AMERICAN UNIVERSITY OF BEIRUT

Fall 2025

EECE 503P/798S: Agentic Systems

C3 - Introduction to Agents

Lesson Objectives

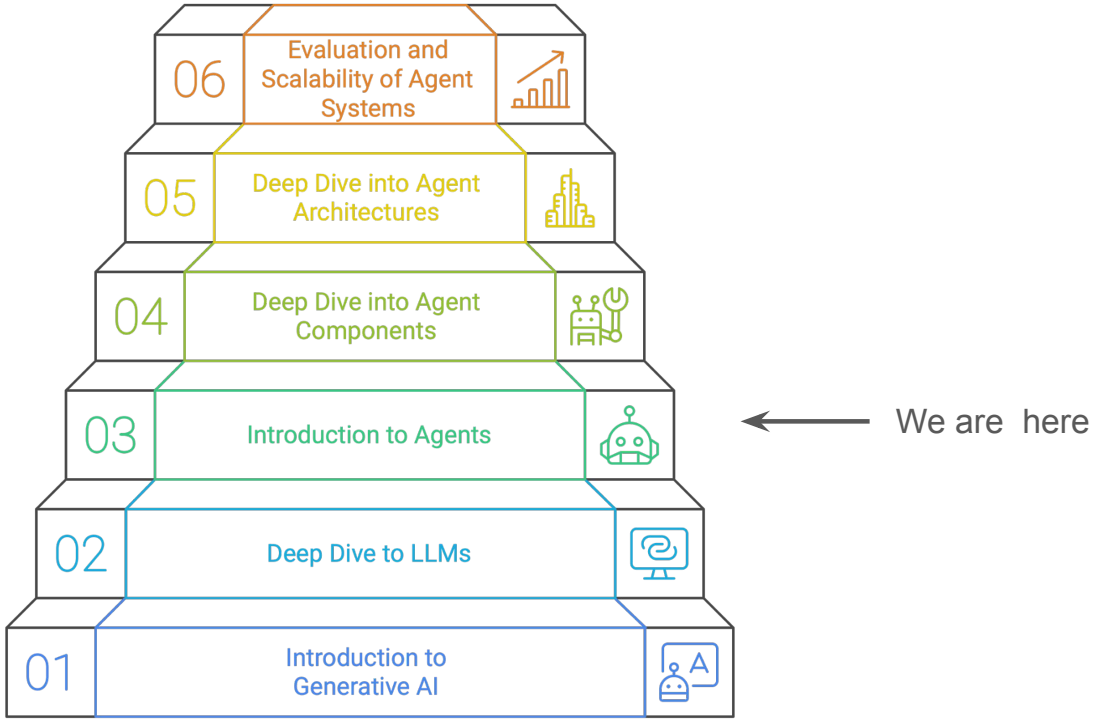
- Define what an **AI agent** is and identify real-world applications
- Explain the **Reinforcement Learning (RL)** setup of an agent
- Understand what makes a system **agentic**
- Break down the **core components** of an LLM-based agentic system (tools, memory, planning, etc.)
- Recognize **key challenges** in deploying agentic systems (e.g., hallucinations, cost, debugging)

Introduction

This chapter is inspired by the Udemy course: The Complete Agentic AI Engineering Course (2025) by Ed Donner.

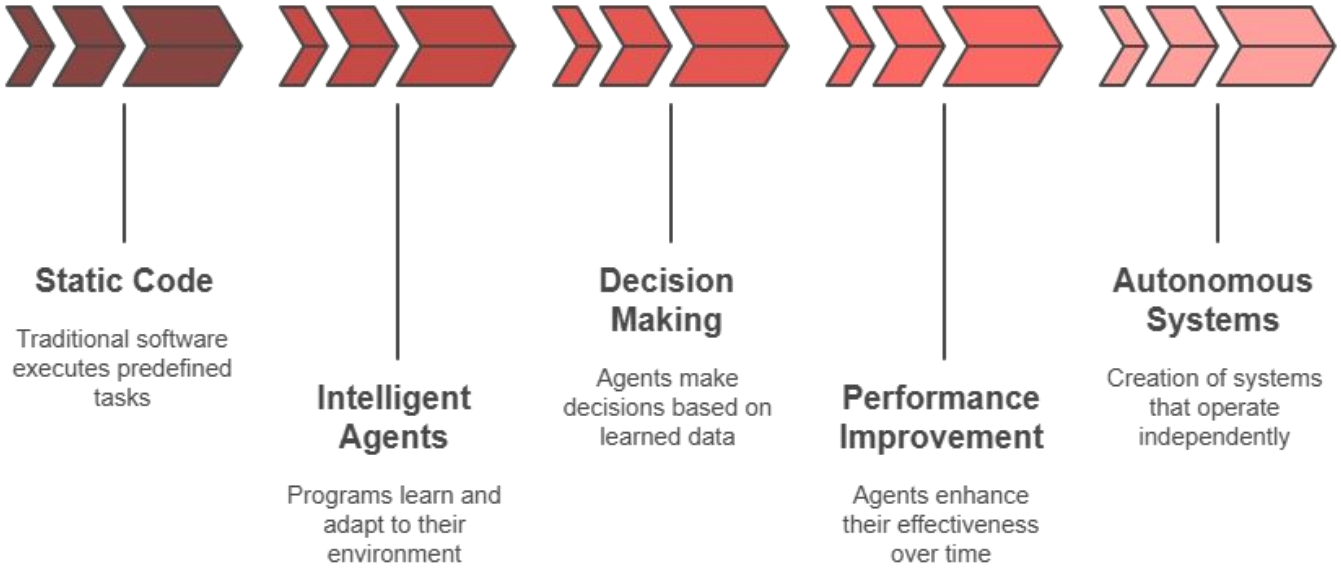
You'll find everything you need to setup your environment in this Github repo: agents by Ed Donner.

Course Timeline



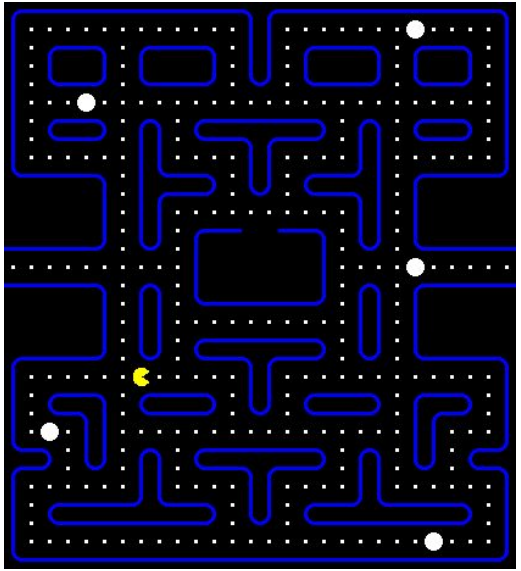
Introduction

From Static Code to Intelligent Agents



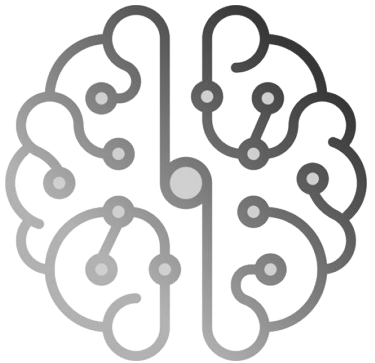
Agents are Not New

- The concept of AI agents **predates** LLMs by decades
- **Early agents (1990s–2000s):**
  - Based on **symbolic AI, rule-based reasoning,** or **classical planning**
  - Used in **robotics, game agents,** and **expert systems**
- Focused on **perception + action** within well-defined **environments**



Why Now?

- Modern LLMs unlocked true agentic behavior
- What LLMs Bring:
  - Language **understanding** and **generation**
  - **Chain-of-thought** reasoning
  - **Generalization** across domains
  - Multi-step **planning** with **feedback**



The Agent Need

Business Needs

Businesses require intelligent automation, not mere scripts.

Developer Needs

Developers desire systems capable of learning and adapting.

User Needs

Users need agents understanding goals, not just command execution.

Definition and Applications of Agents

What is an Agent?

- An agent is an **autonomous system** that:

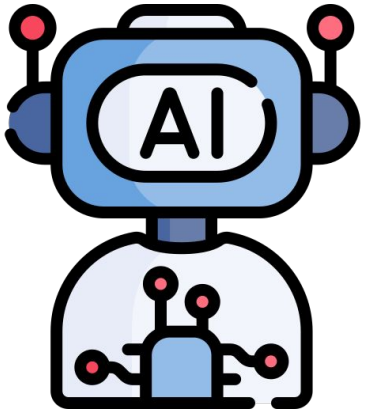
Observes

Reasons

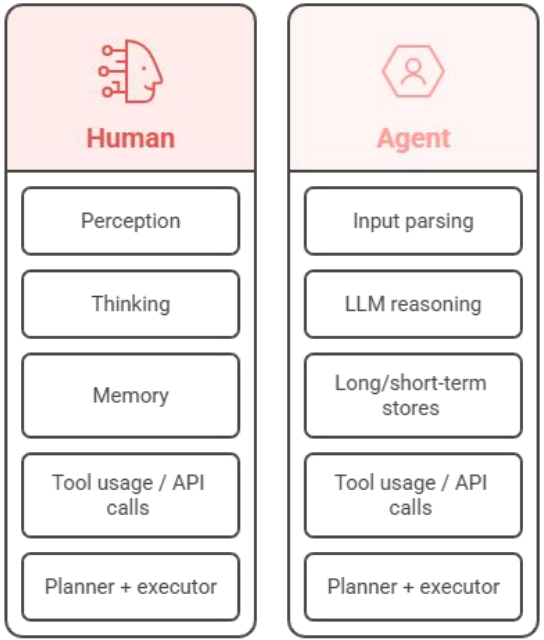
Acts toward achieving goals

What is an Agent?

- **Agent ≠ Chatbot:**
  - Not just conversation, but action
- **Agent ≠ Script:**
  - Not just automation, but goal-driven reasoning
- Varying levels of **autonomy**
  - from tool caller to planner



How Agents Resemble Human Thinking



Is it an Agent?

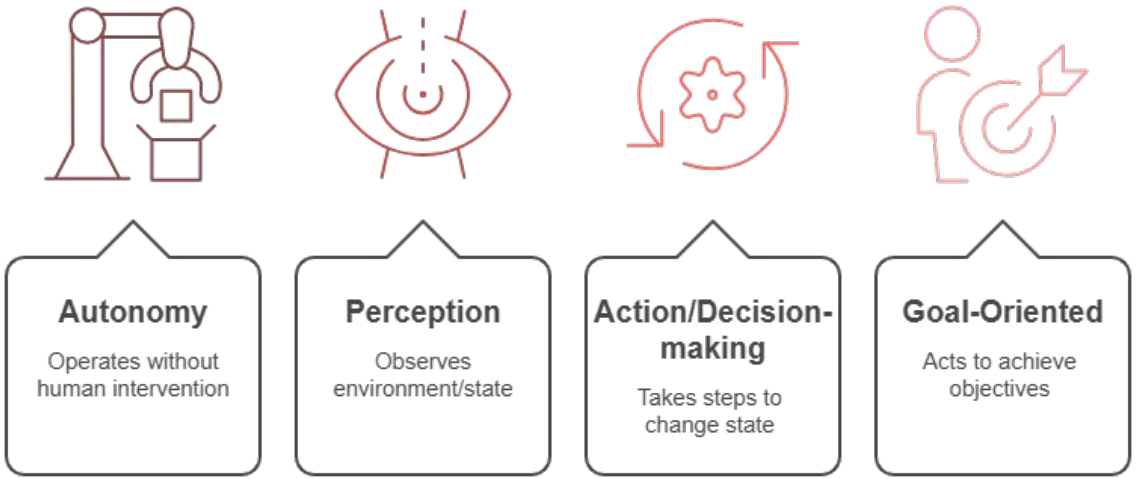
- Is a smart thermostat an agent? 

- Smart Light Timer? 
- Scheduled Email Sender?
- A Roomba (robot Vacuum Cleaner)? 

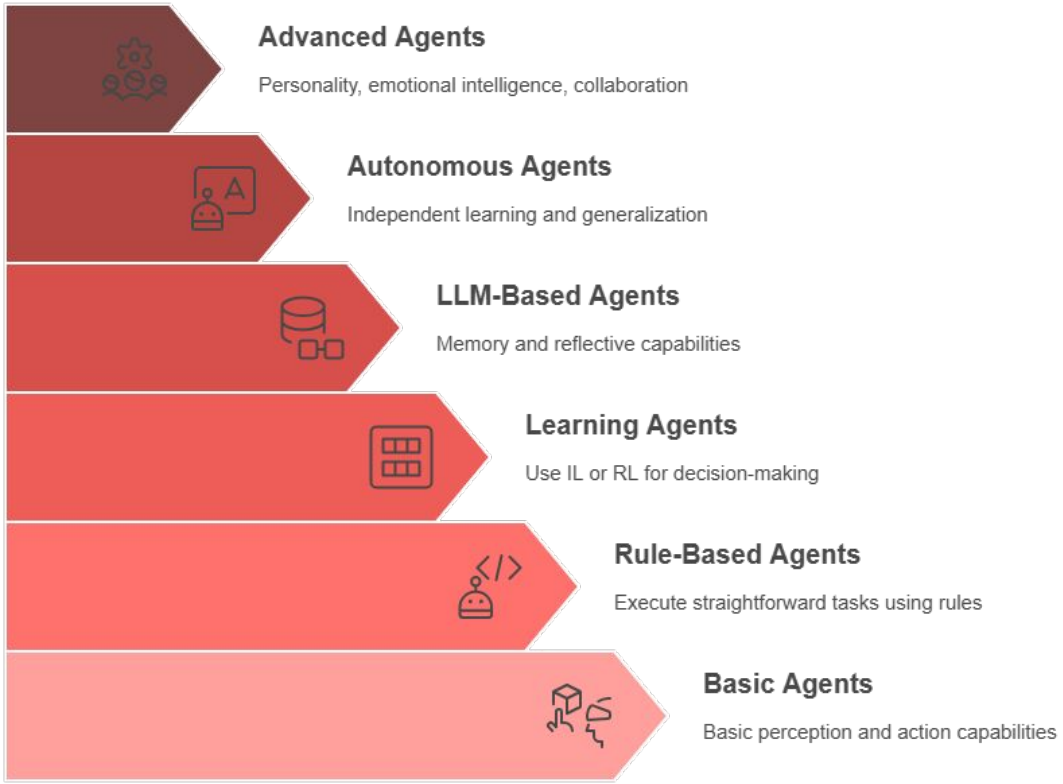
Is it an Agent?

