LangGraph Agentic Framework

Mukesh Mittal

Agenda

- A quick intro to LangChain
- A bit of LangGraph
  - and probably another session or series
  - Replit - A use case
  - Reddit - a small survey
- **Demo/s**
- Preso copy and the code - available afterwards
- Sources:
  - https://www.langchain.com/ and
  - https://langchain-ai.github.io/langgraph
  - unless mentioned

Training - courses

Short Course

AI Agents in LangGraph

Build agentic AI workflows using LangChain's LangGraph and Tavily's agentic search.

LangChain, Tavily

Short Course

Build LLM Apps with LangChain.js

Expand your toolkit with LangChain.js, a JavaScript framework for building with LLMs. Understand the fundamentals of using LangChai...

LangChain

Short Course

Functions, Tools and Agents with LangChain

Learn about the latest advancements in LLM APIs and use LangChain Expression Language (LCEL) to compose and customize chains and agents.

LangChain

Short Course

LangChain: Chat with Your Data

Create a chatbot with LangChain to interface with your private data and documents. Learn from LangChain creator, Harrison Chase.

LangChain

Short Course

LangChain for LLM Application Development

Use the powerful and extensible LangChain framework, using prompts, parsing, memory, chains, question answering, and agents.

LangChain

https://www.deeplearning.ai/

Training - courses

Course Curriculum

- Welcome to the course!
- Module 1: Introduction
- Module 2: State and Memory
- Module 3: UX and Human-in-the-Loop
- Module 4: Building Your Assistant

LangGraph

About this course

- Free
- 40 lessons
- 4 hours of video content

https://academy.langchain.com/courses/intro-to-langgraph

LangChain

LangChain

The biggest developer community in GenAI

Learn alongside the 100K+ practitioners who are pushing the industry forward.

