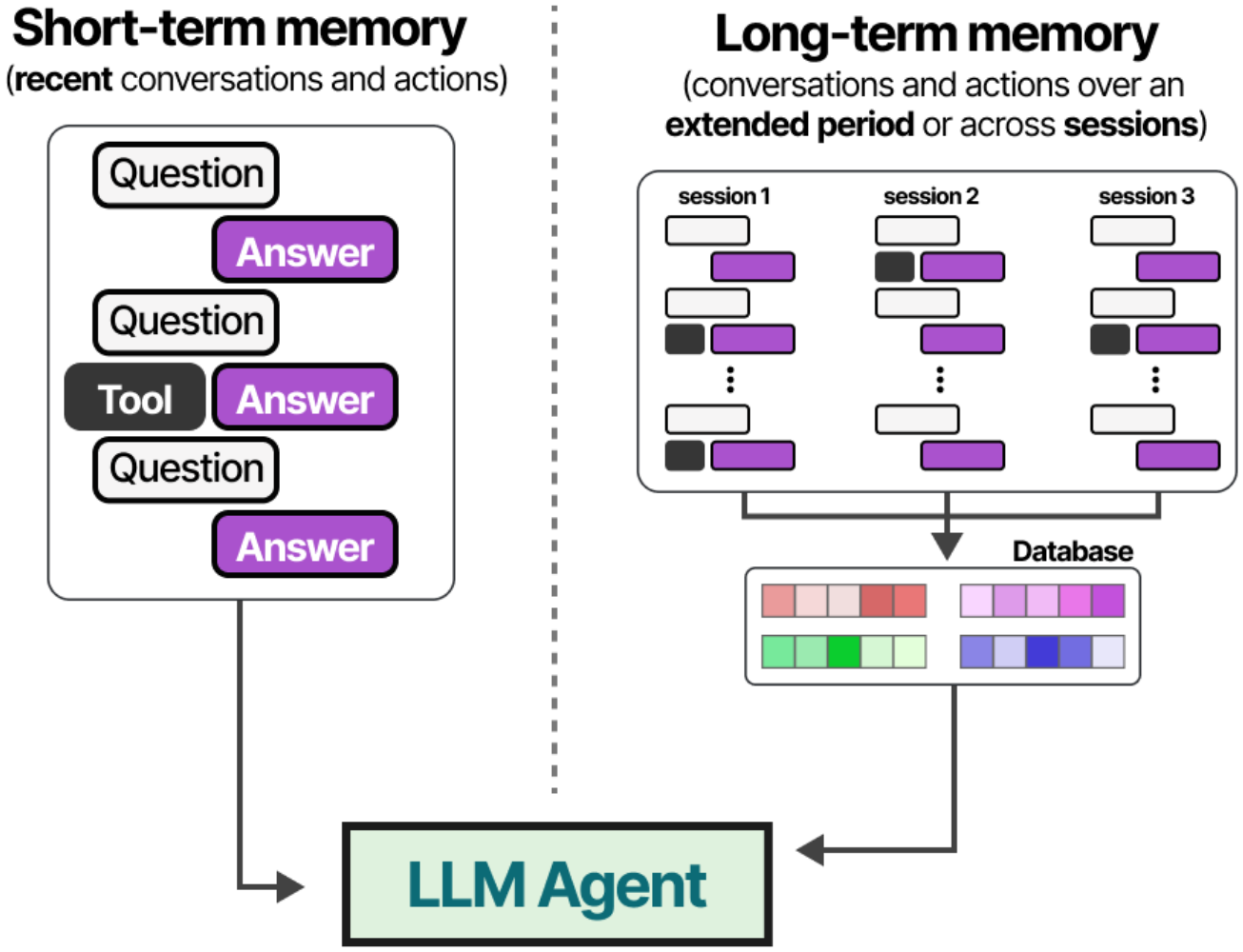
Depending on who you ask, a system is more “agentic” the more the LLM decides how the system can behave.

Memory

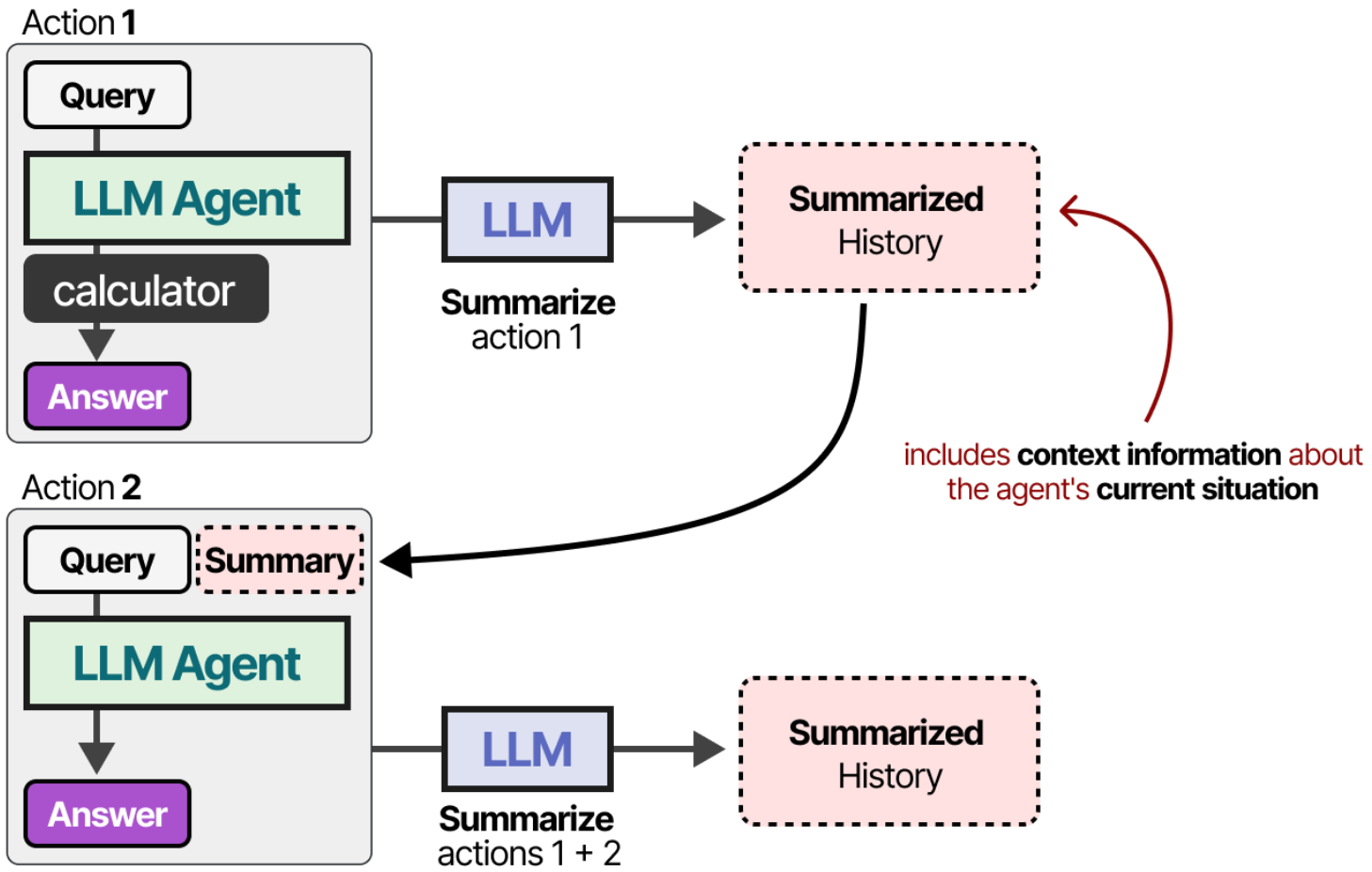
- We typically refer to this as **short-term memory**, also called working memory, which functions as a buffer for the (near-) immediate context. This includes recent actions the LLM Agent has taken.



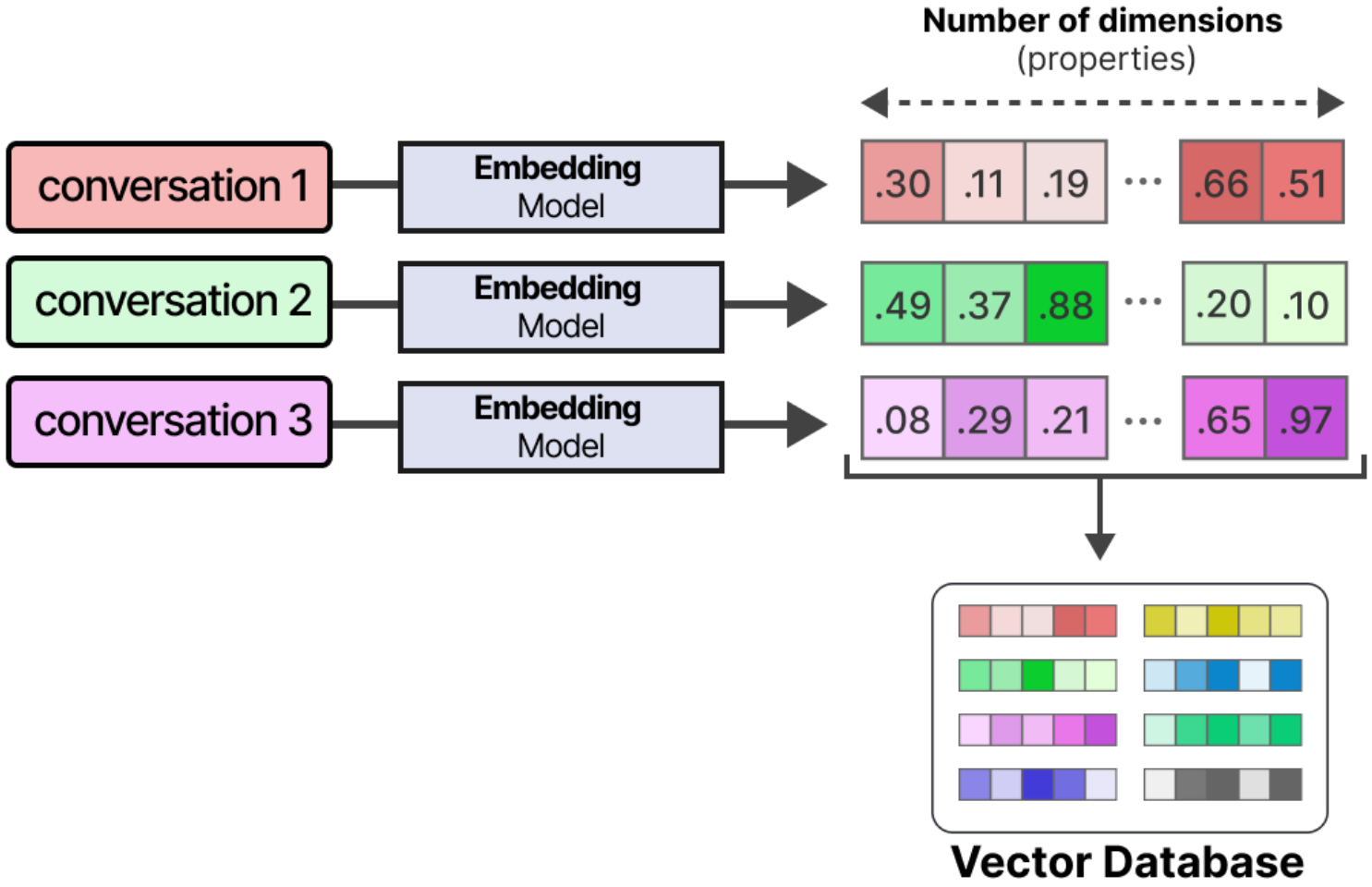
Memory


Short-term memory
(recent conversations and actions)
Question
Answer
Question
Tool
Answer
Question
Answer
Long-term memory
(conversations and actions over an extended period or across sessions)
session 1
session 2
session 3
Database
LLM Agent

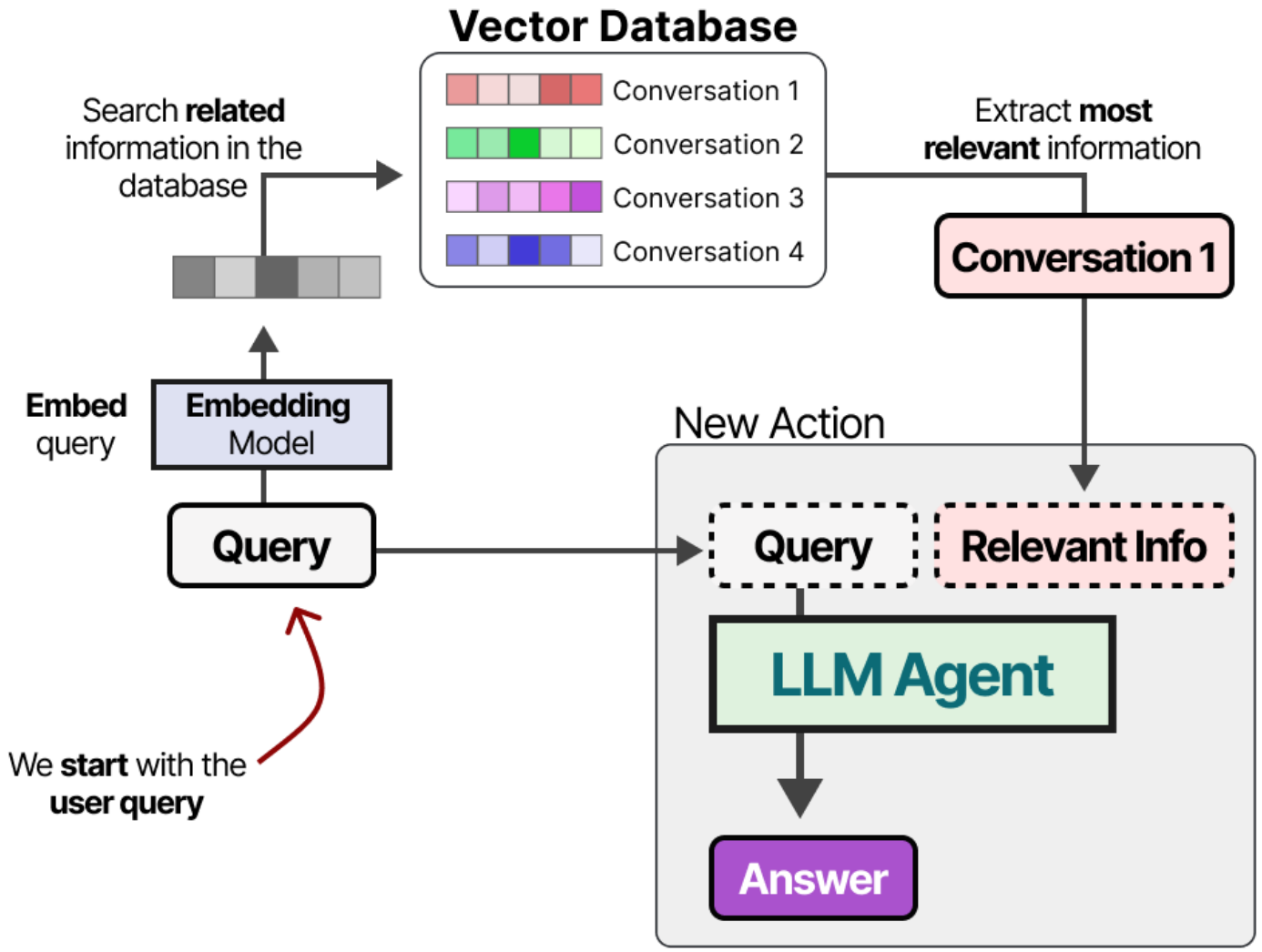
Short-term Memory



Long-term Memory



Long-term Memory

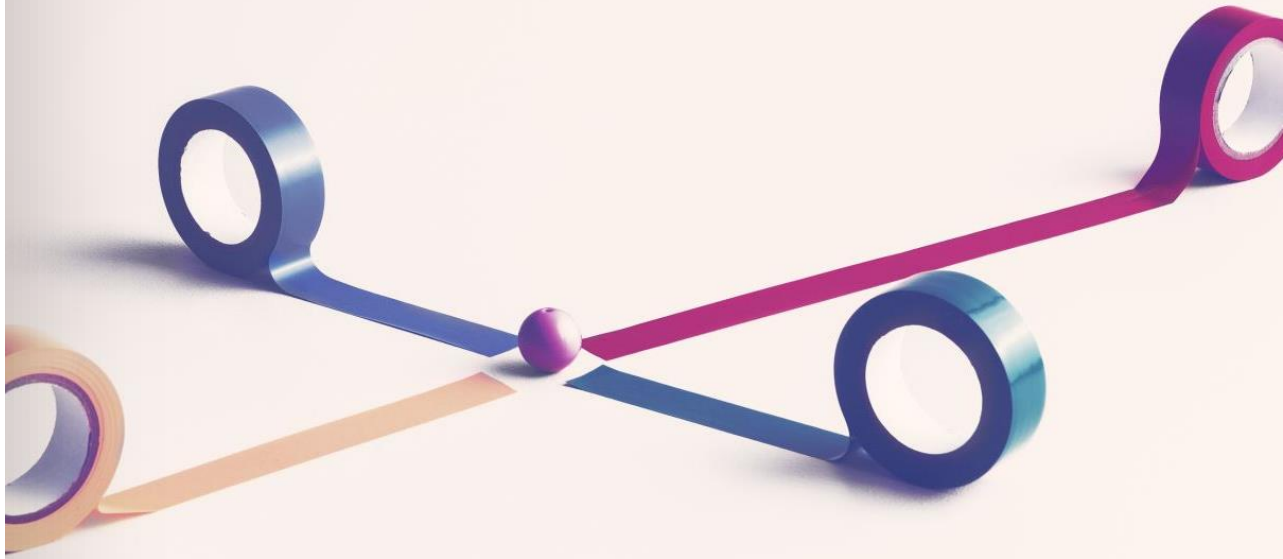


Tools

- Foundational models, despite their impressive text and image generation, remain **constrained** by their inability to interact with the outside world.

