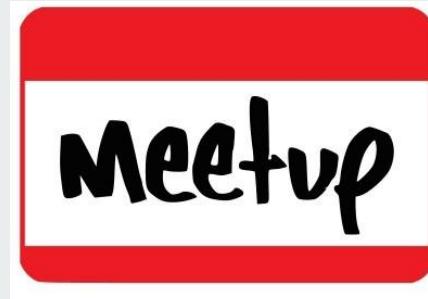


Python Frederick

Python AI Agents: Intro to Agents



June 11th, 2025

7:00pm - 9:00pm

My Top AI Tools

Windsurf

Cursor AI



ChatGPT

Github copilot

Claude

gemini.google.com

perplexity

meta.ai

Manus

bing.com

Microsoft copilot

runnerh.com

notebooklm.google

anythingllm.com

Ollama

Imstudio

<https://github.com/gmossy>

<https://www.linkedin.com/in/gmossy/>

Python Frederick Meetup

Introduction to AI Agents



Glenn Mossy

AI Agent Architect

37+ years turning sci-fi into production code! 🚀 From nuclear plants to autonomous AI agents

💡 Warp Core AI Autonomy Stack

Lead architect for autonomous agent framework serving ISR operations with neurosymbolic reasoning

🧠 ATEM LLM RAG Copilot

Built comprehensive GenAI knowledge assistant for military technical documentation

🌐 DISA's First Production SDN

Delivered software-defined networking system automating critical network management

📞 Hughes Enterprise VoIP Pioneer

Launched North America's first satellite-based enterprise voice service

🎯 Fun Agent Facts

- Taught 2,000+ adults electronics & robotics at Frederick Community College
- Built security systems for nuclear plants (back when AI was just a dream!)
- Lives in Ijamsville, MD - probably your neighbor! 🎉
- Started with Assembly & C, now wielding Python like a wizard 🧙

Python 3.11+

LangChain

LlamaIndex

AWS Bedrock

PyTorch

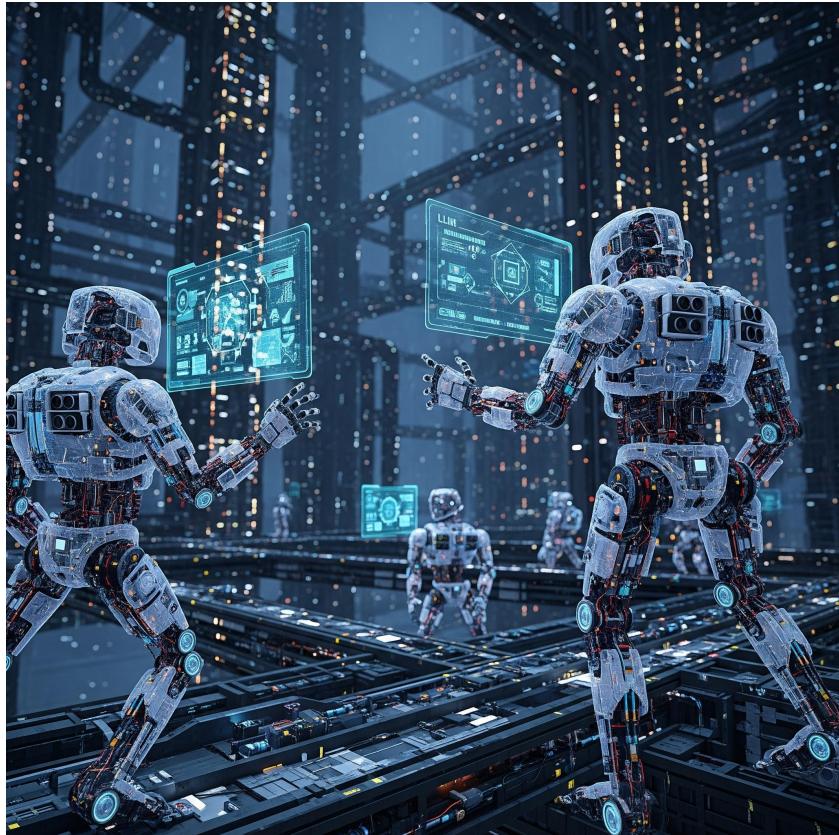
Autonomous Agents

RAG Systems

Neurosymbolic AI

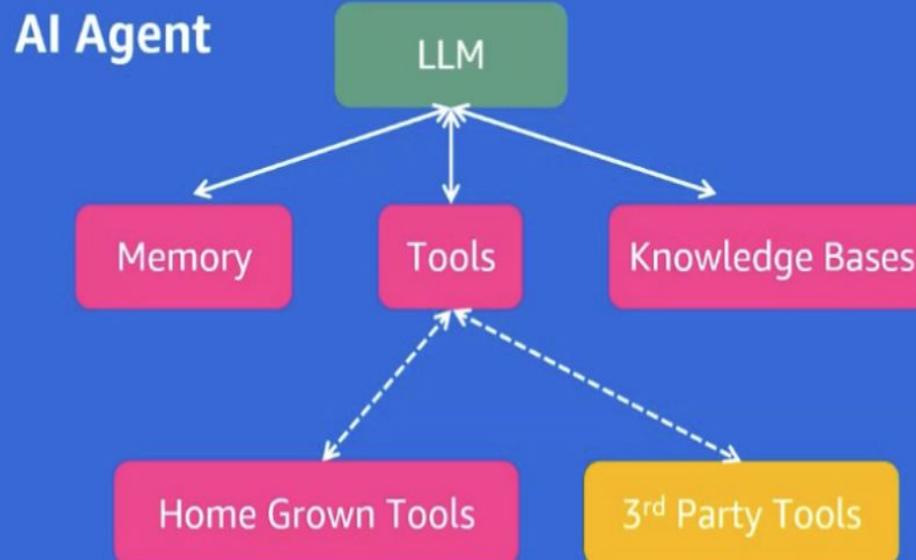
Agents in Action

- AI agents interact with environment via tools.
- Key components are augmented LLMs
- LLMs show varying degrees of autonomy in systems.
- Augmented LLMs use external systems for enhancements.



What are AI agents?

Autonomous software systems that leverage AI to reason, plan, and complete tasks on behalf of humans and systems



Key areas demonstrate real business value:

- Workplace productivity
- Workflow automation
- Innovation and research

AI Agent Frameworks Timeline

Evolution of Intelligent Agent Development



Python Frederick



What Are Agent Frameworks?

🧠 Intelligence Layer

- Reasoning & Planning
- Memory Management
- Decision Making
- Goal Decomposition

🔧 Tool Integration

- API Connections
- Database Access
- External Services
- File Operations

🤝 Collaboration

- Multi-Agent Systems
- Task Distribution
- Communication Protocols
- Workflow Orchestration

⚡ Execution Engine

- Code Generation
- Task Execution
- Error Handling
- Performance Monitoring

- <https://github.com/anthropic/anthropic-cookbook/tree/main/patterns/agents>

Anthropic-cookbook

Engineering at Anthropic



Building effective agents

Published Dec 19, 2024

We've worked with dozens of teams building LLM agents across industries. Consistently, the most successful implementations use simple, composable patterns rather than complex frameworks.

<https://www.anthropic.com/engineering/building-effective-agents>

OpenAI developer platform

<https://platform.openai.com/docs/guides/text>

javascript :

```
1 import OpenAI from "openai";
2 const client = new OpenAI();
3
4 const response = await client.responses.create({
5   model: "gpt-4.1",
6   input: "Write a one-sentence bedtime story about a unicorn.",
7 });
8
9 console.log(response.output_text);
```

Developer quickstart

Make your first API request in minutes. Learn the basics of the OpenAI platform.

⌚ 5 min

Browse models

View all

GPT-4.1

Flagship GPT model for complex tasks

o4-mini

Faster, more affordable reasoning model

o3

Our most powerful reasoning model

Start building

Read and generate text
Use the API to prompt a model and generate text

Use a model's vision capabilities
Allow models to see and analyze images in your application

Generate images as output
Create images with GPT Image 1

Build apps with audio
Analyze, transcribe, and generate audio with API endpoints

Build agentic applications
Use the API to build agents that use tools and computers

Achieve complex tasks with reasoning
Use reasoning models to carry out complex tasks

Get structured data from models
Use Structured Outputs to get model responses that adhere to a JSON schema

Tailor to your use case
Adjust our models to perform specifically for your use case with fine-tuning, evals, and distillation

Agents

...more



Rakesh Goel 
@rakeshgoel01

20-AI Agent Terms

You should know

1  An entity that receives prompts and environment to perceive and act on goals.	2  Context or space where an AI agent operates and interacts with other tools.	3  The AI Agents' ability to understand and interpret environmental data.	4  The current process performed by an AI agent or group of agents.
5  The current condition of an agent's environment or system.	6  Large language Models - the brain behind the Agents to perform.	7  Large Reasoning model - a reasoning type for more context based reasoning.	8  Native or Third Party APIs used by Agents to perform their task.
9  Storage for current context as well as past interactions.	10  Database for Agents' knowledge used to fuel and generate outcomes.	11  Process of developing agents' interaction, starting from input to output.	12  The process of an AI agent determining a sequence of actions to achieve a goal.
13  The assessment of an AI agent's performance, in achieving its goals.	14  The blueprint of an AI agent, defining how its components interact.	15  A reasoning technique where an agent breaks down complex problems.	16  A reasoning Framework for combining reasoning and acting iteratively.
17  Multiple AI agents interacting in a shared space.	18  Collectively exhibit intelligent behavior through self-organized interactions.	19  The transfer of tasks or responsibilities between Multiple AI agents.	20  Engage in structured argumentation to create better outcomes

671

58 comments • 117 reposts



Like



Comment



Repost



Send

Rakesh Goel  Author

Scaling with AI Agents | Expert in Agentic AI & Cloud Native S...