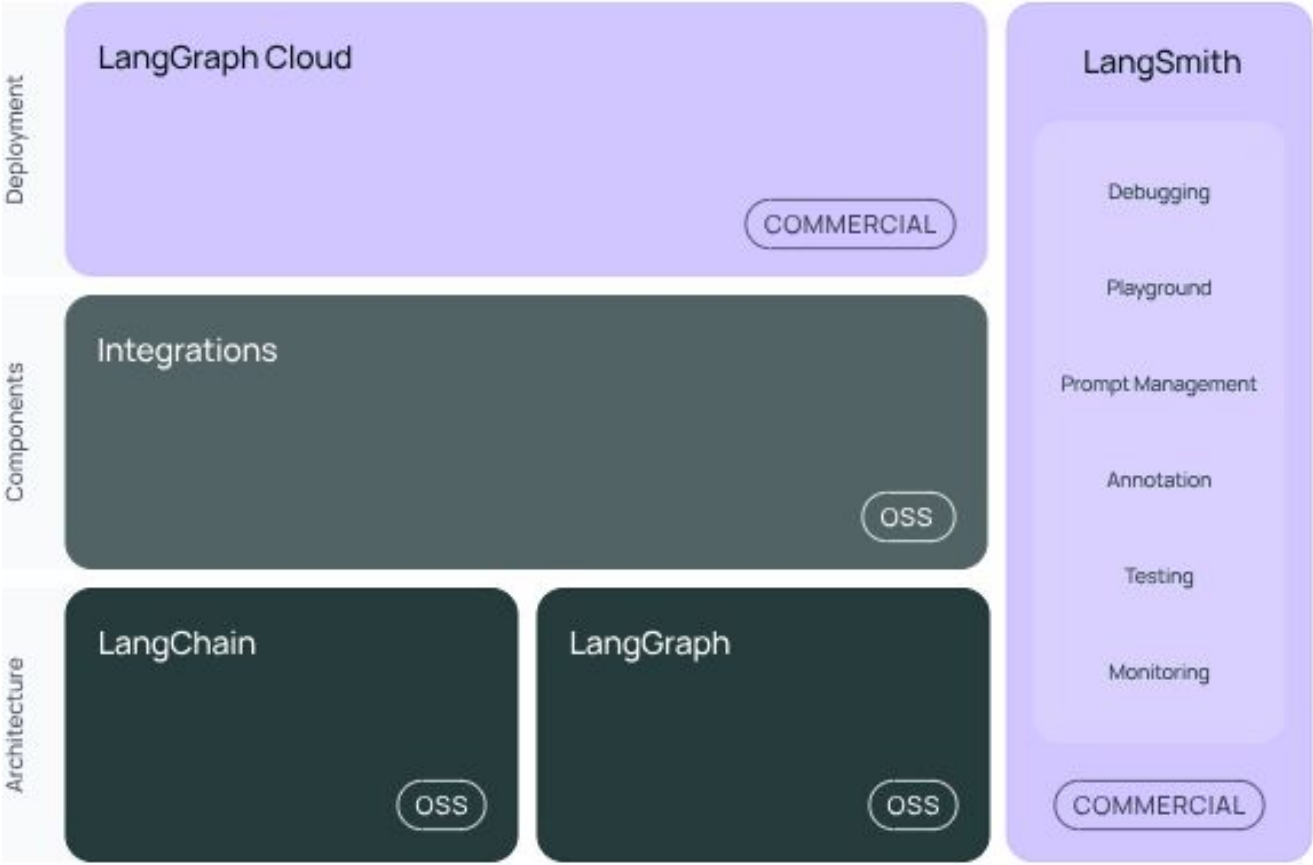
15M+
Monthly Downloads

100K+
Apps Powered

75K+
GitHub Stars

3K+
Contributors

LangChain


Deployment
LangGraph Cloud
COMMERCIAL
Components
Integrations
OSS
Architecture
LangChain
OSS
LangGraph
OSS
LangSmith
Debugging
Playground
Prompt Management
Annotation
Testing
Monitoring
COMMERCIAL

Langchain

- Open-source framework for developing applications powered by language models.
- Integration with various LLMs
- To provide a standard interface for chaining together different components
- Combines LLMs with external data sources and computational tools
- Facilitates the creation of AI agents that can interact with their environment
- Modular architecture for easy customization and extension
- Built-in memory and context management
- Support for external data sources and tools

Components of Langchain

- Models: LLMs and embeddings
- Prompts: Templates and management
- Indexes: For efficient data retrieval
- Chains: Combine LLMs with other components
- Agents: Autonomous task completion

Agents in Langchain

- By themselves, language models can't take actions - they just output text.
- Agents are systems that use an LLM as a reasoning engine to determine
  - which actions to take and
  - what the inputs to those actions should be.
- Uses:
  - Autonomous decision-making entities
  - Tool selection and usage
  - Memory and state management
  - Goal-oriented task completion