- Tools **bridge this gap**, empowering agents to interact with external data and services while unlocking a wider range of actions beyond that of the underlying model alone.

- In summary, tools bridge the gap between the agent's internal capabilities and the external world, unlocking a broader range of possibilities.

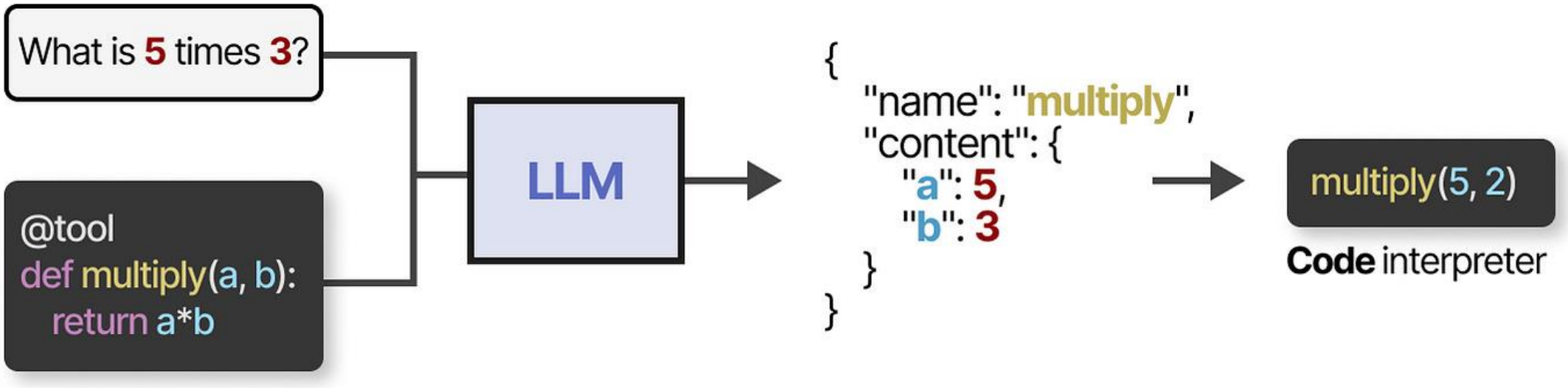


Tools

- Tools allow a given LLM to either interact with an external environment (such as databases) or use external applications (such as custom code to run).
- Tools generally have two use cases: fetching data to retrieve up-to-date information and taking action like setting a meeting or ordering food.

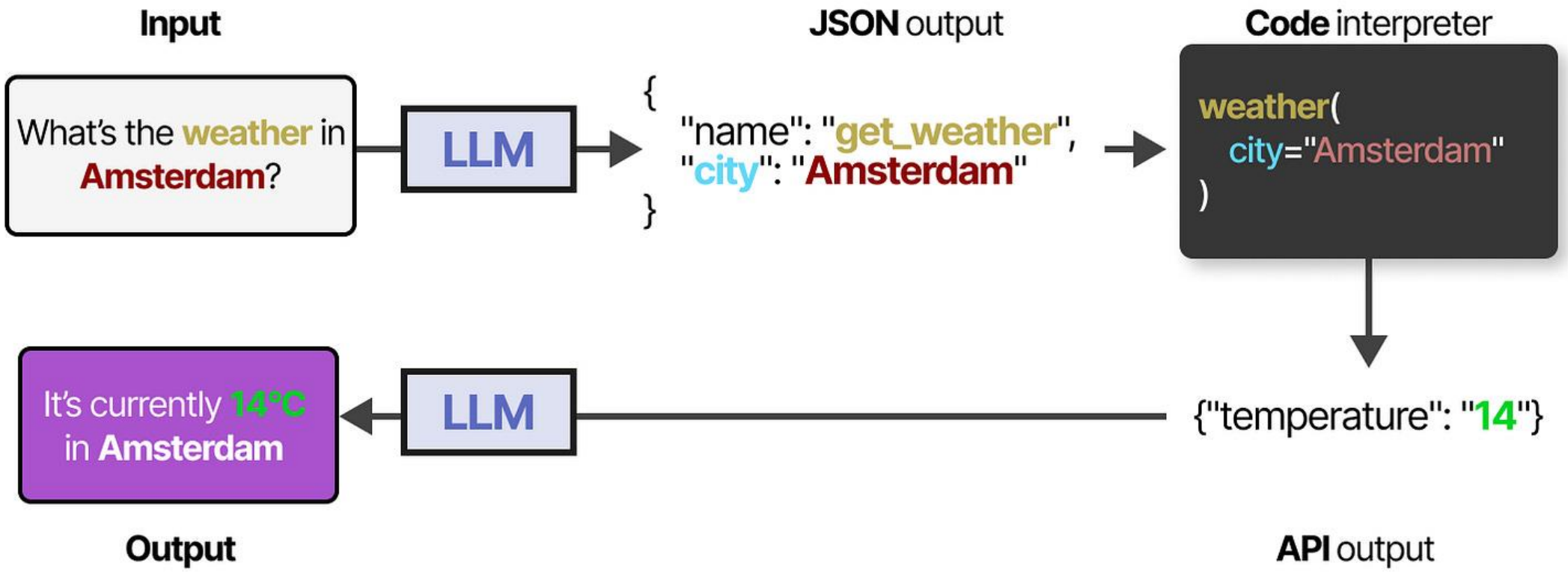
Tools - Function Calling

- You can also generate custom functions that the LLM can use, like a basic multiplication function.
- This is often referred to as **function calling.**



Tools

- The output of intermediate steps is fed back into the LLM to continue processing.



Model Context Protocol (MCP)

MCP is designed specifically for systems, addressing the unique challenges they face when interacting with external tools and data sources.

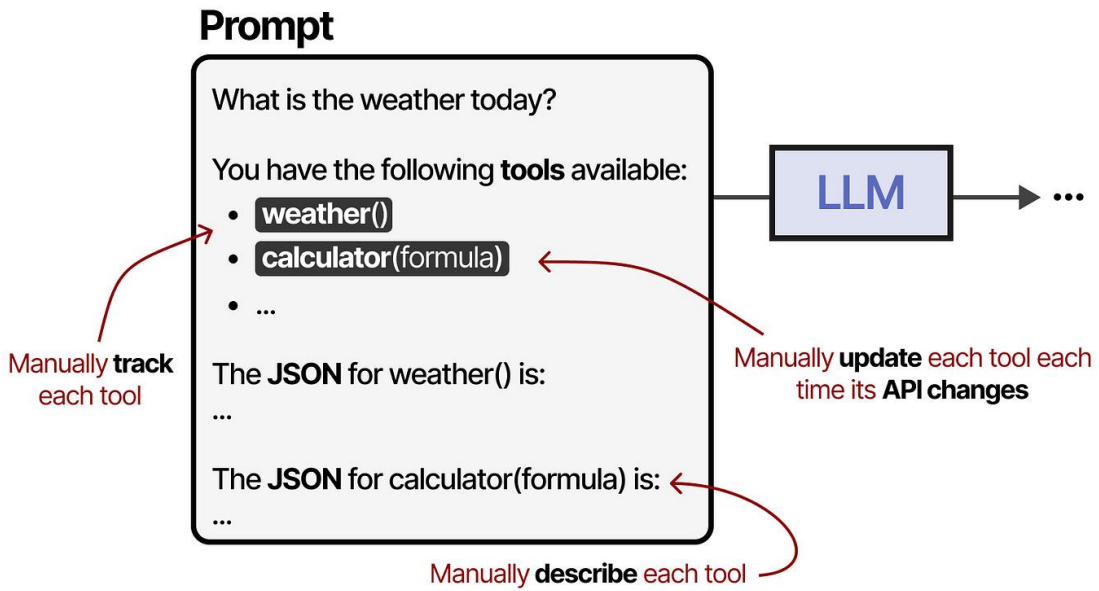
By providing a standardized, context-aware, and real-time communication protocol, MCP enables AI models to perform complex tasks more efficiently and adaptively.

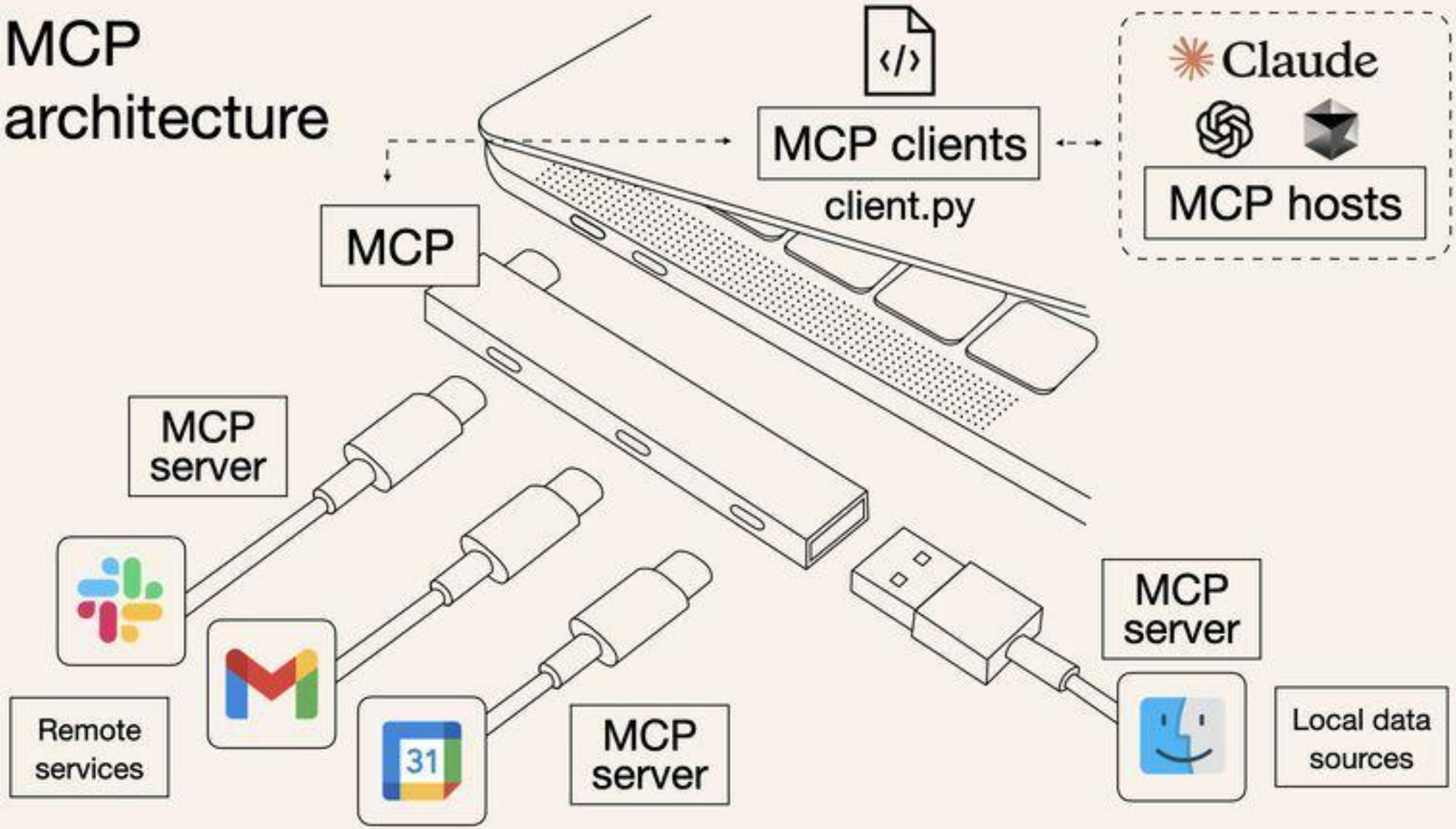
For instance, an AI assistant using MCP can autonomously discover and utilize tools like calendars, email clients, or databases, orchestrating multi-step workflows without the need for hard-coded integrations.

This flexibility is crucial for developing scalable and maintainable AI applications.

Tools – Model Context Protocol

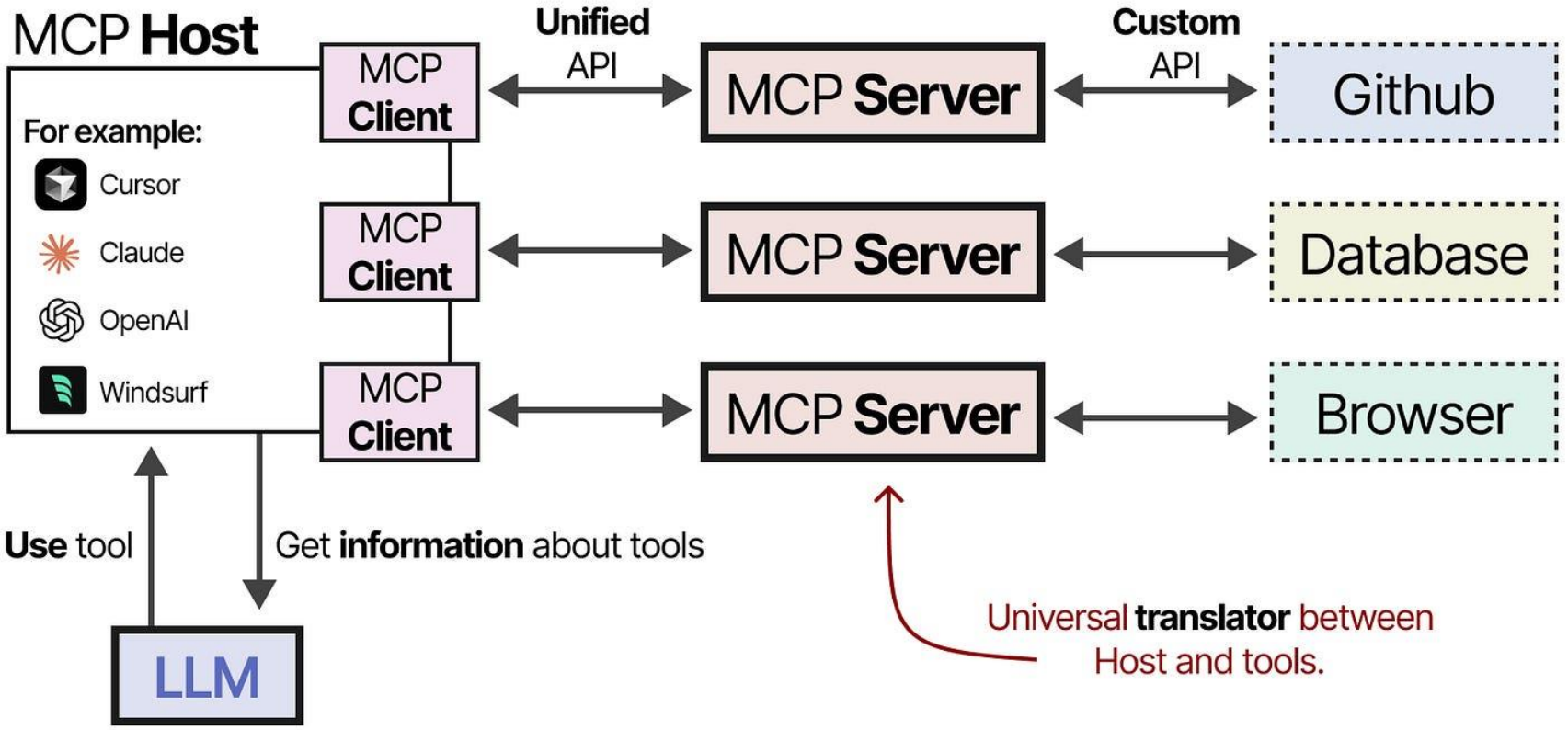
- Tools are an important component of Agentic frameworks, allowing LLMs to interact with the world and extend their capabilities.
- However, enabling tool use when you have many different API becomes troublesome as any tool needs to be:
  - Manually tracked and fed to the LLM
  - Manually described (including its expected JSON schema)
  - Manually updated whenever its API changes




MCP
architecture
MCP clients
client.py
Claude
MCP hosts
MCP
MCP
server
MCP
server
MCP
server
Remote
services
Local data
sources