System	Autonomy	Perception	Action	Goal-Oriented	Agent?
Smart Thermostat	✅	✅	✅	✅	✅
Roomba	✅	✅	✅	✅	✅
Smart Light Timer	✅	❌	✅	❌	❌
Scheduled Email Sender	✅	❌	✅	❌	❌

Key Characteristics of Agents

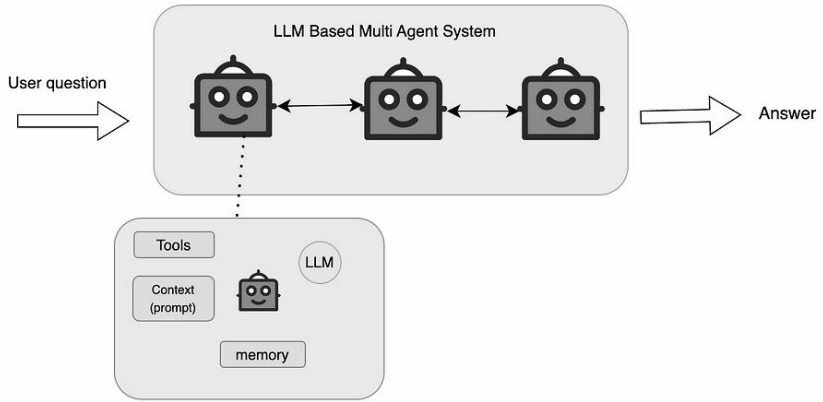


Categorization Based on Task Complexity



AI/LLM-Based Agents

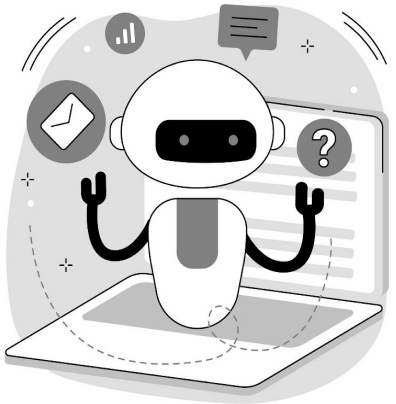
- Combine the **classic** agent loop with **natural language understanding**
- Use **LLMs** (e.g., ChatGPT, Claude, Gemini) as **reasoning** engine
- Can **interpret goals, choose tools,** and **plan actions** dynamically
- **Examples:** LangChain agents, AutoGPT, Devin, CrewAI



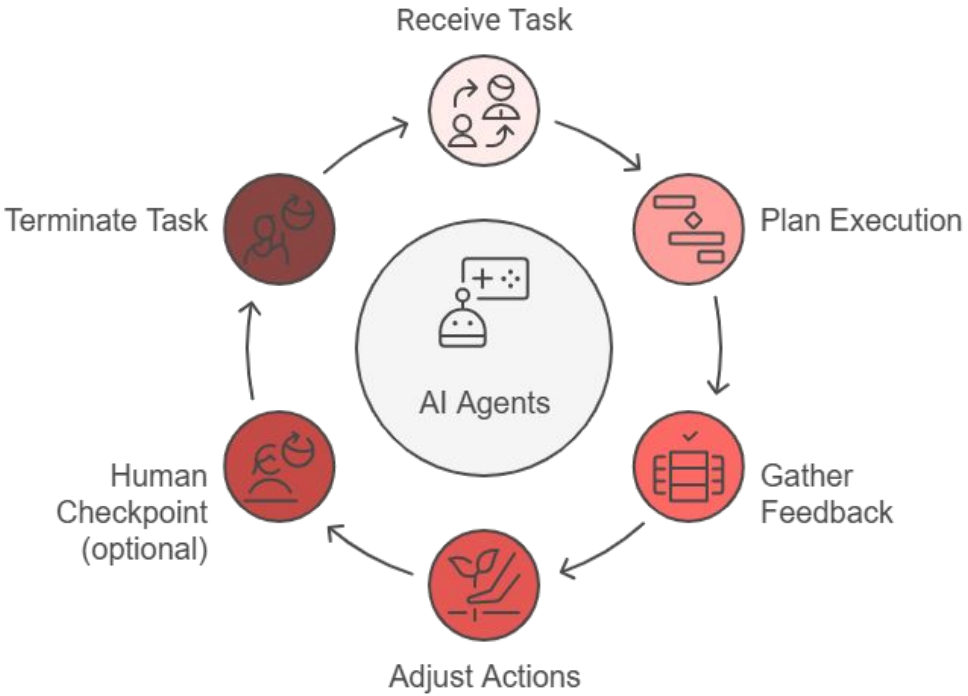
AI Agents Definition

AI agents are programs where LLM **outputs** govern the **workflow**

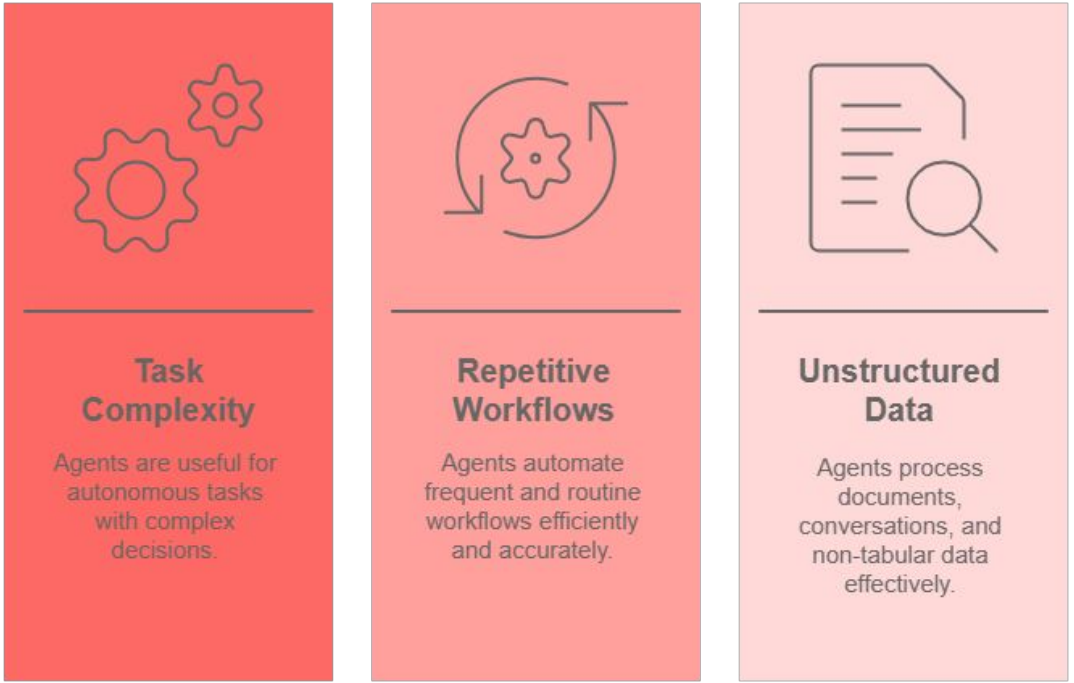
1. **Multiple** Calls to **LLMs**
2. LLMs with the Ability to Use **Tools**
3. An **Environment** Where LLMs Interact
4. A **Planner** Coordinating Activities
5. **Autonomy**



Core Lifecycle of an Agent



When to use Agents



Customer Support Agents

- Blend **conversational flow** with **action-taking** via integrated tools
- Access **user data, order history,** and **knowledge base** articles
- **Automate** actions (e.g., refunds, ticket updates)
- Human oversight ensures **trust** and **quality**

Coding Agents

- Solve programming problems **autonomously**
- Use **automated tests** for verification and feedback
- Operate in structured, well-defined **environments** (e.g., GitHub issues)
- Human review ensures **alignment** with broader system goals

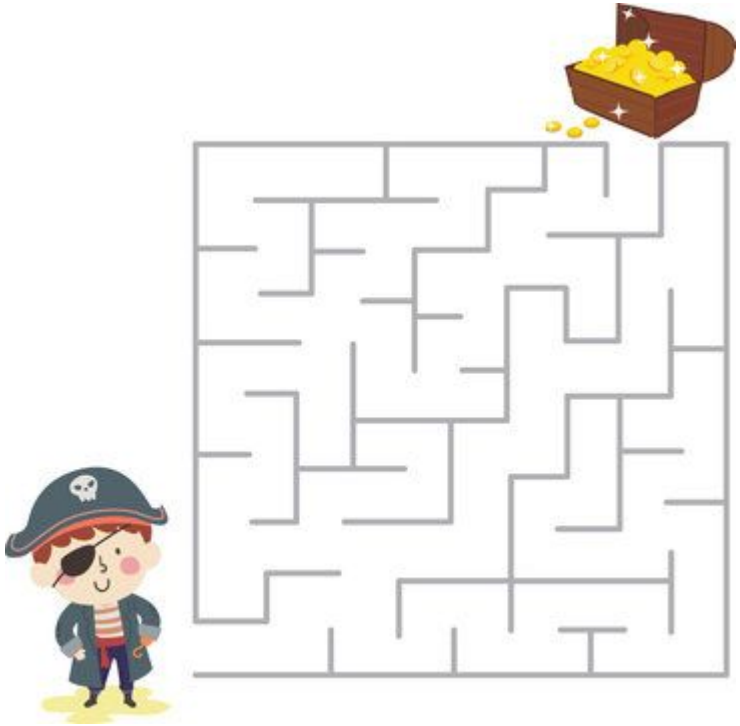
The RL setup of agents

Find the Treasure

No instructions. No map.

Just one goal: **“Find the treasure.”**

- You try going left — *dead end*
- You try going right — *trap!*
- You learn from failure
- You try again — *you get better*
- Eventually, you succeed
- That’s not luck — ***that’s learning***



What Is Reinforcement Learning?

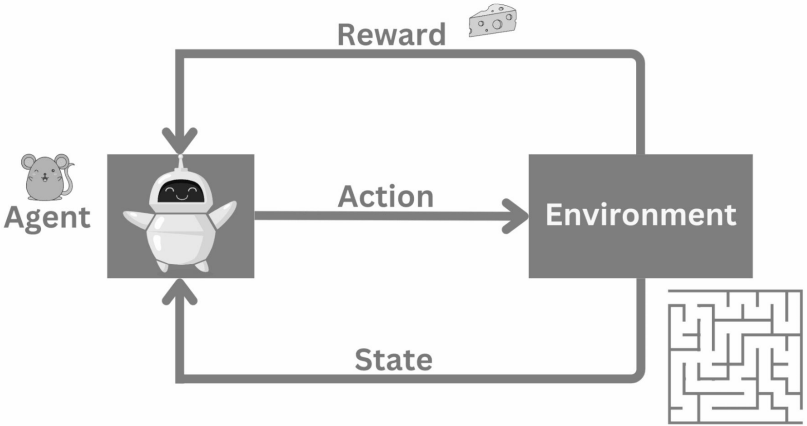
RL is like teaching a pet tricks — give it treats (**rewards**) when it does the right thing!

Agents **learn** what to do by:

- Trying actions
- Seeing what works (reward)
- Trying again (better next time!)

What Is Reinforcement Learning?

