Getting Started with LangGraph

By Lucas Soares

11/07/2024

Lucas Soares

- AI Engineer

Lucas Soares

- AI Engineer
- Curious about Building Tools with AI

Understanding Agentic Systems

- **Agentic Systems**: Systems that utilize large language models (LLMs) to manage the control flow of applications.

Understanding Agentic Systems

- **Agentic Systems**: Systems that utilize large language models (LLMs) to manage the control flow of applications.
- **Key Functions**: Routing decisions, tool selection, and evaluating output sufficiency.

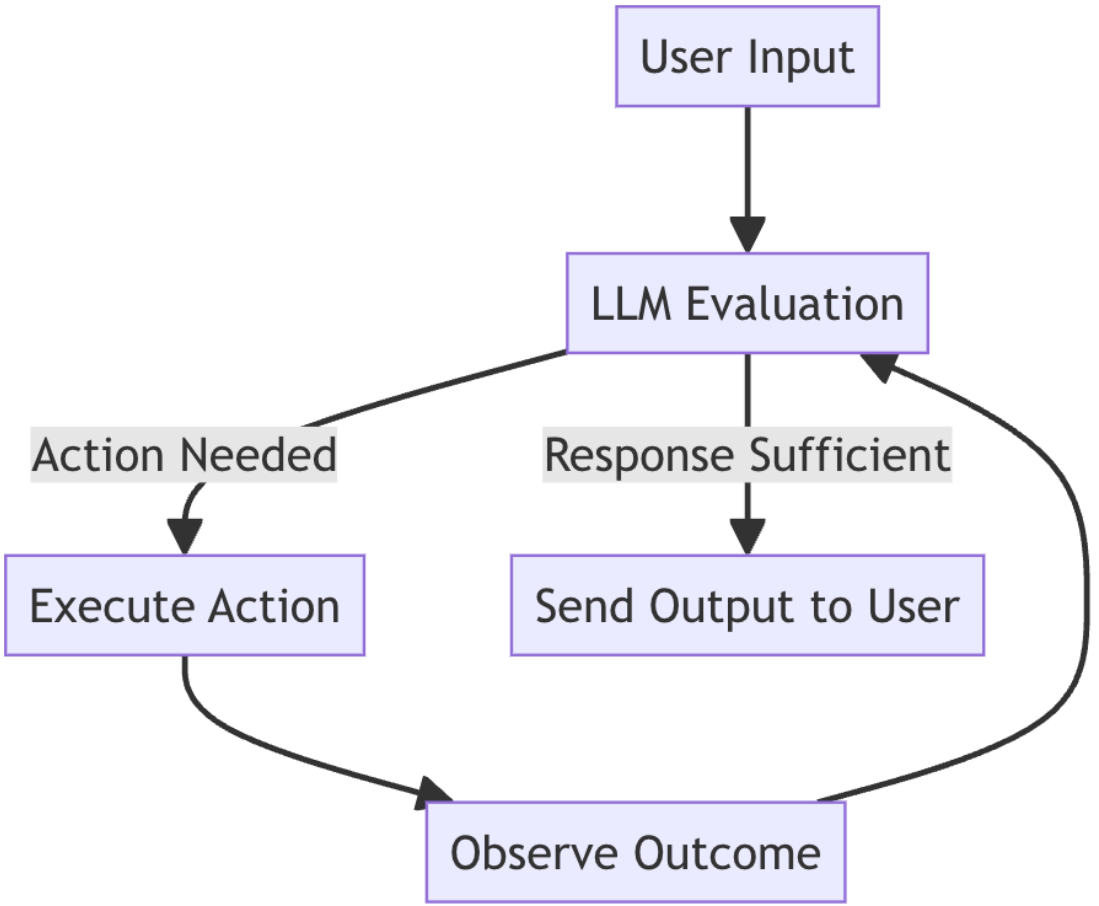
Understanding Agentic Systems

- **Agentic Systems**: Systems that utilize large language models (LLMs) to manage the control flow of applications.
- **Key Functions**: Routing decisions, tool selection, and evaluating output sufficiency.
- **Agent Loop**: Continuous decision-making process that enables agents to solve complex tasks.

