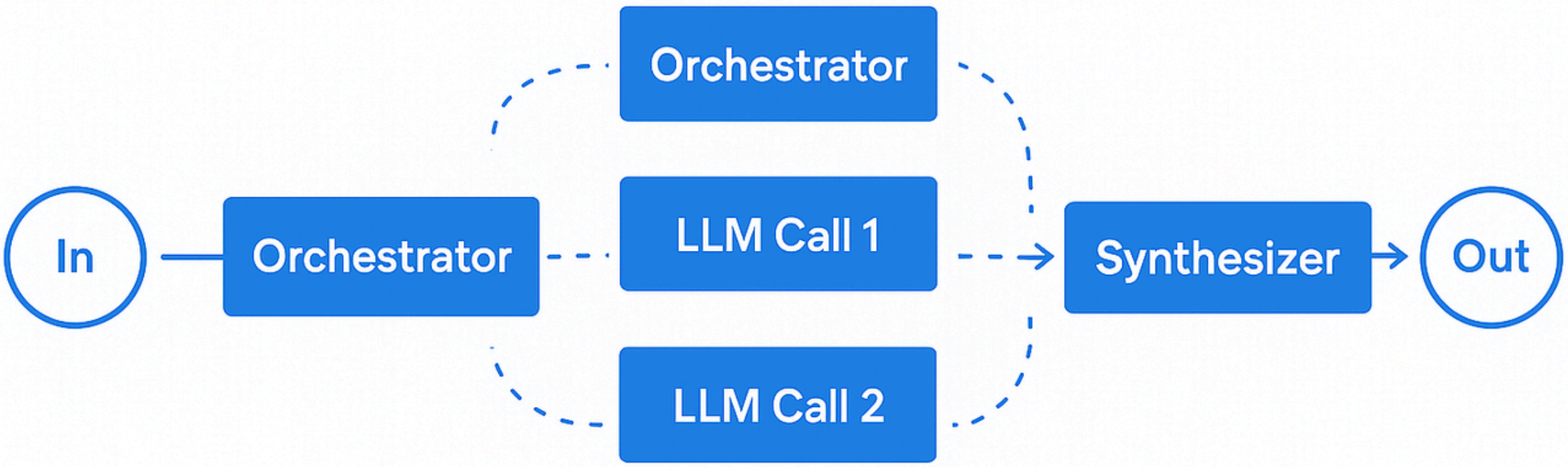
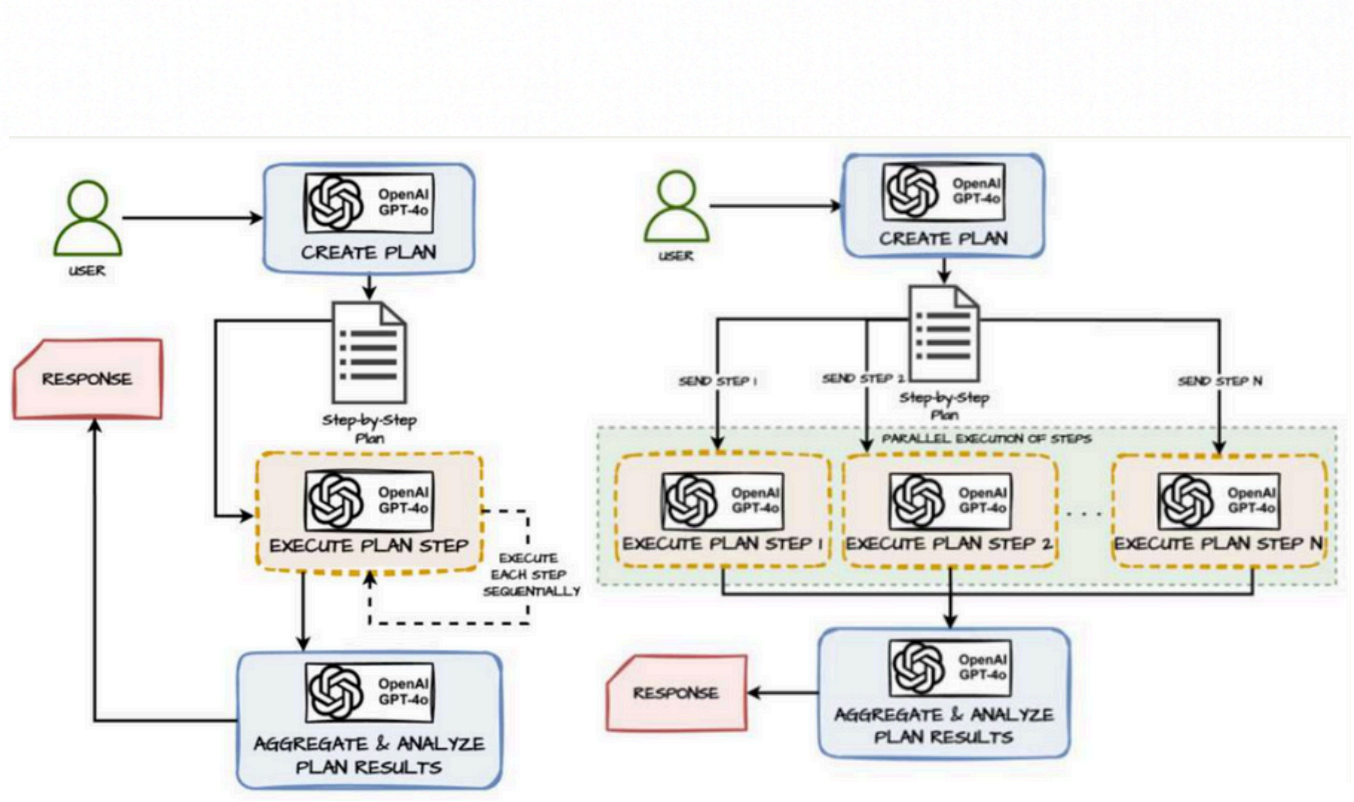
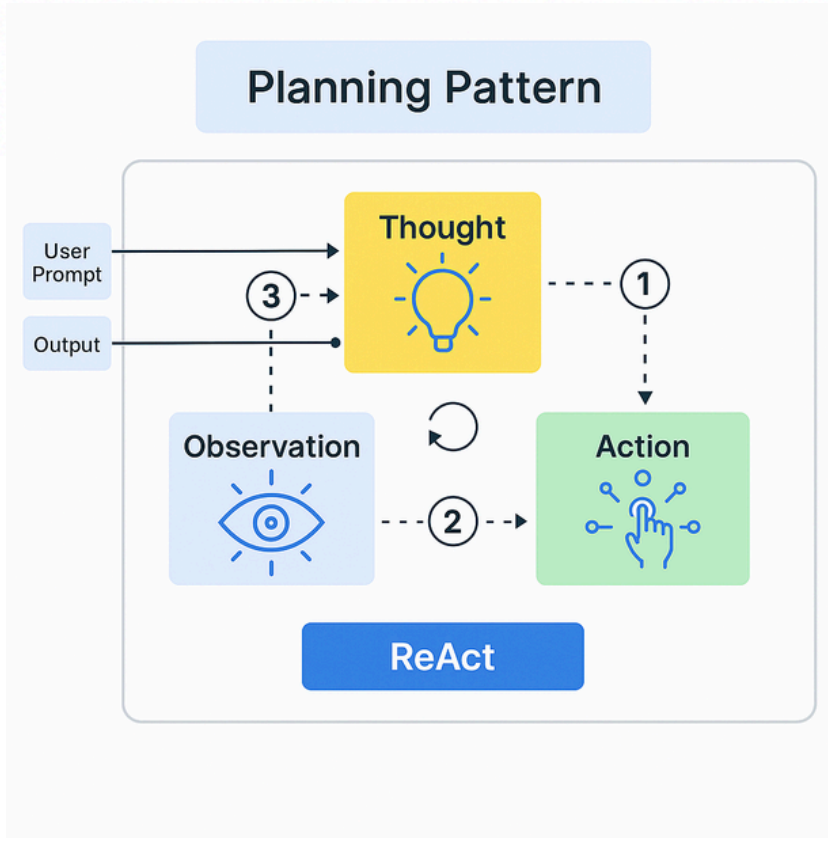
Analytics Vidhya