 Help us build the learning path YOU wish existed
for mastering AI agents

<https://forms.gle/qbtarifWLD7BPgMb6>


Home



Video



My Network



20+ Notifications



Jobs

Technology Enabling Agentic AI: Grounding and Reasoning

Reliable Data Foundations + Advanced Reasoning = Trustworthy AI Agents

Foundation Technologies Working Together

RAG

Retrieval Augmented Generation

Purpose:
Grounds LLMs with external, up-to-date knowledge to prevent hallucinations and ensure accuracy

Key Features:

- Dynamic knowledge retrieval
- Real-time data integration
- Context-aware responses
- Multi-source synthesis

Agentic Enhancement:
Intelligent query planning and autonomous source verification

Advanced Reasoning

Multi-Step Cognitive Processing

Core Patterns:

- Chain-of-Thought (CoT) reasoning
- Tree-of-Thoughts exploration
- ReAct (Reasoning + Acting)
- Self-reflection loops

Agentic Features:

- Autonomous planning & decomposition
- Dynamic strategy adaptation
- Error detection & correction

Enables complex problem-solving with human-like reasoning depth

Knowledge Graphs

Structured Relationship Networks

Structure:
Entity-relationship networks that represent context, semantics, and complex interconnections

Agent Benefits:

- Enhanced understanding
- Relationship reasoning
- Knowledge inference
- Dependency mapping

GraphRAG:
Combines graph relationships with retrieval for richer context and improved reasoning

Reinforcement Learning

Trial & Error Optimization

Core Mechanism:
Agents learn optimal behavior through environment interaction, maximizing cumulative rewards

Key Algorithms:

- Q-Learning (value functions)
- Policy gradients
- Actor-critic methods
- Monte Carlo tree search

Enables:
Autonomous adaptation, goal optimization, and strategic decision-making over time

Memory Systems

Persistent Learning & Context

Memory Types:

- Working memory (current task)
- Episodic memory (experiences)
- Semantic memory (knowledge)
- Long-term storage

Agent Capabilities:

- Continuous learning
- Context preservation
- Pattern recognition
- Adaptive behavior

Enables long-horizon tasks and learning

Intelligent Agents

What Are Agent Frameworks?

🧠 Intelligence Layer

- Reasoning & Planning
- Memory Management
- Decision Making
- Goal Decomposition

🔧 Tool Integration

- API Connections
- Database Access
- External Services
- File Operations

🤝 Collaboration

- Multi-Agent Systems
- Task Distribution
- Communication Protocols
- Workflow Orchestration

⚡ Execution Engine

- Code Generation
- Task Execution
- Error Handling
- Performance Monitoring

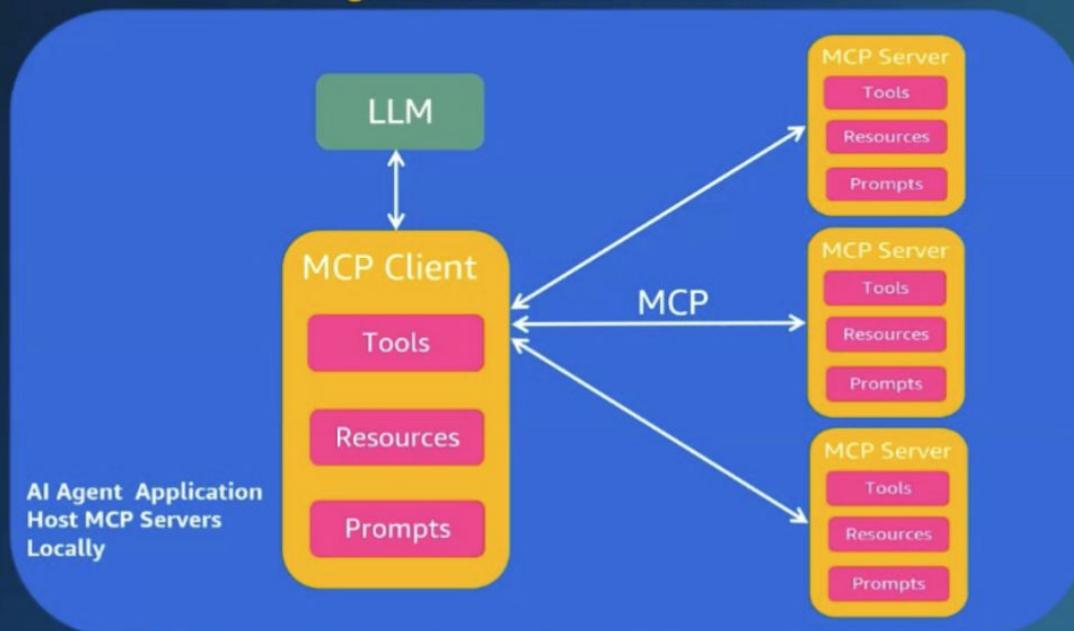
Consider exploring the dangers of relying on one large agent doing everything as opposed to segregating tasks to multiple agents all managed by an orchestrator.

One Slide Introduction to Model Context Protocol

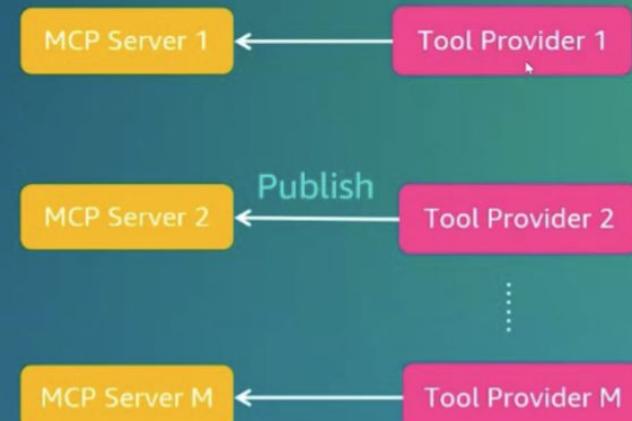
Developing AI agents with MCP

Standardized development

MCP Clients dynamically discover capabilities from a configurable list of MCP Servers