Understanding agentic systems in LangGraph.

The Agent Loop

The Agent Loop



Explanation of the agent loop in cognitive architectures.

Practical Use Case: Customer Support Agent

Practical Use Case: Customer Support Agent

- **Scenario**: An LLM-powered customer support agent.

Practical Use Case: Customer Support Agent

- **Scenario**: An LLM-powered customer support agent.
- **User Input**: Customer asks about order status.

Practical Use Case: Customer Support Agent

- **Scenario**: An LLM-powered customer support agent.
- **User Input**: Customer asks about order status.
- **LLM Decision**: Determines if it can provide the status directly or if it needs to fetch data from the database.

Practical Use Case: Customer Support Agent

- **Scenario**: An LLM-powered customer support agent.
- **User Input**: Customer asks about order status.
- **LLM Decision**: Determines if it can provide the status directly or if it needs to fetch data from the database.
- **Action Taken**: If data fetch is needed, the agent queries the database and updates the user with the order status.

Practical use case of LLM agents in customer support.

Advantages of LLM Agents

Advantages of LLM Agents

- **Flexibility**: Agents can adapt to various tasks by determining the best action to take.

Advantages of LLM Agents

- **Flexibility**: Agents can adapt to various tasks by determining the best action to take.
- **Specialization**: LLM agents can be specialized with a set of tools to perform niche tasks.

Advantages of LLM Agents

- **Flexibility**: Agents can adapt to various tasks by determining the best action to take.
- **Specialization**: LLM agents can be specialized with a set of tools to perform niche tasks.
- **Multi-Agent Collaboration**: Specialized LLM agents can collaborate to perform complex tasks.

Key Components of Agentic Systems

Key Components of Agentic Systems

1. **Tool Calling**: Utilizing external tools to perform tasks.

Key Components of Agentic Systems

1. **Tool Calling**: Utilizing external tools to perform tasks.
2. **Action Taking**: Executing actions based on LLM outputs.

Key Components of Agentic Systems

1. **Tool Calling**: Utilizing external tools to perform tasks.
2. **Action Taking**: Executing actions based on LLM outputs.
3. **Memory**: Keeping track of interactions for context-aware responses.

Key Components of Agentic Systems

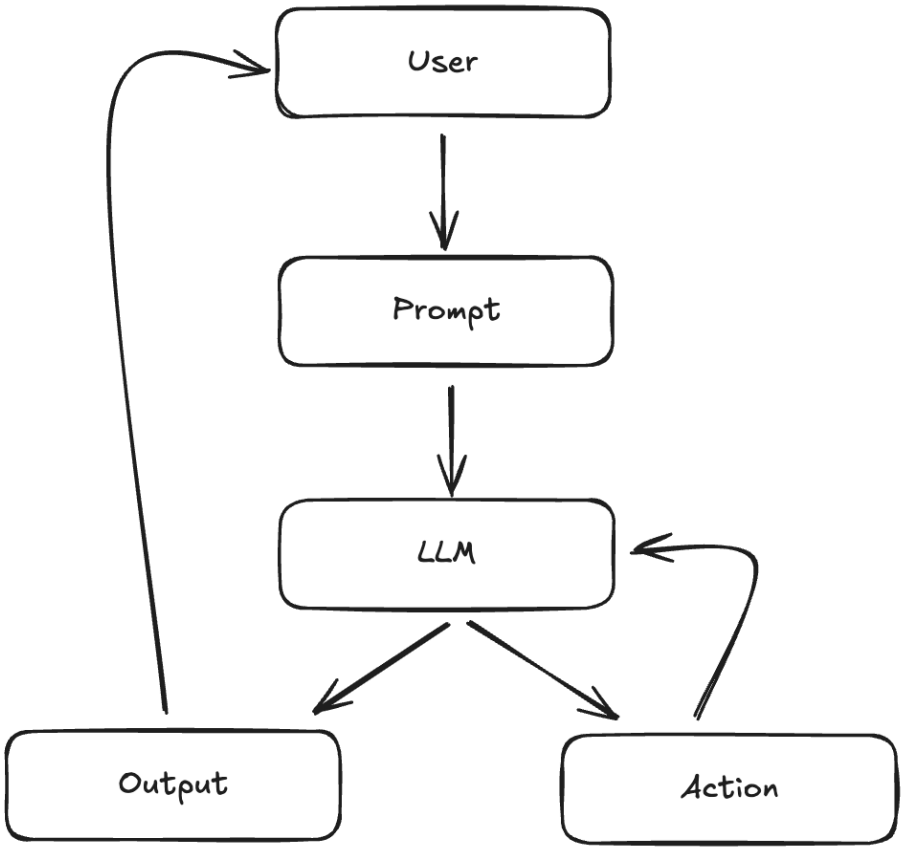
1. **Tool Calling**: Utilizing external tools to perform tasks.
2. **Action Taking**: Executing actions based on LLM outputs.
3. **Memory**: Keeping track of interactions for context-aware responses.
4. **Planning**: Structuring steps to ensure optimal decision-making.