Guide to Building Effective Agentic AI Systems



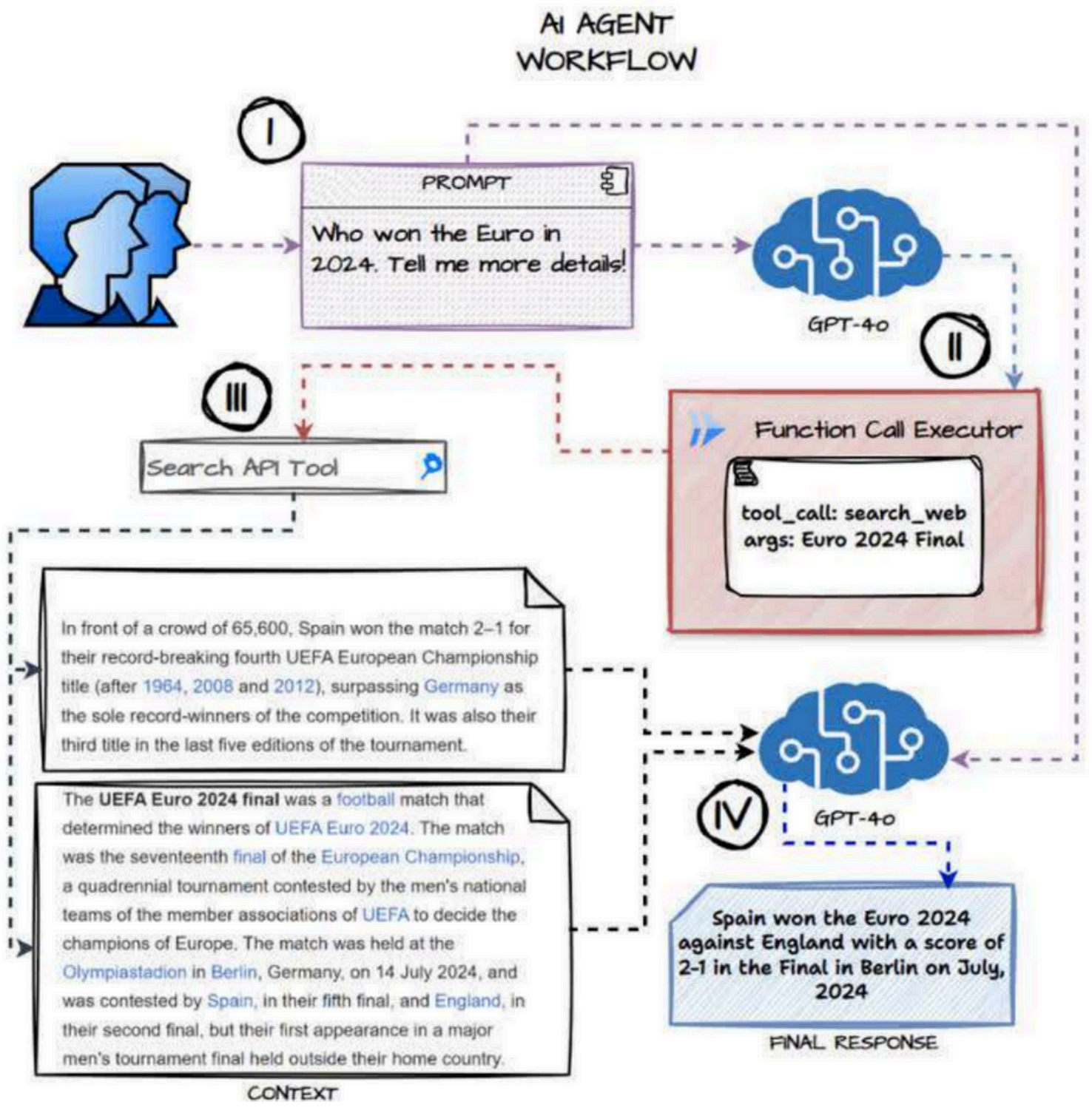




Free Course Inside

What are Agents?

An Agentic AI System is usually an autonomous system that operates independently over extended periods, using various tools and flows to accomplish complex tasks