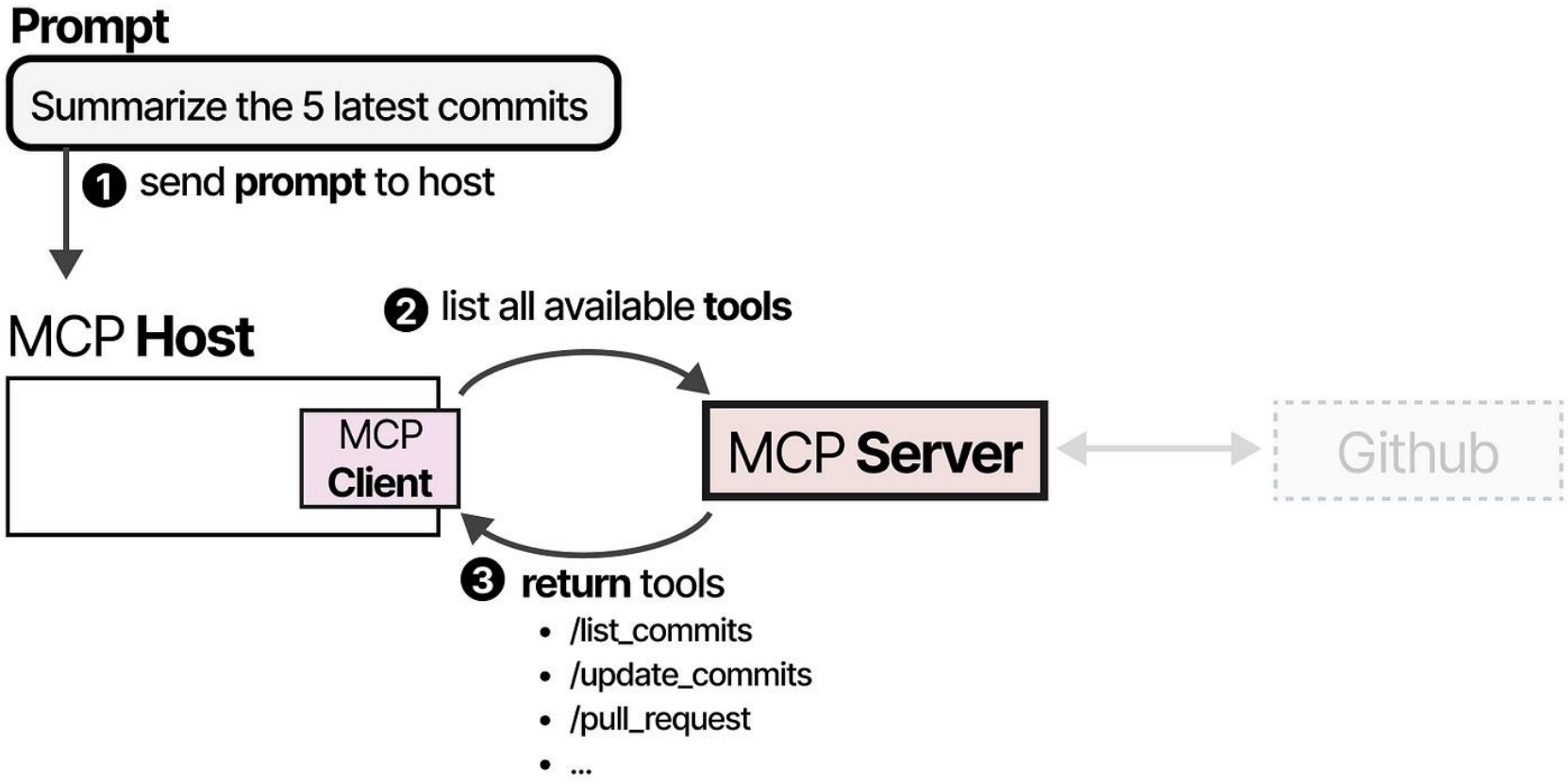
MCP

- MCP Host — LLM application (such as Cursor) that manages connections
- MCP Client — Maintains 1:1 connections with MCP servers
- MCP Server — Provides context, tools, and capabilities to the LLMs



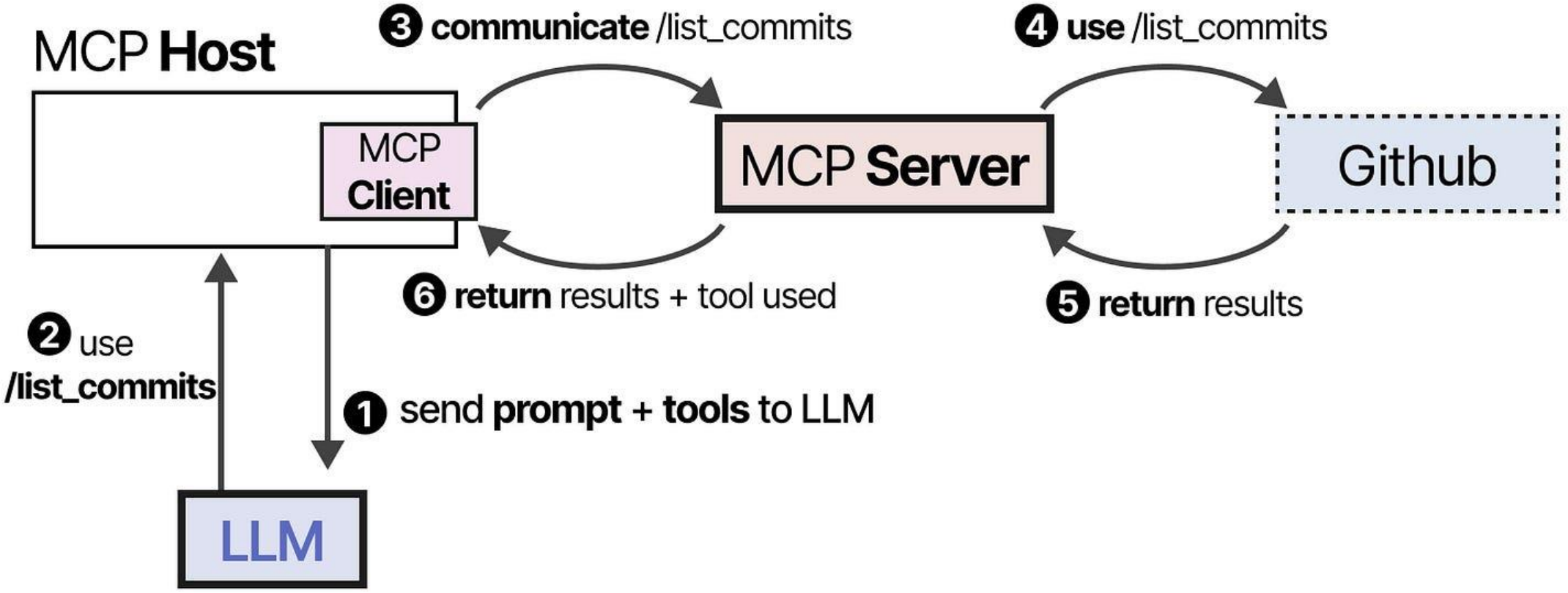
MCP

- For example, let's assume you want a given LLM application to summarize the 5 latest commits from your repository.
- The MCP Host (together with the client) would first call the MCP Server to ask which tools are available.



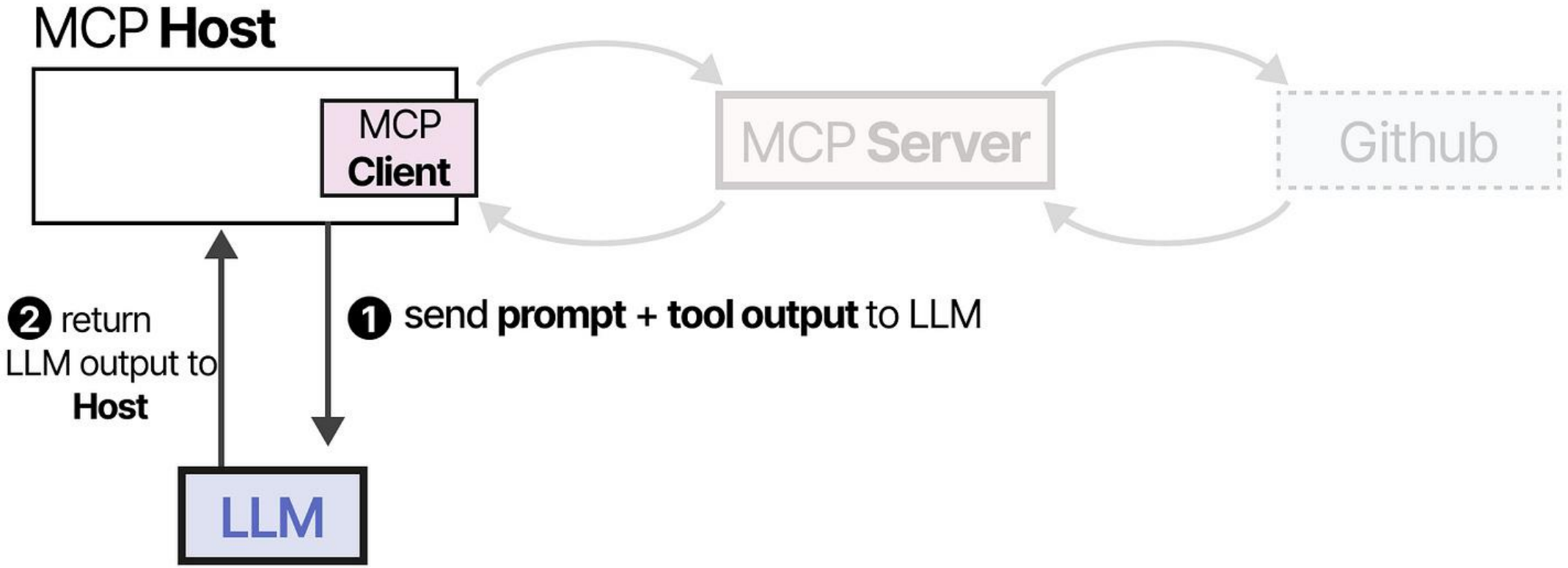
MCP

- The LLM receives the information and may choose to use a tool.
- It sends a request to the MCP Server via the Host, then receives the results, including the tool used.

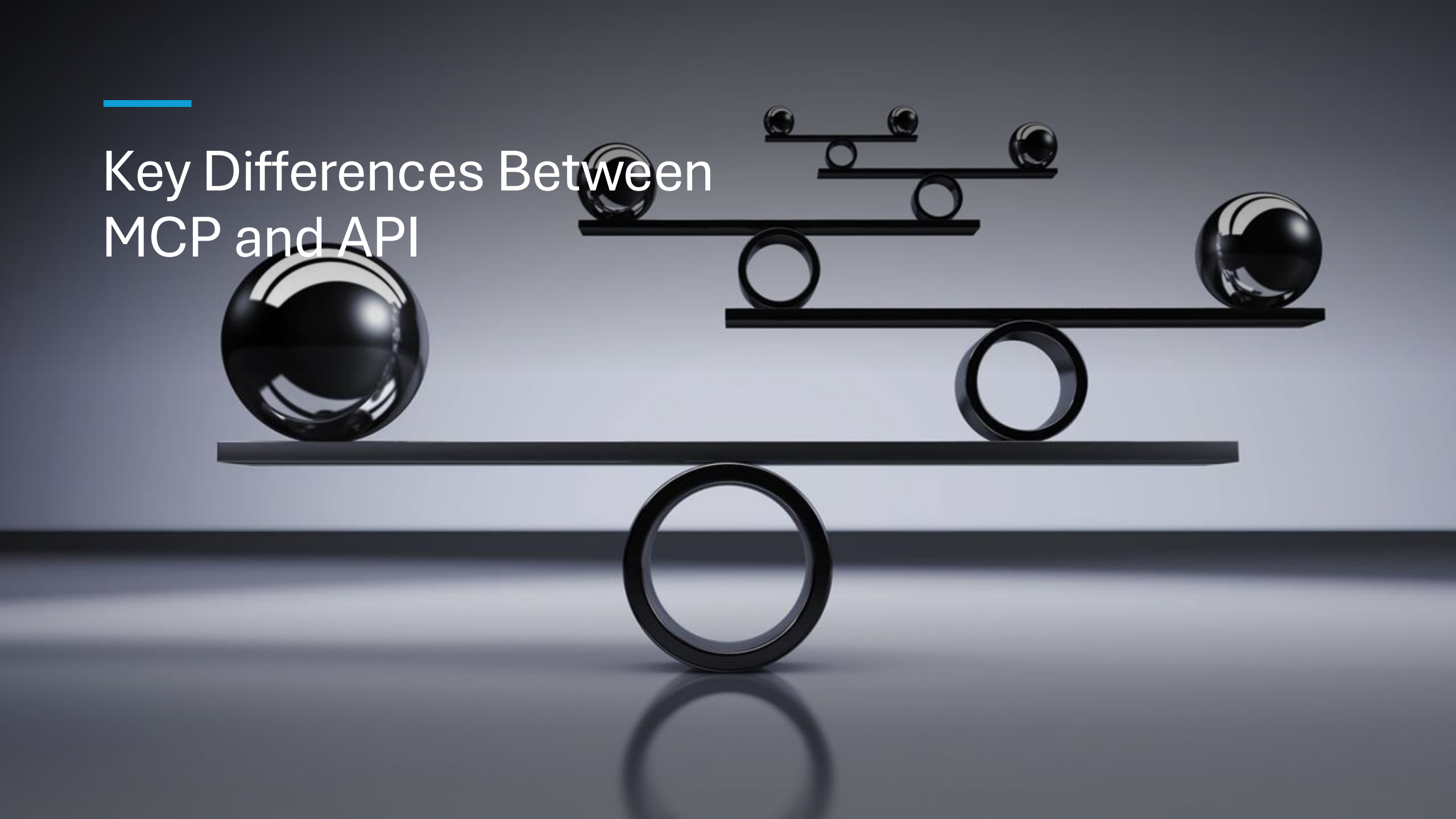


MCP

- Finally, the LLM receives the results and can parse an answer to the user.
- This framework makes creating tools easier by connecting to MCP Servers that any LLM application can use.
- So, when you create an MCP Server to interact with Github, any LLM application that supports MCP can use it.

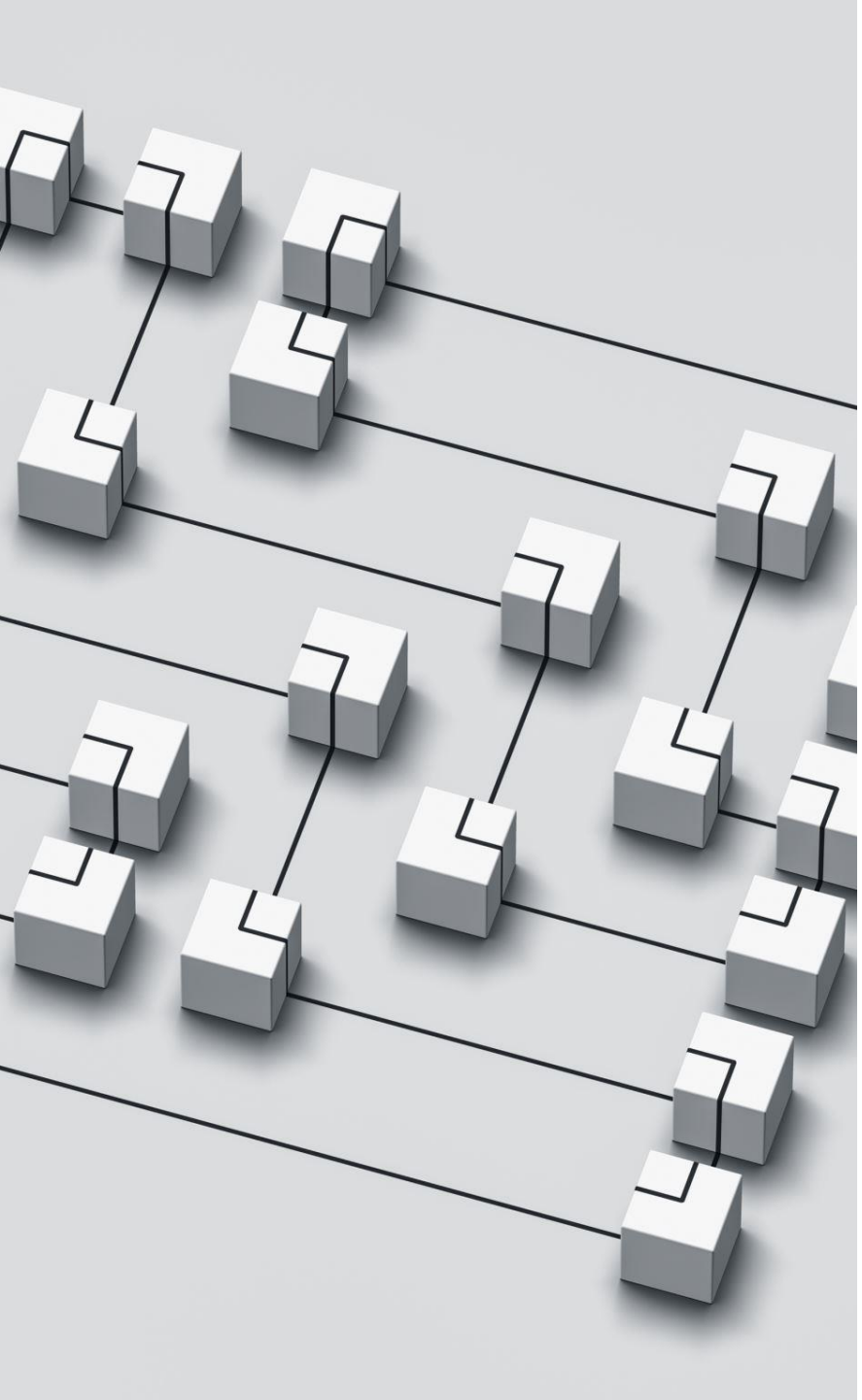


Key Differences Between MCP and API



Dynamic Tool Discovery

- **Traditional APIs**: Developers must manually configure and maintain knowledge of each API's capabilities.
- **MCP**: Enables AI models to dynamically discover available tools and resources at runtime. This self-describing nature allows models to adapt to new tools without requiring code changes.



Standardized Integration

- **Traditional APIs**: Each service requires a custom integration, with unique endpoints, authentication methods, and data formats. This leads to repetitive and time-consuming development efforts.

- **MCP**: Offers a unified protocol where AI models can interact with various tools and services through a single, standardized interface. Once an AI system supports MCP, it can seamlessly connect to any MCP-compliant server without additional custom coding.