Key components of agentic systems.

Agents as Graphs

Agents as Graphs

- Workflows built with agents are usually structured as graphs!!



Why LangGraph?

LangGraph is designed for building agentic applications with some core principles:

Why LangGraph?

LangGraph is designed for building agentic applications with some core principles:

- **Controllability**: Offers low-level control which increases reliability in agentic systems.

Why LangGraph?

LangGraph is designed for building agentic applications with some core principles:

- **Controllability**: Offers low-level control which increases reliability in agentic systems.
- **Human-in-the-Loop**: Built-in persistence layer enhances human-agent interaction patterns.

Why LangGraph?

LangGraph is designed for building agentic applications with some core principles:

- **Controllability**: Offers low-level control which increases reliability in agentic systems.
- **Human-in-the-Loop**: Built-in persistence layer enhances human-agent interaction patterns.
- **Streaming First**: Supports streaming of events and tokens, providing real-time feedback to users.

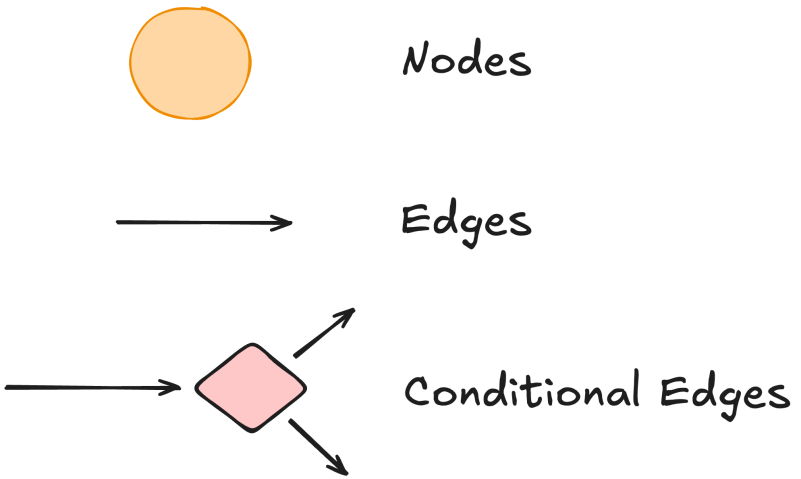
Overview of LangGraph's purpose and principles.

The Basic Components of LangGraph

LangGraph models agent workflows as graphs:

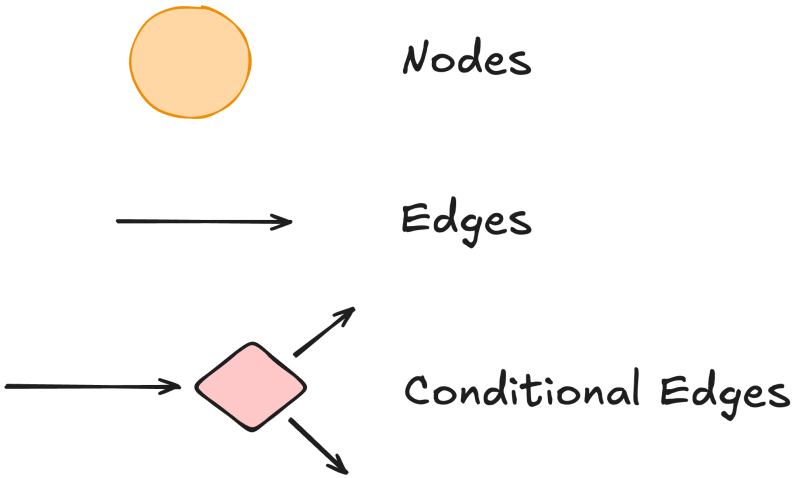
The Basic Components of LangGraph

LangGraph models agent workflows as graphs:



The Basic Components of LangGraph

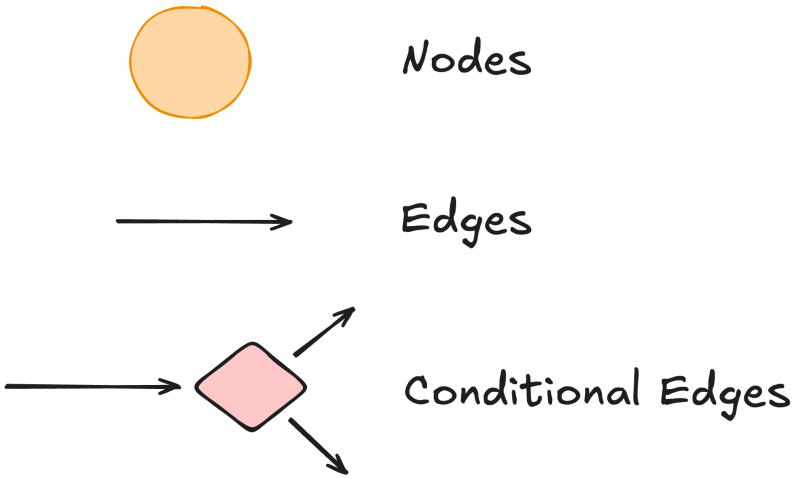
LangGraph models agent workflows as graphs:



- **Nodes**: Python functions that implement the logic of agents, taking the current State as input and returning an updated State.

The Basic Components of LangGraph

LangGraph models agent workflows as graphs:



- **Nodes**: Python functions that implement the logic of agents, taking the current State as input and returning an updated State.
- **Edges/Conditional Edges**: Functions that implement fixed/conditional transitions to determine which Node to execute next based on the current State.

Explanation of LangGraph's components.

States in LangGraph

These graphs in LangGraph are driven by:

States in LangGraph

These graphs in LangGraph are driven by:

- **States**: Shared data structures that evolve over time as Nodes execute and pass messages along Edges.

States in LangGraph

These graphs in LangGraph are driven by:

- **States**: Shared data structures that evolve over time as Nodes execute and pass messages along Edges.
- **Message Passing**: Nodes send messages to activate other Nodes, facilitating the execution of workflows in discrete iterations or "super-steps".

Overview of how graphs and states interact in LangGraph.

Notebook Demo: Introduction to LangGraph

Nodes

In LangGraph, Nodes are the core functional units:

Nodes

In LangGraph, Nodes are the core functional units:

- **Functionality**: Each Node is a Python function that processes the current State and outputs an updated State.

Nodes

In LangGraph, Nodes are the core functional units:

- **Functionality**: Each Node is a Python function that processes the current State and outputs an updated State.
- **Execution**: Nodes can run synchronously or asynchronously, and are added to the graph using the `add_node` method.

Nodes

In LangGraph, Nodes are the core functional units:

- **Functionality**: Each Node is a Python function that processes the current State and outputs an updated State.
- **Execution**: Nodes can run synchronously or asynchronously, and are added to the graph using the `add_node` method.
- **Special Nodes**: Includes START and END Nodes to manage the flow of execution in the graph.

Details on the functionality and structure of Nodes in LangGraph.