001

Plan-and-execute

002

Multi-agent

003

Critique Revise

004

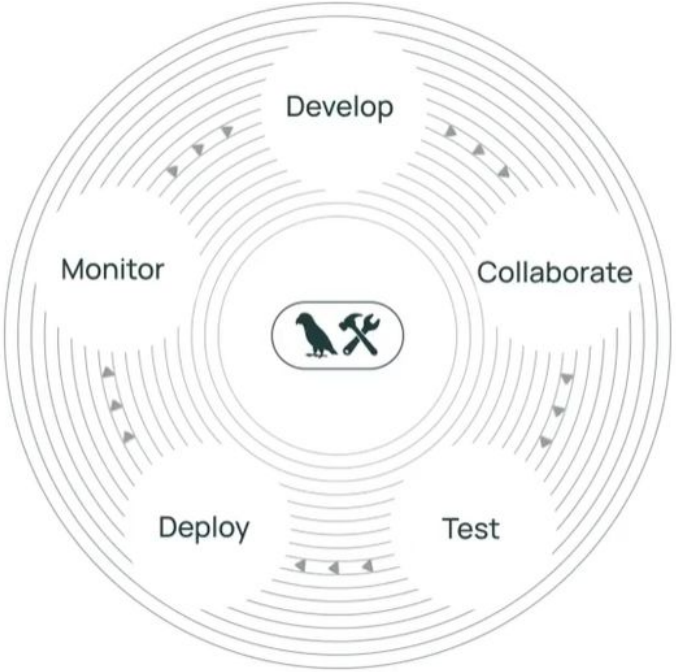
ReAct

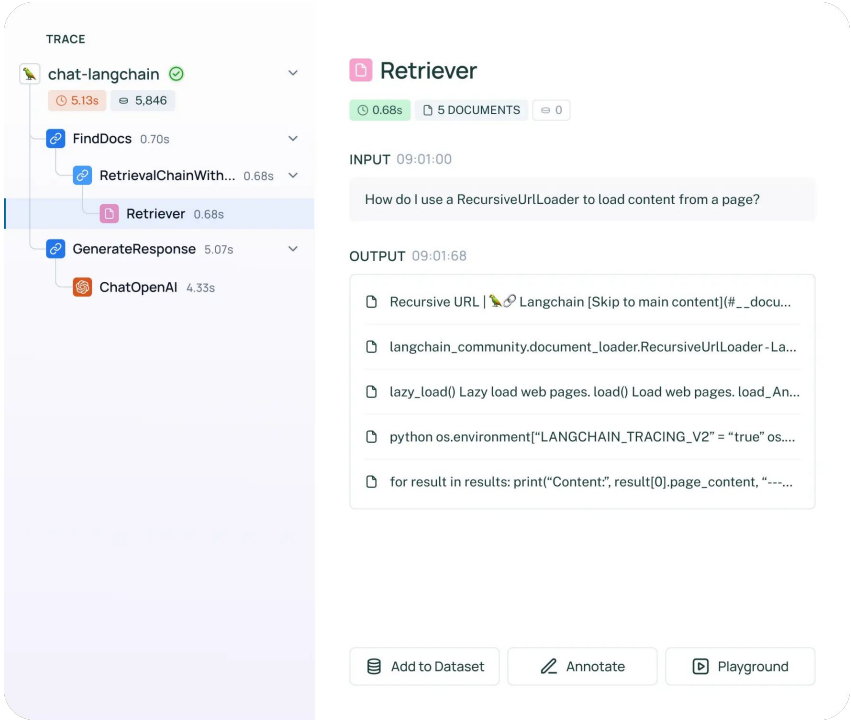
005

Self-ask

LangSmith

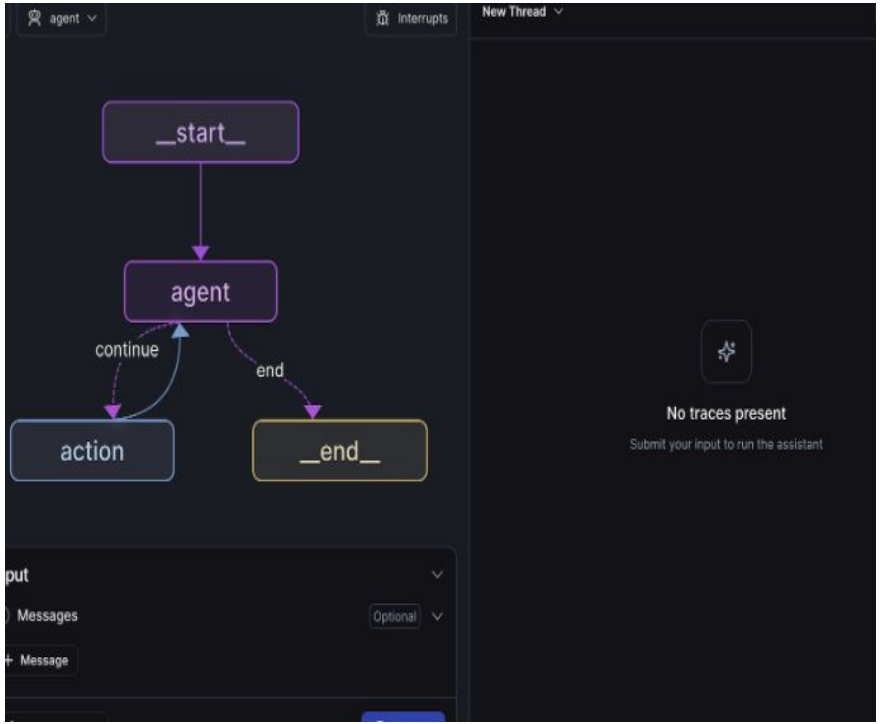
a platform for building production-grade LLM applications. It allows you to closely **monitor and evaluate** your application