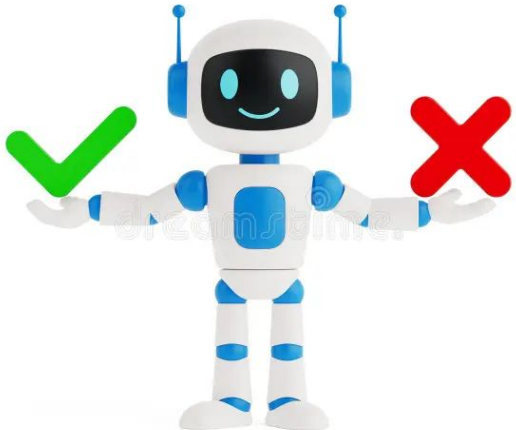
- An agent is a **decision-making** entity
- It interacts with an **environment** through **trial and error**
- Its goal is to **maximize rewards** over time
- Inspired by how humans and animals **learn** from experience



The Agent-Environment Loop

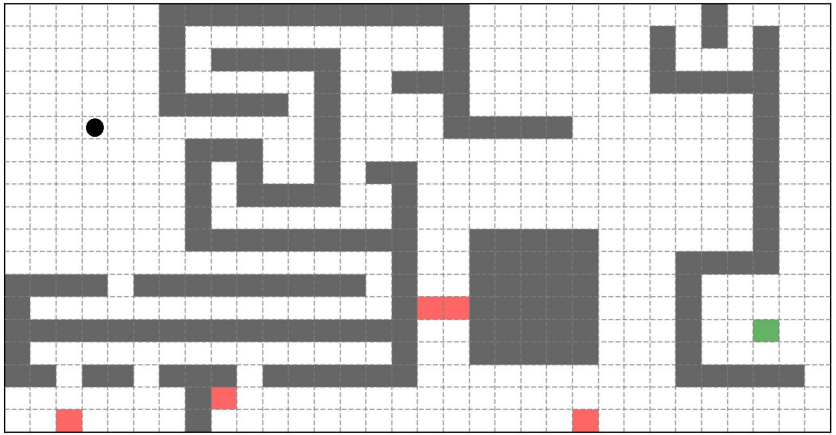
Core Components of RL- Agent

- The AI system that makes **decisions** in order to achieve a **goal**
- Can be a robot, game character, chatbot, or software



Core Components of RL- Environment

- Everything the agent **interacts with** and receives **feedback** from
- Provides the current **state** and **reward**
- Can be **physical** (robot's world) or **digital** (video game)



Core Components of RL- State

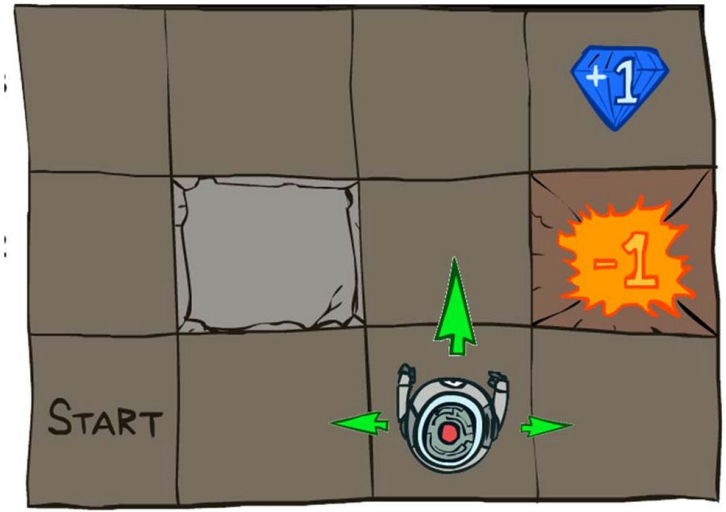
- The agent's view of the **current situation** in the environment
- Can be **full** (chess board) or **partial** (robot with limited sensors)
- Influences which **action** is chosen
- Often represented as **vectors, images,** or **structured data**

Core Components of RL- Action

- A **decision** or **behavior** the agent performs at a **specific** time
- Chosen based on the **current state** and **policy**
- **Examples:** Move left, turn off light, buy stock, answer question
- Some actions may be **better** than others depending on the context

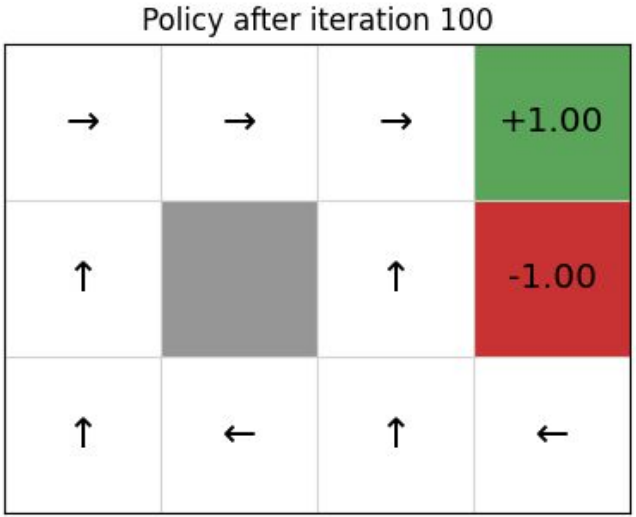
Core Components of RL- Reward

- A signal telling the agent how **good** or **bad** its action was
- **Example:**
  - Positive = good action (e.g., +10 for success)
  - Negative = bad action (e.g., -10 for failure)
- The goal is to **maximize** the **cumulative reward**

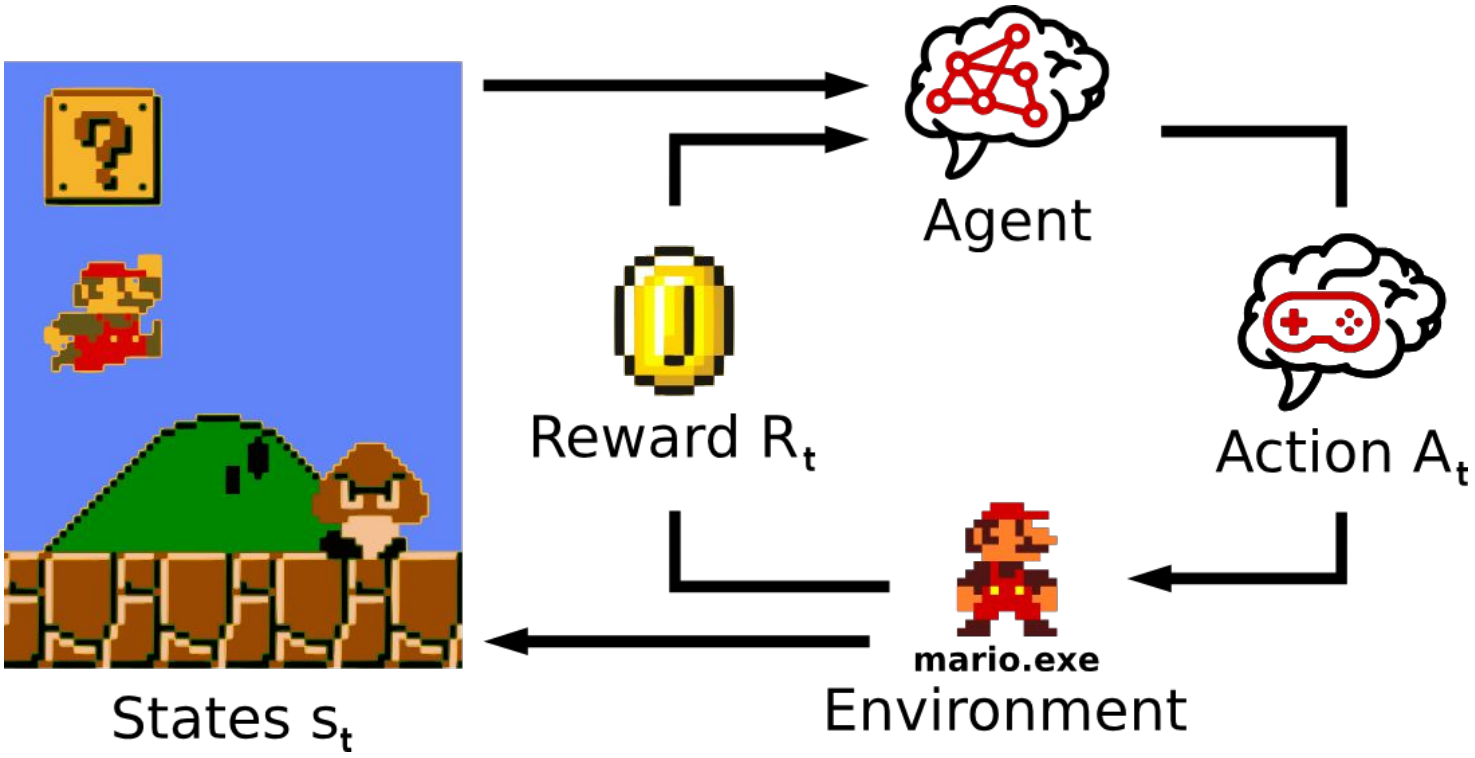


Core Components of RL- Policy

- Mapping from **state to action**: "In this situation, do this."
- The agent's learned or programmed **decision plan**
- **Updated over time** as the agent learns
- In RL, often represented as **π(a|s) = probability of action a in state s**



Reminder: Putting it together



Example: Smart Vacuum Cleaner

State:
room is dirty

Action:
clean or move

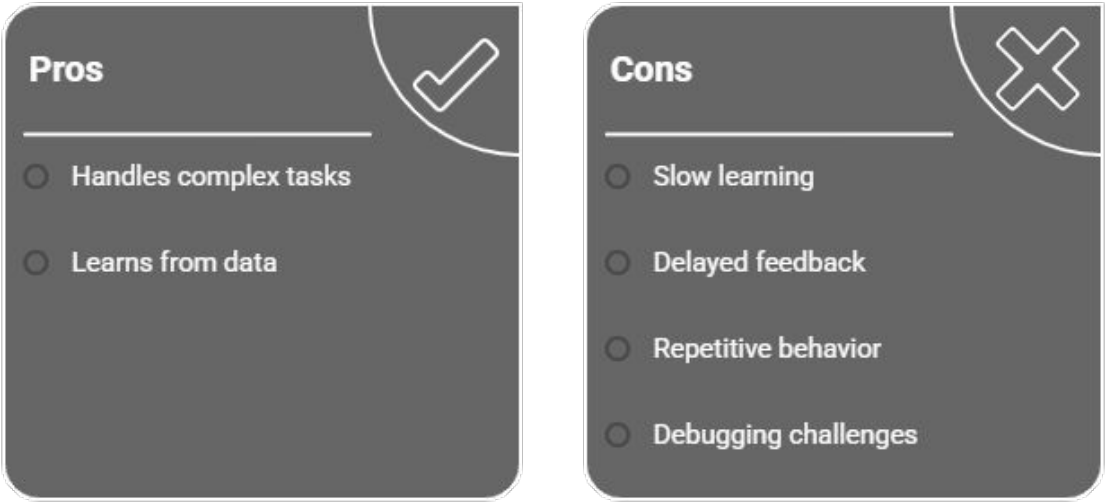
Reward:
+ 1 for cleaning

Policy:
If room is dirty → Clean
If room is clean → Move

RL Vs. LLM Agents

Feature	RL Agent	LLM Agent
Learning Method	Rewards & Trials	Prompts, Tools, Memory
Policy	Trained Function	Emergent from LLM Behavior
Tool Use	Rare	Frequent
Flexibility	Learns from Data	Generalizes via Pretraining

Challenges in RL



Hands_on: Cleaning Agent Navigation in a Grid World

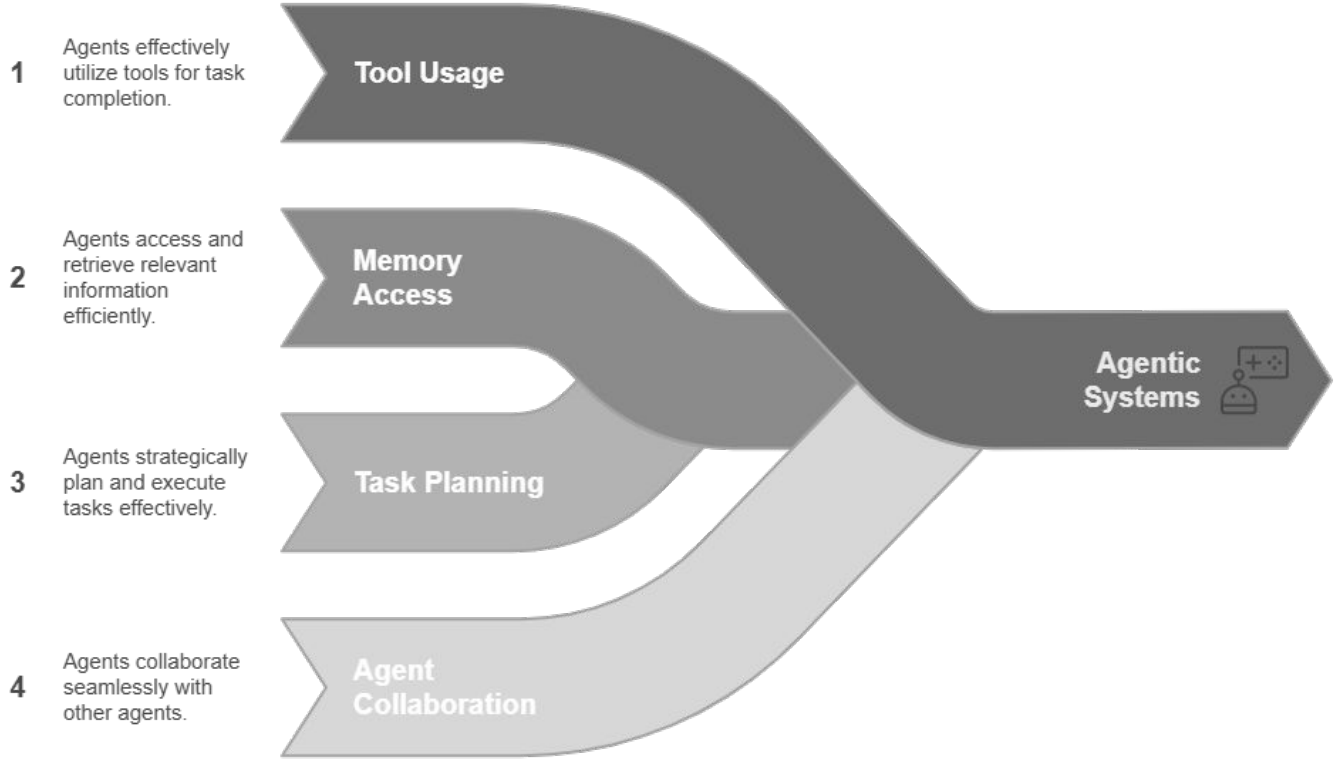
AMERICAN UNIVERSITY OF BEIRUT

Agentic Systems

What does “agentic” even means?