Notebook Demo: Nodes in LangGraph

Edges

Edges define the routing logic in LangGraph:

- **Types of Edges**:

Edges

Edges define the routing logic in LangGraph:

- **Types of Edges**:
  - **Normal Edges**: Direct transitions from one Node to another.

Edges

Edges define the routing logic in LangGraph:

- **Types of Edges**:
  - **Normal Edges**: Direct transitions from one Node to another.
  - **Conditional Edges**: Determine the next Node(s) to execute based on a function's output.

Edges

Edges define the routing logic in LangGraph:

- **Types of Edges**:
  - **Normal Edges**: Direct transitions from one Node to another.
  - **Conditional Edges**: Determine the next Node(s) to execute based on a function's output.
  - **Entry Points**: Specify which Node to invoke first based on user input.

Edges

Edges define the routing logic in LangGraph:

- **Types of Edges**:
  - **Normal Edges**: Direct transitions from one Node to another.
  - **Conditional Edges**: Determine the next Node(s) to execute based on a function's output.
  - **Entry Points**: Specify which Node to invoke first based on user input.
- **Parallel Execution**: Multiple outgoing edges from a Node can trigger parallel execution of destination Nodes.

Explanation of Edges and their role in LangGraph.

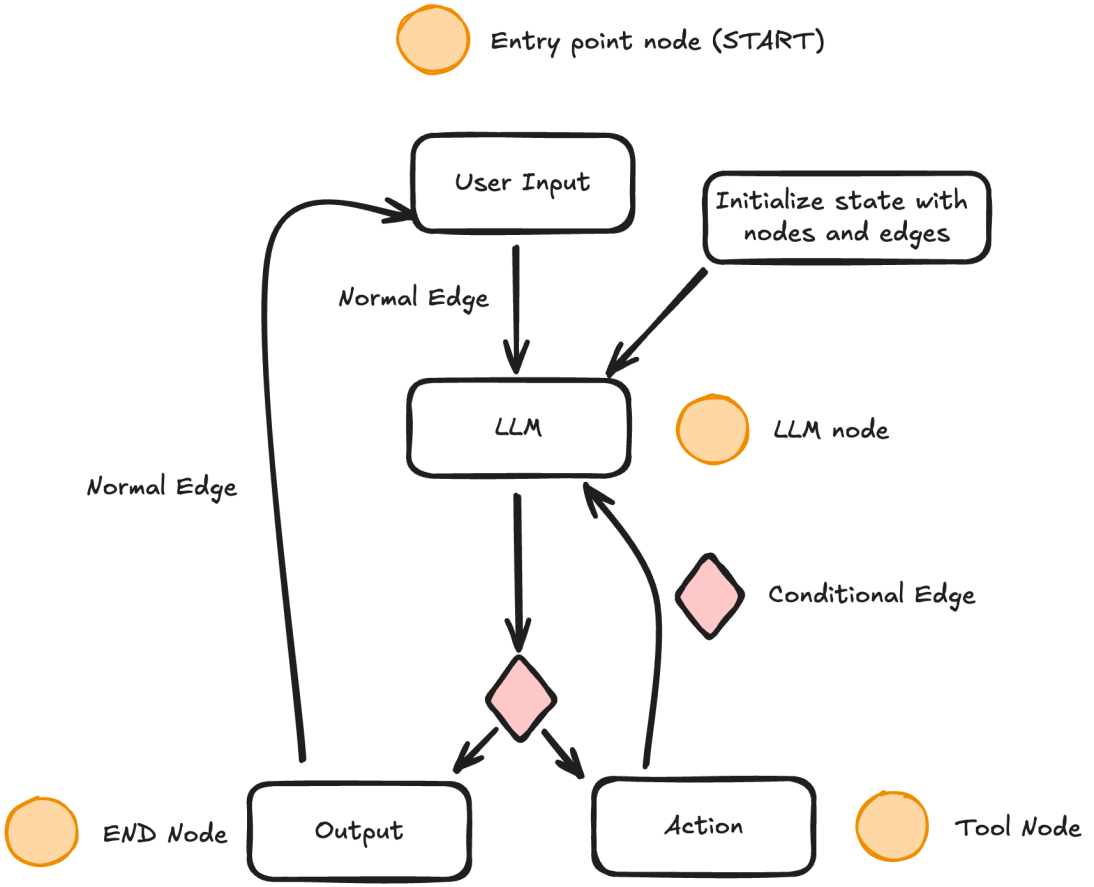
Notebook Demo: Edges in LangGraph

Agent Loop in LangGraph

- Outline of an basic agent loop in langgraph:

Agent Loop in LangGraph

- Outline of an basic agent loop in langgraph:

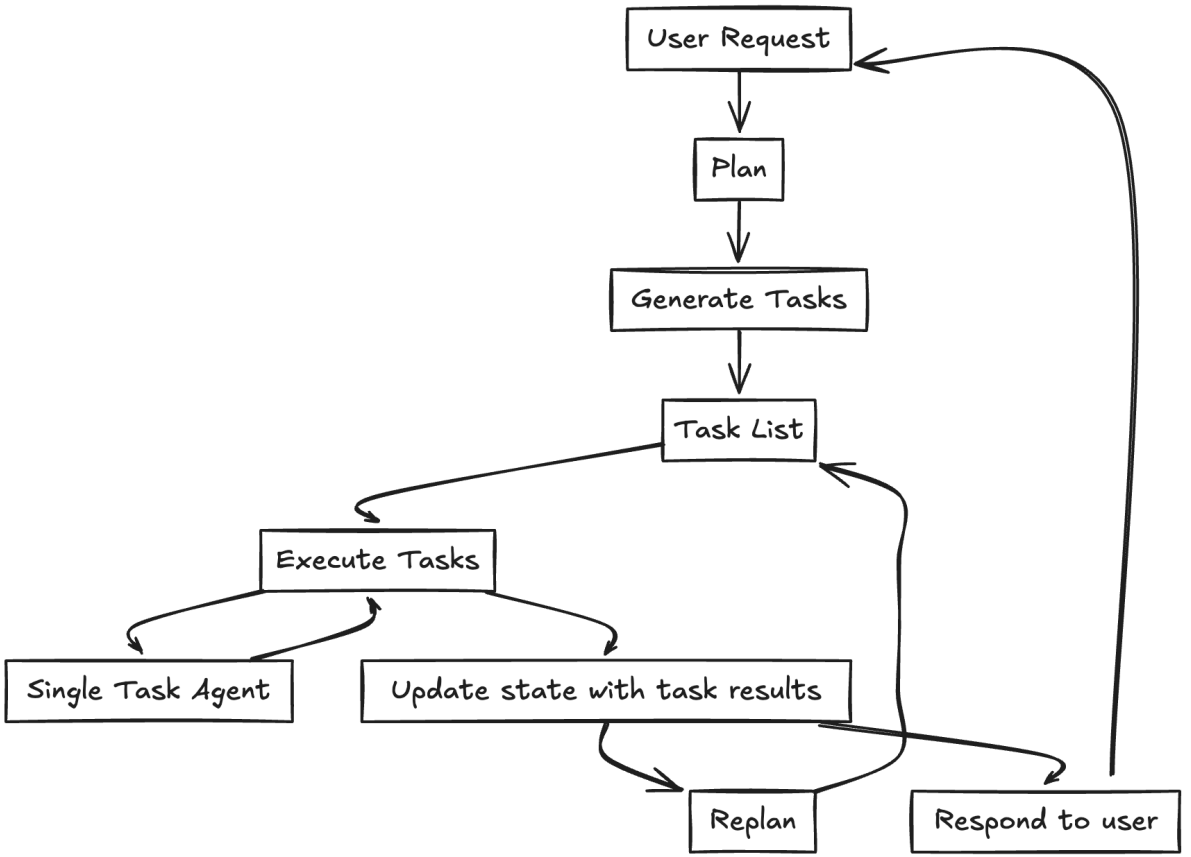


Notebook Demo: A Basic Research Agent in LangGraph

Agentic Patterns

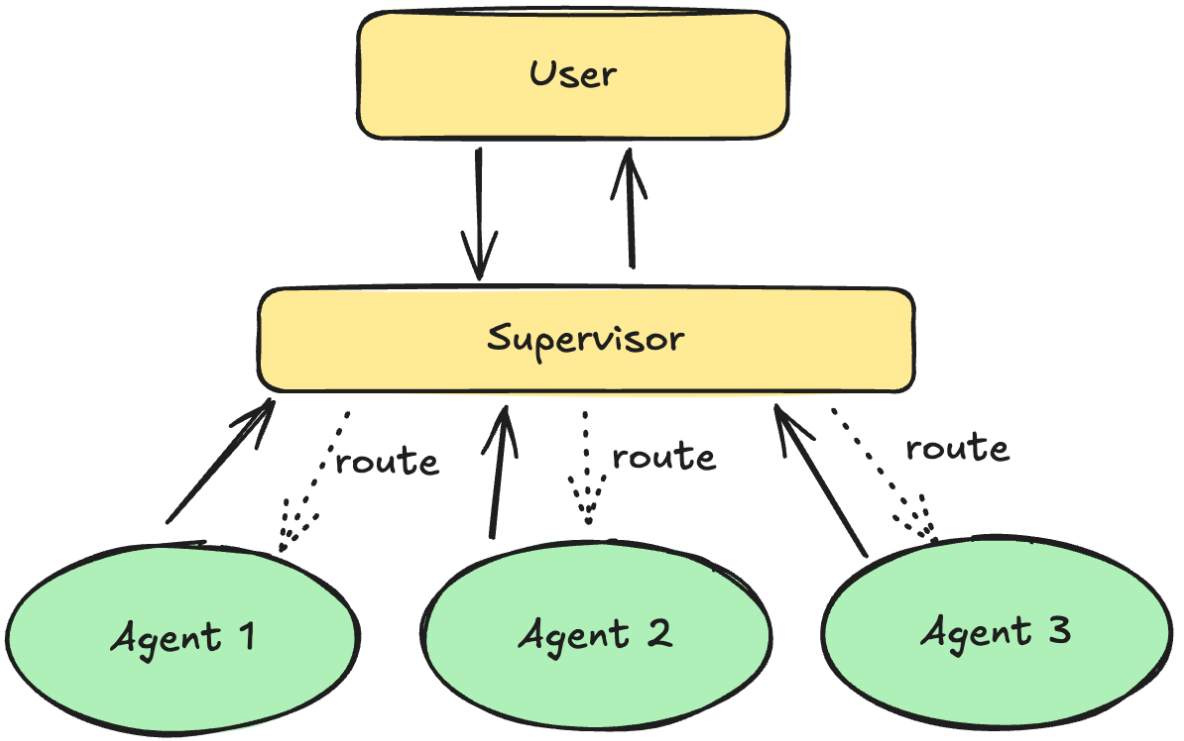