Prompt Engineering

(2023 Era)

- Short task descriptions
- Basic instructions
- Simple API calls
- "ChatGPT Wrapper"

evolves to

Key Insights

- MCP standardizes AI-to-tool connections like USB-C
- Context engineering fills windows with precise info
- Too little → poor performance
- Too much → high cost & lag
- Protocol-driven architecture

Evolution Timeline

2023	Nov 2024	2025
"ChatGPT Wrapper"	MCP Released	Protocol-Driven AI

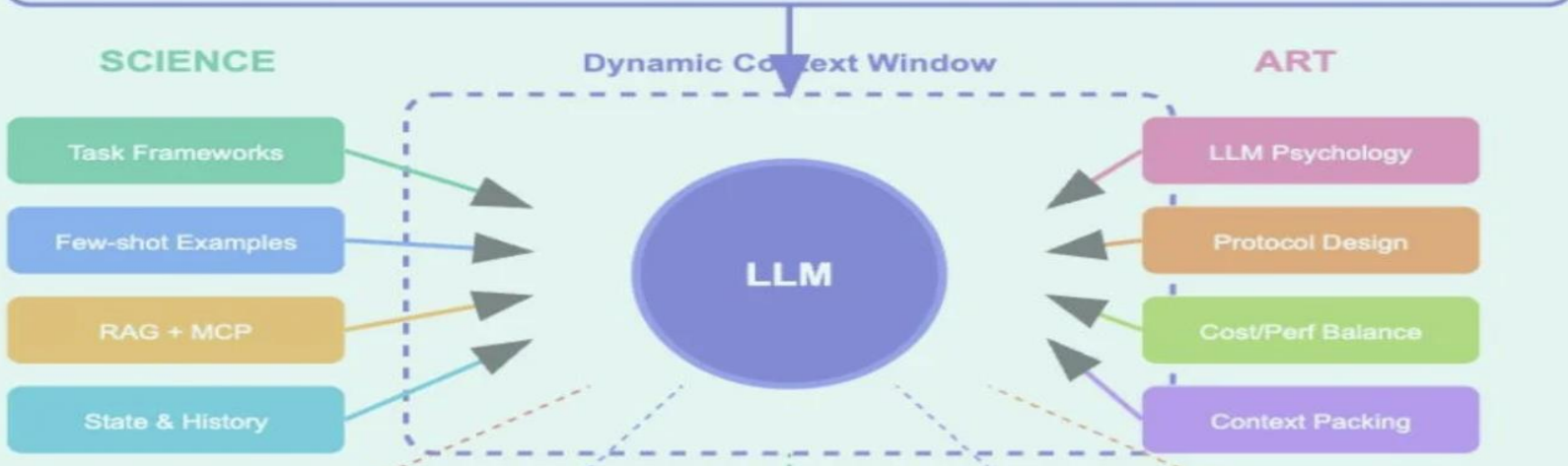
Context Engineering with MCP

(Protocol-Driven Agentic AI Era)

Model Context Protocol (MCP)

"Like a USB-C port for AI applications"

- Standardized connections
- Bidirectional data flow
- Security & permissions
- Tool discovery & invocation
- Universal protocol
- Industry adoption



MCP-Connected Tools & Data Sources

Google Drive	Slack	Postgres	GitHub	Custom APIs
Documents	Communications	Databases	Code & Issues	Internal Tools

Orchestration & Infrastructure Layer

"The thick layer of non-trivial software"

Control & Flow
- Problem decomposition
- Dependency graphs
- State machines
- Error recovery

Model & Routing
- Model selection
- Cost optimization
- Load balancing
- Fallback chains

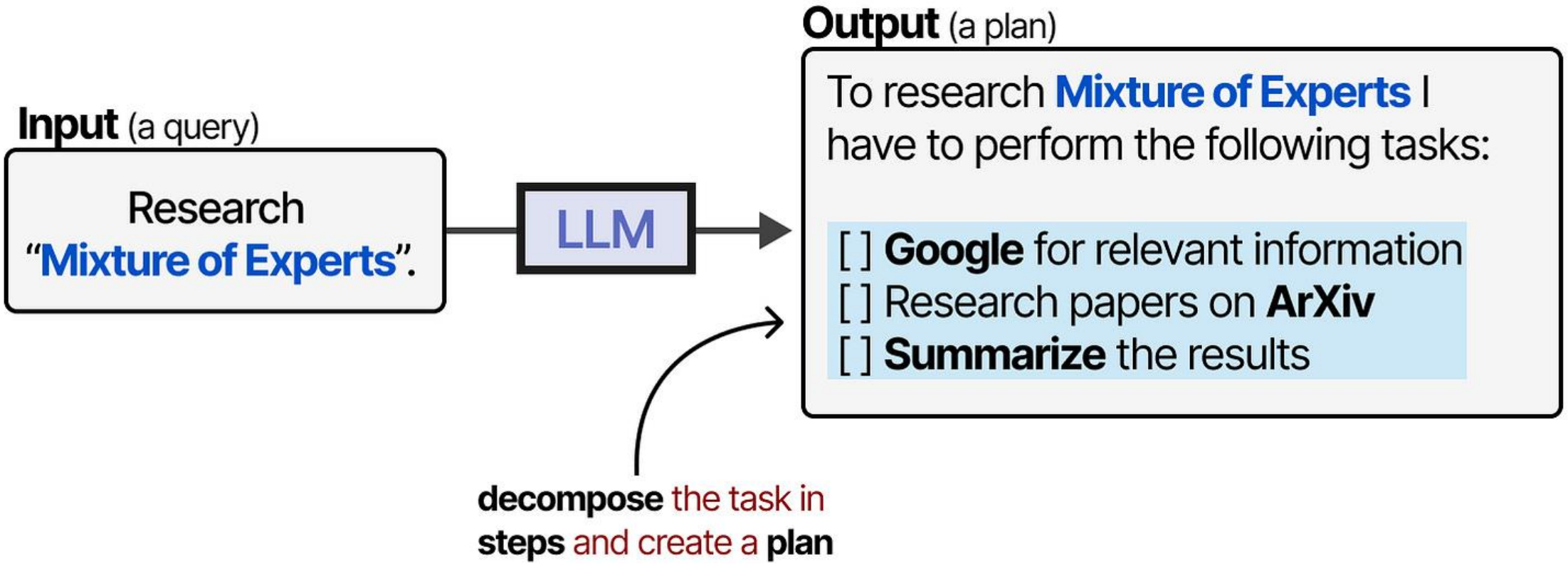
Security & Ops
- MCP permissions
- Audit trails
- Guardrails
- Monitoring

Performance
- Parallelism
- Prefetching
- Caching
- Token optimization

≠ "Claude Wrapper"

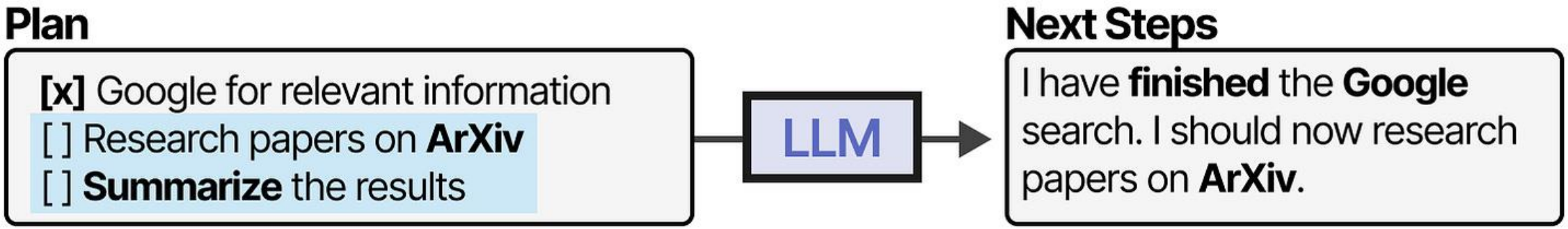
Planning

- Planning in LLM Agents involves breaking a given task up into actionable steps.



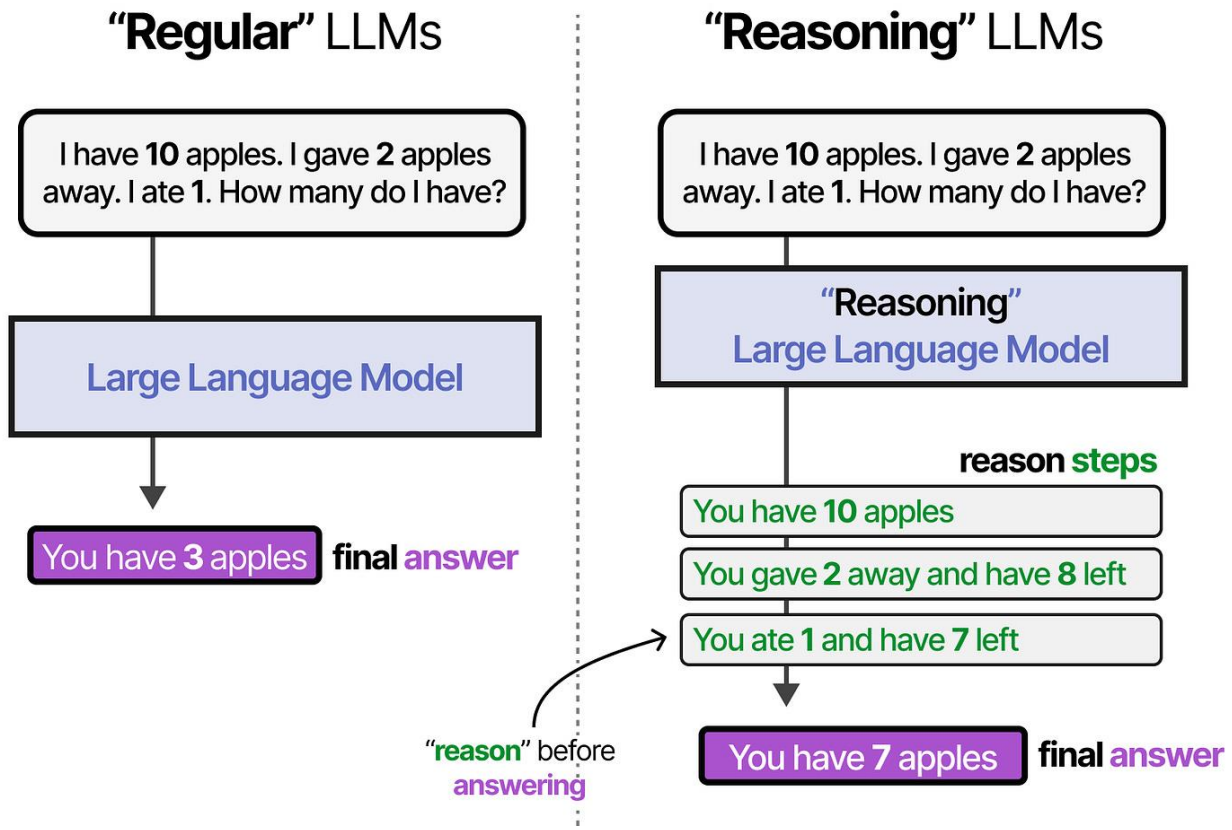
Planning

- This plan allows the model to iteratively reflect on past behavior and update the current plan if necessary.



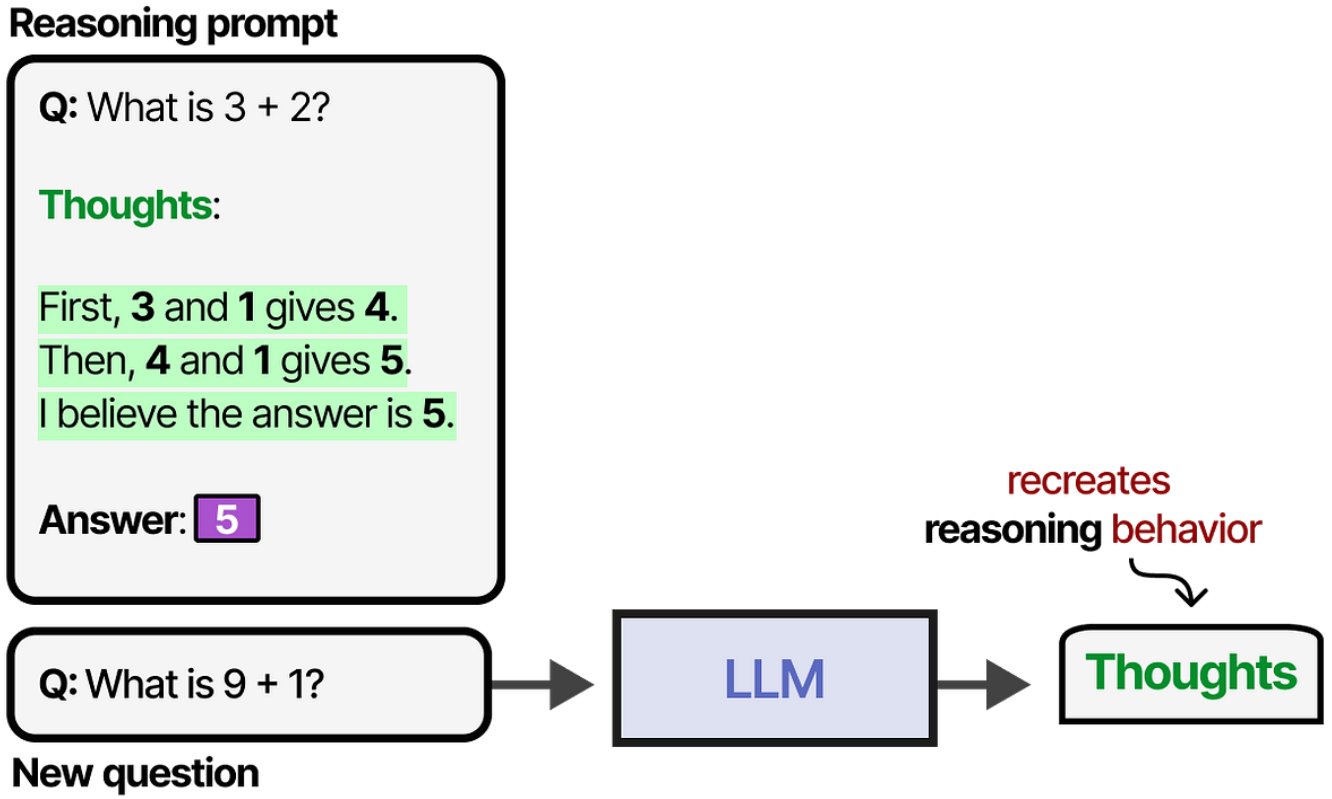
Reasoning

- Planning actionable steps requires complex reasoning behavior. As such, the LLM must be able to showcase this behavior before taking the next step in planning out the task.
- “Reasoning” LLMs are those that tend to “think” before answering a question.



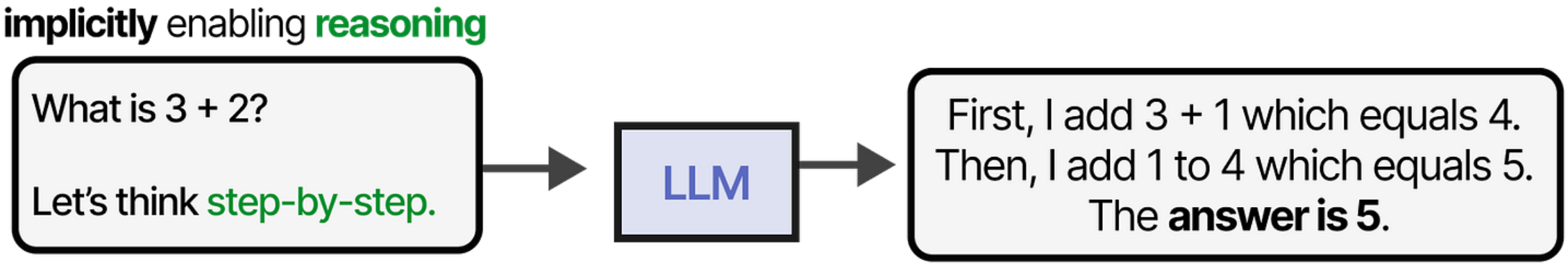
Reasoning - Prompt

- Providing examples (also called few-shot prompting) is a great method for steering the LLM's behavior.



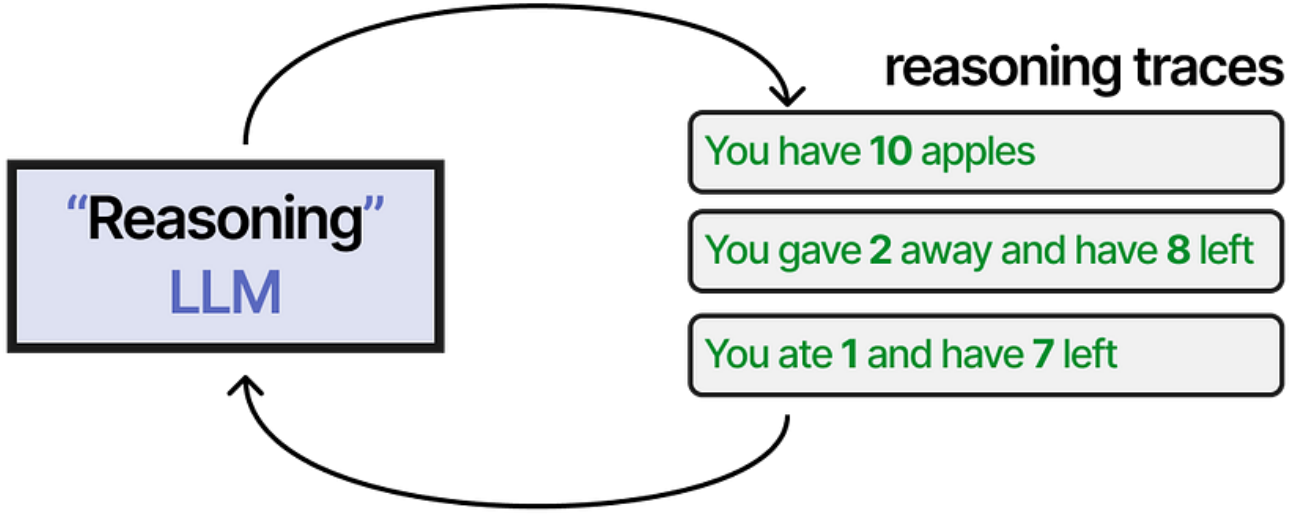
Reasoning - Prompt

- Chain-of-thought can also be enabled without any examples (zero-shot prompting) by simply stating “Let’s think step-by-step”.