LangGraph

- An extension of LangChain
- specifically aimed at creating
  - highly controllable and customizable agents.



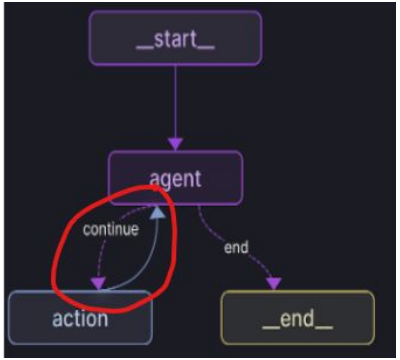
LangGraph

LangGraph

pypi v0.2.37 | downloads/month 1M | open issues 44 | docs latest

Building language agents as graphs

- A library - used to create **agent** and multi-agent **workflows**.
- Core benefits: cycles, controllability, and persistence
- a very low-level framework
  - fine-grained control over both the flow and
  - state of your application
- includes built-in persistence, enabling
  - advanced human-in-the-loop and
  - memory features
- Streaming Support
- LangGraph integrates seamlessly with LangChain and LangSmith
  - But can be used without LangChain.



LangGraph - Replit Agent

- Platform that simplifies writing, running, and collaborating on code
  - for over 30+ million developers.
- Complex workflow built on LangGraph,
  - enables a highly custom agentic workflow
    i. with a high-degree of control and
    ii. parallel execution.
- seamless integration with LangSmith,
  - deep visibility into their agent interactions (to debug tricky issues).
- With reviewing and writing code, also performs a wider range of functions :
  - planning,
  - creating dev environments,
  - installing dependencies, and
  - deploying applications for users.

Source:
https://blog.langchain.dev/customers-replit/

A Study - LangGraph Use Cases

- **Content Generation:** multiple AI agents collaborate to draft, fact-check, and refine research papers in real-time.
- **Customer Service:** dynamic response systems that analyze sentiment, retrieve relevant information, and generate personalized replies with built-in clarification mechanisms.
- **Financial Modeling:** valuation models in real estate that adapt in real-time based on market fluctuations and simulated scenarios.
- **Academic Research:** adaptive research assistants capable of gathering data, synthesizing insights, and proposing new hypotheses within a single integrated system.

Source: https://www.reddit.com/r/LangChain/comments/1eh0ly3/spoke_to_22_langgraph_devs_and_heres_what_we_found/

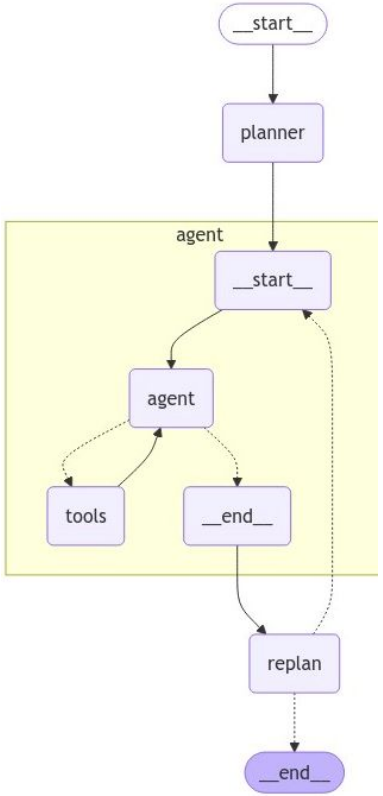
LangGraph vs Alternatives (CrewAI and Microsoft's Autogen)

- Handling Complex Workflows: including **cycles** (CrewAI can only handle DAGs)
- Developer Control: LangGraph offers **a level of control** that many find unmatched, especially for custom use cases.
- **Mature Ecosystem**: LangGraph's longer market presence has resulted in more resources, tools, and infrastructure.
- The ability to use **LangSmith** in conjunction with LangGraph for debugging and performance analysis is a significant differentiator."
- Leader in functionality and tooling for developing workflows.