Agentic Patterns

Plan & Execute



Agentic Patterns

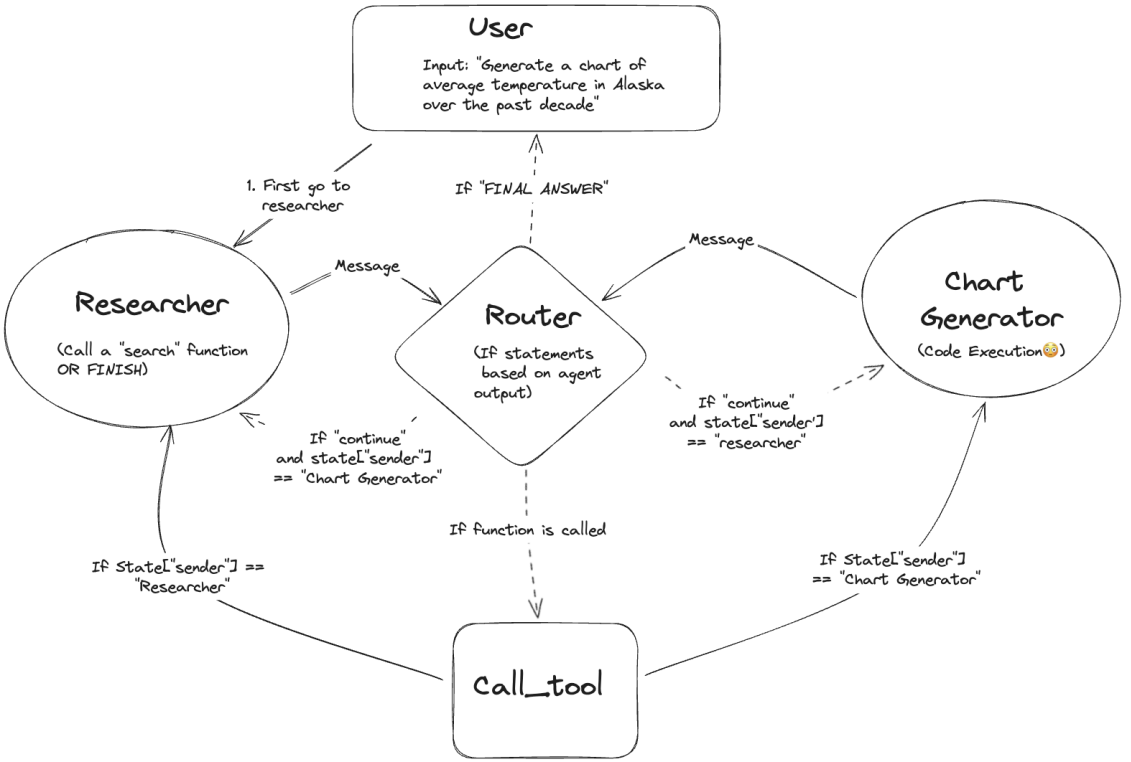
Supervisor



LangGraph: Multi-Agent Workflows

Agentic Patterns

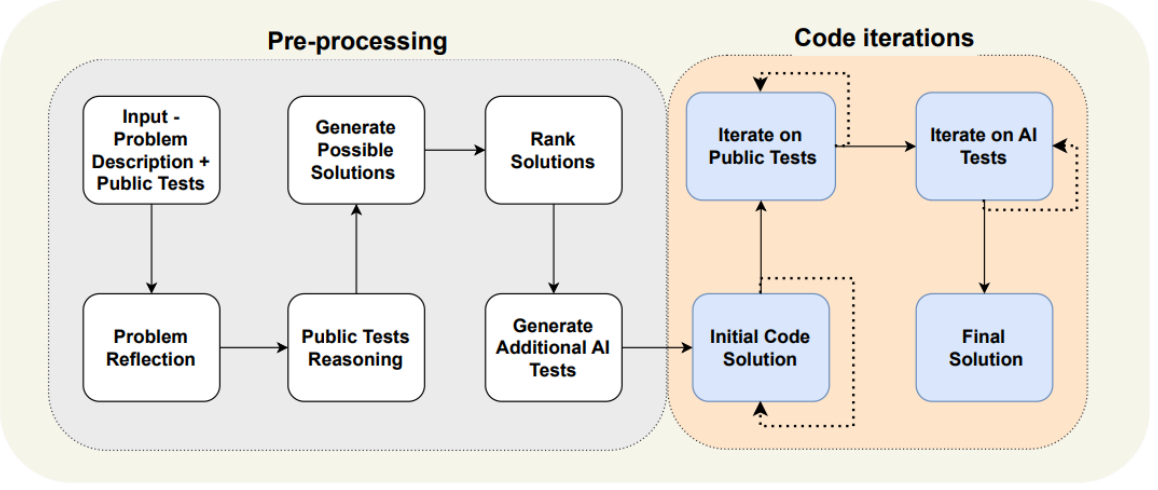
Multi-Agent



[LangGraph: Multi-Agent Workflows]

Agentic Patterns

Flow Engineering



(a) The proposed AlphaCodium flow.

[Code Generation with AlphaCodium]

Notebook Demo: Self-Reflective RAG Agent with LangGraph