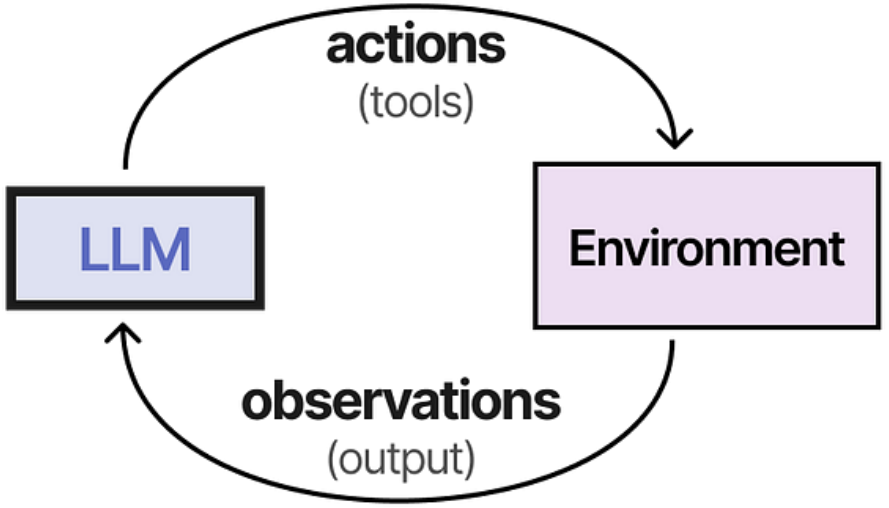


Reasoning and Acting

- Enabling reasoning behavior in LLMs is great but does not necessarily make it capable of planning **actionable** steps.
- The techniques we focused on thus far either showcase reasoning behavior or interact with the environment through tools.



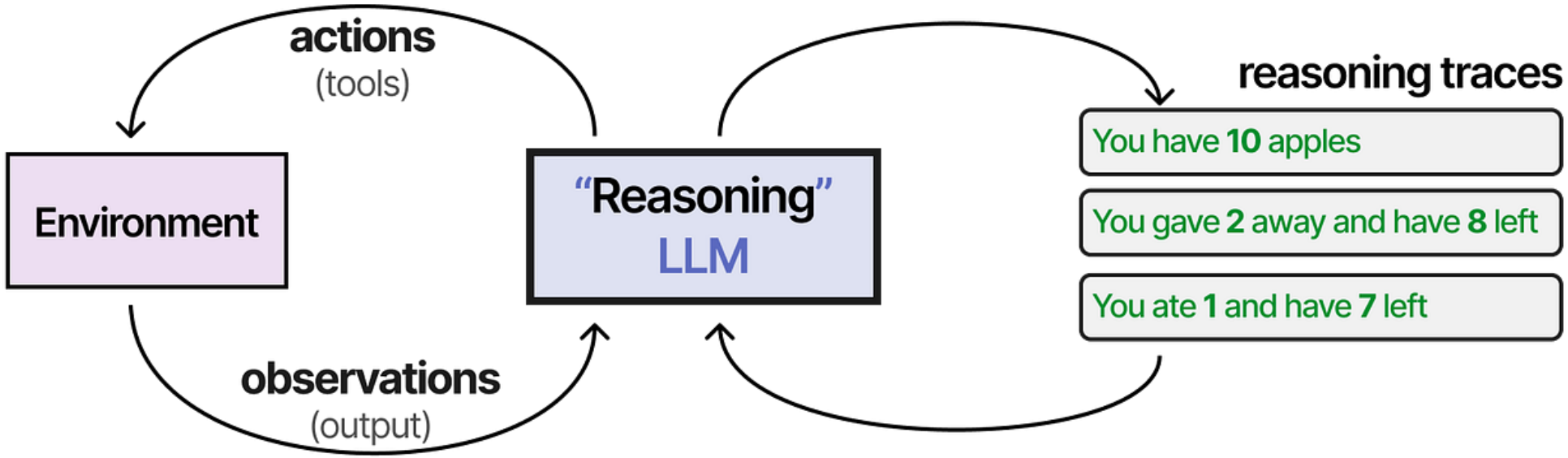
Reasoning Only
(e.g., chain-of-thought)



Act Only
(e.g., Toolformer)

ReAct

- Chain-of-Thought, for instance, is focused purely on reasoning.
- One of the first techniques to combine both processes is called ReAct (Reason and Act).



ReAct
(Reason + Act)

ReAct Framework - “**Re**asoning” (Think) with “**Act**ing” (Act)

- A **prompt engineering framework** that provides a **thought process strategy** for language models to Reason and take action on a user query, with or without in-context examples.

- Let’s consider an agent that is programmed to use the ReAct framework to choose the correct actions and tools for the user query.

ReAct Framework – Sequence of Events

- User sends query to the agent
- Agent begins the ReAct sequence
- The agent provides a prompt to the model, asking it to generate one of the next ReAct steps and its corresponding output:

1. **Question**: The input question from the user query, provided with the prompt.

2. **Thought**: The model's thoughts about what it should do next.

3. **Action**: The model's decision on what action to take next (This is where tool choice can occur).

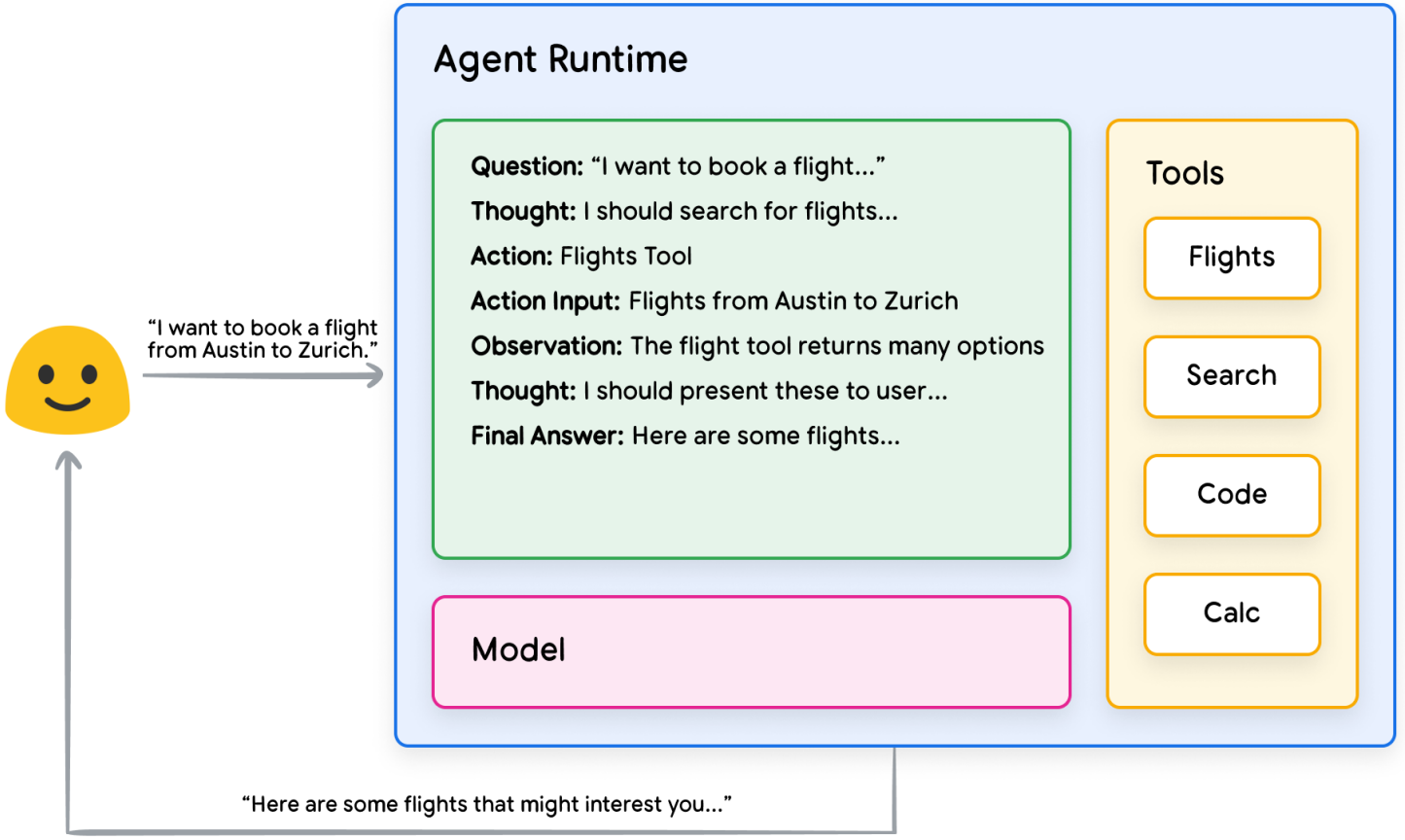
   For example, an action could be one of [Flights, Search, Code, None], where the first 3 represent a known tool that the model can choose, and the last represents "no tool choice".

ReAct Framework – Sequence of Events

- The agent provides a prompt to the model, asking it to generate one of the next ReAct steps and its corresponding output:
  1. **Question**: The input question from the user query, provided with the prompt
  2. **Thought**: The model's thoughts about what it should do next
  3. **Action**: The model's decision on what action to take next
  4. **Action input**: The model's decision on what inputs to provide to the tool (if any)
  5. **Observation**: The result of the action / action input sequence
     - This thought / action / action input / observation could repeat N-times as needed
  6. **Final answer**: The model's final answer to provide to the original user query.

ReAct Framework

The ReAct loop concludes, and a final answer is provided back to the user



ReAct Framework

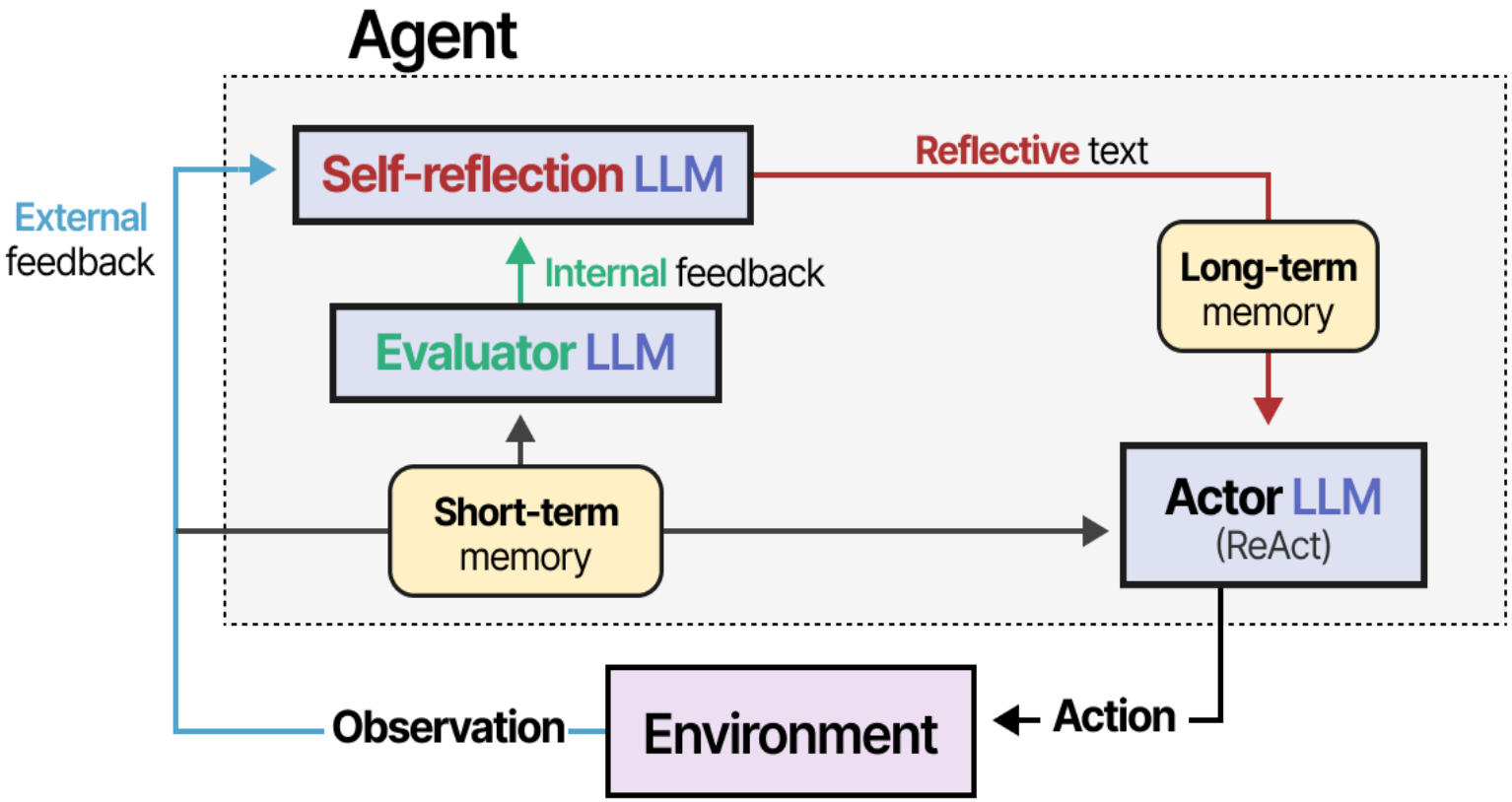
- The model, tools, and agent configuration **work together** to provide a grounded, concise response back to the user based on the user's original query.

- While the model could have guessed at an answer (hallucinated) based on its prior knowledge, it instead used a **tool** (Flights) to search for real-time external information.

- In summary, the **quality** of agent responses can be tied directly to the model's ability to reason and act about these various tasks, including the ability to select the right tools, and how well that tools has been defined.

Reflecting

- Nobody, not even LLMs with ReAct, will perform every task perfectly.
- Failing is part of the process as long as you can reflect on that process.
- This process is missing from ReAct and is where Reflexion comes in.
- Reflexion is a technique that uses verbal reinforcement to help agents learn from prior failures.
- Memory modules are added to track actions (short-term) and self-reflections (long-term), helping the Agent learn from its mistakes and identify improved actions.

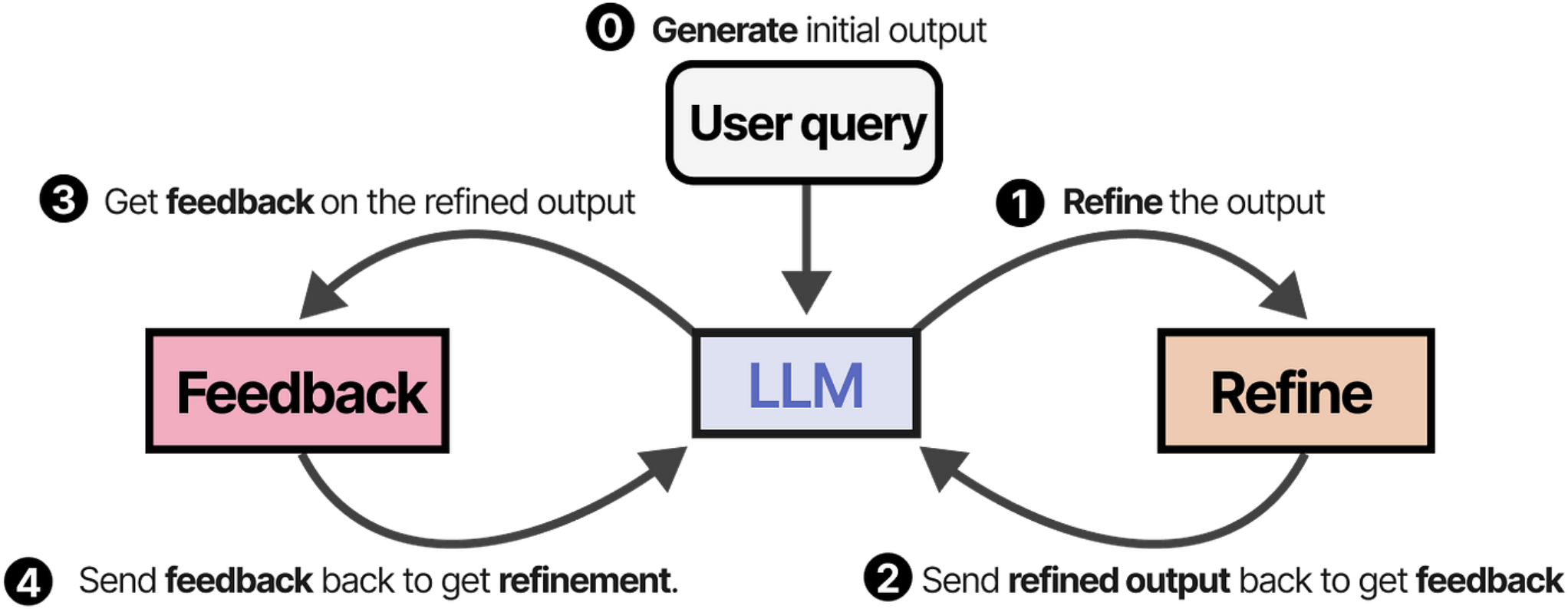


The method assumes three LLM roles:

- **Actor** — Chooses and executes actions based on state observations. We can use methods like Chain-of-Thought or ReAct.
- **Evaluator** — Scores the outputs produced by the Actor.
- **Self-reflection** — Reflects on the action taken by the Actor and scores generated by the Evaluator.