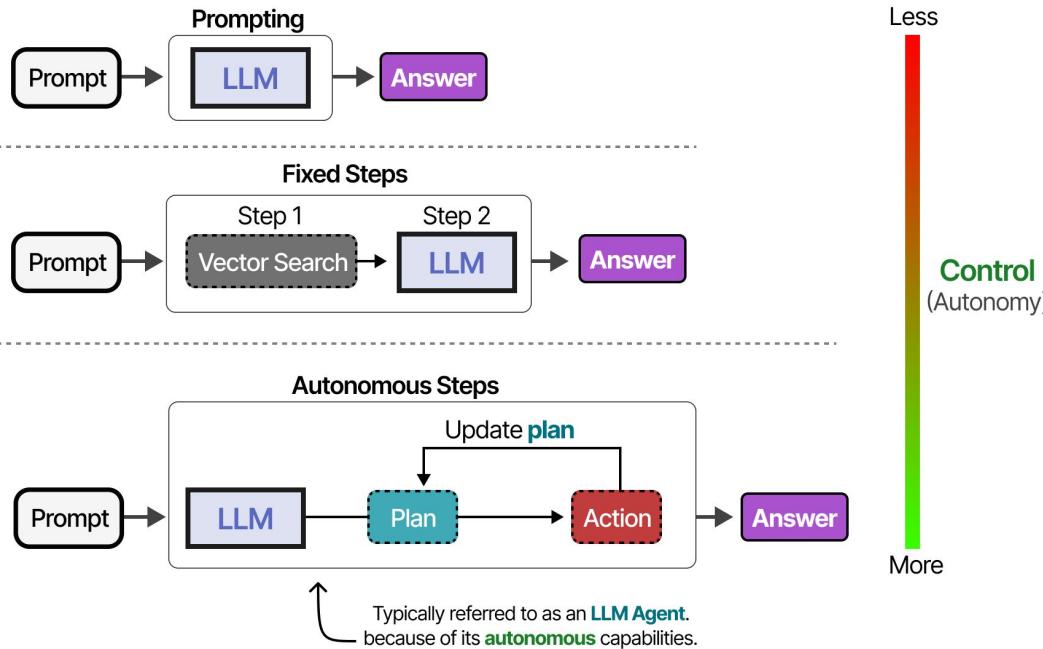


Tool providers develop MCP Servers for exposing capabilities



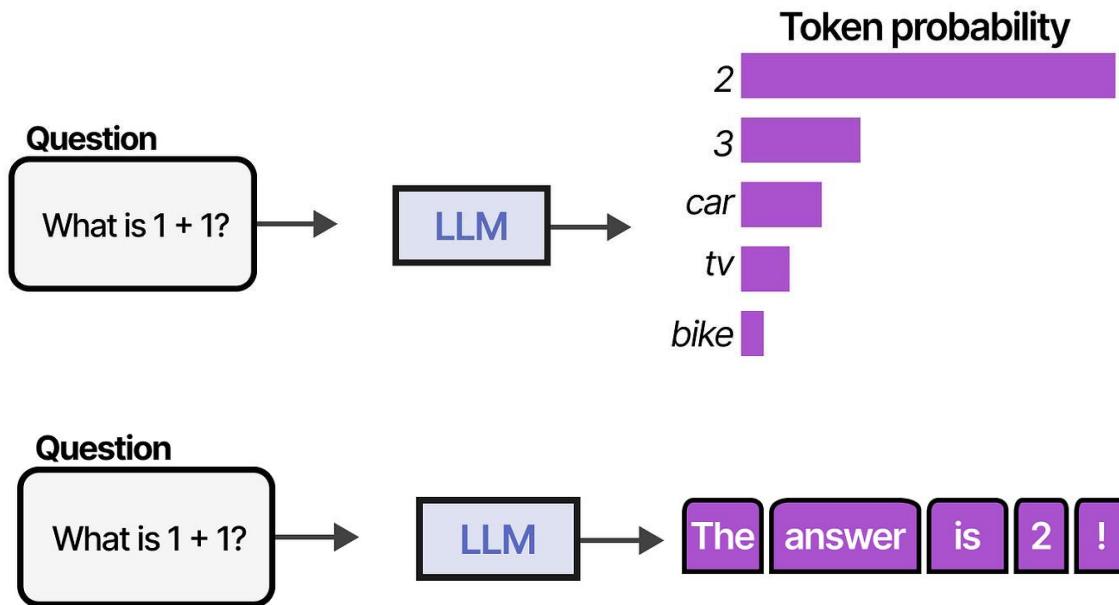
Degrees of Autonomy

Depending on the system, you can have LLM Agents with varying degrees of autonomy.



How LLMs work

By sampling many tokens in a row, we can mimic conversations and use the LLM to give more extensive answers to our queries.



LLMs have no memory

However, when we continue the “conversation”, any given LLM will showcase one of its main disadvantages. It does not remember conversations!

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.

```
from openai import OpenAI

client = OpenAI(api_key=os.getenv("OPENAI_API_KEY"))

completion = client.chat.completions.create(
    model="gpt-4o",
    messages=[
        {"role": "system", "content": "You're a helpful assistant."},
        {
            "role": "user",
            "content": "Write a limerick about the Python programming language,  
also say hello to the Frederick Python Meetup developers"
        },
    ],
)

response = completion.choices[0].message.content
print(response)
```

Basic ChatBot Demo

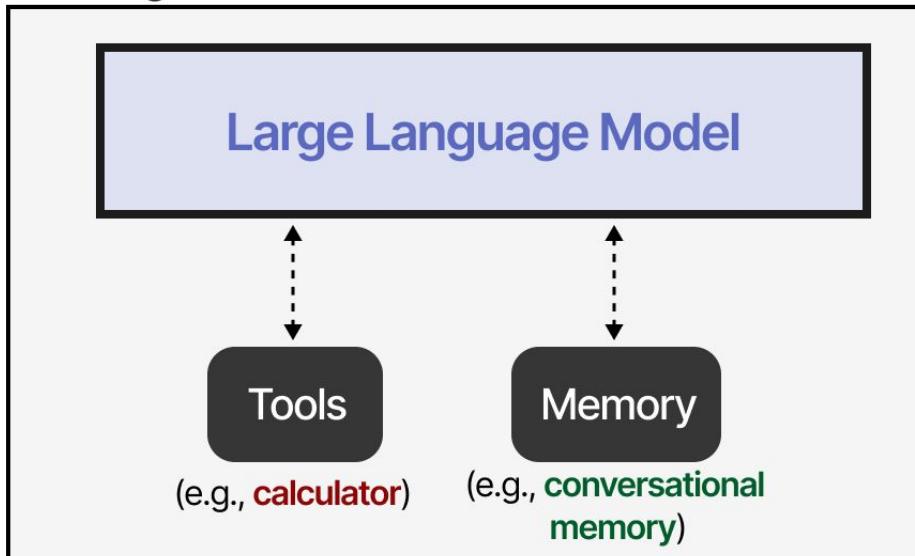
<https://platform.openai.com/docs/guides/text>

Demo / Talk about
2-structured.py

The Augmented LLM

Through external systems, the capabilities of the LLM can be enhanced. Anthropic calls this “The Augmented LLM”.

The Augmented LLM

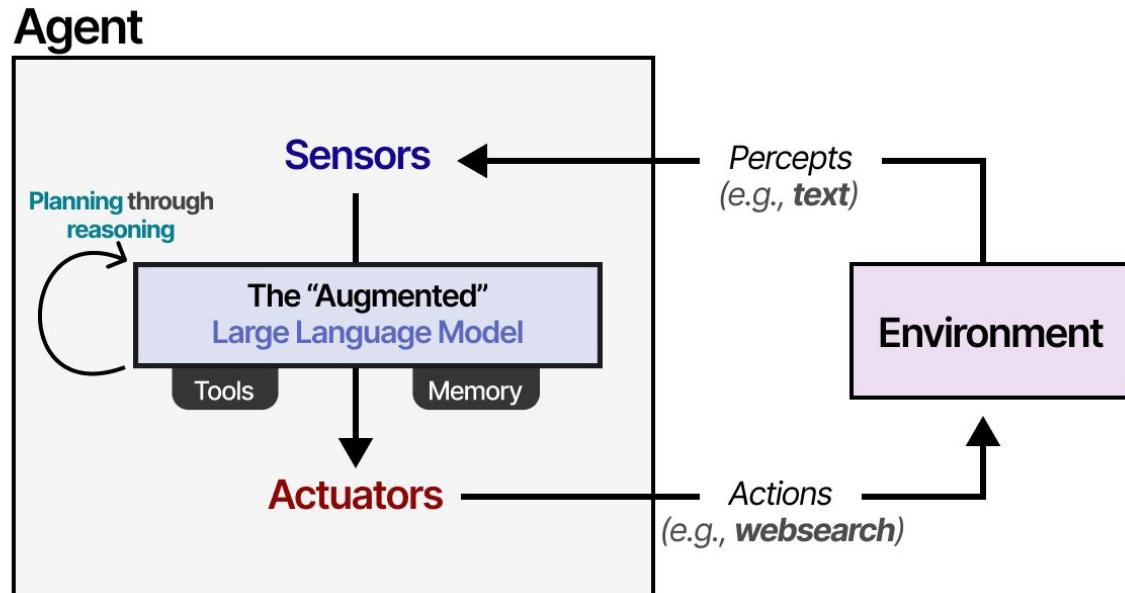


The Augmented LLM

Agents interact with their environment and typically consist of several important components:

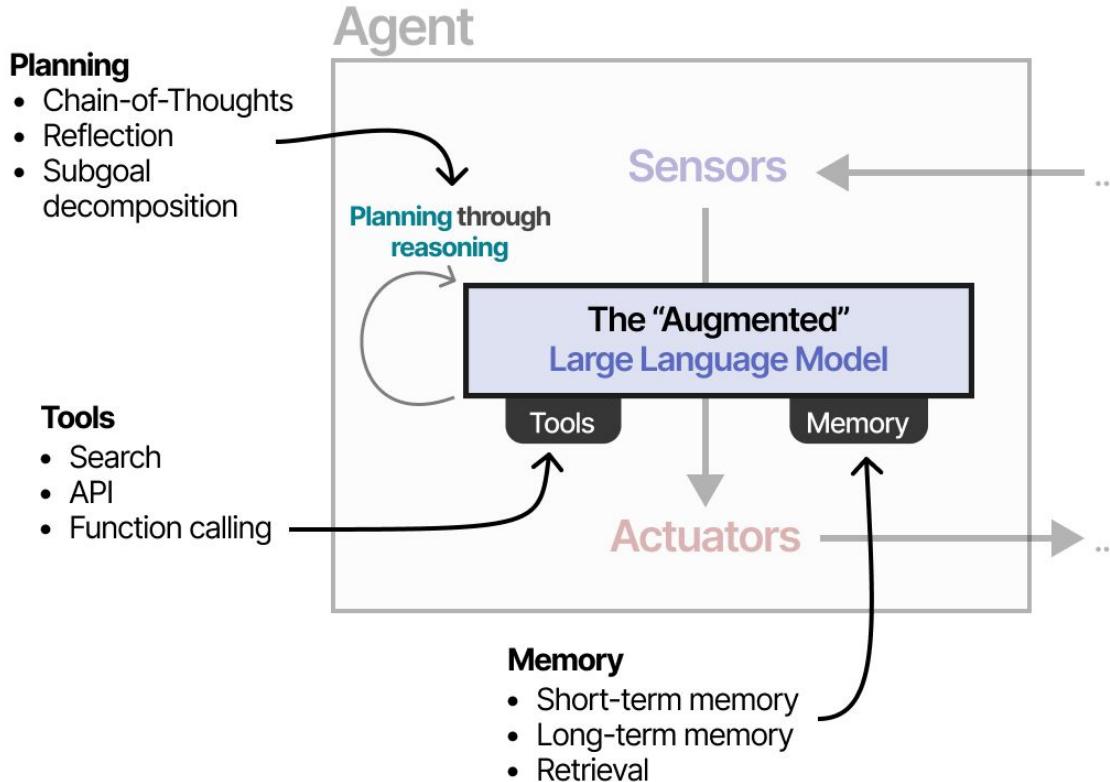
- **Environments** — The world the agent interacts with
- **Sensors** — Used to observe the environment
- **Actuators** — Tools used to interact with the environment
- **Effectors** — The “brain” or rules deciding how to go from observations to actions

Through external systems, the capabilities of the LLM can be enhanced. Anthropic calls this “The Augmented LLM”.