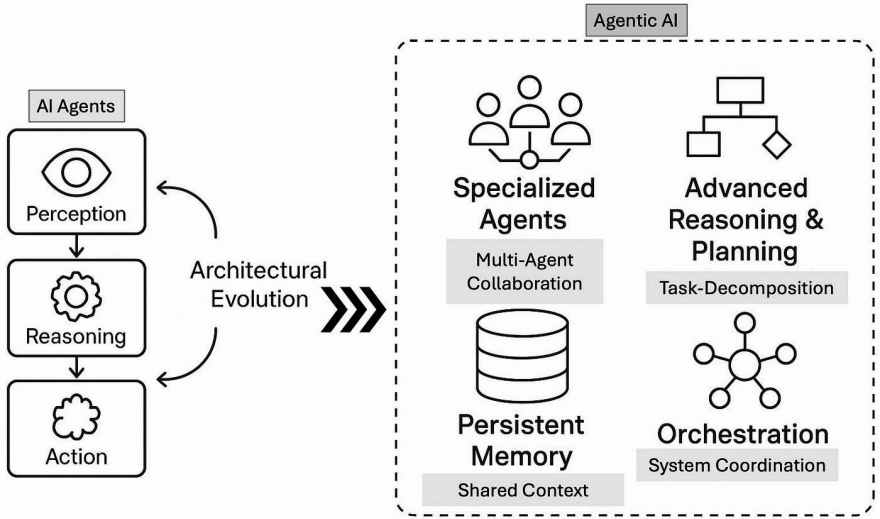
- Derived from "agent" — an entity capable of acting **independently**
- Refers to systems that exhibit **autonomy, goal pursuit,** and **decision-making**
- **Example:**
  - **Non-Agentic AI:** A calculator (responds to direct input)
  - **Agentic AI:** A personal assistant (understands your intent and takes steps to fulfill it)

The Rise of Agentic Systems



Agentic Systems

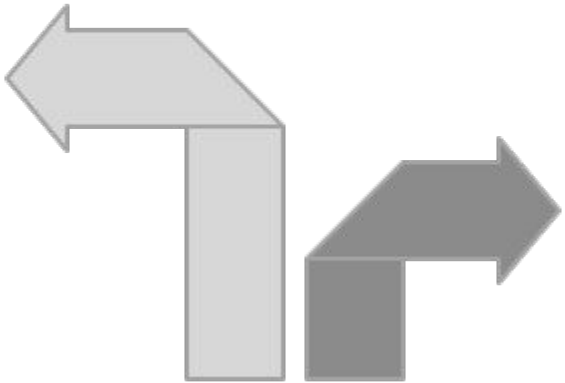
- Systems that coordinate multiple **autonomous** agents
- May include shared **memory**, **planning** components, or **role** assignment
- Can **divide** and **conquer** tasks (task planning + execution + verification)
- Often involve **feedback loops** and **multi-step tool** use



Example: Fraud Detection

Agentic Workflow

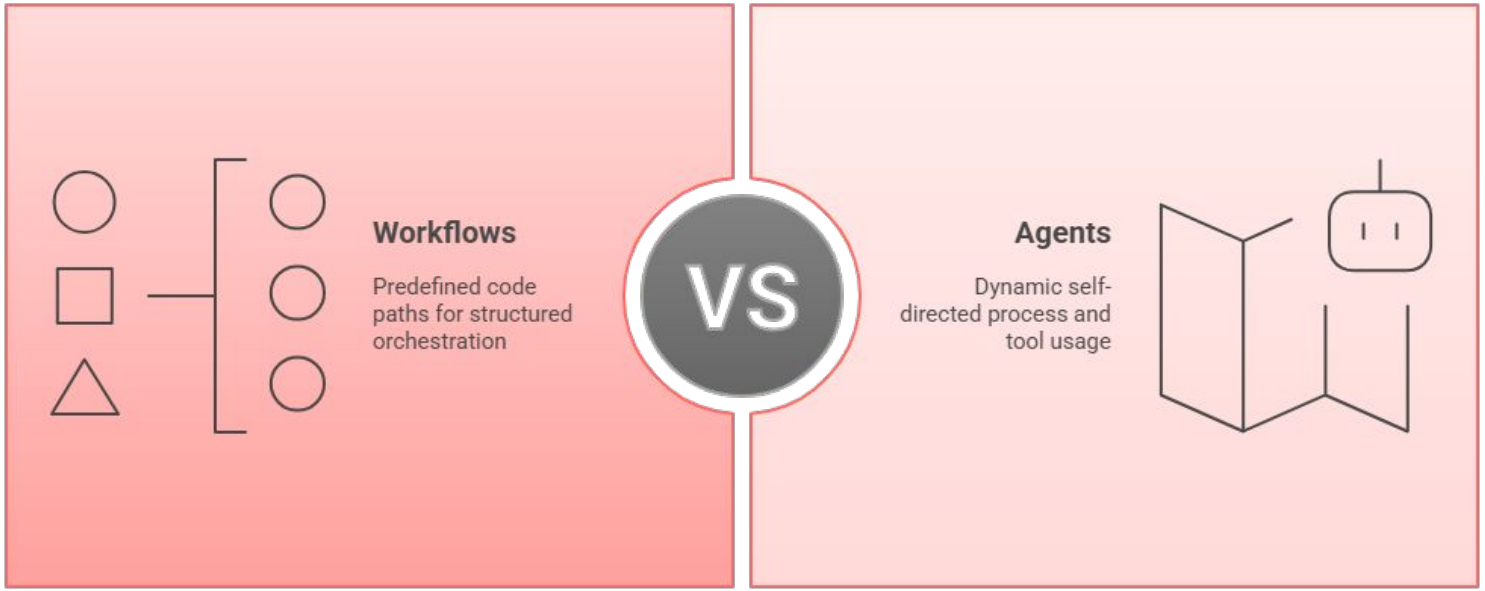
The AI detects an unusual transaction, sends a notification to the customer, and verifies the purchase through a quick app interaction



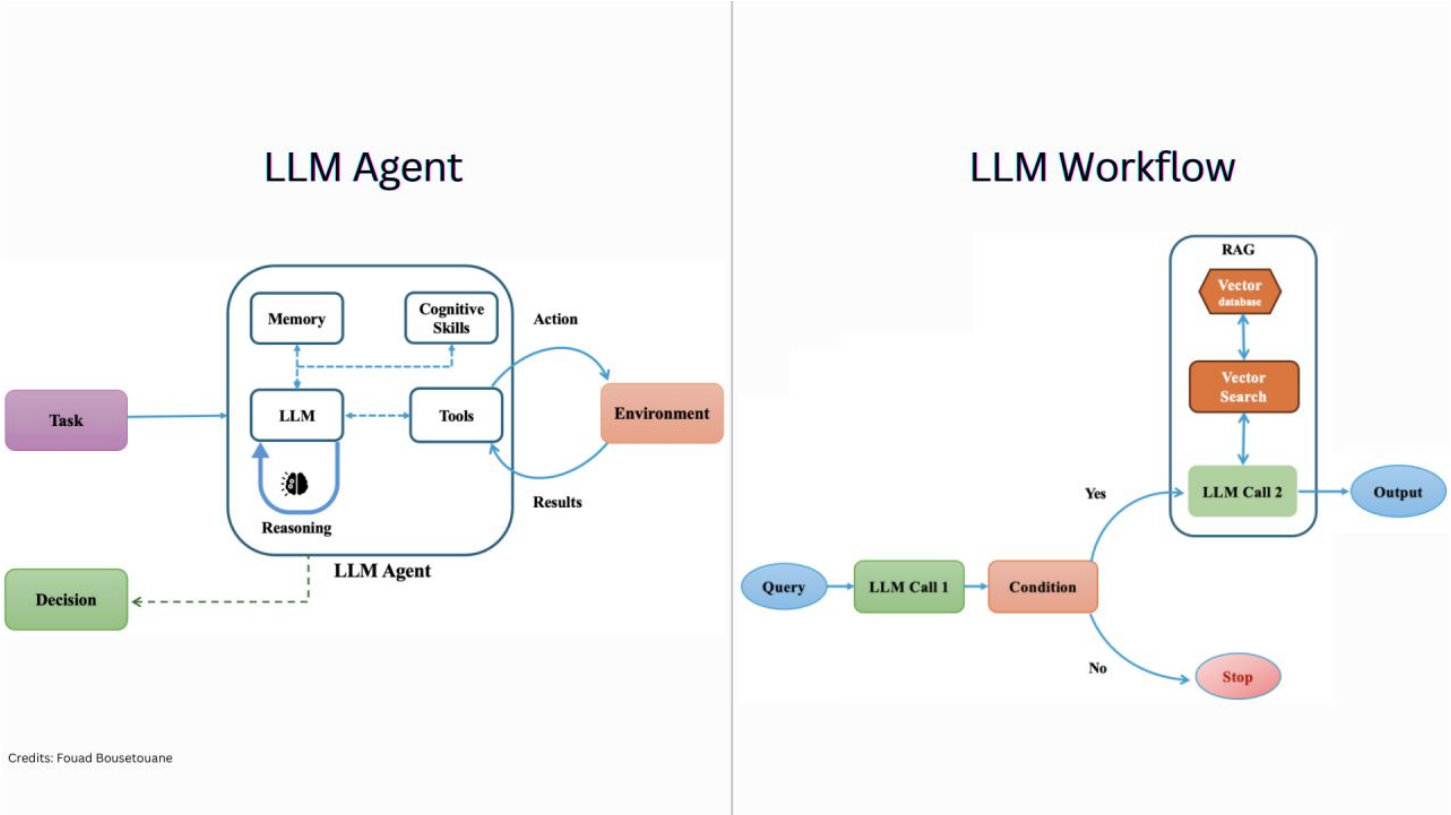
Non-Agentic Workflow

A customer's credit card purchase is flagged as fraudulent, requiring them to call the bank and verify the transaction