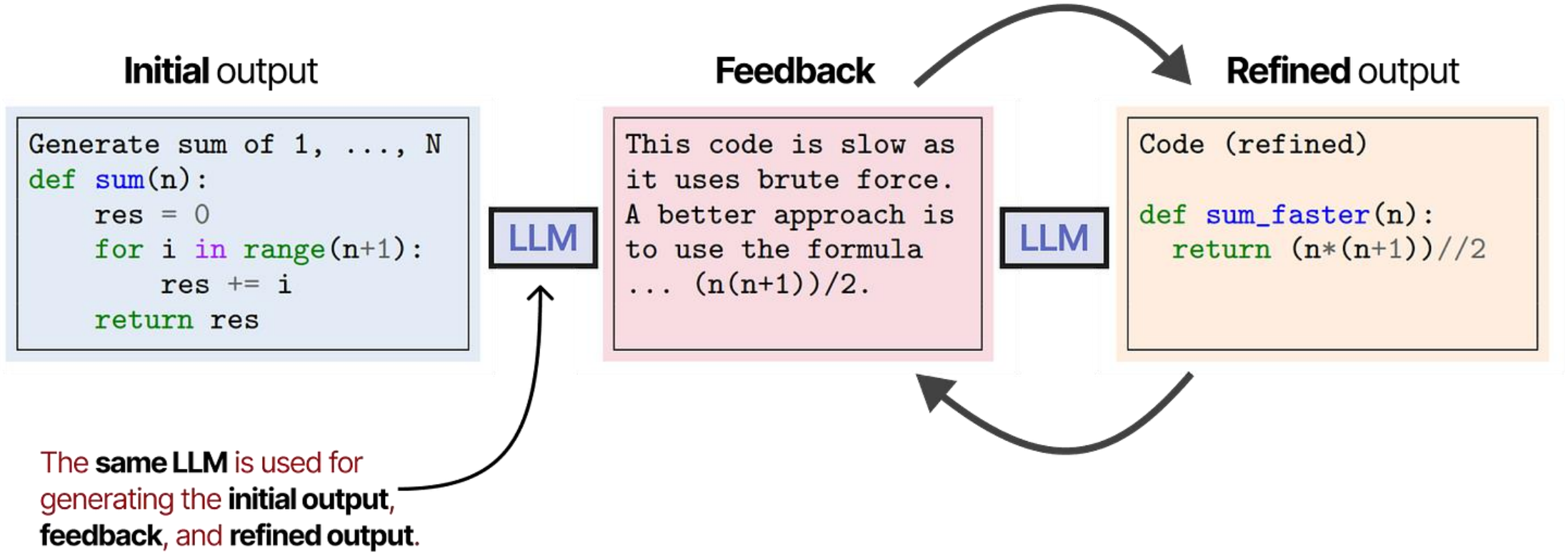
Reflecting

- A similar and elegant technique is called SELF-REFINE, where actions of refining output and generating feedback are repeated



Reflecting

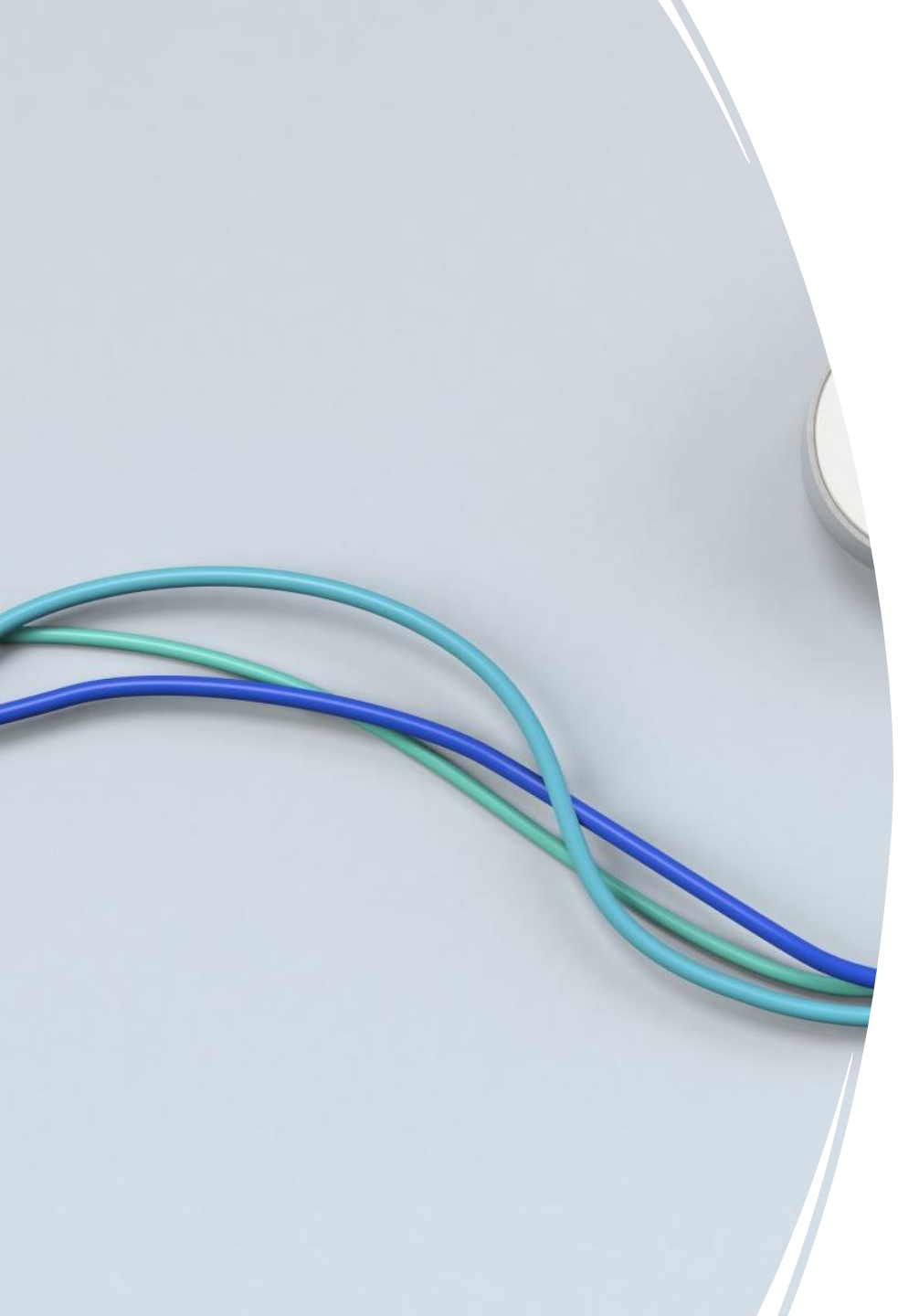
- The same LLM is in charge of generating the initial output, the refined output, and feedback.
- Interestingly, this self-reflective behavior, both Reflexion and SELF-REFINE, closely resembles that of reinforcement learning where a reward is given based on the quality of the output.



Move towards Agentic Models

Function Calling – Pushing actions to models

- Function-calling is a way for an **LLM to take actions** on its environment.
- Just like the tools of an Agent, function-calling gives the model the capacity to take an action on its environment. However, the function calling capacity is **learned by the model**, and **relies less on prompting** than other agents' techniques.
- We observed that the **Agent didn't learn to use the Tools**, we just provided the list, and we relied on the fact that the model was able to generalize on defining a plan using these tools.
- While here, **with function-calling**, the **Agent is fine-tuned (trained) to use Tools**.



Function calling and Tools

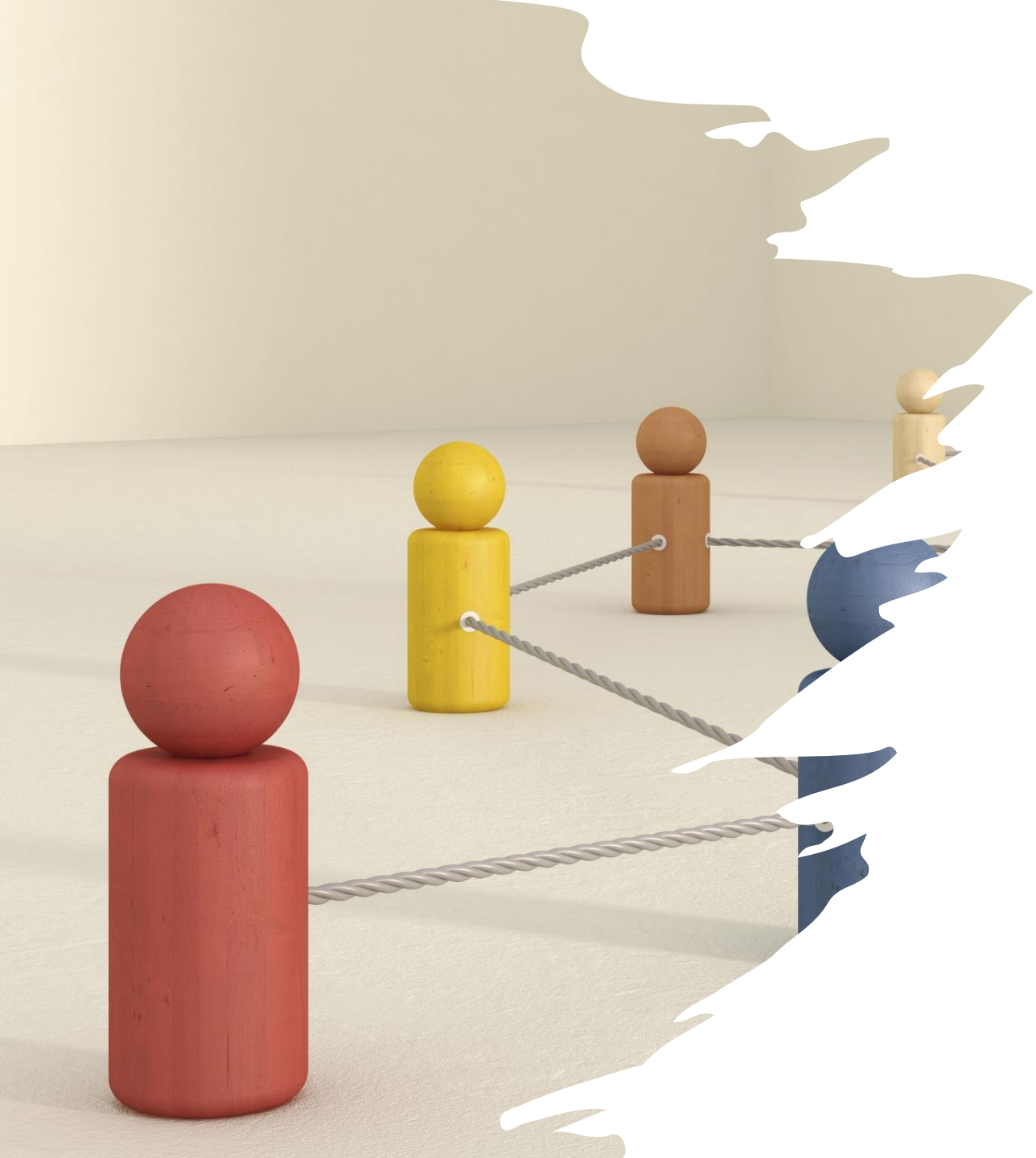
- Function calling is a **built-in capability** in LLMs that allows them to **invoke specific functions** with **structured outputs**.
- It is typically **used for**:
  - Structured data retrieval (e.g., calling a weather API and getting JSON output).
  - Processing user input and routing it correctly to different backend functions.
  - Enabling AI agents to **perform tasks dynamically** based on real-time user interactions.

Tools (Plugins or External APIs)

- Tools are **external systems, APIs, or integrations** that the AI model can call.
- They provide **extended capabilities** beyond what the model can do on its own.
- Tools often include:
  - Web search (for real-time information).
  - Code execution environments.
  - Third-party API integrations (e.g., databases, SaaS products).

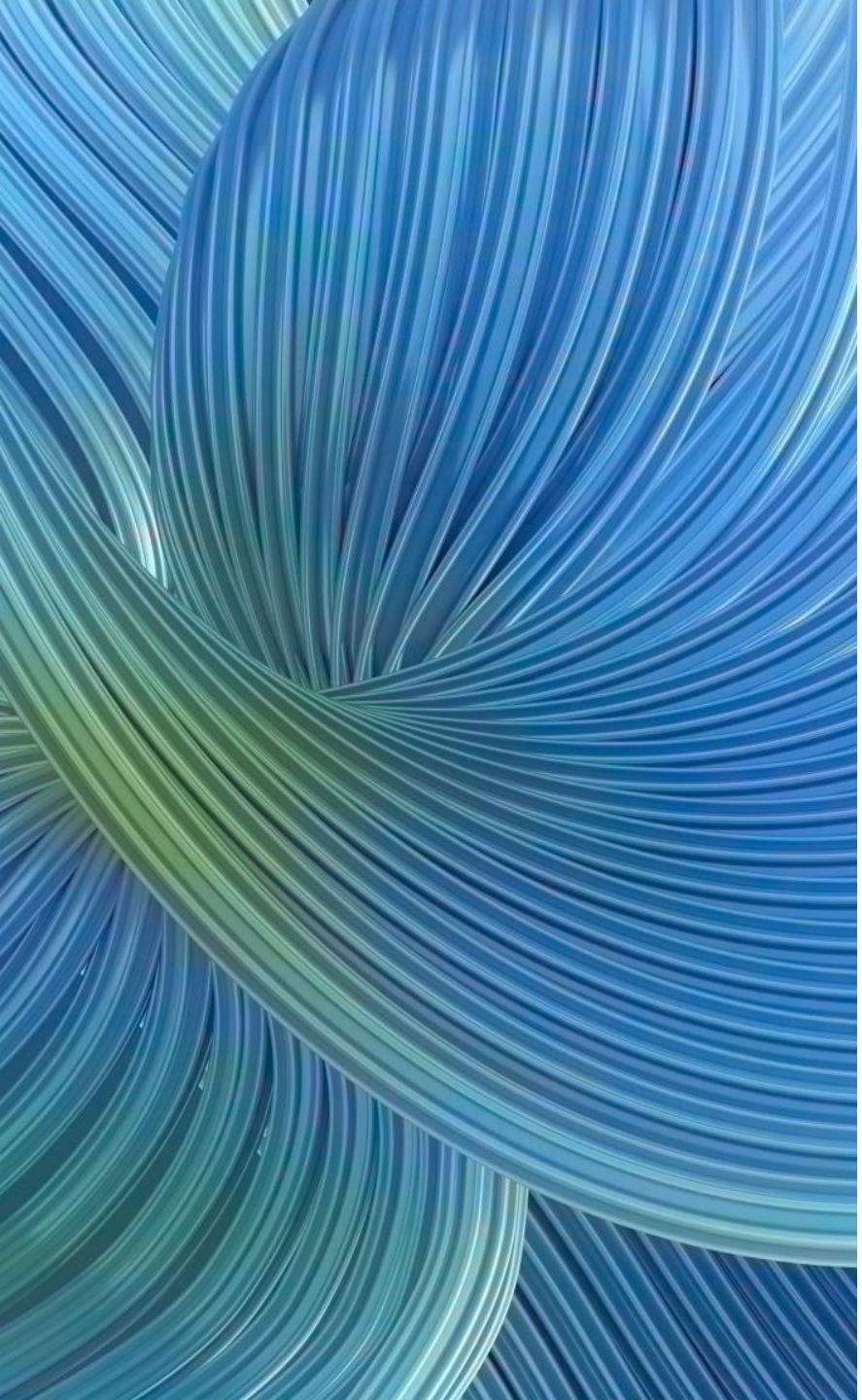
Key Differences

Feature	Function Calling	Tools (Plugins/APIs)
Purpose	Calls pre-defined functions with structured outputs	Connects to external APIs, databases, or services
Execution	Model determines function parameters and returns structured JSON	Tool performs the task and returns the result
Dependency	Requires function definitions in the environment	Relies on third-party APIs or external integrations
Example Use	Calling a `get_weather` function with parameters	Searching for real-time weather updates via a web tool
Flexibility	Structured, model knows the schema	More powerful, can integrate various services



Reasoning Models

- **Reasoning models**, like OpenAI o1 and o3-mini, are new large language models trained with reinforcement learning to perform complex reasoning.
- Reasoning models think before they answer, producing a long internal chain of thought before responding to the user.
- The performance of o1 consistently improves with more reinforcement learning (**train-time compute**) and with more time spent thinking (**test-time compute**).
- Reasoning models excel in complex problem solving, coding, scientific reasoning, and **multi-step planning for agentic workflows**.