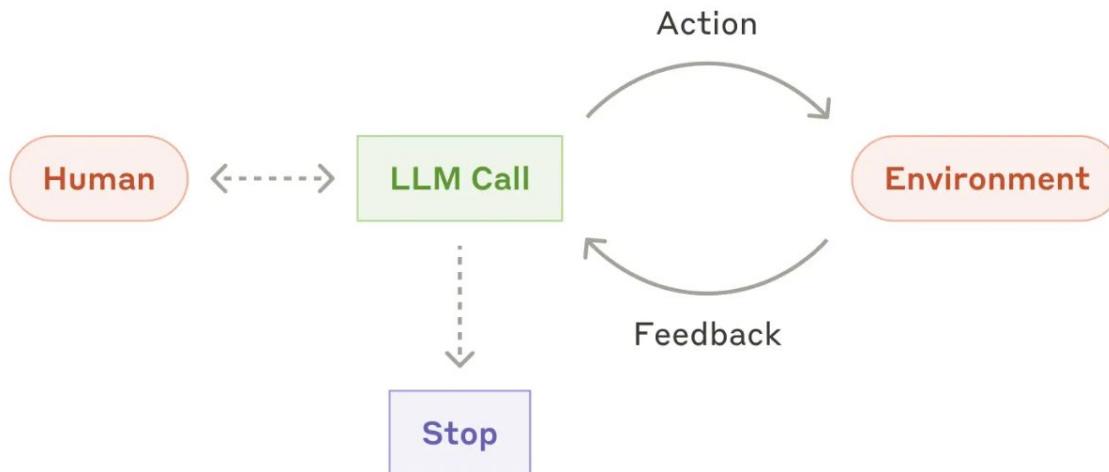


The Augmented LLM

Through external systems, the capabilities of the LLM can be enhanced. Anthropic calls this “The Augmented LLM”.



AGENTS



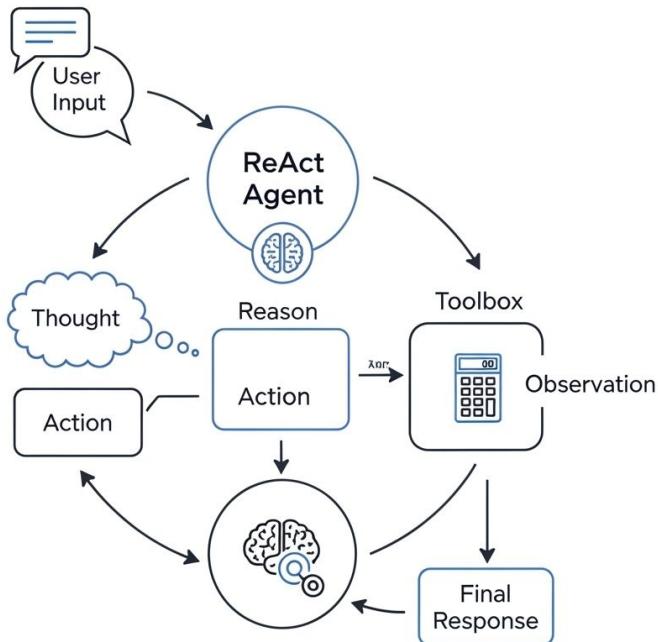
Tools

How to Incorporate Tools



CALCULATOR DEMO

Calculator Agent: A system optimized for executing complex mathematical computations with precision and speed



What "ReAct" Means

ReAct stands for "**R**eason and **A**ct". It's a framework that enables an agent to solve complex problems by mimicking a human-like thought process. Instead of just trying to give a final answer immediately, the agent follows an iterative loop:

1. **Thought:** The agent (the LLM) examines the user's input and its available tools. It then forms a "thought" about what it needs to do to solve the problem.
2. **Action:** Based on its thought, it chooses an **action** to take. An action consists of selecting one of its tools (like your **CalculatorTool**) and deciding on the input for that tool.
3. **Observation:** The action is executed (your **_run** method is called), and the result is returned to the agent as an **observation**.
4. **Repeat:** The agent looks at the new observation and goes back to step 1 (Thought). It thinks about whether the problem is solved or if it needs to take another action. This loop (**Thought** → **Action** → **Observation** → **Thought**...) continues until it has a final answer.

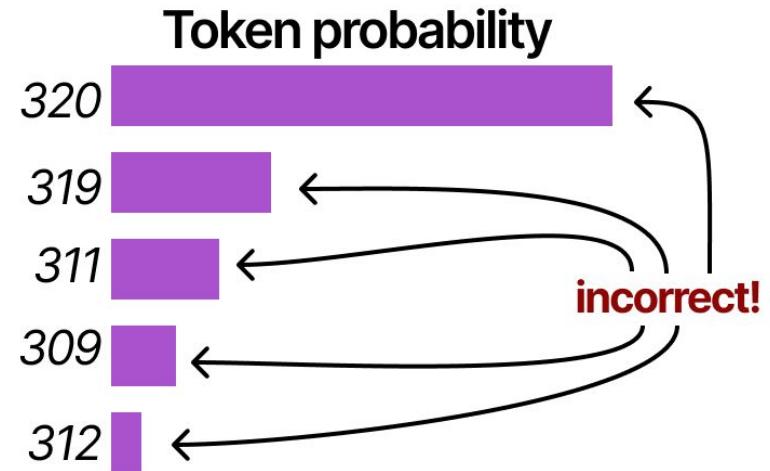
Math Questions

Math Question

What is
 $8 * 9 * 2 * 3 / 1.34?$

LLM

There are many other tasks that LLMs often fail at, including basic math like multiplication and division:



Tools use

Using Calculator Tools

For instance, when faced with a math question, the LLM may decide to use the appropriate tool (a **calculator**).

Math Question

What is
 $8 * 9 * 2 * 3 / 1.34?$

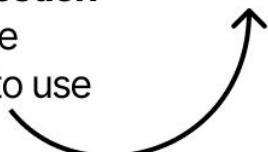
The “Augmented”
Large Language Model

322.389

identifies a **math question**
and chooses the
corresponding **tool** to use

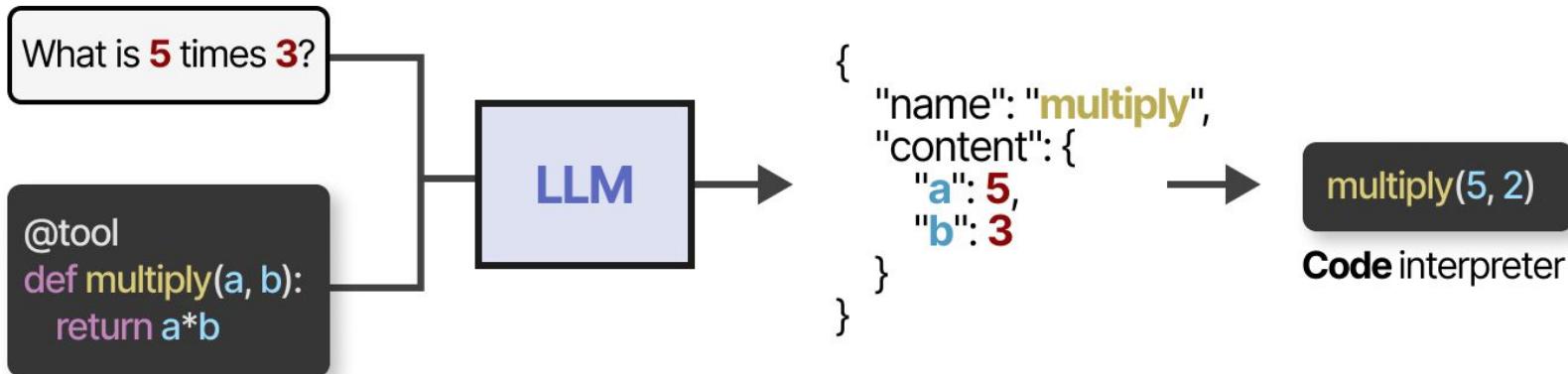
Tools
(calculator)

Memory
(conversational)