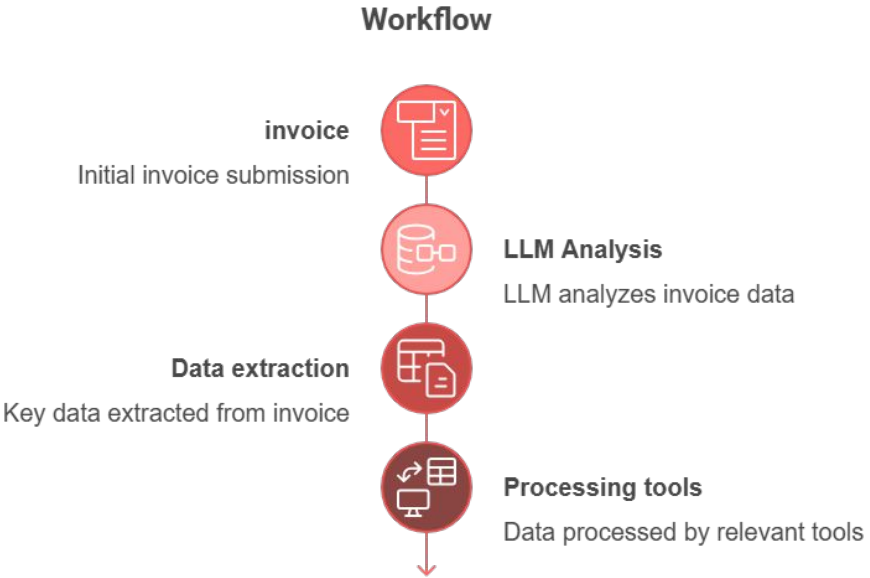
Agentic Systems: Workflows and Agents

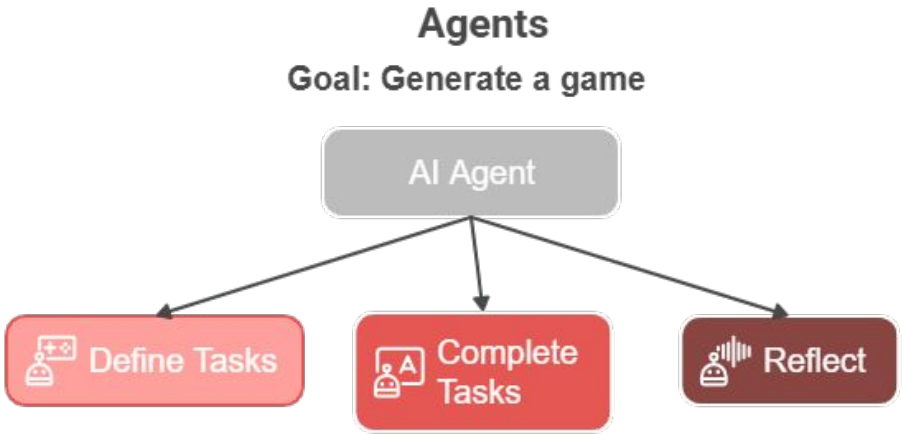


Agentic Systems: Workflows and Agents



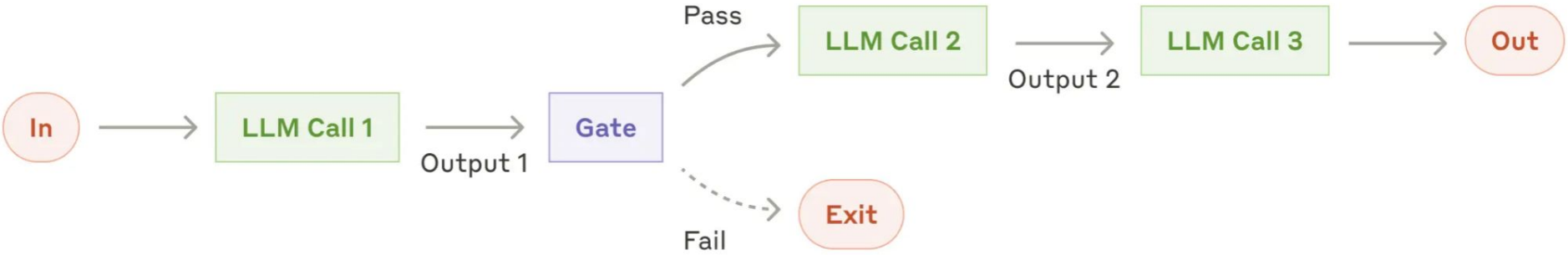
Agentic Systems: Workflows and Agents





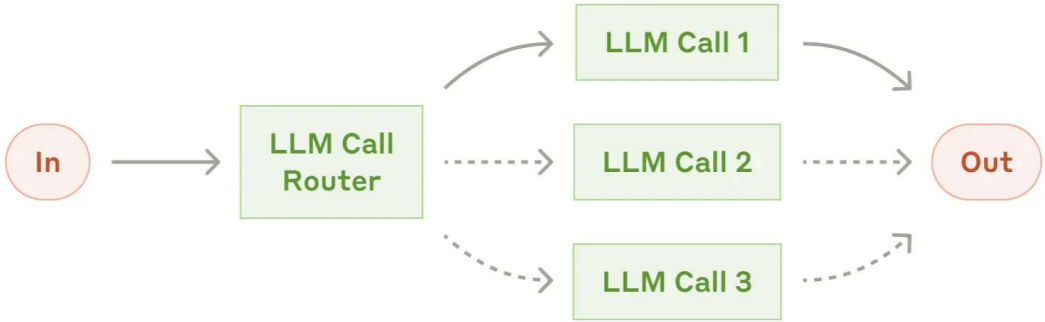
Workflow Design Patterns: Prompt Chaining

- Breaks a **complex task** into a sequence of **smaller LLM steps**
- Use when tasks can be cleanly **decomposed** into fixed subtasks
- **Example:** Write document outline → Validate outline → Generate document



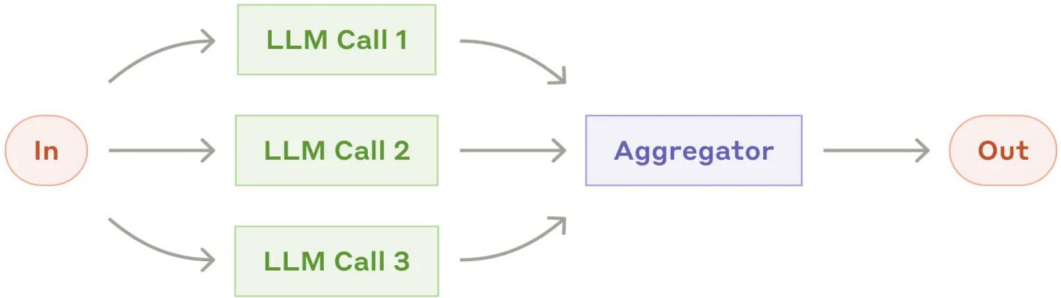
Workflow Design Patterns: Router

- Direct an input into **specialized subtasks**
- Use to **separate concerns** and **improve performance** per category
- **Example:** route customer queries general vs. refund vs. tech support to different workflows



Workflow Design Patterns: Parallelization

- Breaking down tasks and running multiple subtasks **concurrently**
- **Types:**
  - **Sectioning** – Split task into independent parts, run in parallel (Each LLM scores a different eval metric)
  - **Voting** – Run the same task multiple times, then combine outputs (Multiple reviews of code for bugs)
- Use when tasks can be **parallelized**

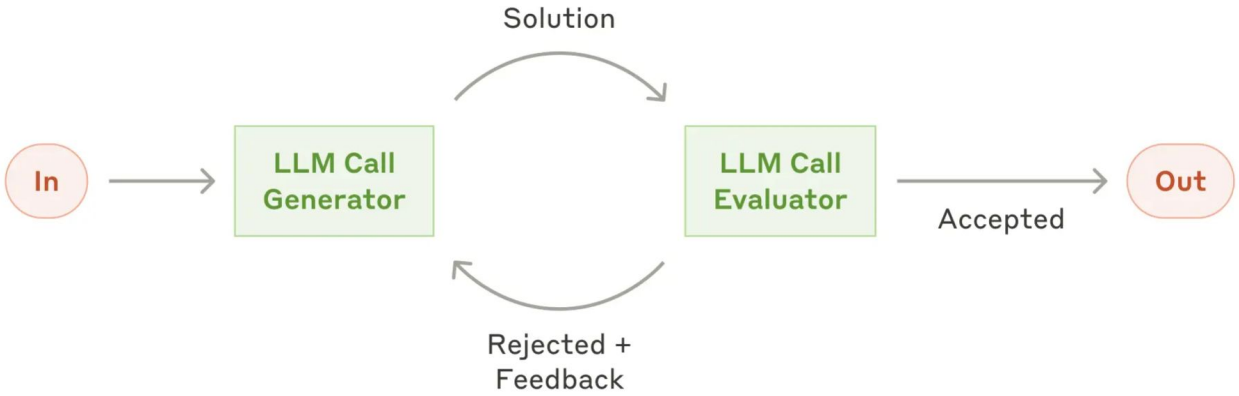


Workflow Design Patterns: Orchestrator-workers

- A central **orchestrator LLM** dynamically breaks down a task, delegates to **worker LLMs**, then **combines** their outputs
- Use when **subtasks vary** depending on the input
- **Example:**Research agents that gather, filter, and synthesize information from diverse sources

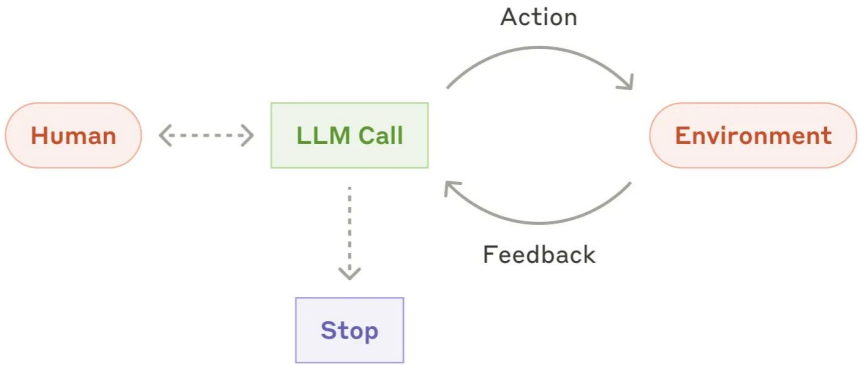
Workflow Design Patterns: Evaluator-optimizer

- LLM output is **validated** by another
- Use when **clear evaluation** criteria exist and **iterative improvement** adds real value
- **Example:** Search tasks refined through repeated evaluation and reruns

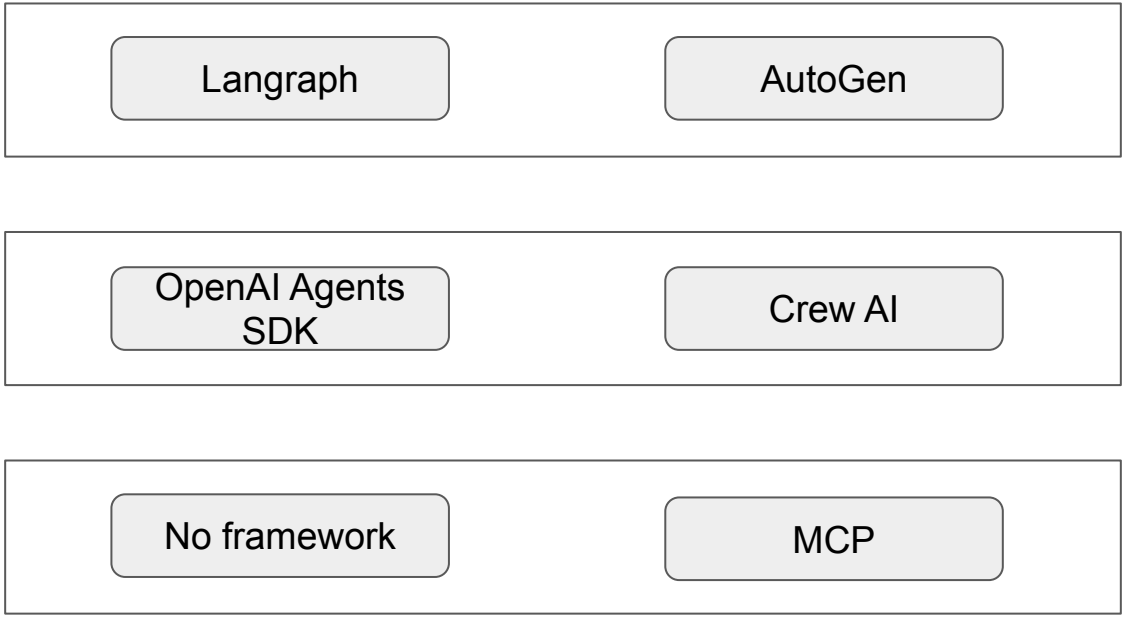


Agents

- **When to Use:**
  - Open-ended tasks with **unpredictable** steps
  - Tasks that need **reasoning, adaptation,** and **tool use**
- Needs **guardrails, testing,** and **clear tool documentation**
- **Example:** Coding agent editing multiple files from a task description



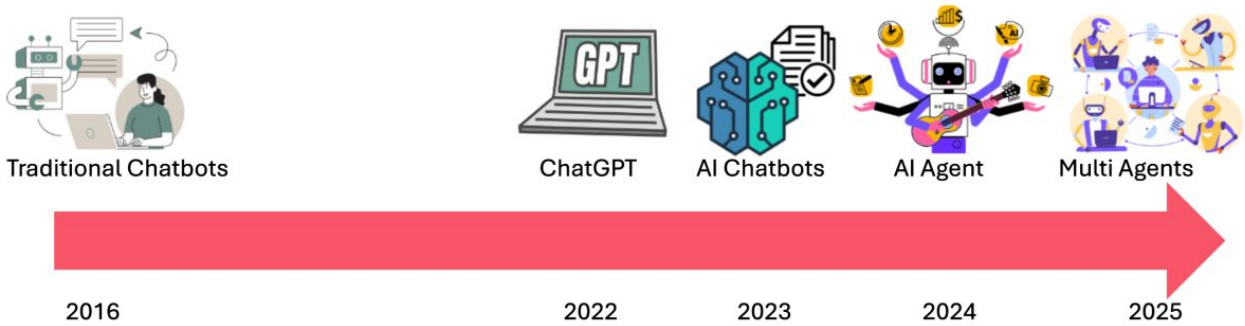
Agentic AI Framework



Core Components of an LLM-based Agentic System

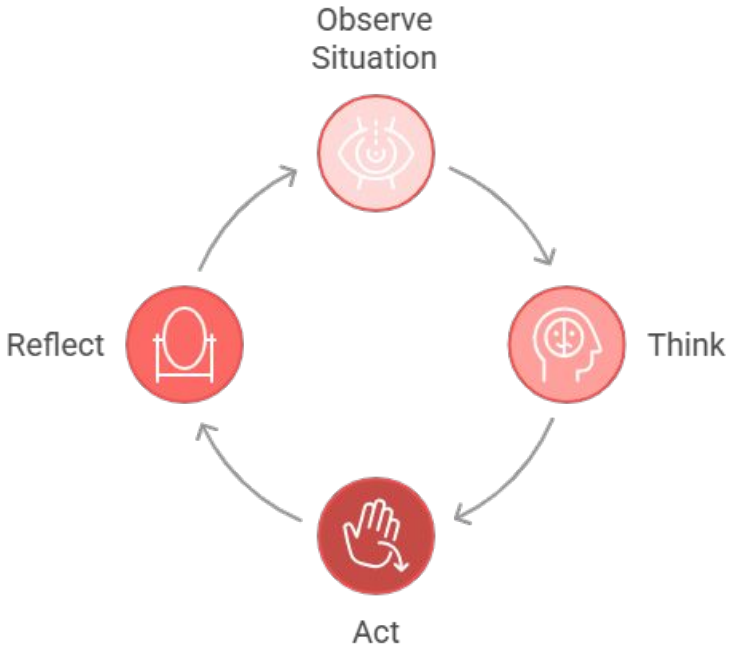
The Evolution of LLM Agents

- Initially, LLMs were used in **static settings** (answering questions, generating text)
- Researchers began embedding LLMs into **agent loops**
- Gave rise to the first **LLM-based autonomous agents**
- These agents transitioned from passive responders to **adaptive cognitive** engines