Reasoning Models – Behind the scenes

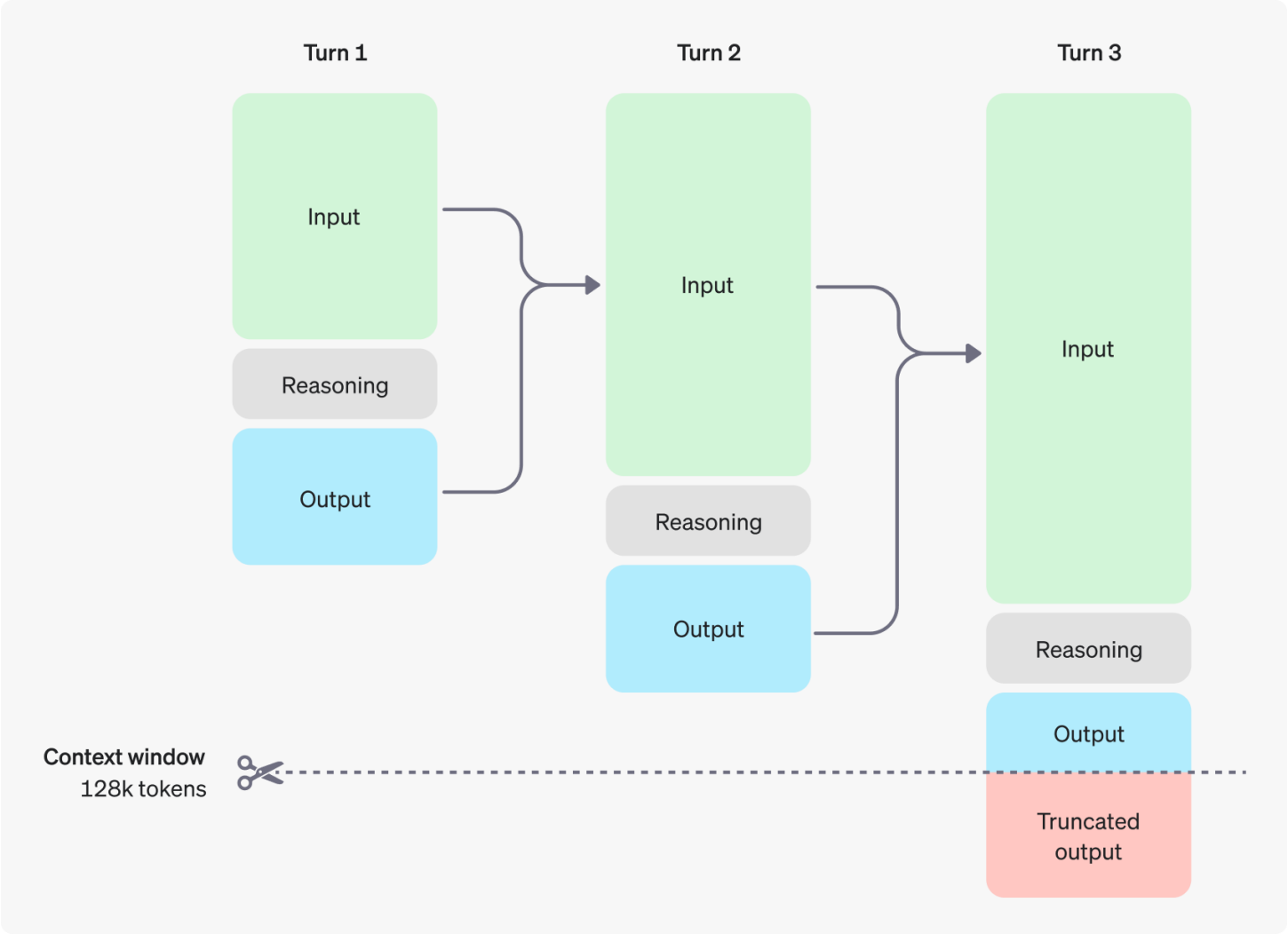
- We have recently seen a lot of interest for reasoning strategies.
- This is what's behind models like Deepseek R1 or OpenAI's o1, which have been fine-tuned to "think before answering".
- These models have been trained to always include specific ***thinking*** sections (enclosed between <think> and </think> special tokens).
- This is not just a prompting technique like ReAct, but a training method where the model learns to generate these sections after analyzing thousands of examples that show what we expect it to do.

Reasoning Models: How reasoning works

- Reasoning models introduce **reasoning tokens** in addition to input and output tokens.
- The models use these reasoning tokens to "think", breaking down their understanding of the prompt and considering multiple approaches to generating a response.
- After generating reasoning tokens, the model produces an answer as visible completion tokens and discards the reasoning tokens from its context.

How reasoning works

- Here is an example of a multi-step conversation between a user and an assistant.
- Input and output tokens from each step are carried over, while reasoning tokens are discarded.



Managing the context window

- It's important to ensure there's enough space in the context window for reasoning tokens when creating completions.

- Depending on the problem's complexity, the models may generate anywhere from a **few hundred to tens of thousands** of reasoning tokens. (Recommendation to reserve 25,000 tokens for reasoning and outputs to start with).

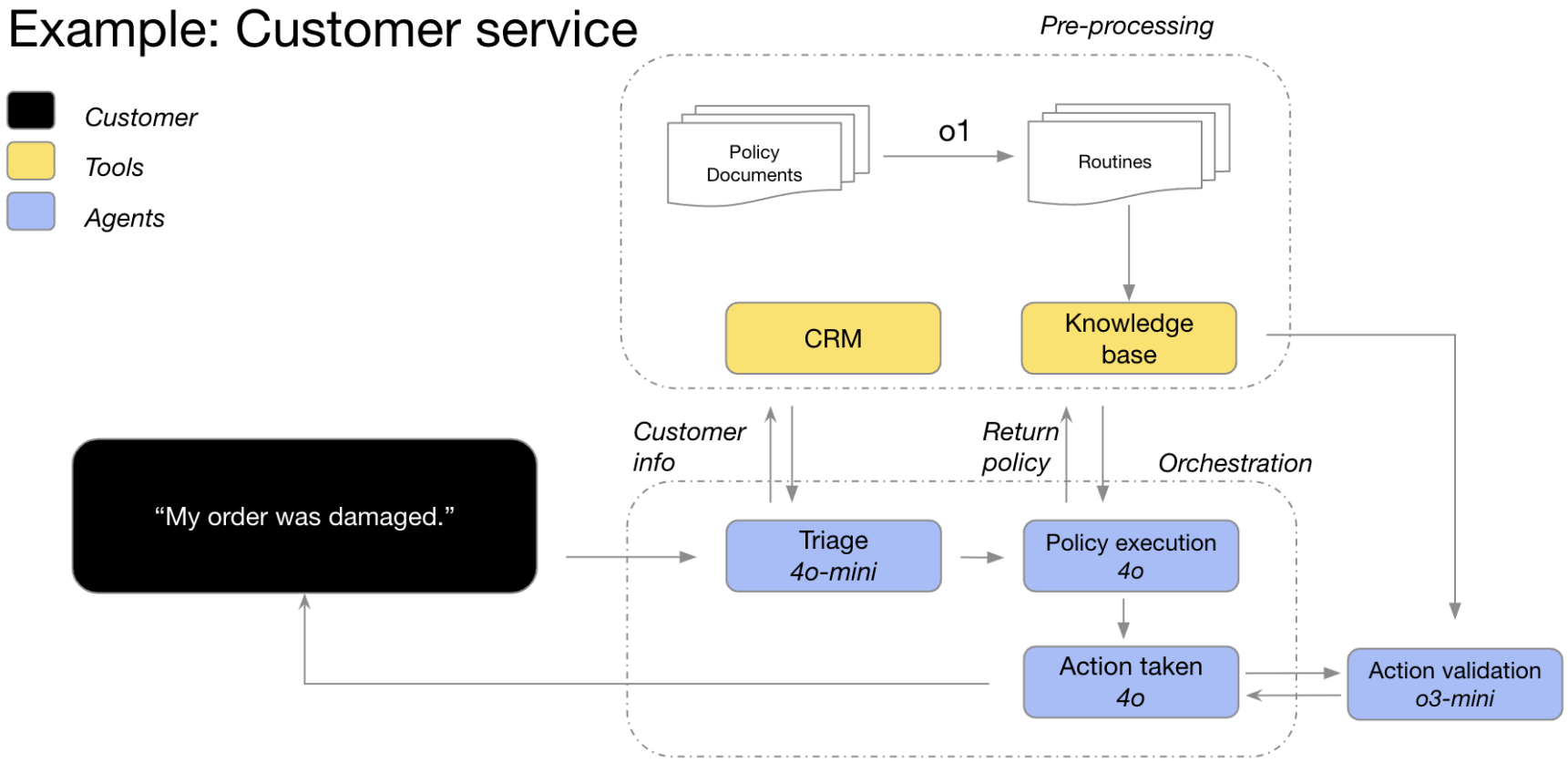

```
{
  "usage": {
    "prompt_tokens": 9,
    "completion_tokens": 12,
    "total_tokens": 21,
    "completion_tokens_details": {
      "reasoning_tokens": 0,
      "accepted_prediction_tokens": 0,
      "rejected_prediction_tokens": 0
    }
  }
}
```

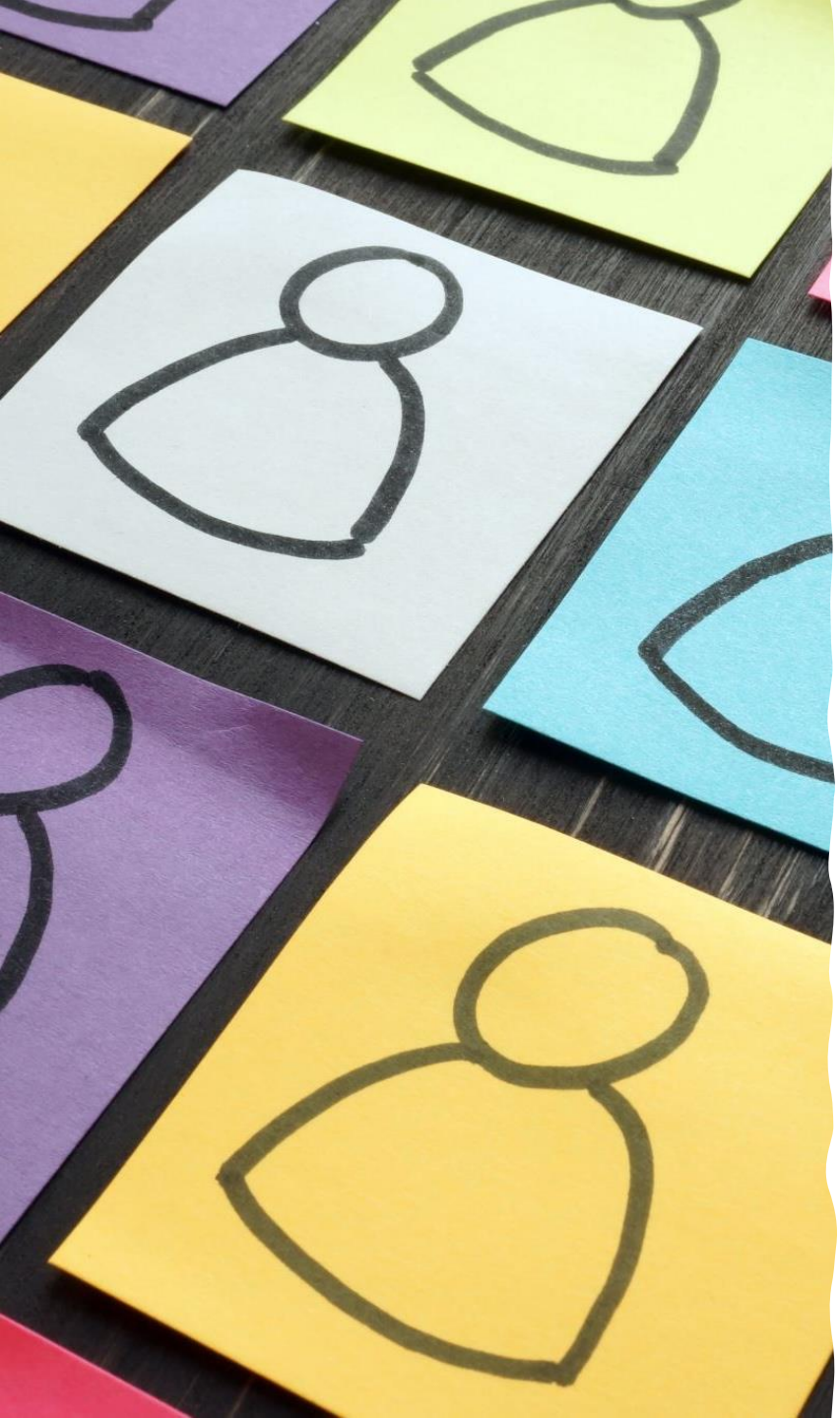
Reasoning models vs. GPT models

- **“o-series”** models (“the **planners**”) were trained to think longer and harder about complex tasks, making them effective at strategizing, planning solutions to complex problems, and making decisions based on large volumes of ambiguous information.
    - These models can also execute tasks with high accuracy and precision, making them ideal for domains that would otherwise require a human expert—like math, science, engineering, financial services, and legal services.
- On the other hand, the lower-latency, more cost-efficient GPT models (“the **workhorses**”) are designed for straightforward execution.
- An application might use **o-series models to plan** out the strategy to solve a problem, and **use GPT models to execute** specific tasks, particularly when speed and cost are more important than perfect accuracy.

How to choose?

GPT-4o and GPT-4o mini models triage order details with customer information, identify the order issues and the return policy, and then feed all of these data points into o3-mini to make the final decision about the viability of the return based on policy.

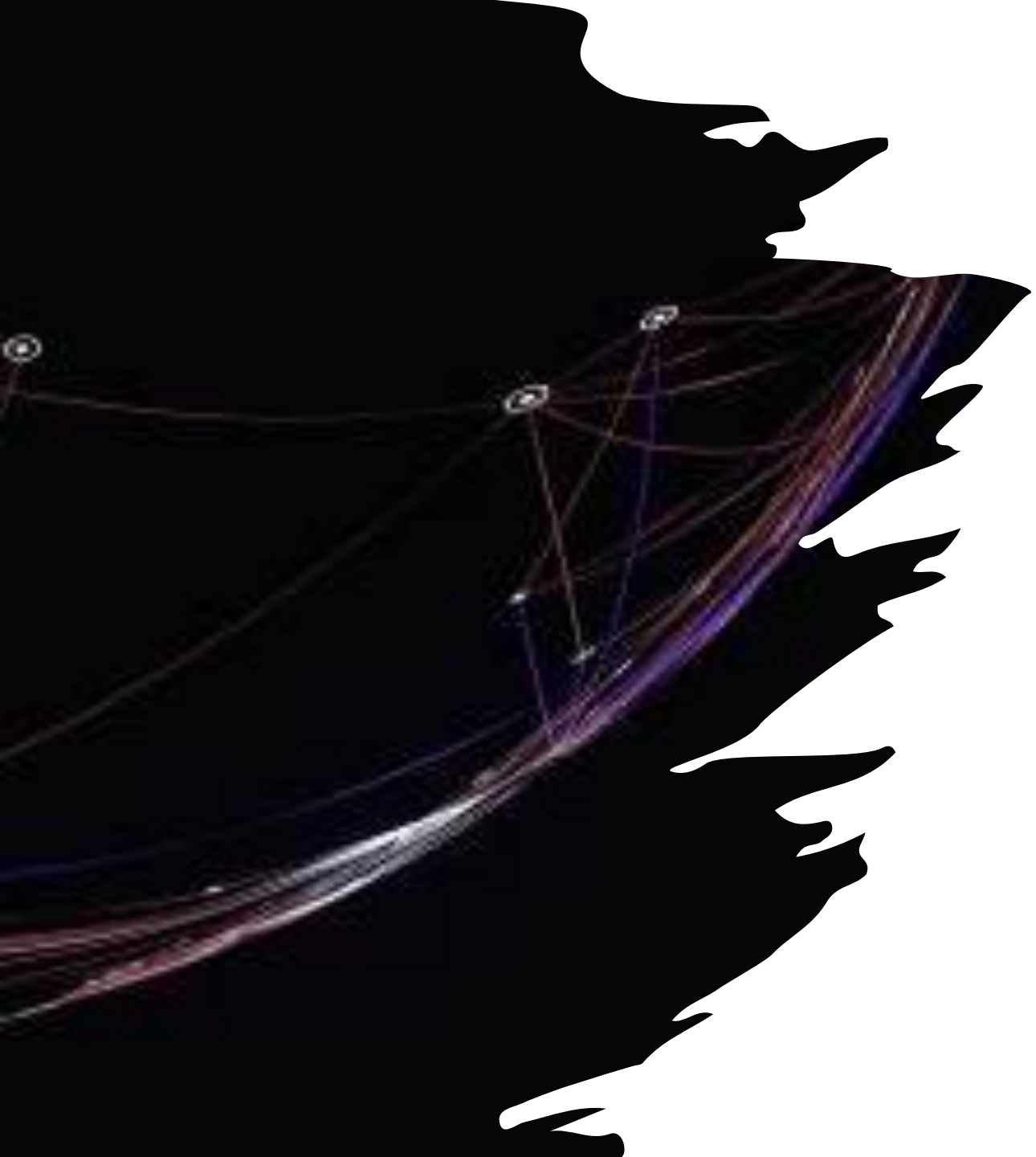




How to prompt reasoning models effectively

- Keep prompts simple and direct
- Avoid chain-of-thought prompts
- Use delimiters for clarity
- Try zero shot first, then few shot if needed
- Provide specific guidelines
- Be very specific about your end goal

Frameworks



LangGraph

- At its core, LangGraph models agent workflows as **graphs**.
- By composing Nodes and Edges, you can create complex, looping workflows that evolve the State over time.
- The real power, though, comes from how LangGraph manages that **State**.
- To emphasize: Nodes and Edges are nothing more than Python functions - they can contain an LLM or just good old Python code.
- **In short: *nodes do the work. Edges tell what to do next.***

LangChain vs. LangGraph: Key Differences

Feature	LangChain 🏗️	LangGraph 🔄
Purpose	Framework for building LLM-powered apps	Graph-based framework for handling **multi-step reasoning** with AI
Core Concept	Chains & Agents (Sequential execution of tasks)	Directed Acyclic Graphs (DAG) for **complex workflows**
Execution	Linear or branching logic	**Non-linear** execution (looping, branching, async execution)
Best for	Simple chatbots, RAG, API calls	Complex reasoning, multi-step processes, AI agents
State Handling	Limited state persistence	**Persistent state across nodes**
Parallelism	Mostly sequential	Supports **parallel execution**
Use Cases	Chatbots, RAG, tool calling	Agent-based workflows, decision-making, multi-agent systems
Performance	Good for simple tasks	Optimized for **efficiency in complex workflows**

Multi-agent Systems

- An agent is a system that uses an LLM to decide the control flow of an application.
- As you develop these systems, they might grow more complex over time, making them harder to manage and scale.
- For example, you might run into the following problems:
  - agent has too many tools at its disposal and makes poor decisions about which tool to call next
  - context grows too complex for a single agent to keep track of
  - there is a need for multiple specialization areas in the system (e.g. planner, researcher, math expert, etc.)
- To tackle these, you might consider breaking your application into multiple smaller, independent agents and composing them into a multi-agent system.