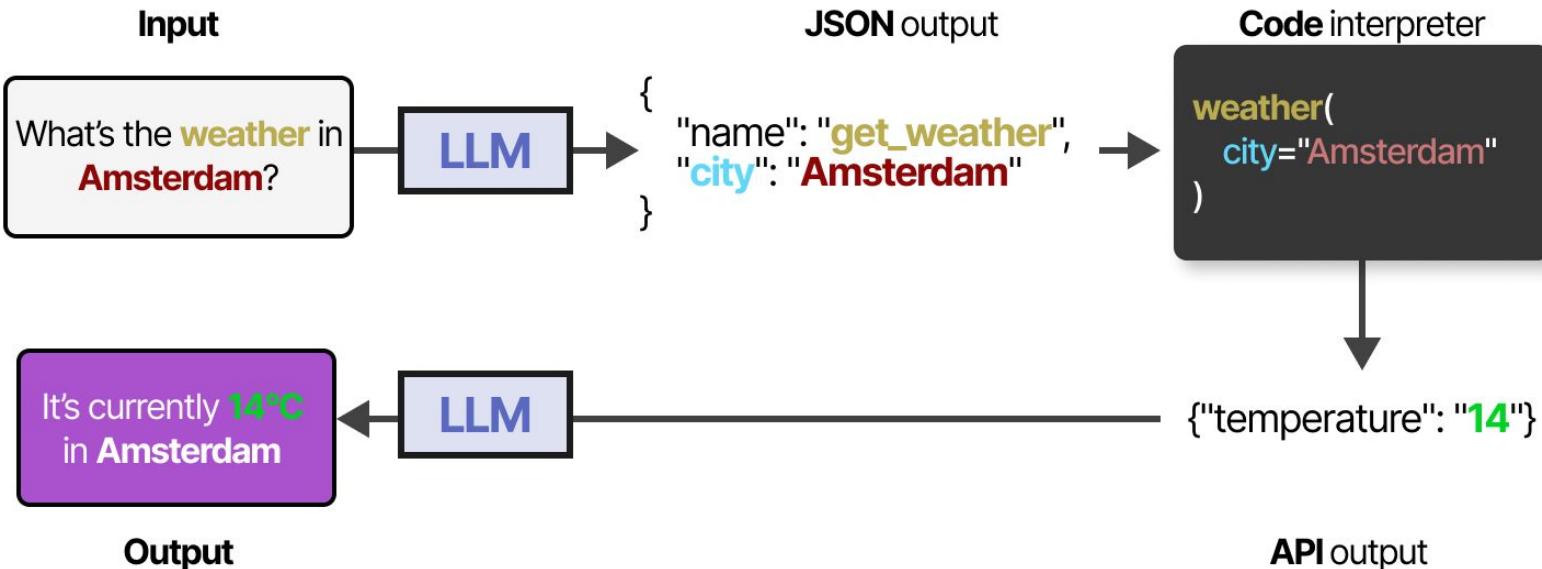
Tools function calling

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



Tools fetching data

- 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.
- To actually use a tool, the LLM has to structure the output for the API of the given tool. We tend to expect strings that can be formatted to **JSON** so that it can easily be fed to a **code interpreter**.





Memory

with the

basic_agent_demo

Lead with the basic_agent_tutorial.py



Key Highlights

- Uses GPT-3.5-turbo with OpenAI API
- Maintains conversation history with ConversationBufferMemory
- Has a system prompt defining personality
- Graceful error messages
- Stores both user and AI messages

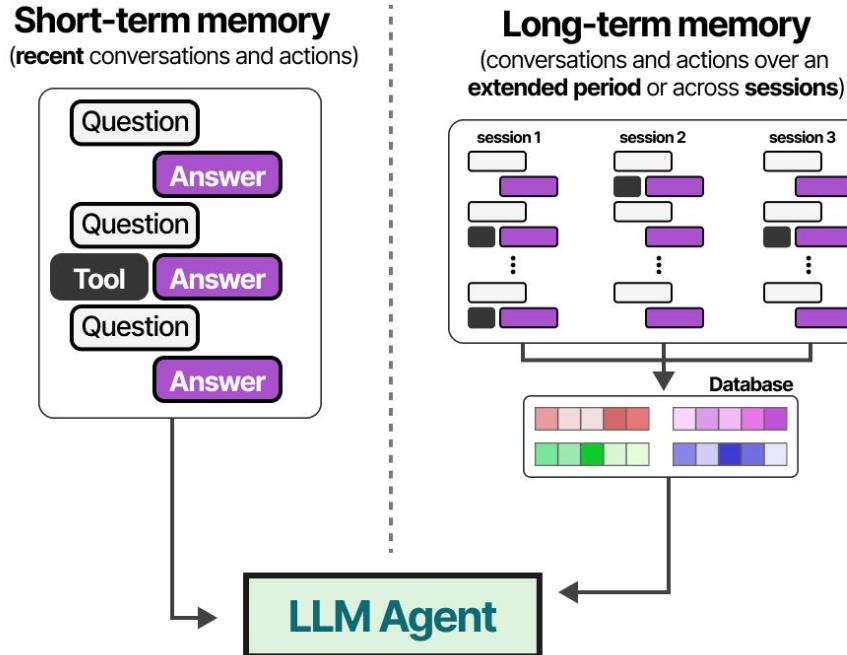
Potential Drawbacks

- Error handling relies on try-except blocks, which may miss complex issues
- ConversationBufferMemory may have storage limitations for long conversations

1. "list for me the presidents of the US"
2. "how many were assassinated?"
3. "what did i ask first?"
4. "what questions did i ask"

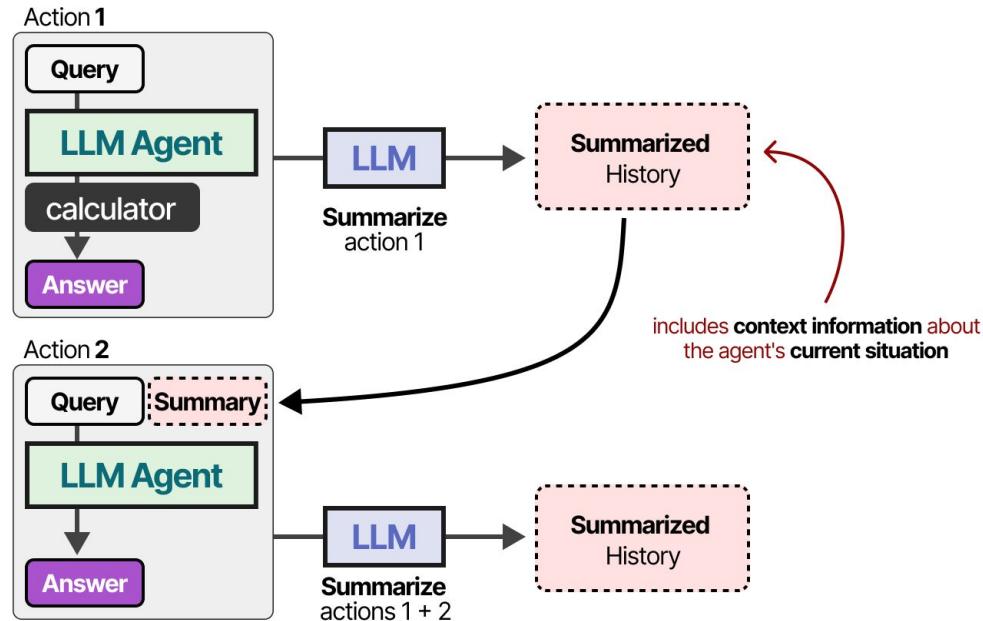
Memory

The most straightforward method for enabling short-term memory is to use the model's context window, which is essentially the number of tokens an LLM can process. Otherwise you can use database storage for Long-term memory.



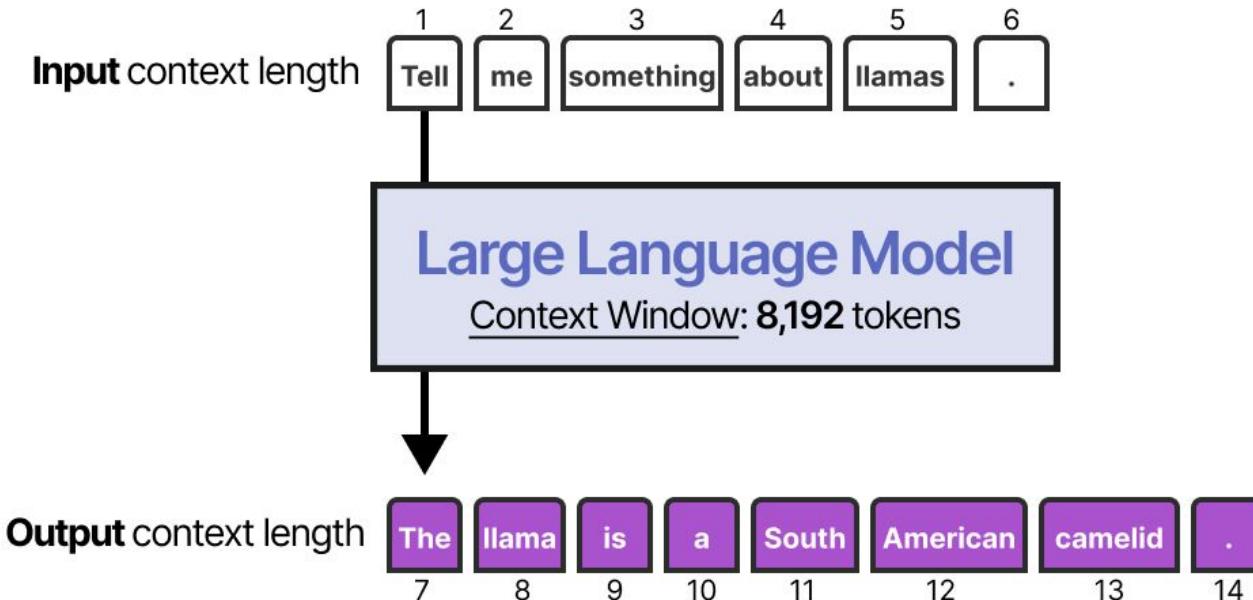
Short-Term Memory

For models with a smaller context window, or when the conversation history is large, we can instead use another LLM to summarize the conversations that happened thus far.