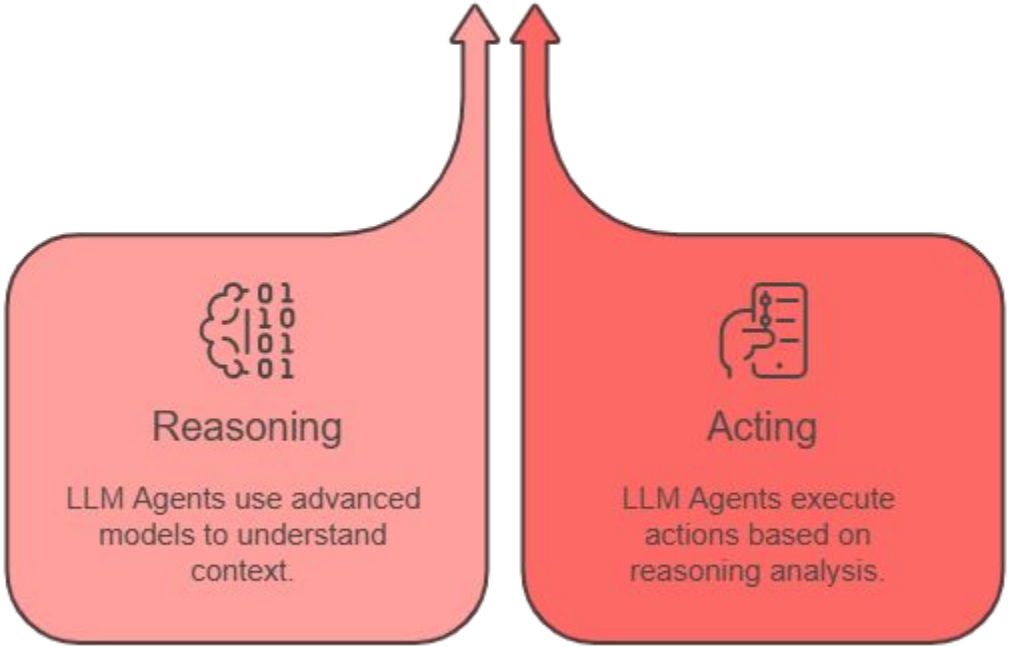


The Agent Loop Revisited

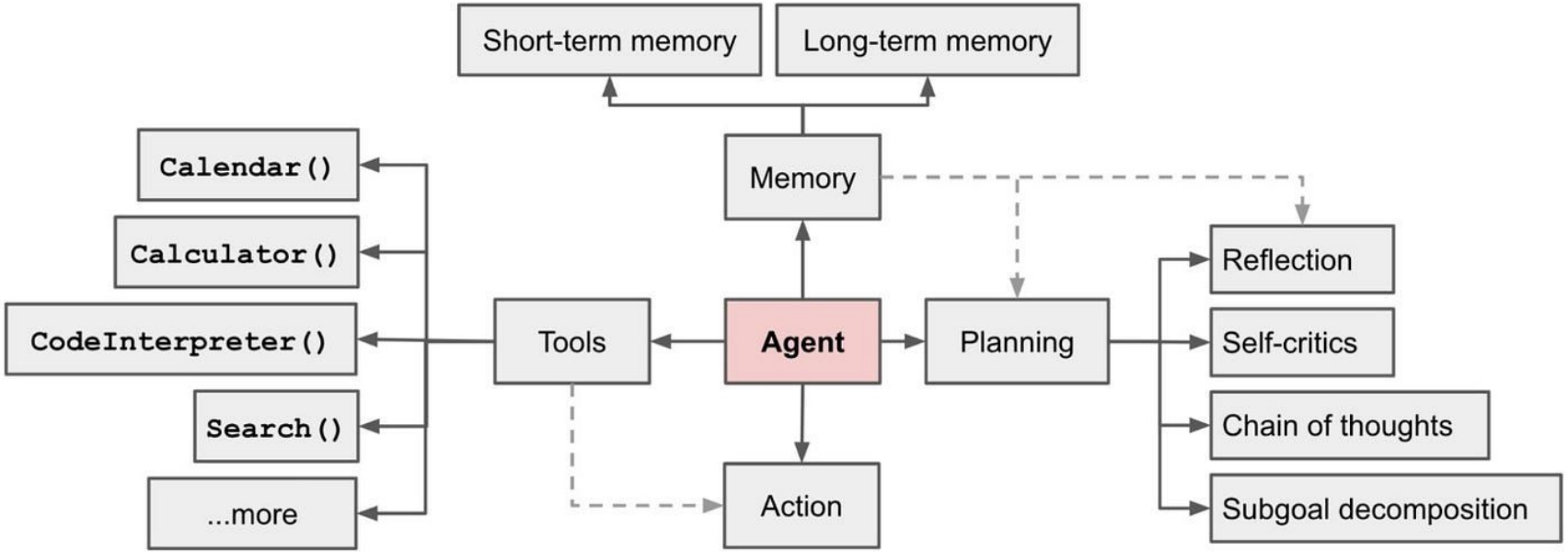
- LLM is **not a one-off responder**, but part of a **feedback** loop
- Each **component** supports one or more **stages** of this loop



How Do LLM Agents Work?

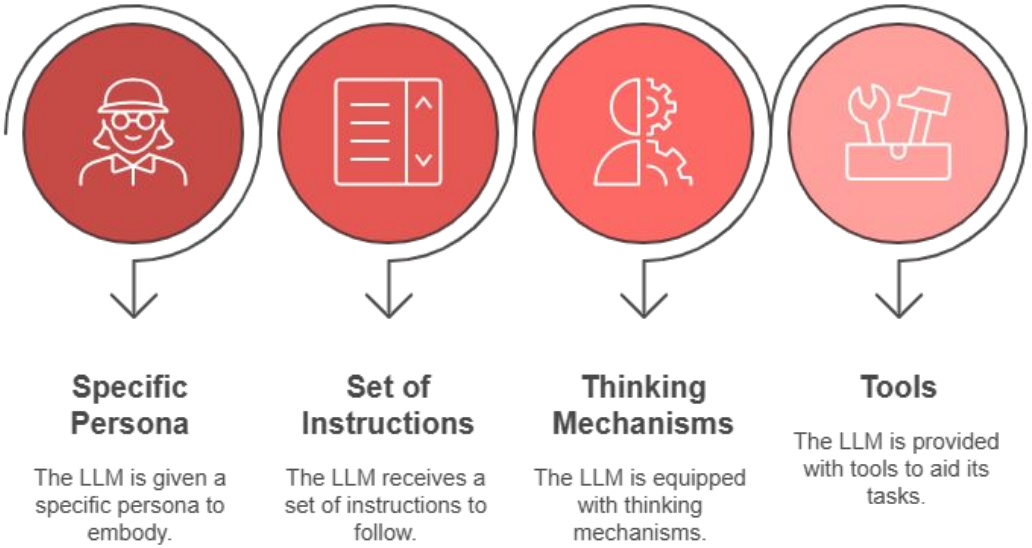


Core Components of LLM-based Agents



Component 1: LLM-based Agent

- When LLMs are given a:

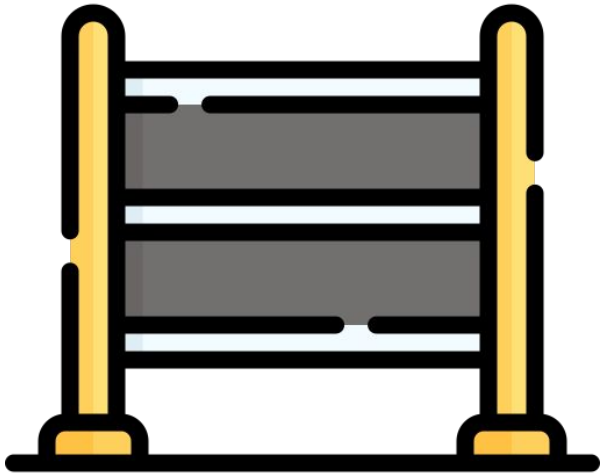


Persona — The Agent's Identity

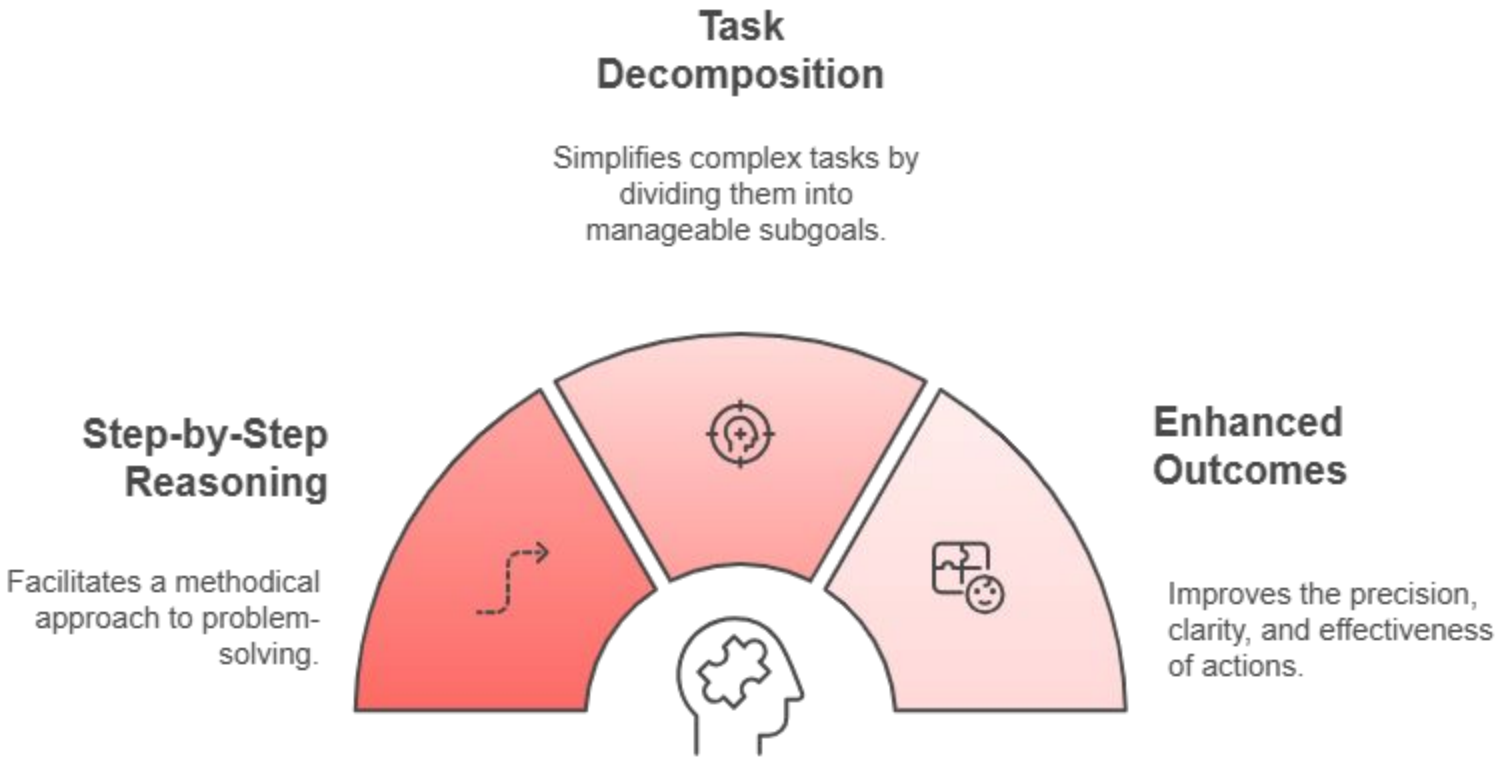
- Defines the **role** the agent plays (e.g., teacher, developer, assistant)
- Shapes **tone, style,** and **behavior**
- **Examples:**
  - "You are a friendly tutor helping a student."
  - "You are a cybersecurity expert detecting threats."

Instructions — Guardrails for Behavior

- Direct the agent's **actions** and **boundaries**
- Include **goals, forbidden actions**, ..
- Act as an **operational manual** for the LLM
- **Examples:**
  - “Avoid making assumptions; ask for clarification.”
  - “Do not access external APIs without permission.”