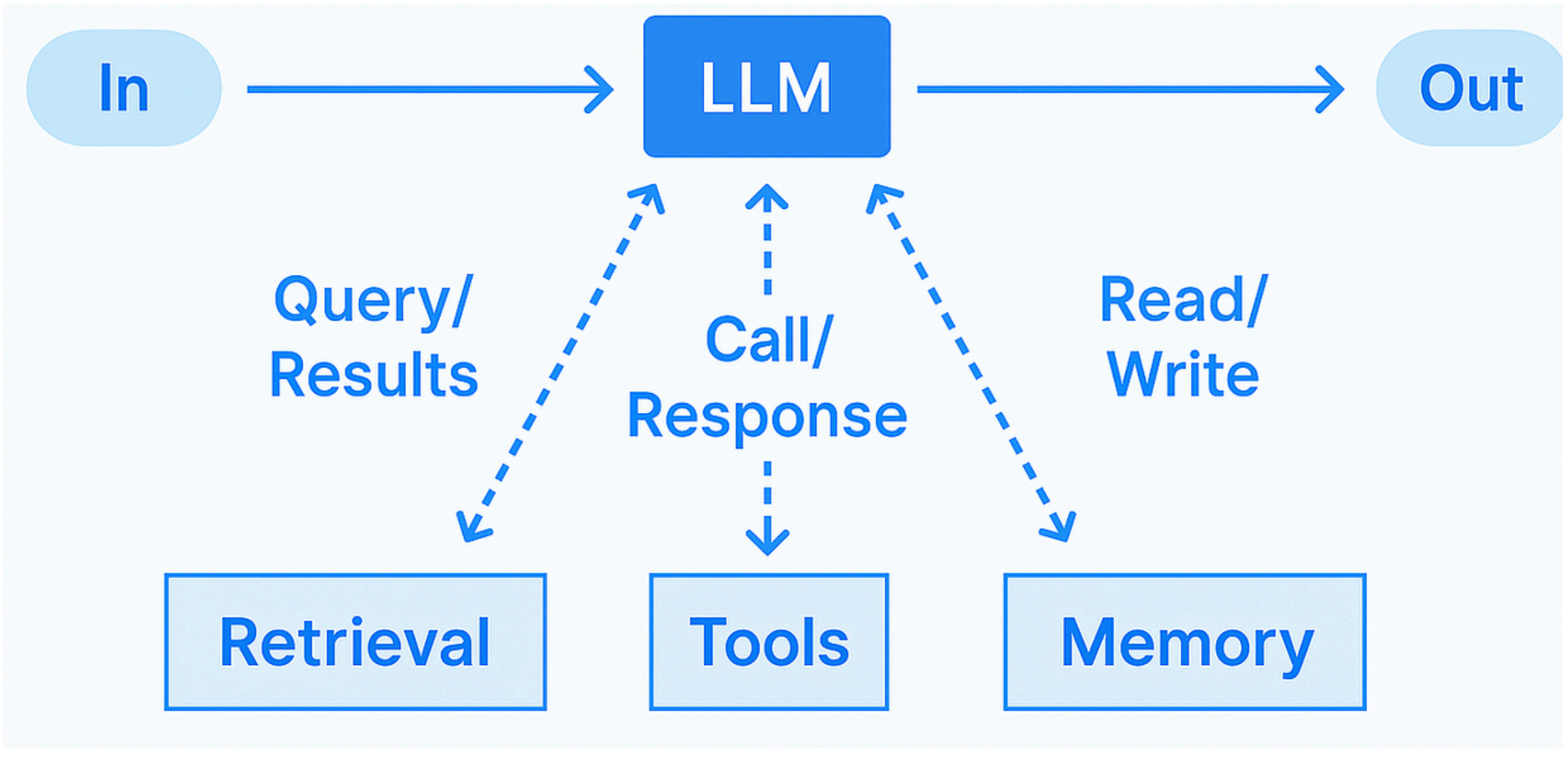


Agentic AI Systems can be further categorized as:

- **Workflows** are systems where LLMs and tools are orchestrated through predefined paths.
- **Agents**, on the other hand, are systems where LLMs dynamically direct their own processes and tool usage, maintaining control over how they accomplish tasks.

Build the key components for your Agent

- The basic building block of Agentic AI Systems is an LLM enhanced with augmentations such as **retrieval**, **tools**, and **memory**.
- Powerful LLM platforms have these in-built. When using **APIs** you would need to connect the LLM with relevant tools, memory and databases so that they can generate their own search queries, select appropriate tools, and determine what information to retain.

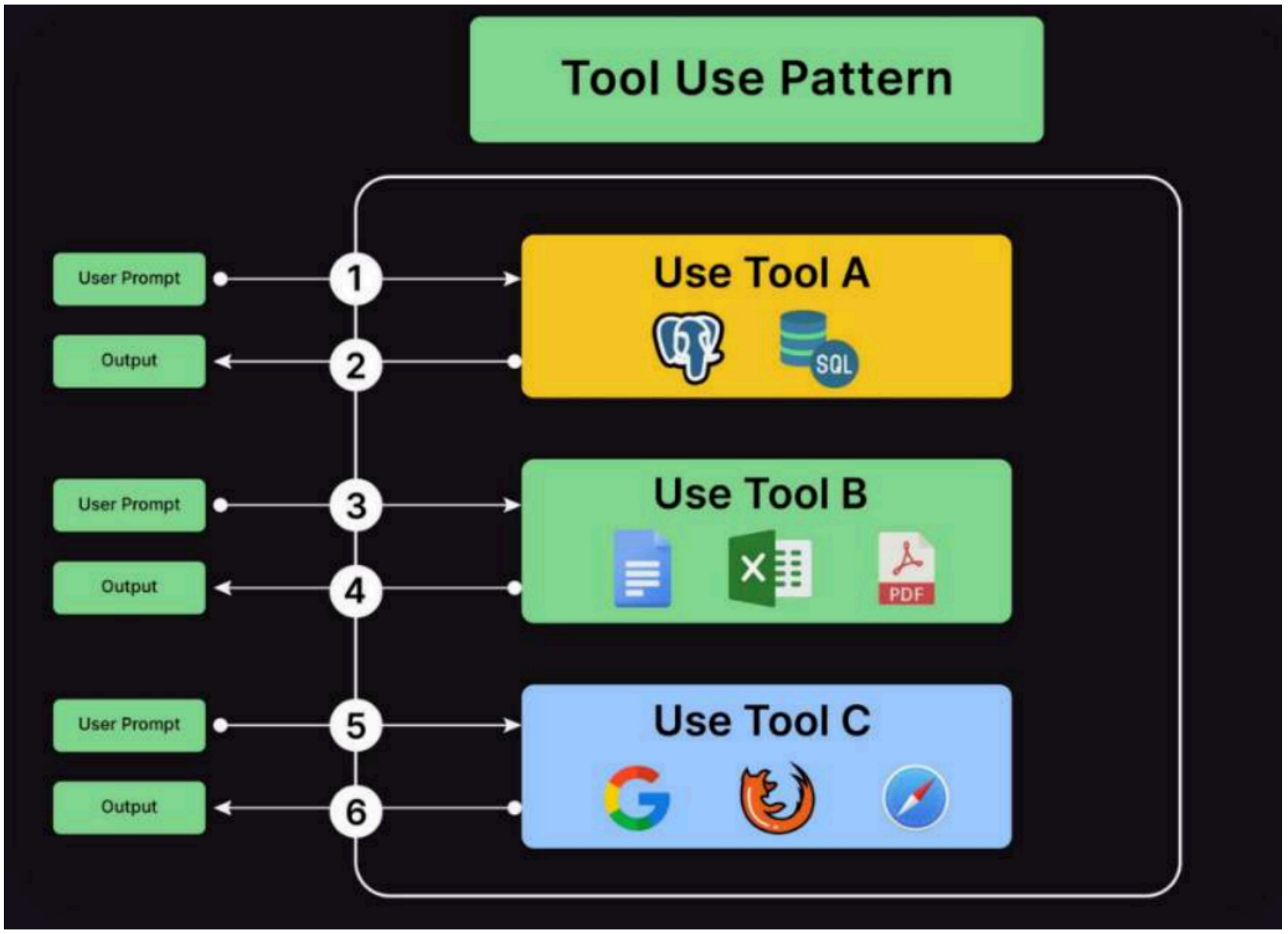


Anthropic recommends focusing on two key aspects of the **implementation**:

- Tailoring these capabilities to your specific use case
- Ensuring they provide an easy, well-documented interface for your LLM

Start with Tool-Use Single ReAct Agents

- Most ReAct Tool-Use Agents already have planning capabilities.
- These systems can easily handle 10-15 tools easily Can also handle multi-step and multi-tool call executions easily.



Key drivers here include:

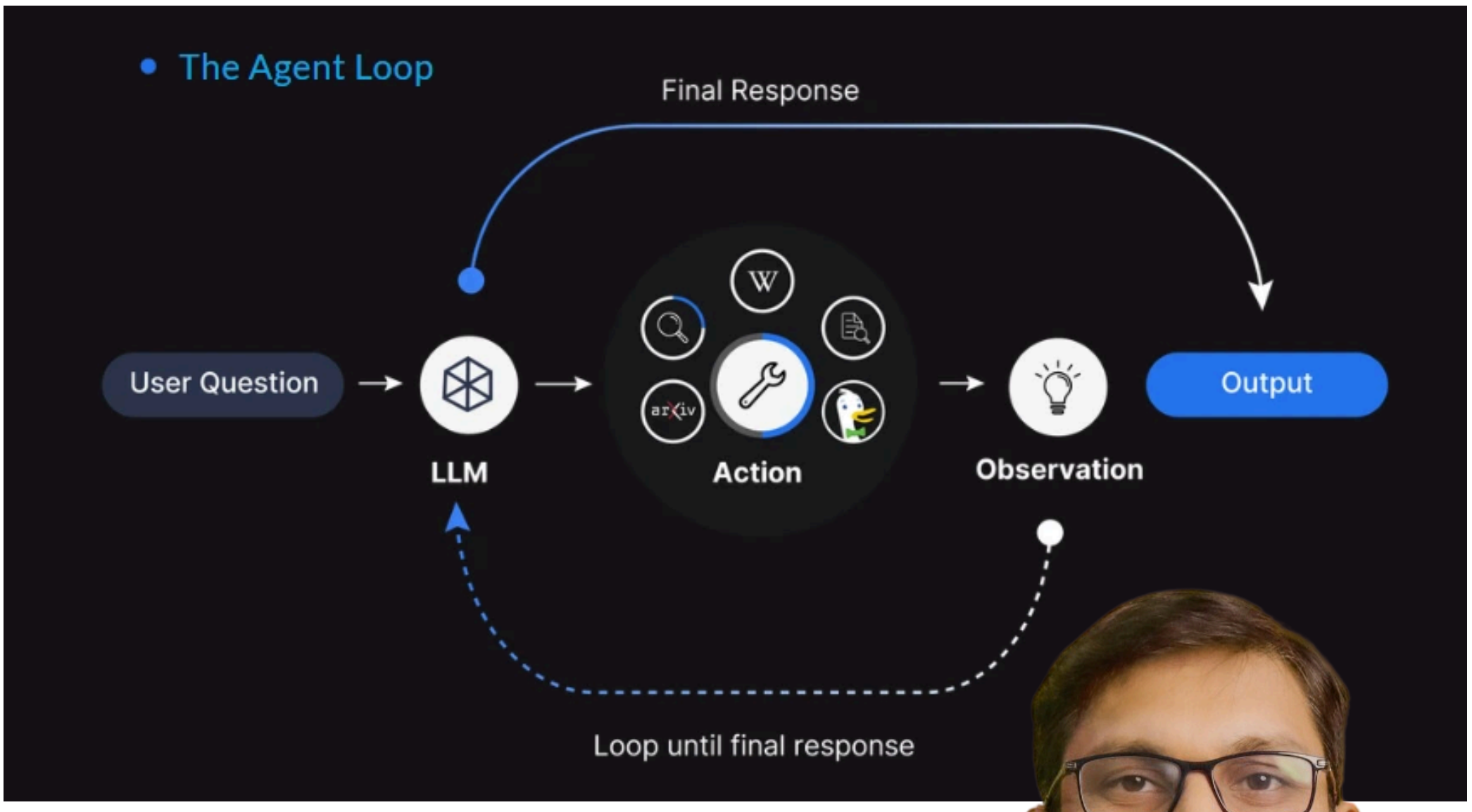
- Well-defined tool schemas for accurate function calling
- Well-structured System prompt with detailed instructions
- Powerful LLMs already trained for function (tool) calling

Do **NOT** jump into **multi-agent systems** immediately - they are notoriously hard to control and debug regardless of present advancements