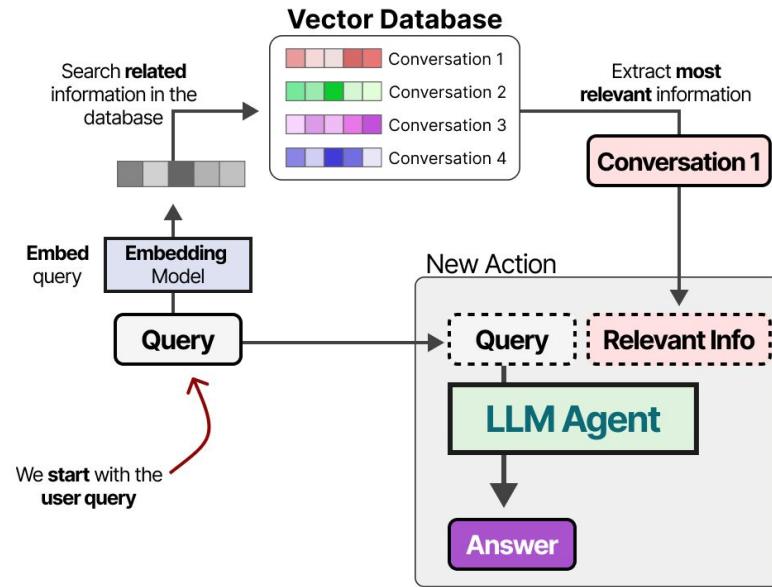
Short-Term Memory

The most straightforward method for enabling short-term memory is to use the model's context window, which is essentially the number of tokens an LLM can process.



Retrieval-Augmented Generation (RAG).

After building the database, we can embed any given prompt and find the most relevant information in the vector database by comparing the prompt embedding with the database embeddings.





Python Frederick Developers



Ask Questions about Generative AI Agents

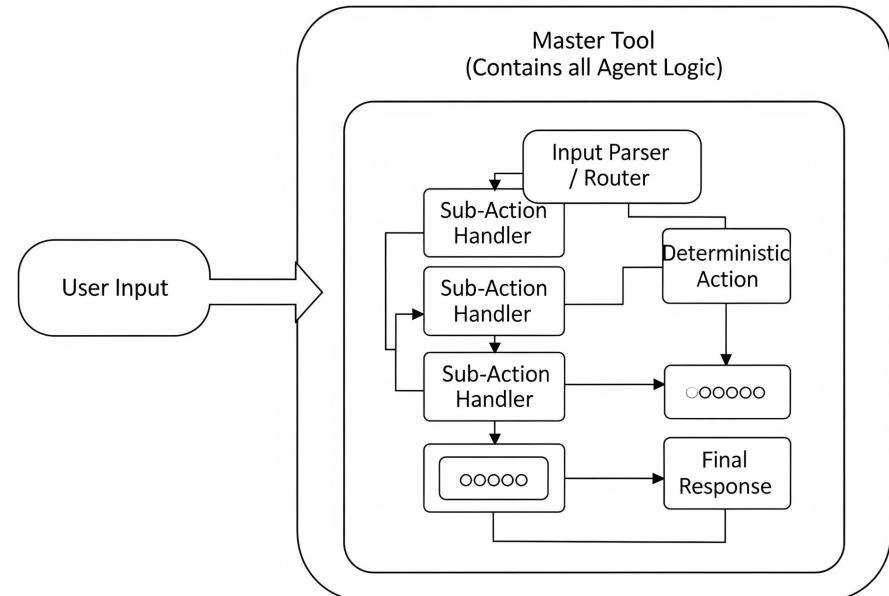
WorkFlows

Rules-Based Tool-as-Agent Workflow Demo

This code represents a **Rules-Based, Tool-as-Agent Workflow**.

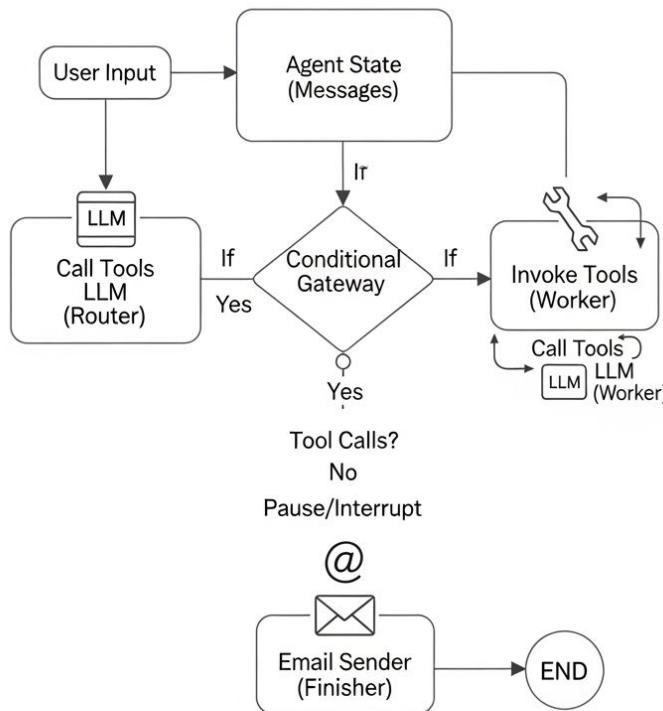
- **Rules-Based:** The core logic for understanding commands is based on a predefined set of rules (`if/elif` statements and JSON parsing), not on the probabilistic reasoning of an LLM.
- **Tool-as-Agent:** The `FileExplorerTool` is so comprehensive that it contains the entire agentic logic (parsing, routing, and execution). The `FilesAgent` class is merely a shell or a host for this "master tool." The tool *is* the agent.

Tool-as-Agent



Ai-travel-agent Demo

LangGraph Agent





`tool_agent_tutorial.py`:

An advanced AI agent demonstrating multi-tool usage for complex problem-solving.

Key Components:

Custom Tools:

`CalculatorTool`: Performs precise mathematical calculations.

`WebSearchTool`: Uses DuckDuckGo API to search the web.

`PythonCodeExecutor`: Safely executes simple Python code with safety checks.

ToolUsingAgent Class:

Powered by GPT-4.1 with low temperature (0.1) for consistent responses.

Built on LangChain's OpenAI Functions Agent framework.

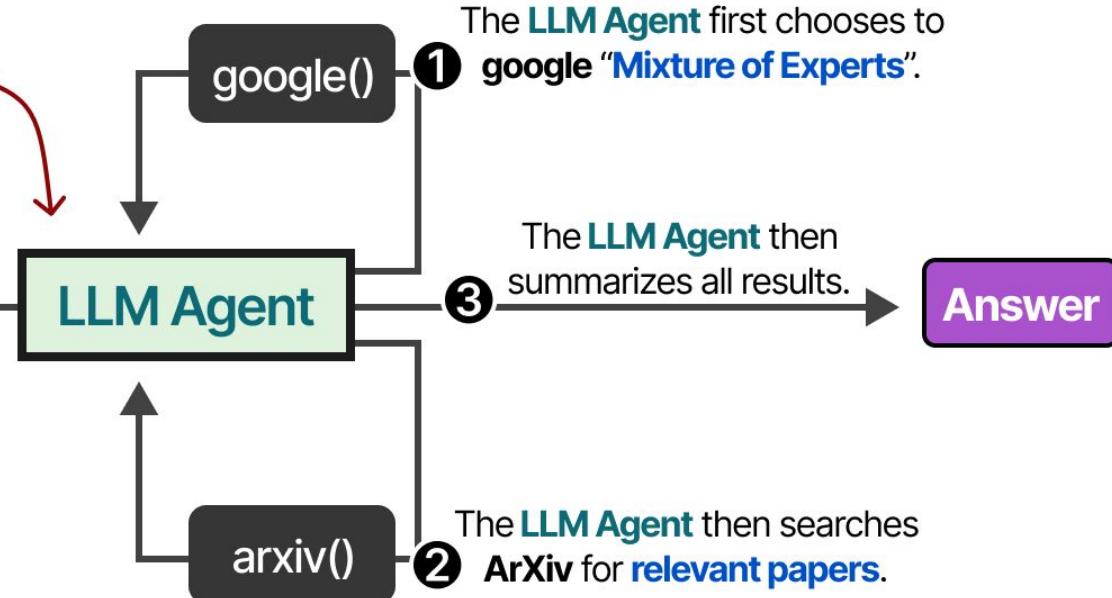
Maintains detailed conversation history for context

Tool Workflow

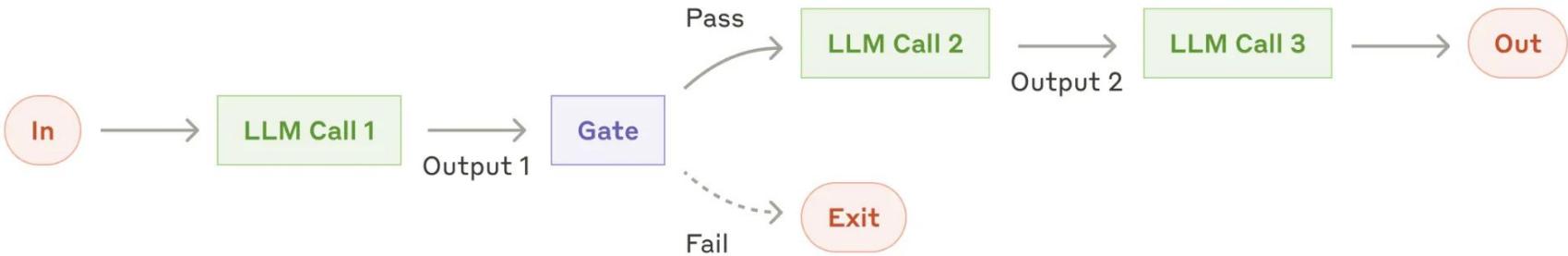
The LLM Agent chooses which tools to use and their order.