Multi-agent Systems - Benefits

- **Modularity**: Separate agents make it easier to develop, test, and maintain agentic systems.
- **Specialization**: You can create expert agents focused on specific domains, which helps with the overall system performance.
- **Control**: You can explicitly control how agents communicate (as opposed to relying on function calling).

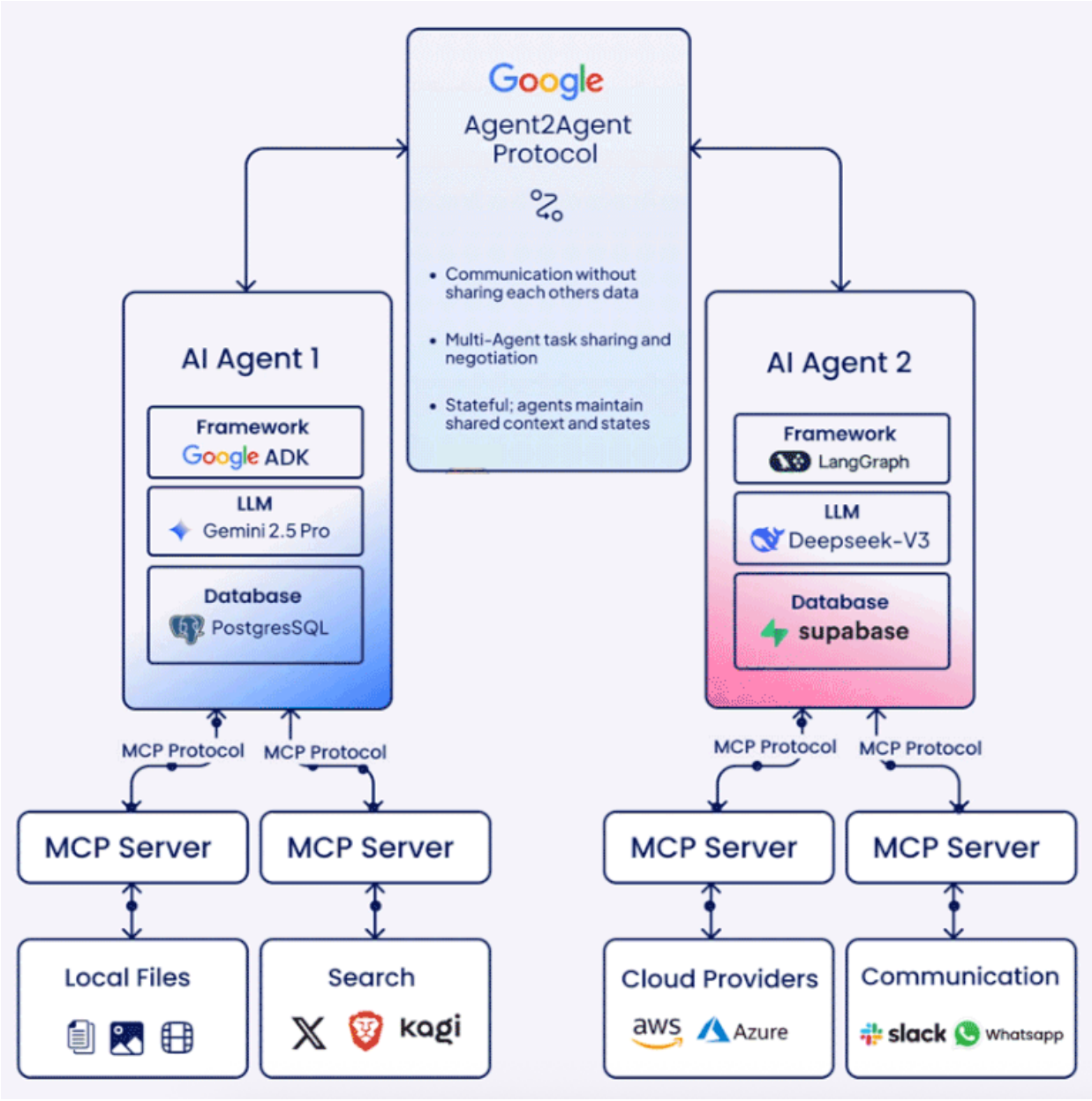
Multi-agent Systems - Architectures

- **Network**: each agent can communicate with every other agent. Any agent can decide which other agent to call next.
- **Supervisor**: each agent communicates with a single supervisor agent. Supervisor agent makes decisions on which agent should be called next.
- **Hierarchical**: you can define a multi-agent system with a supervisor of supervisors. This is a generalization of the supervisor architecture and allows for more complex control flows.

Agent2Agent Protocol (A2A)

Feature	Model Context Protocol (MCP)	Agent2Agent Protocol (A2A)
Primary Purpose	Connects AI models to external tools and data sources.	Enables communication and coordination between AI agents.
Interaction Type	Facilitates AI-to-tool/data interactions.	Facilitates AI-to-AI interactions.
Architecture	Follows a client–host–server model: AI applications (hosts) use clients to connect to servers that expose tools and resources.	Focuses on peer-to-peer communication between autonomous AI agents.
Use Cases	Integrating AI with IDEs, databases, CRM systems, and other tools to provide context-aware assistance.	Enabling multiple AI agents to collaborate on complex tasks, such as coordinating actions across enterprise applications.
Standardization	Provides a standardized protocol for AI models to access various tools and data sources, reducing the need for custom integrations.	Establishes a framework for AI agents to interoperate, share information, and coordinate actions seamlessly.
Security Focus	Emphasizes secure connections between AI models and external tools, incorporating authentication and permission models.	Prioritizes secure communication between AI agents, ensuring data integrity and confidentiality during inter-agent exchanges.

A2A Protocol



Why most AI Agents fail and how to fix them?

DEVELOPMENT ISSUES

Poorly Defined Prompts
- Define Clear Objectives
- Craft Detailed Personas
- Use Effective Prompting

Evaluation Challenges
- Continuous Evaluation
- Use Real-World Scenarios
- Incorporate Feedback Loops

LLM ISSUES

Difficult to Steer
- Specialized Prompts
- Hierarchical Design
- Fine-Tuning Models

High Cost of Running
- Reduce Context Size
- Use Smaller Models
- Cloud-Based Solutions

Planning Failures
- Task Decomposition
- Multi-Plan Selection
- Reflection and Refinement

Reasoning Failures
- Enhance Reasoning Capabilities
- Fine-Tune LLMs with Feedback
- Use Specialized Agents

Tool Calling Failures
- Define Clear Parameters
- Validate Tool Outputs
- Tool Selectio VerificationLoops

PRODUCTION ISSUES

Guardrails
- Rule-Based Filters & Validation
- Human-in-the-Loop Oversight
- Ethical & Compliance Frameworks

Agent Scaling
- Scalable Architectures
- Resource Management
- Monitor Performance

Fault Tolerance
- Redundancy
- Automated Recovery
- Stateful Recovery

Infinite Looping
- Clear Termination Conditions
- Enhance Reasoning & Planning
- Monitor Agent Behavior

When to Use Agents?

- Agents are highly beneficial when tasks require complex decision-making, autonomy, and adaptability.
- They excel in environments where the workflow is dynamic and involves multiple steps or interactions that can benefit from automation.
- **Ex:** Customer Support, Research and Data Analysis, Personalized learning experiences, Software Development (Code generation, debugging, and testing)

When Not to Use Agents?

- If the tasks you're dealing with are **straightforward**, occur **infrequently**, or require only **minimal automation**, the complexity and cost of implementing AI agents might not make sense for you.

- Also, if your tasks require **deep domain-specific knowledge or expertise**—like conducting complex legal analyses, making intricate medical diagnoses, or handling high-stakes decision-making in unpredictable environments—these are typically better left to experienced professionals.

- That said, fields like **psychotherapy**, **counseling**, etc. thrive on the nuances of human emotion and the creative process—areas where agents largely fall short.

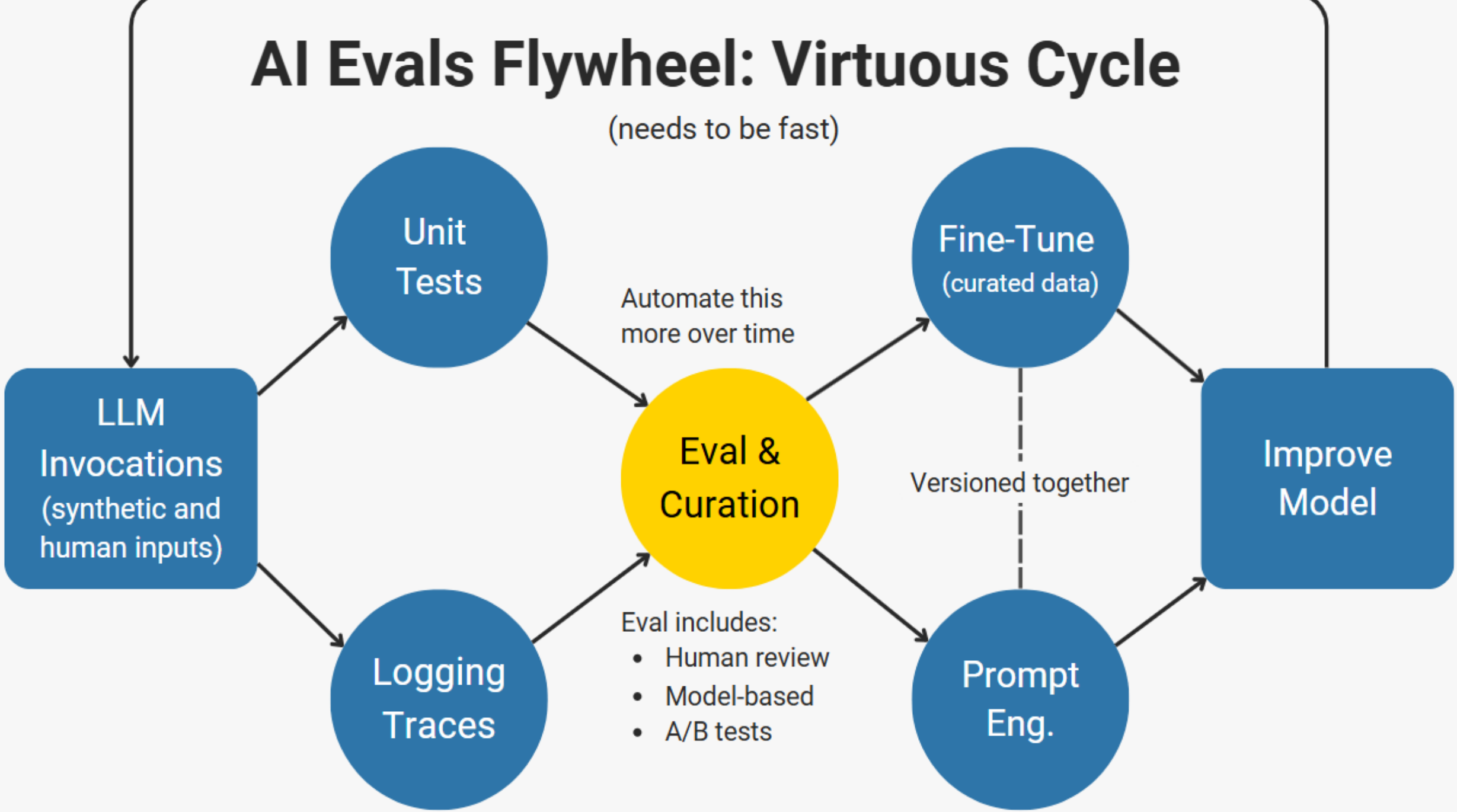
When Not to Use Agents?

- Implementing agents also requires a significant investment from you in terms of time, resources, and expertise.
- If you’re running a small business or managing a project with a tight budget, the costs of developing and maintaining these agents may not justify the benefits.
- In highly regulated industries, your use of agents might be restricted due to compliance and security concerns as well, and ensuring agents adhere to stringent regulatory requirements can be very challenging and resource-intensive.

Questions to Ask Before You Consider an AI Agent

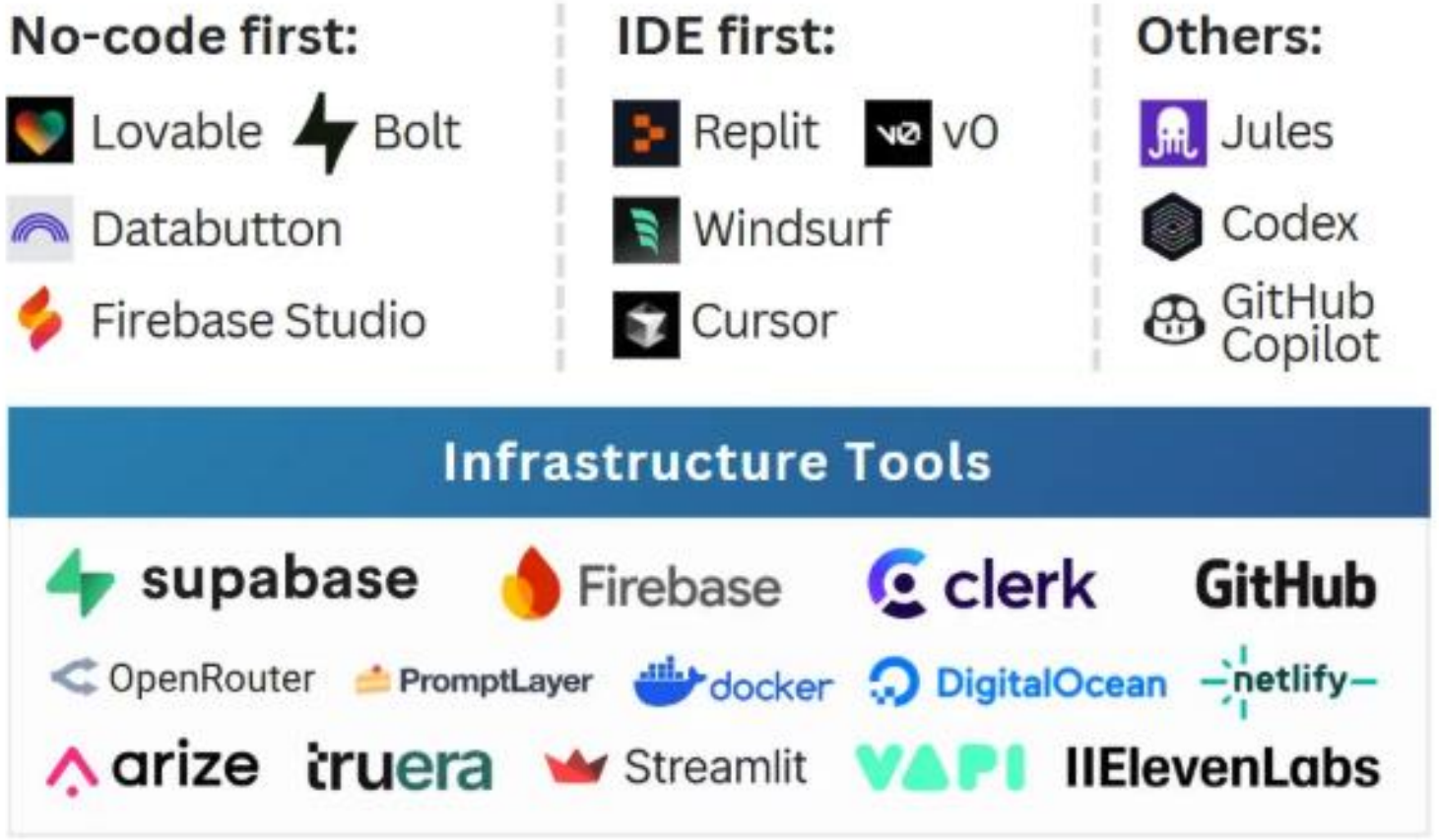
- What is the complexity of the task?
- How often does the task occur?
- What is the expected volume of data or queries?
- Does the task require adaptability?
- Can the task benefit from learning and evolving over time?
- What level of accuracy is required?
- Is human expertise or emotional intelligence essential?
- What are the privacy and security implications?
- What are the regulatory and compliance requirements?
- What is the cost benefit analysis?

Appendix


AI Evals Flywheel: Virtuous Cycle
(needs to be fast)
LLM Invocations (synthetic and human inputs)
Unit Tests
Logging Traces
Automate this more over time
Eval & Curation
Eval includes:
Human review
Model-based
A/B tests
Fine-Tune (curated data)
Prompt Eng.
Versioned together
Improve Model

AI Agents & Agentic Workflows

AI Prototyping & AI Building



AI Tools for Product Managers

Market & competitors research	perplexity
Conduct user surveys	THEYSAID
Transcribe user interviews	
Analyze customer feedback	survicate
Write PRDs	ChatPRD
Build functional prototypes	v0
Generate product design	uizard
Write SQL queries	AI 2sql
A/B testing and analysis	Optimizely
Presentations	Gamma presentations.ai
Copywriting	copy.ai Claude
Brainstorm	Napkin
Master English speaking skills	Fluently