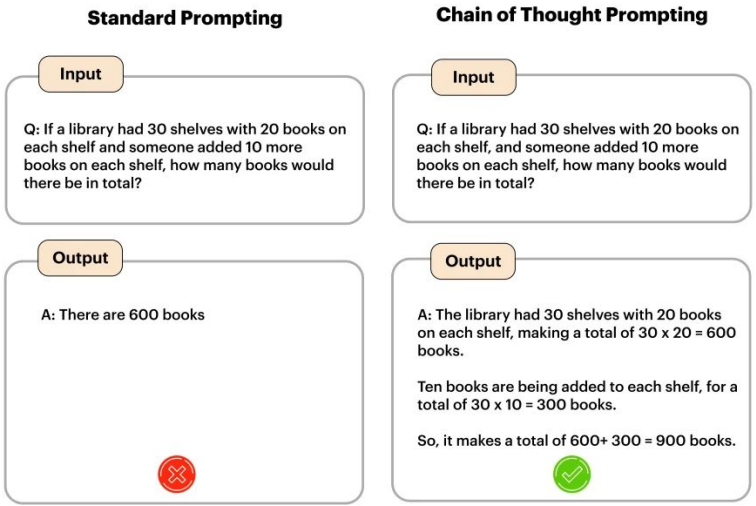


Component 2: Planning/Thinking Mechanisms



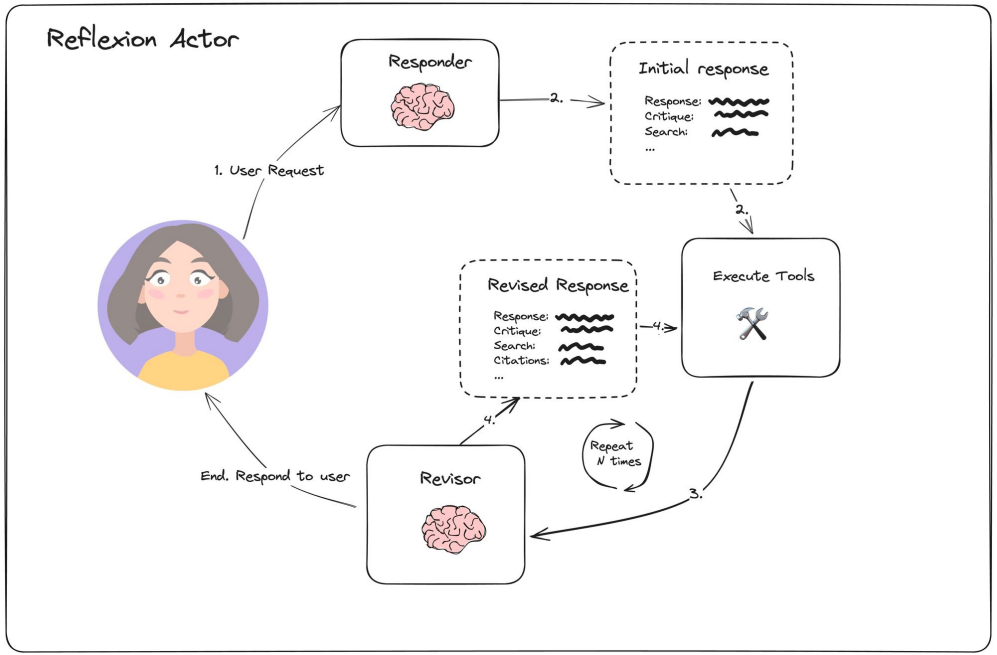
Planning Techniques: Chain of Thought Reasoning

- The agent **explains** its **reasoning** step-by-step
- Encourages **transparency** and better **problem-solving**



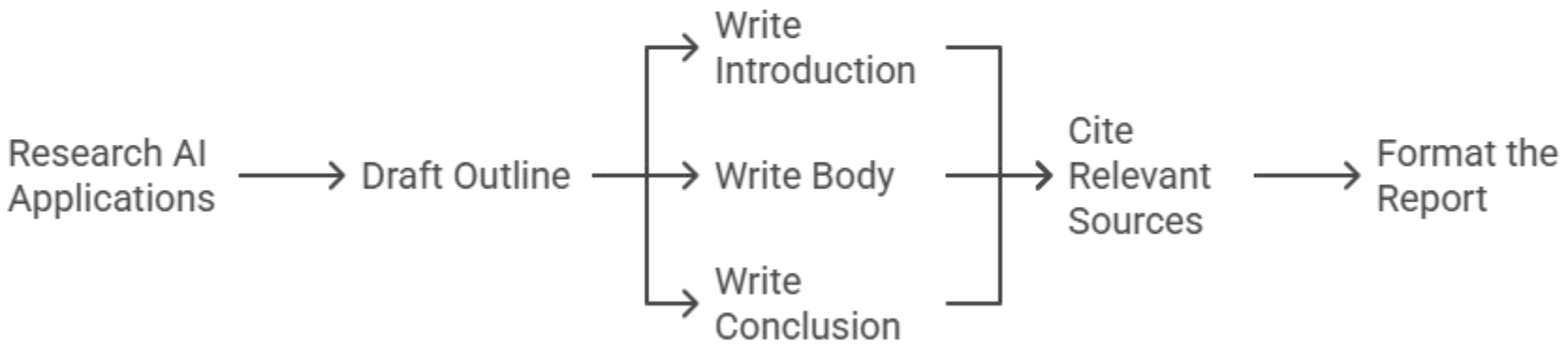
Planning Techniques: Reflection & Self-Criticism

- Agent **evaluates** its own performance
- **Detects** possible mistakes and **rethinks** answers



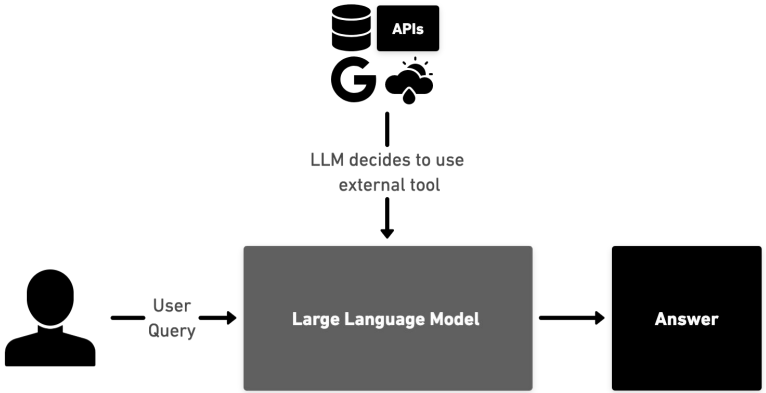
Planning Technique: Subgoal Decomposition

- Agent **breaks** complex goals into **smaller manageable** tasks
- Each subtask can be delegated to **tools** or other **agents**
- Critical for long-term, multi-step **workflows**

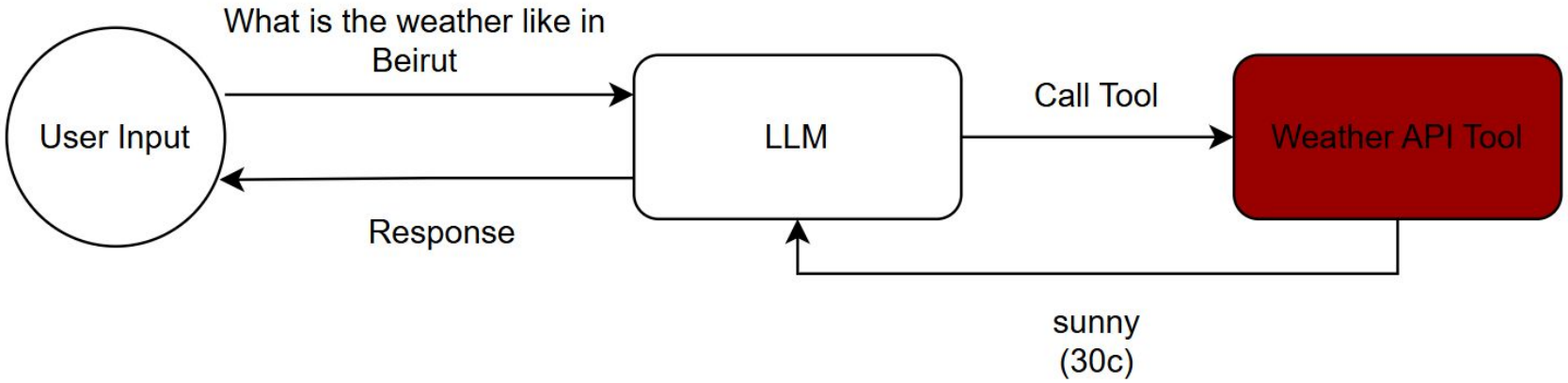


Component 3: Tools

- **External functions or APIs** the agent can invoke to **solve problems** it cannot handle with text generation alone
- Give LLMs the ability to carry **actions**
- Used for precision tasks (e.g., search, code, math)



Tools Example

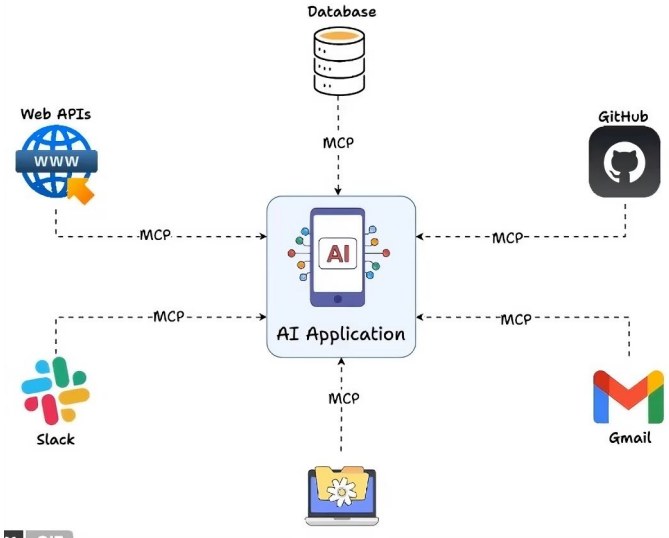


Component 4: Action

- This is where decisions become **reality** — the agent calls a tool or emits a response
- **Prompts (or fine-tuning)** guide tool use
- **Action schemas** define valid **tools** and **formats**
- **Model Context Protocol (MCP)** standardizes how LLMs call tools

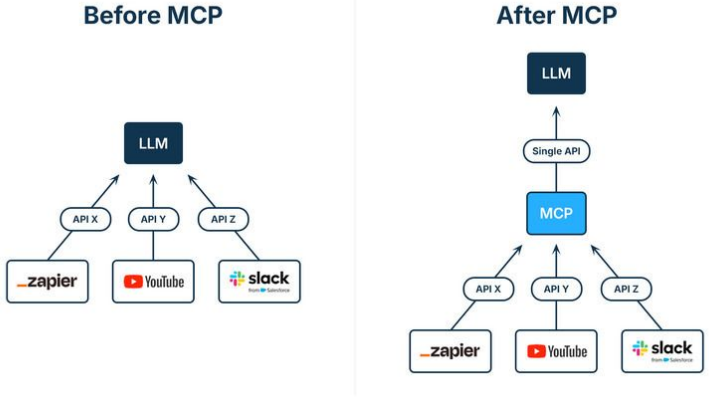
What is Model Context Protocol (MCP)?

- A **standard interface** for LLMs to **call tools**
- Defines how **models declare, request,** and **execute actions**
- Supports **tool schemas, parameters,** and **expected output** formats
- Helps avoid **ambiguous** or **inconsistent tool** calls

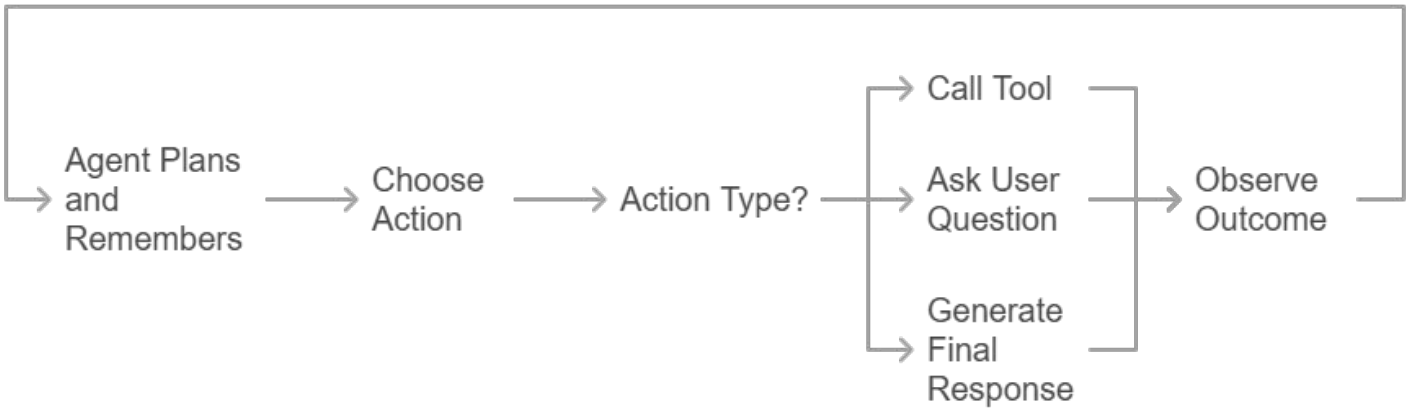


Before vs After Model Context Protocol (MCP)

- **Before MCP**
  - **No standard way** to describe tools
  - Each framework had its **own tool format**
  - **Difficult to reuse** tools across agents or platforms
- **After MCP**
  - Tools defined using standard **JSON schemas**
  - LLMs know **what inputs** to expect and how to **format outputs**
  - Actions can be **validated, parsed,** and **executed automatically**