Research
“Mixture of Experts”.

The LLM can autonomously choose which tool to use and or use a workflow of LLM calls (but with autonomous selection of actions/tools/etc.).

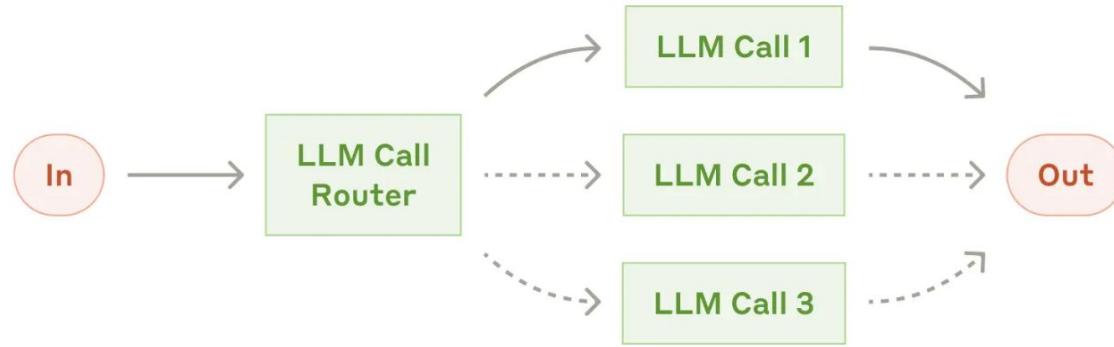


Workflow: Prompt chaining



Prompt chaining decomposes a task into a sequence of steps, where each LLM call processes the output of the previous one. You can add programmatic checks (see "gate" in the diagram below) on any intermediate steps to ensure that the process is still on track.

Workflow: Routing



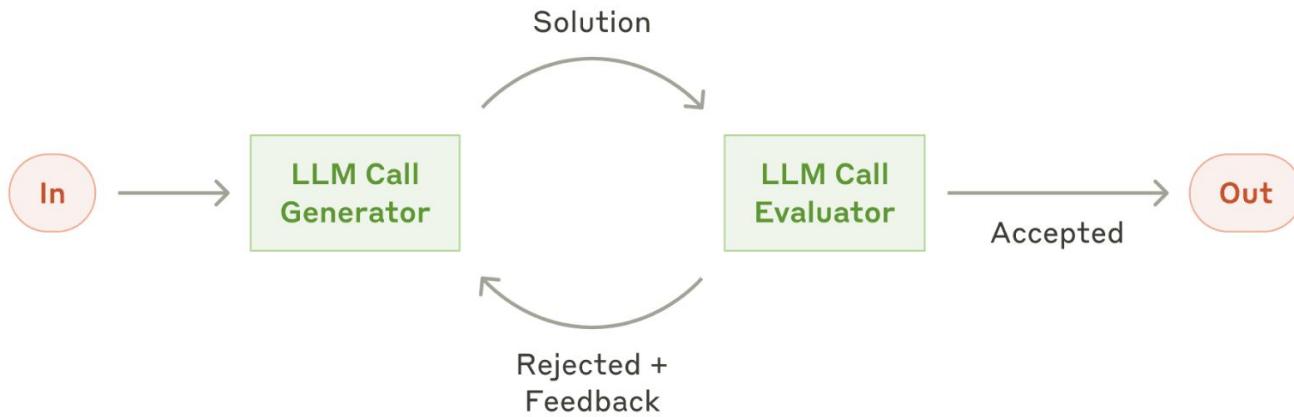
Routing classifies an input and directs it to a specialized followup task. This workflow allows for separation of concerns, and building more specialized prompts. Without this workflow, optimizing for one kind of input can hurt performance on other inputs.

Workflow: Parallelization



When to use this workflow: Parallelization is effective when the divided subtasks can be parallelized for speed, or when multiple perspectives or attempts are needed for higher confidence results. For complex tasks with multiple considerations, LLMs generally perform better when each consideration is handled by a separate LLM call, allowing focused attention on each specific aspect.

Workflow: Evaluator-optimizer



When to use this workflow: This workflow is particularly effective when we have clear evaluation criteria, and when iterative refinement provides measurable value. The two signs of good fit are, first, that LLM responses can be demonstrably improved when a human articulates their feedback; and second, that the LLM can provide such feedback. This is analogous to the iterative writing process a human writer might go through when producing a polished document.

Workflow: Orchestrator-workers



In the orchestrator-workers workflow, a central LLM dynamically breaks down tasks, delegates them to worker LLMs, and synthesizes their results.



Python Frederick Developers

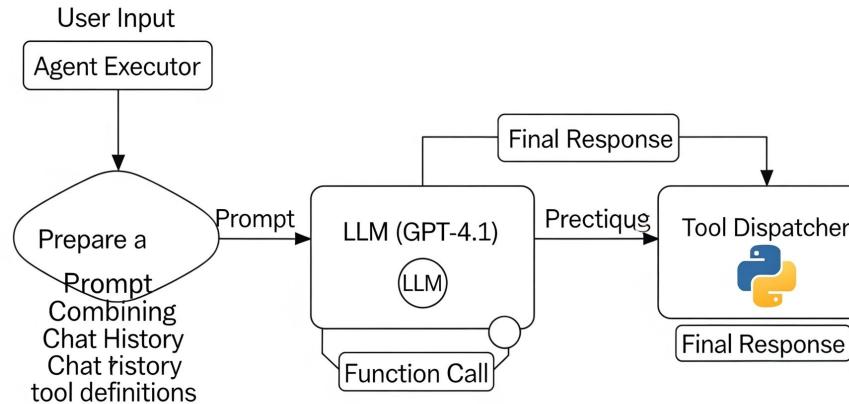


Ask Questions about Generative AI Agents

Tool_agent_tutorial

Agentic Workflow Diagram

This image shows how the agent uses OpenAI's Function Calling capability to decide when and how to use the tools you've provided.



Planning and Reasoning



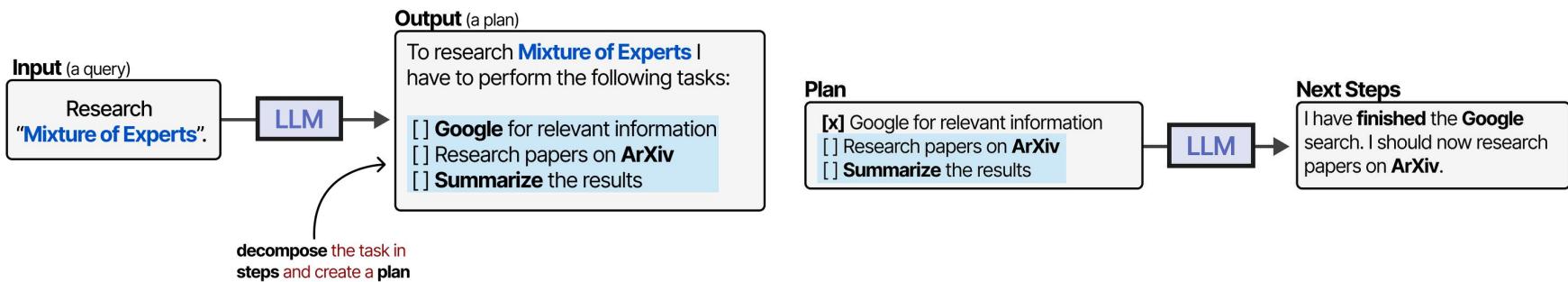
Planning

This is where planning comes in. Planning in LLM Agents involves breaking a given task up into actionable steps.

Tool use allows an LLM to increase its capabilities. They are typically called using JSON-like requests.

But how does the LLM, in an agentic system, decide which tool to use and when?