Source: https://www.reddit.com/r/LangChain/comments/1eh0ly3/spoke_to_22_langgraph_devs_and_heres_what_we_found/

LangGraph - Three key components

- State:
  - A shared data structure that represents **the current snapshot** of your application. It can be any Python type, but is typically a TypedDict or Pydantic BaseModel.
- Nodes:
  - Python functions that encode **the logic of your agents**. They receive the current State as input, perform some computation or side-effect, and return an updated State.
- Edges:
  - Python functions that determine **which Node** to execute next based on the current State. They can be **conditional** branches or **fixed** transitions.
- Nodes and Edges are nothing more than Python functions (contain LLM or code)
- Summary: Nodes **do the work**. Edges tell **what to do next**.



State

```
class State(TypedDict):
    # Messages have the type "list". The `ad
    # in the annotation defines how this sta
    # (in this case, it appends messages to
    messages: Annotated[list, add_messages]
```

- Consists of
  - the schema of the graph as well as
  - reducer functions which specify how to apply updates to the state.
- The schema of the State will be **the input schema** to **all Nodes and Edges** in the graph
- All Nodes will **emit updates to the State** which are then applied using the specified **reducer function**.

LangGraph - Nodes

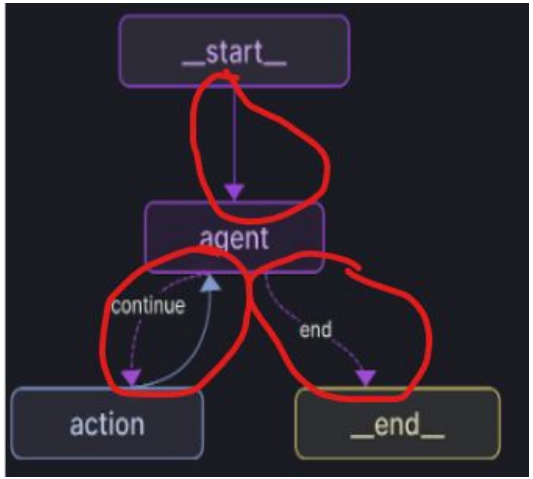
```python
def my_node(state: dict, config: RunnableConfig):
    print("In node: ", config["configurable"]["user_id"])
    return {"results": f"Hello, {state['input']}!"}


# The second argument is optional
def my_other_node(state: dict):
    return state


builder.add_node("my_node", my_node)
builder.add_node("other_node", my_other_node)
```

- START Node is a special node that represents the node sends user input to the graph
- The END Node is another special node - a terminal node i.e. no actions after