Action Example

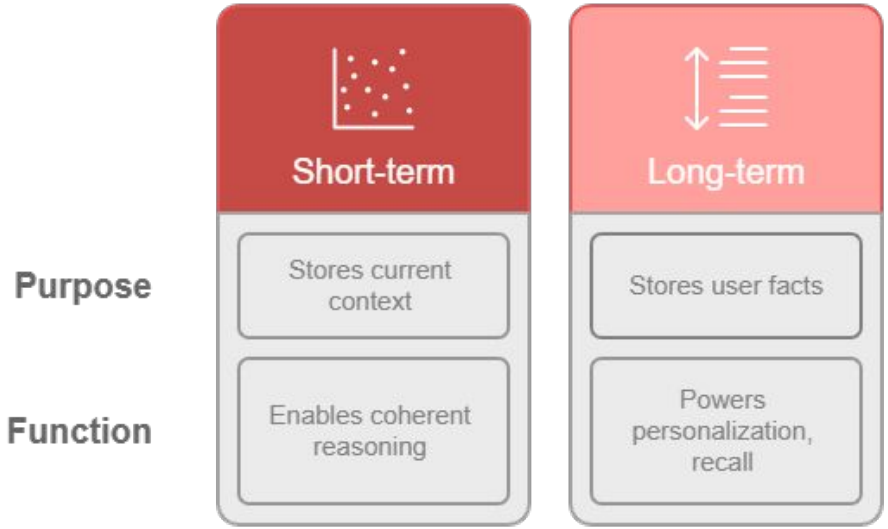


Hands_on: Experimenting_with_LLM_Configurations_and_Personas

Component 5: Memory

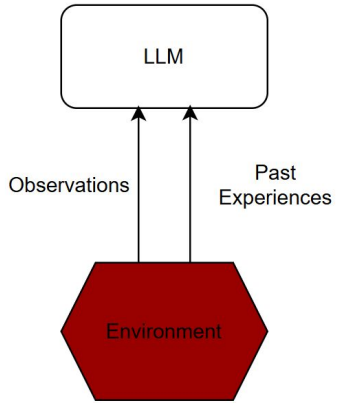
- Stores **key info**:
  - internal thoughts, tool outputs, user data
- Retains **context** across steps and sessions
- Helps agent learn from **past interactions**
- Crucial for **personalization** and **long-term tasks**

Types of Memory



Observations & Feedback

- Capture **what worked** and **what failed**
- Store **tool results, task completions,** or **user feedback**
- Enable agents to **refine** their strategy or escalate



Prompt Engineering as Policy

- Prompts can act like **behavioral blueprints** for the agent
- **System prompts = policies**
- Guide:
  - Role and tone
  - Constraints and permissions
  - Decision-making strategies
- **Structured prompting:**
  - XML, JSON modes, tool selection guidelines

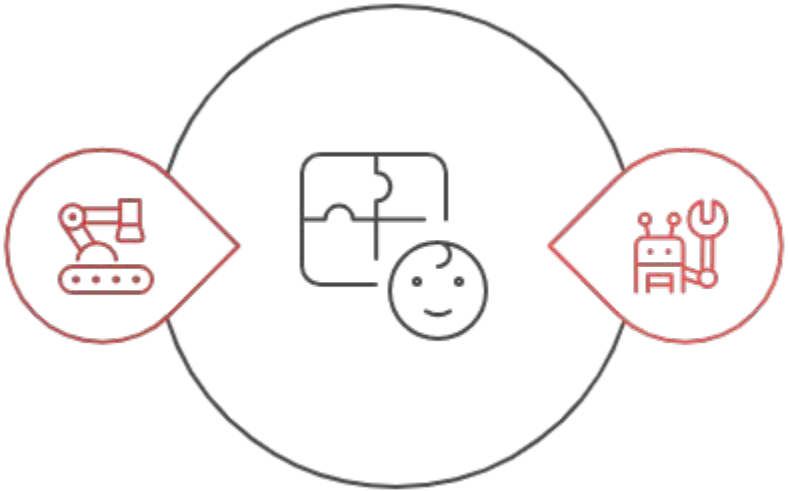
```
You have access to the following tools:
- calculator: for math operations
- search: for current or factual info
- code_executor: for running Python code

When choosing a tool, respond using this JSON format:
{
  "tool": "tool_name",
  "input": "input string"
}
```

Orchestration Patterns in Agents

Multi-Agent System

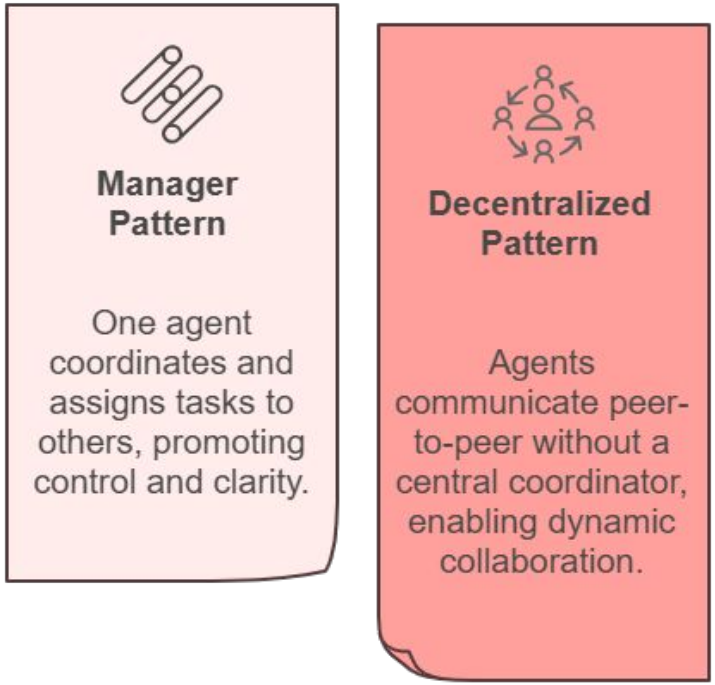
Multiple specialized agents coordinating. These agents collaborate, passing messages and refining each other's outputs.



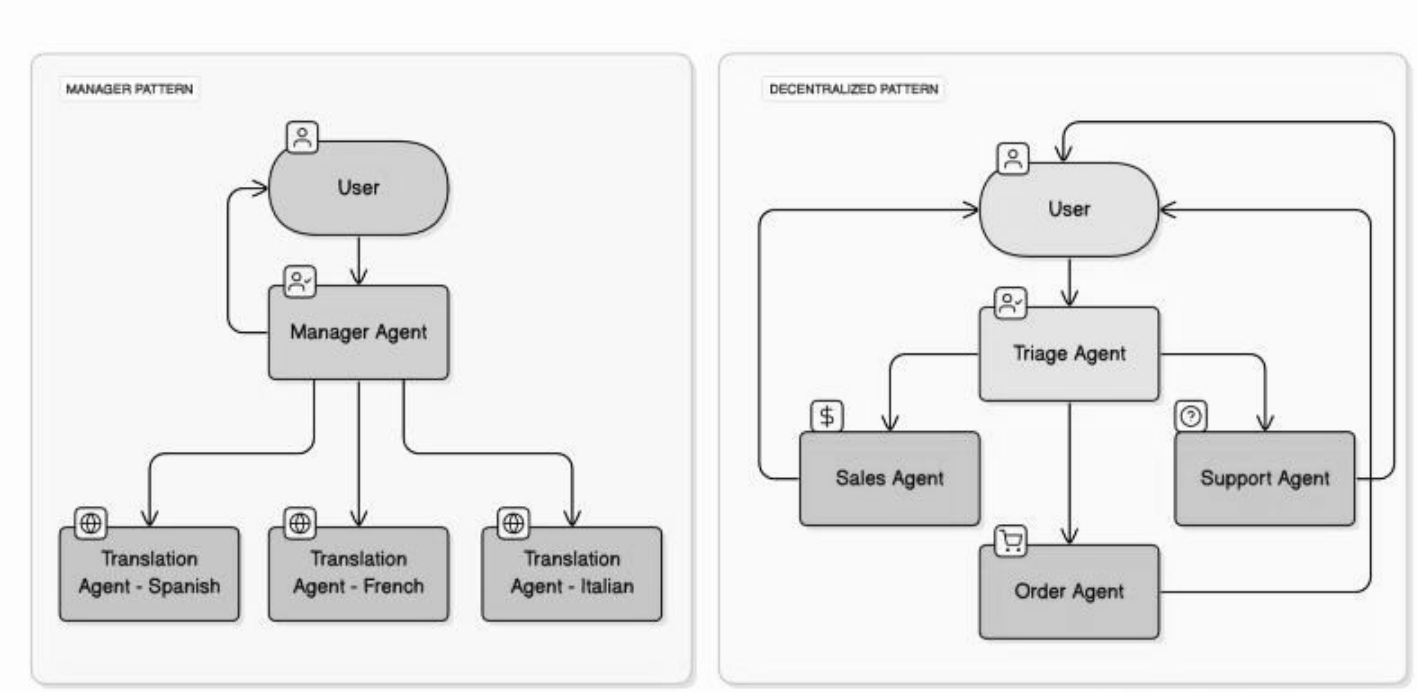
Single-Agent System

A single model looping with tools and instructions

Multi-Agent Patterns



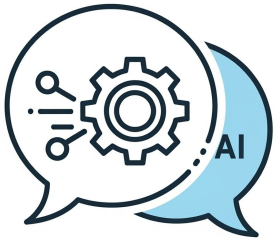
Multi-Agent Patterns



Challenges of Agentic Systems

Poor Context Understanding

- Agents may **misinterpret** vague or nuanced instructions
- **Lack of grounding** in user goals or domain-specific language
- **Solution:**
  - Use **clarification prompts** before acting (e.g., “Can you confirm if you meant X or Y?”)
  - **Fine-tune** on domain-specific tasks or instruction-following datasets



Hallucination and Reasoning Limits

LLMs may fabricate tools, steps, or results

reason incorrectly or overcomplicate

Limited memory = loss of critical context

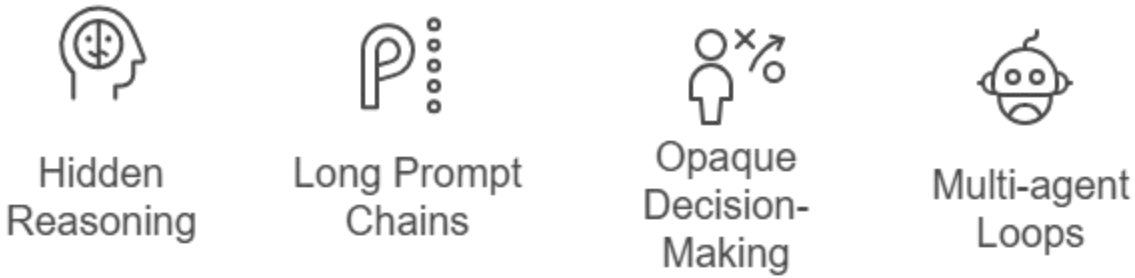
Poor self-evaluation

Tool Misuse and Failure Recovery

- Agents may call **incorrect** tools or **misuse** parameters
- Lack of **error handling** causes **failure** cascades
- **Example:** API returns 404 or malformed response → agent gets stuck
- **Solution:** Add tool schema validation, fallback options, retries, tool metadata

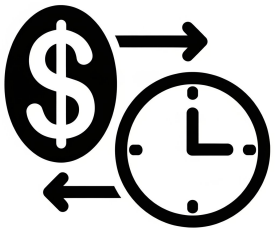
Observability and Debugging

- Debugging agents is **harder** than traditional scripts
- Agents involve:



Cost and Latency Trade-offs

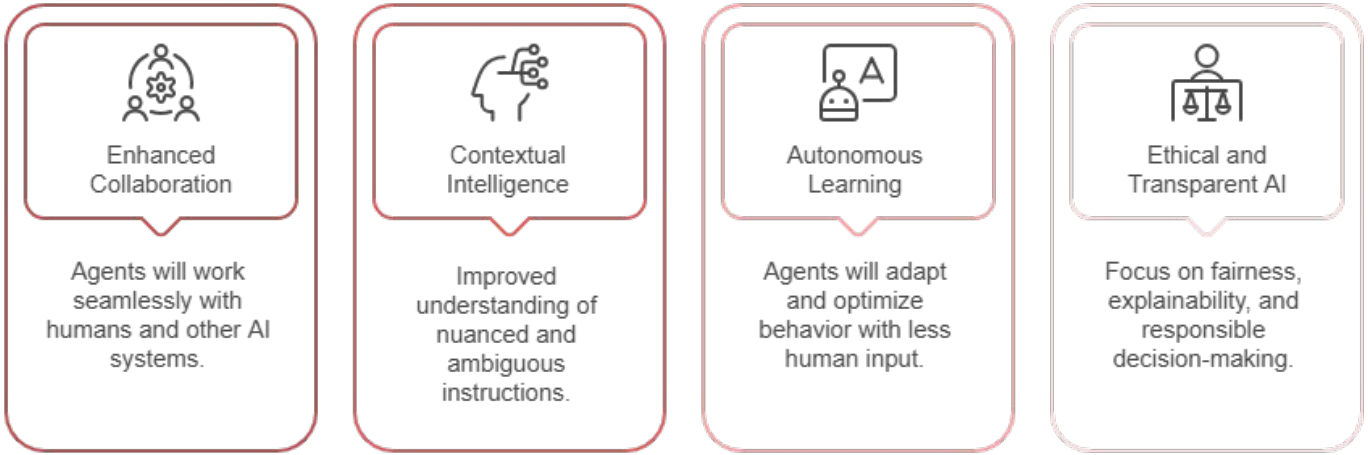
- Tool use = more LLM calls = **more tokens**
- Planning + reflection = **longer latency**
- Agents may **retry, repeat,** or **loop** → unexpected cost spikes
- **Solution:** Use token limits, prompt summarization, and smart tool gating



AMERICAN UNIVERSITY OF BEIRUT

Conclusion

Future Directions



Best Practices

Start with narrow tasks and expand scope incrementally

Use tool gating, max steps, and cost monitoring

Log all steps and tool calls for debugging and transparency

Best Practices

Design clear prompt policies for consistent agent behavior

Include fallbacks and recovery strategies for safety

Continuously evaluate and fine-tune based on real user feedback

Agents are not just smarter chatbots — they are the foundation of a new way to automate thinking and doing

Hands_on: Exploring_and_Comparing_Large_Language_Model_Thinking_Strategies

AMERICAN UNIVERSITY OF BEIRUT

Thank you!