Analytics Vidhya

Free Course on

A B C of Coding to Build AI Agents

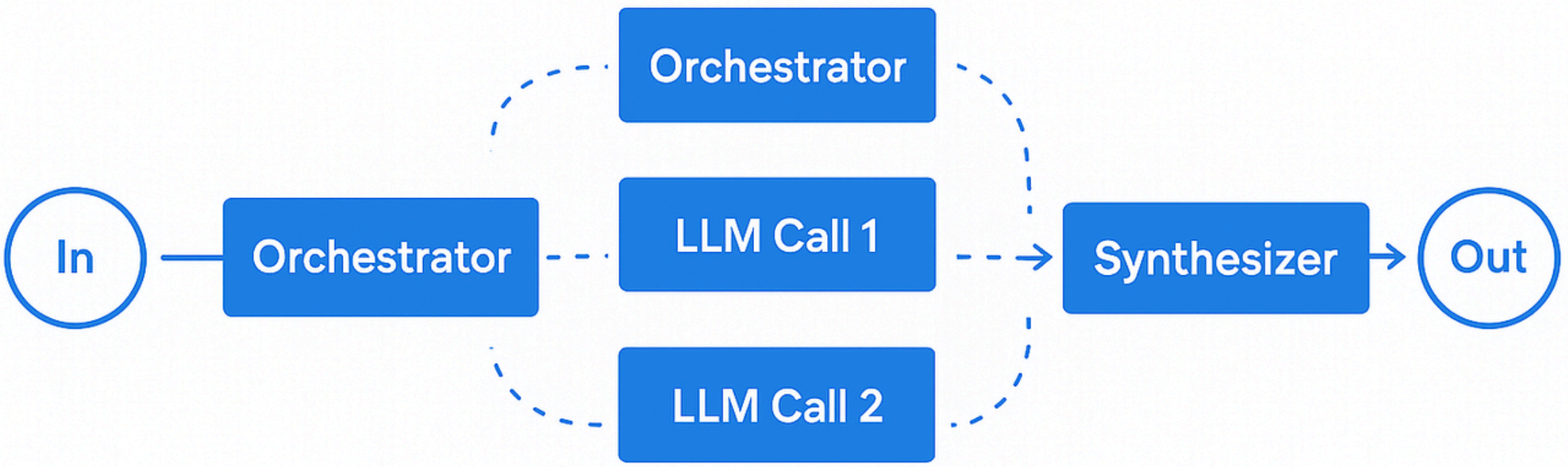


Prashant Sahu

Data Science Manager

Router Agents for Reliability

- Router Agents use an LLM to classify and route execution flow to a specific pathway or route.
- Each route can be a sequential workflow or a sub-agent itself.
- This workflow allows for separation of concerns. modularity and reusability (e.g multiple RAG workflows reusing similar pipelines but different DBs)
- Popular open-source router frameworks include Route0x & Semantic Router.



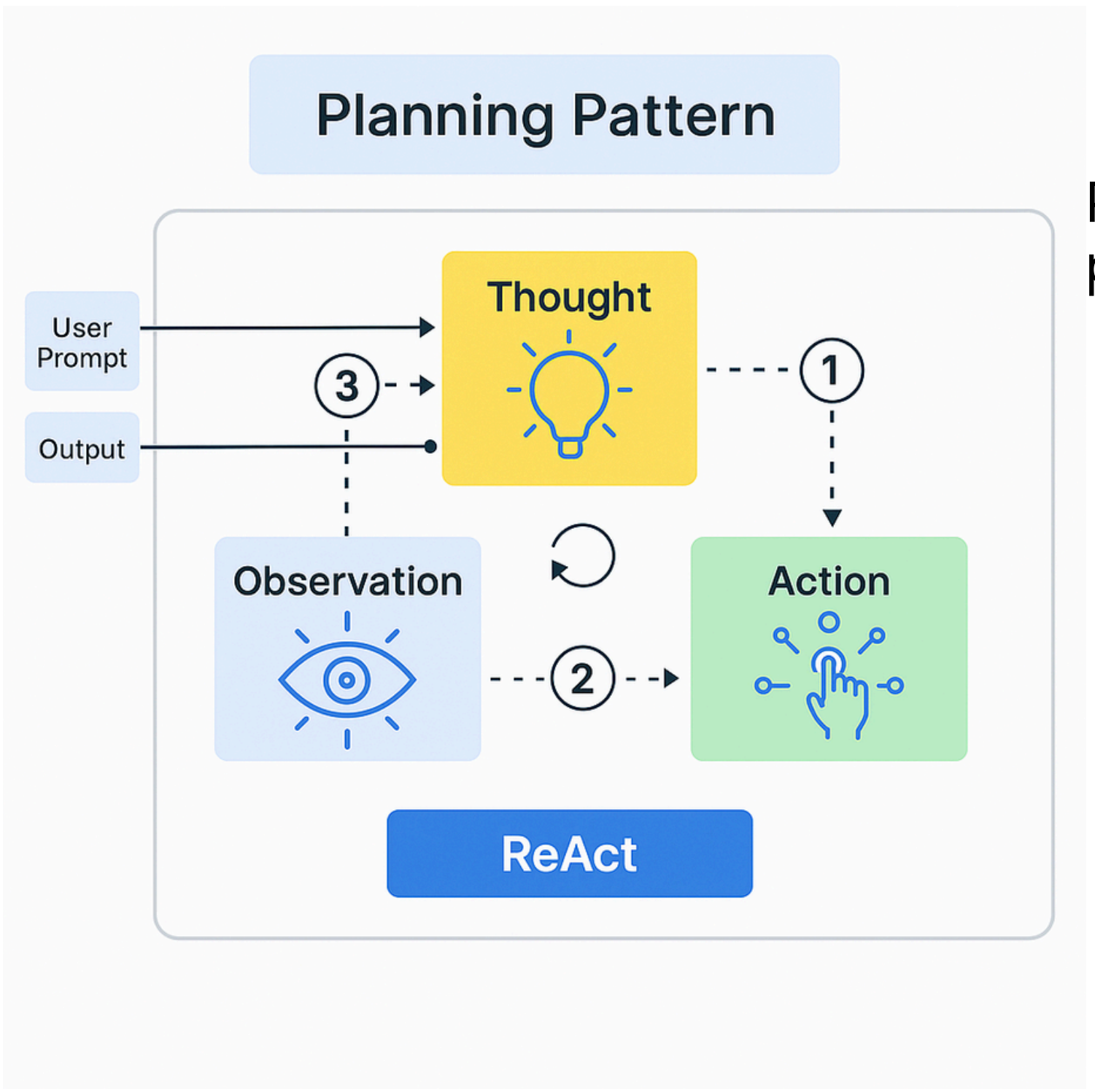
Examples where routing is useful:

- Directing different types of customer service queries (general questions, refund requests, technical support) into different downstream processes, prompts, and tools.
- Routing easy/common questions to smaller models and hard/unusual questions to more capable models to optimize cost and speed.

Often preferred to traditional supervisor or collaborative multi-agent systems as this is usually more reliable