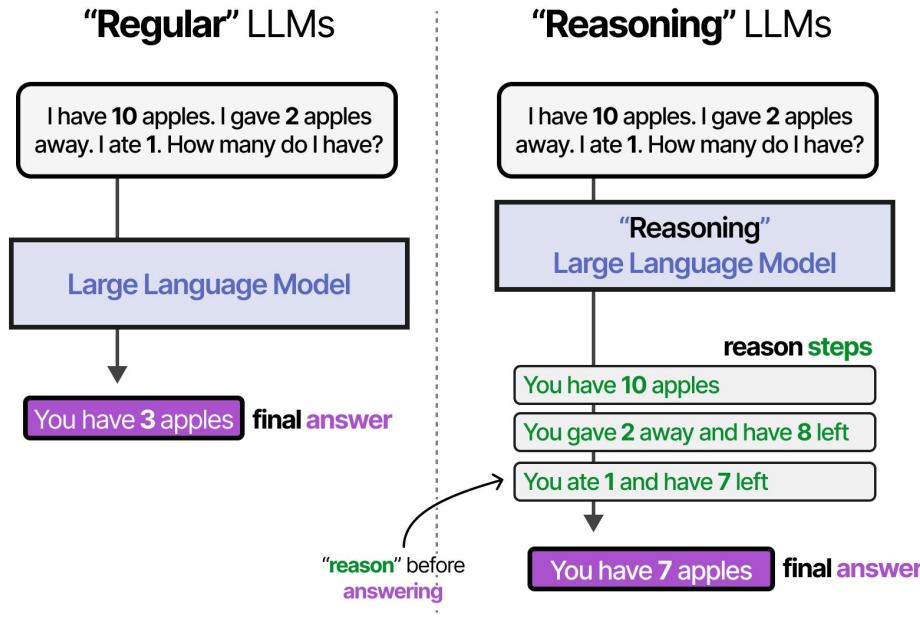
.



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.

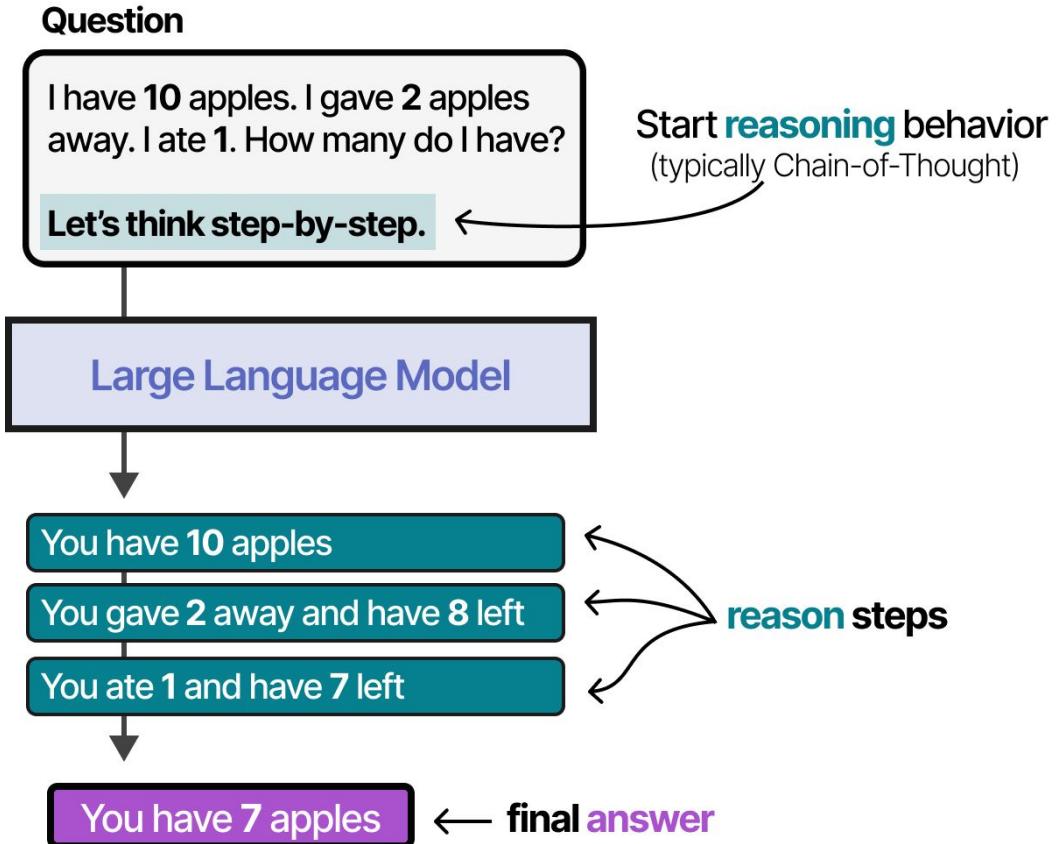
“Reasoning LLMs” are those that tend to “think” before answering a question



Reasoning behavior

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**).

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.



What "ReAct" Means



- ReAct stands for "Reason and Act". It's a framework that enables an agent to solve complex problems by mimicking a human-like thought process.
- Instead of just trying to give a final answer immediately, the agent follows an iterative loop:
- Thought: The agent examines the user's input and its available tools, forming a thought about what needs to be done.
- Action: Based on its thought, it chooses an action to take, selecting a tool.
- Observation: The action is executed, and the result is returned to the agent as an observation.

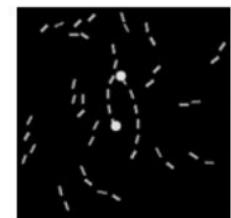
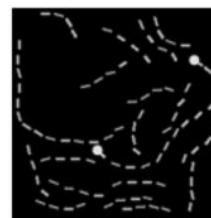
Neurosymbolic Programming with Scallop

<https://www.scallop-lang.org/>

Pathfinder, Long Range Connectivity Reasoning

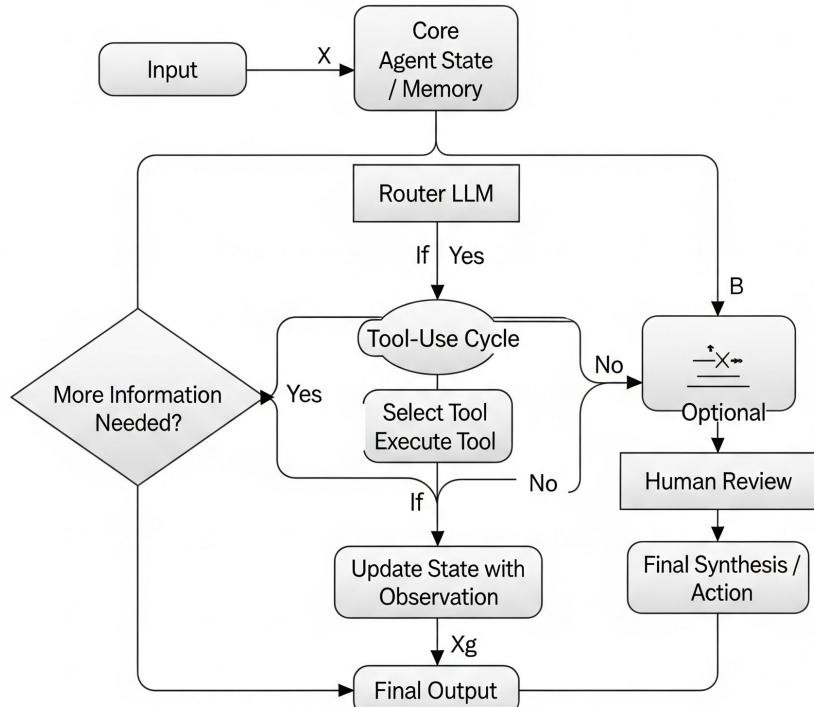
In this task, we are provided with black-and-white images containing two dots and dashed lines. The goal is to determine whether the two dots are connected by a dashed line. This task can be programmed in just a few lines in Scallop using a very simple neural architecture for detecting dots and dashes, together with the logic rules below, and outperforms state-of-the-art transformers.

```
rel path(x, y) = dash(x, y)
rel path(x, y) = path(x, z), dash(z, y)
rel is_connected() = dot(x), dot(y), path(x, y), x != y
```



Stateful, Cyclical, Graph-Based Agent

General Agentic Workflow



Stateful: The agent maintains a memory of the entire conversation and all tool outputs ([AgentState](#)). It doesn't just react to the last message; it has the full context.

Graph-Based: Instead of a simple linear chain, the workflow is defined as a graph (a state machine) with nodes and edges. This allows for complex logic, branching, and cycles, which is managed by [LangGraph](#).

Cyclical Tool-Use Loop: This is the core of the information-gathering process. The agent can use a tool, look at the result, and decide if it needs to use another tool (or the same one again). It loops through this "Think -> Act -> Observe" cycle until it determines it has enough information to complete the final task.

Separation of Concerns: A key feature of this pattern is using different components for different jobs. One LLM acts as the "Router" to decide which tool to use, while another acts as a "Generator" to format the final email.

File System Agent

- File System Agent: An intelligent assistant designed for organizing files.

This code represents a **Rules-Based, Tool-as-Agent Workflow**.

- **Rules-Based:** The core logic for understanding commands is based on a predefined set of rules (`if/elif` statements and JSON parsing), not on the probabilistic reasoning of an LLM.
- **Tool-as-Agent:** The `FileExplorerTool` is so comprehensive that it contains the entire agentic logic (parsing, routing, and execution). The `FilesAgent` class is merely a shell or a host for this "master tool." The tool *is* the agent.