Planning Agents for Complex Task Execution

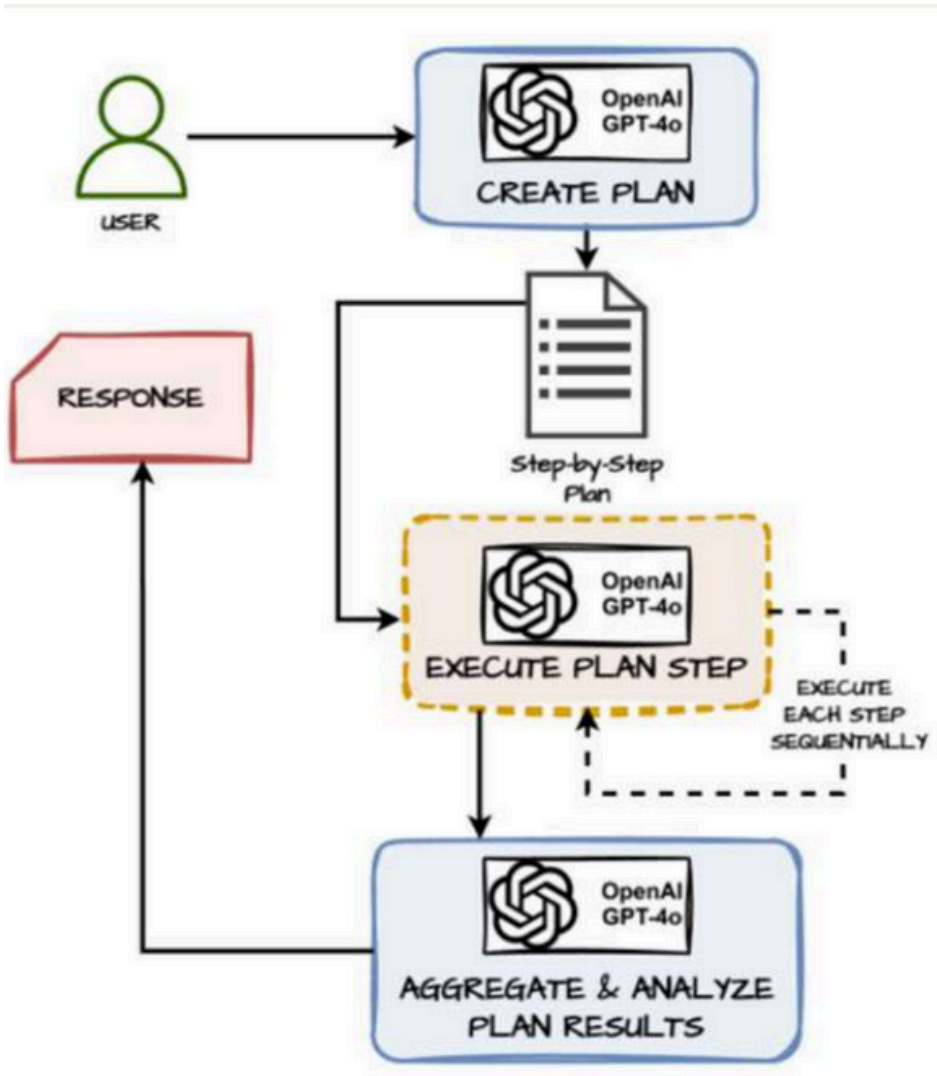
- Most ReAct Agents already have planning built-in so first start with simple ReAct Agents.
- If tasks are more complex and require explicit planning consider adding in additional planning modules in the Agent



Planning modules or patterns are typically:

- Static Planners with Parallel Task Execution & Synthesis
- Dynamic Planners with Task Execution, Reflection & Replanning

- Do **not** add **extra planning steps** or modules unless absolutely necessary as this increases system latency
- Especially useful in complex reasoning, inference and validation scenarios

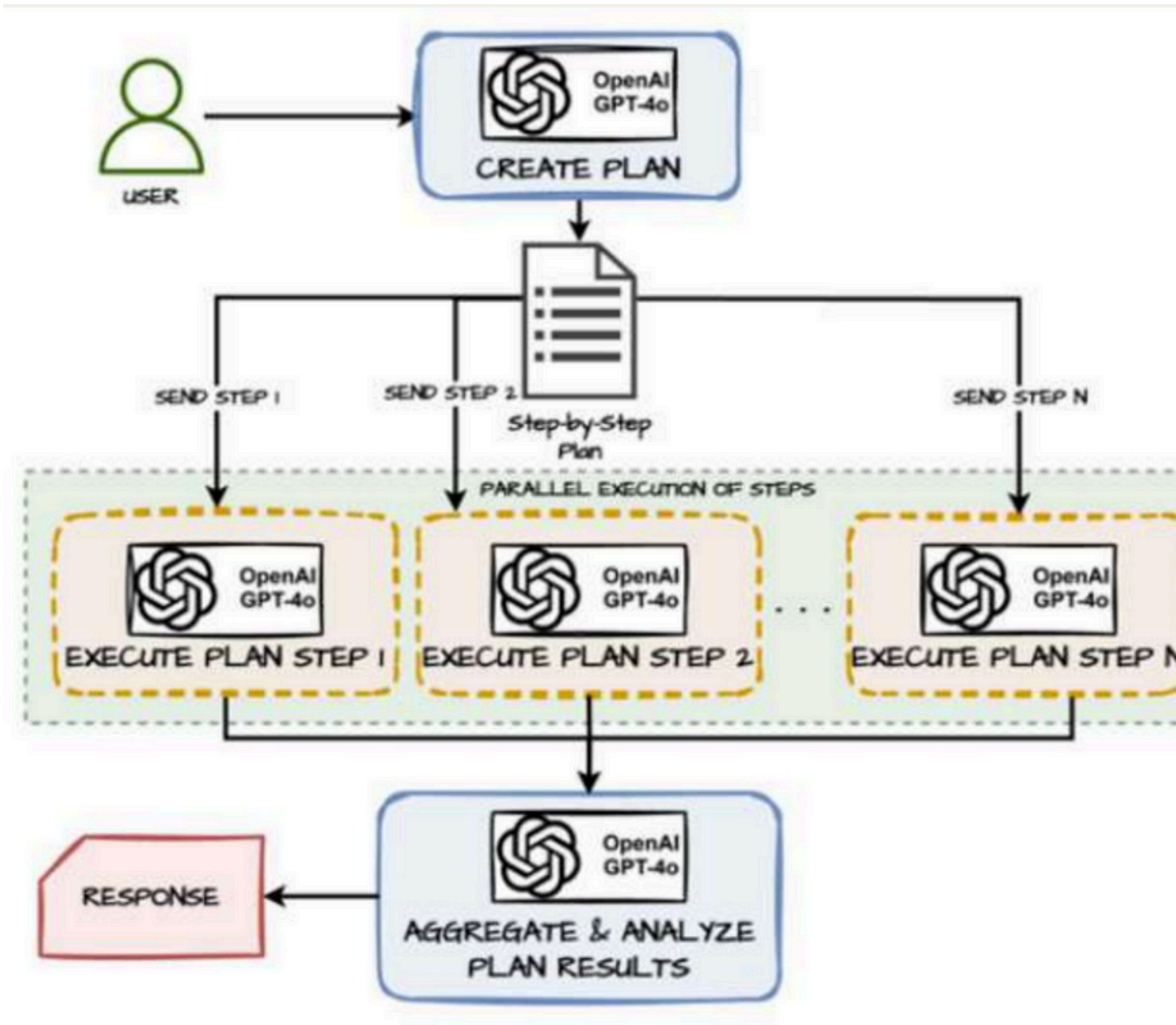


Dynamic Planners:

- Use planning to break down a task into multiple steps
- Executes one step at a time
- Reflects on results of steps already executed
- Uses reflection to replan remaining steps (if needed)
- Repeats till all steps are executed
- Synthesizes results from all steps and generates final response
- Useful when tasks may have dependencies among each other

Static Planners:

- Use planning to break down a task into multiple steps
- Execute all steps in parallel
- Synthesize results from all steps and generate final response (map-reduce)
- Useful when steps do not have dependencies



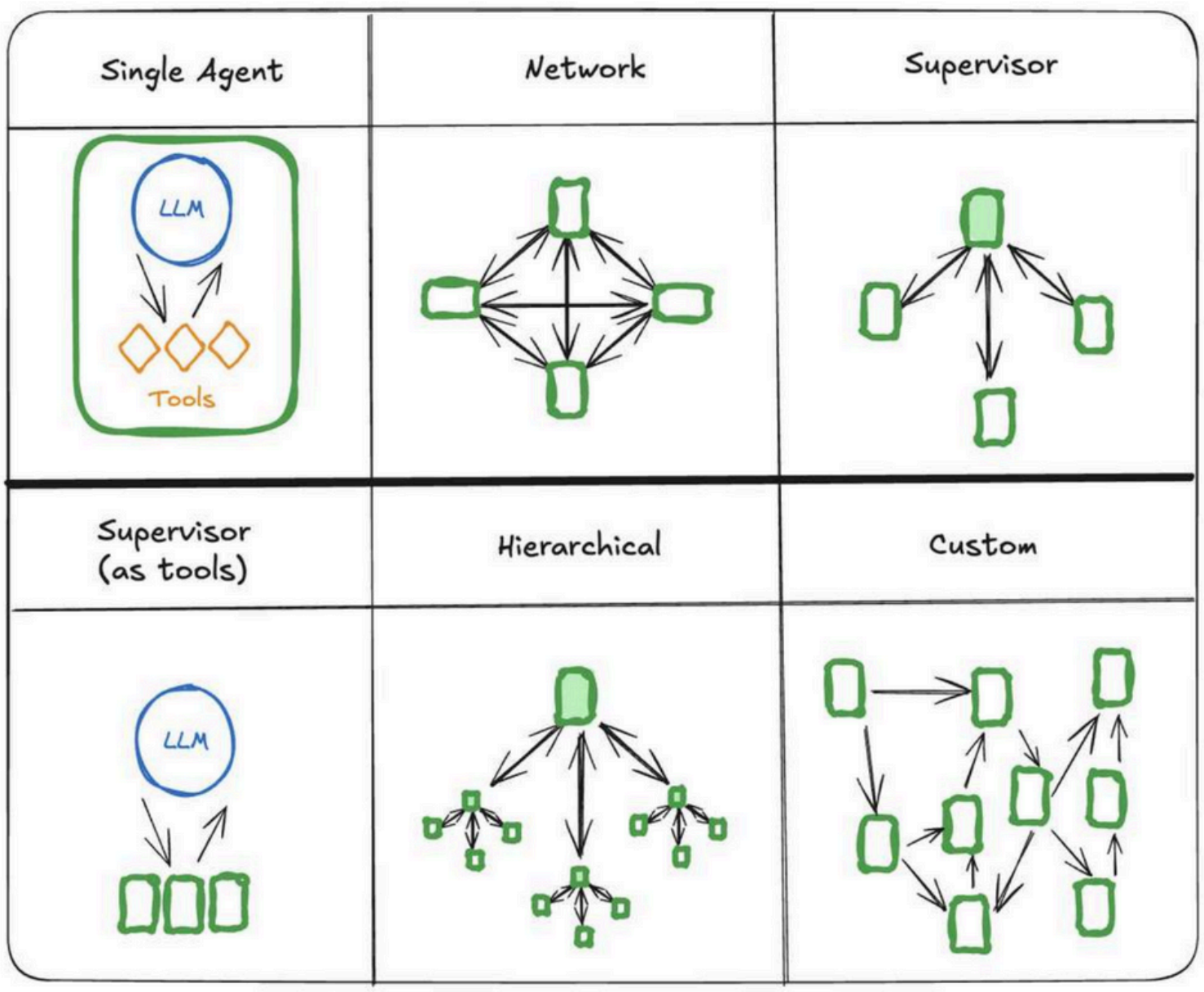
Multi-Agent Systems for Efficient Task Distribution

Analytics Vidhya

- **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

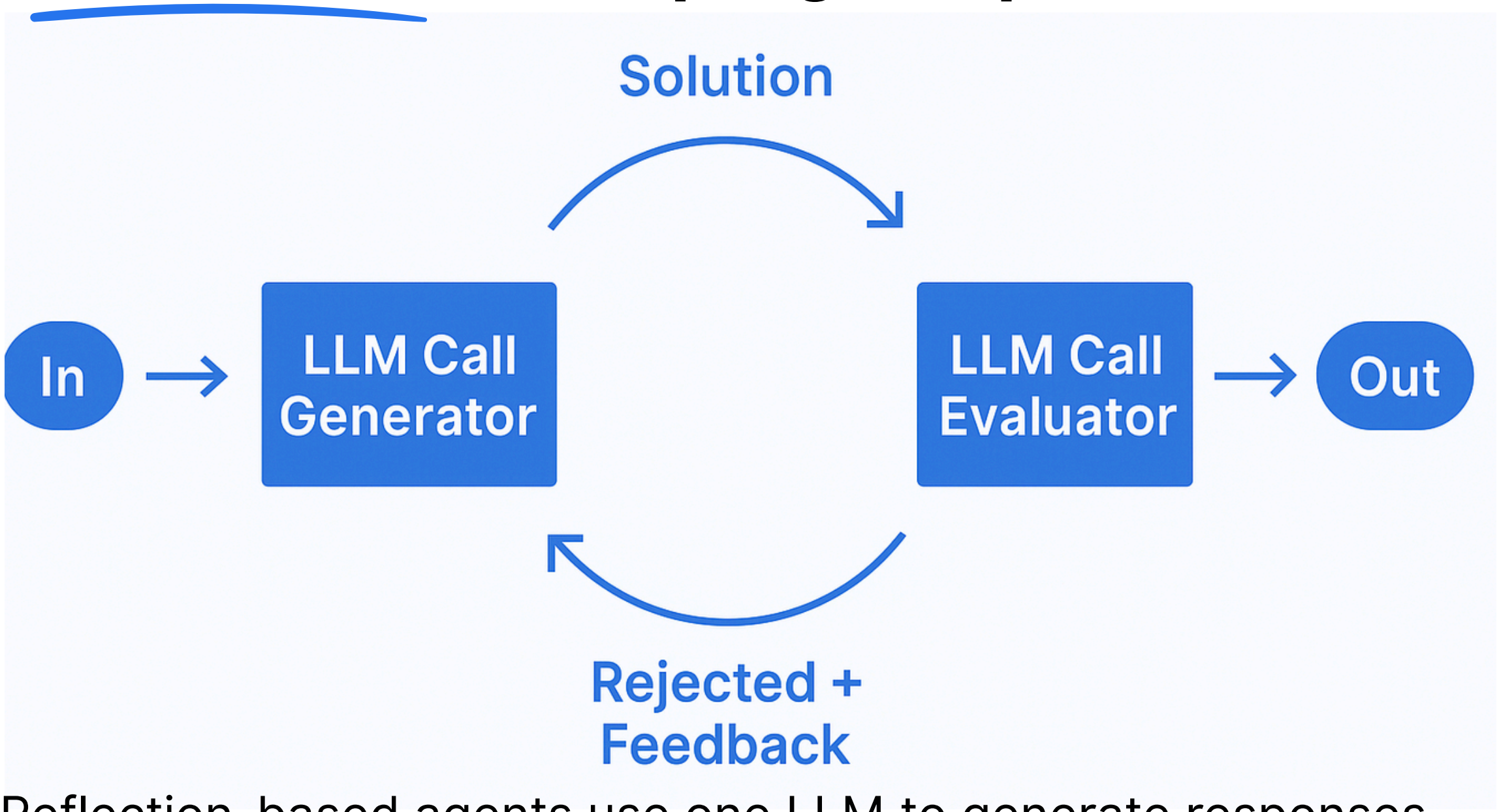
- **Hierarchical:** Multi-agent system with a supervisor of supervisors. This is a generalization of the supervisor architecture and allows for more complex control flows



- Always start with simple supervisor or network architecture and then expand

- Create separate agents based on specific processes, tasks and flows

Reflection for Critiquing & Improvements



Reflection-based agents use one LLM to generate responses and another to evaluate and provide feedback in a loop. This is effective when clear evaluation criteria exist and iterative refinement adds value.

Ideal conditions:

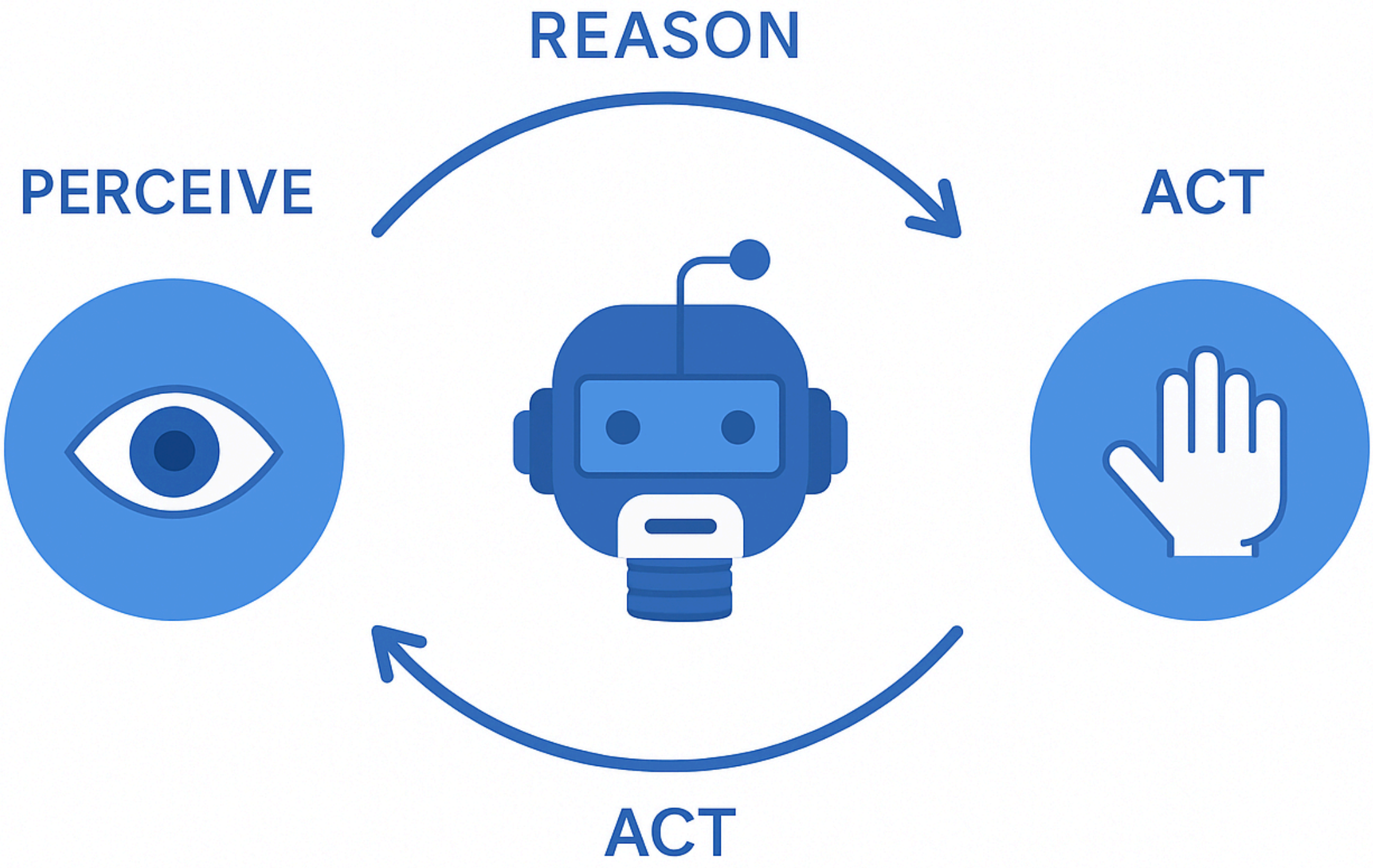
- LLM responses improve with human-like feedback.
- The LLM can generate such feedback.

This mirrors how humans refine writing or code through review and testing.

Use cases include:

- Enhancing RAG retrieval and responses
- Grading or critiquing LLM outputs
- Validating criteria (e.g., claims processing)

Final Words



- Start by building key components of your Agentic AI System
- Start with Simple Tool-Use ReAct Single Agent Systems
- Add Planning only if the above fails and you need to add explicit planning modules for complex tasks
- Add Reflection for handling any tasks around grading, critiquing, improving responses at any step in the agent
- Add Routing to handle multiple flows or agents reliably
- Consider Multi-Agent systems when you have multiple processes, workflows, too many tools to handle and you can segregate tasks to specific agents with a set of tools
- Do not add in too many Agents unless absolutely necessary
- Monitoring and Debugging is super useful to check for common failure patterns