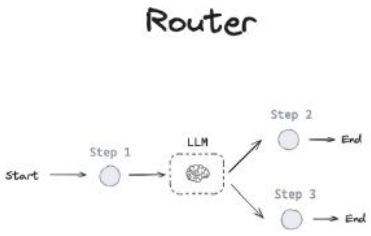
Type of Edges

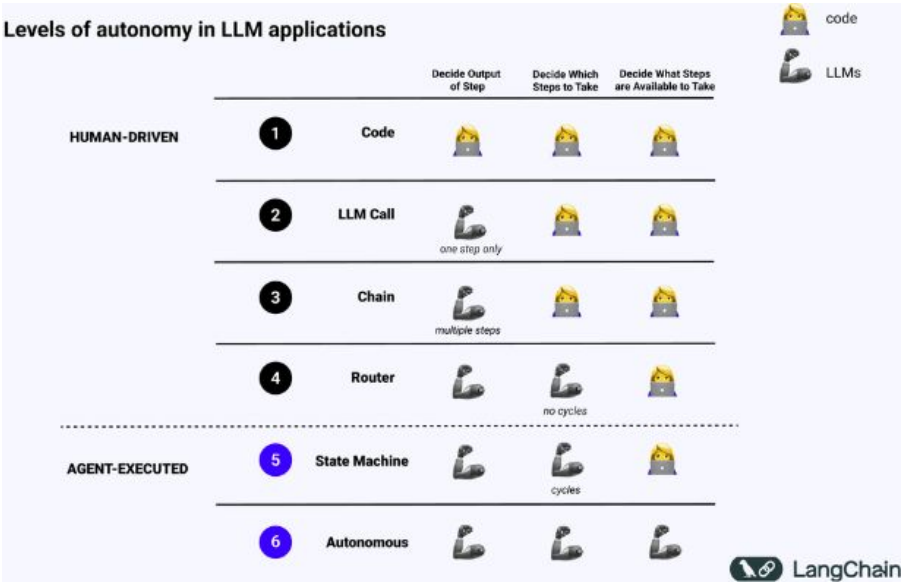
- Normal Edges:
  - Go directly from one node to the next.
- Conditional Edges:
  - Call a function to determine which node(s) to go to next.
- Entry Point:
  - Which node to call first when user input arrives.
- Conditional Entry Point:
  - Call a function to determine which node(s) to call first when user input arrives.

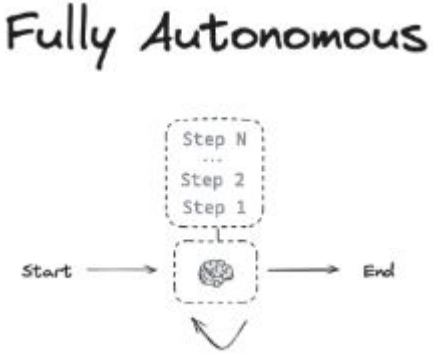
```
# Set the entrypoint as `agent`
# This means that this node is the fir
workflow.add_edge(START, "agent")

# We now add a conditional edge
workflow.add_conditional_edges(
    # First, we define the start node.
    # This means these are the edges t
    "agent",
    # Next, we pass in the function th
    should_continue,
```



Agent Architectures

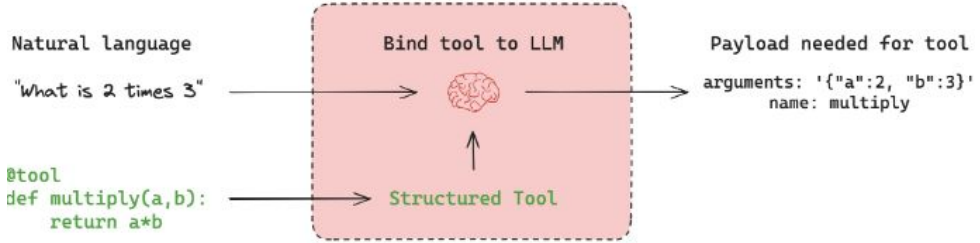






https://blog.langchain.dev/what-is-a-cognitive-architecture/

Tool calling agent (ReAct)

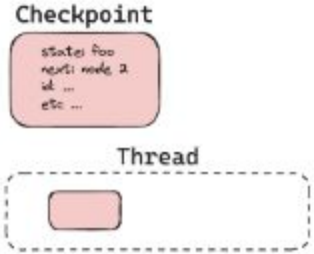


Human-in-the-loop - Approval

(1) Hit a breakpoint

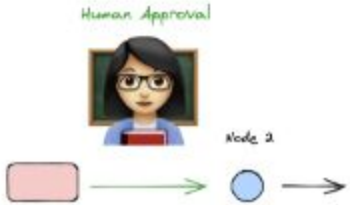
Graph stops at breakpoint before Node 2

(2) Wait for approval



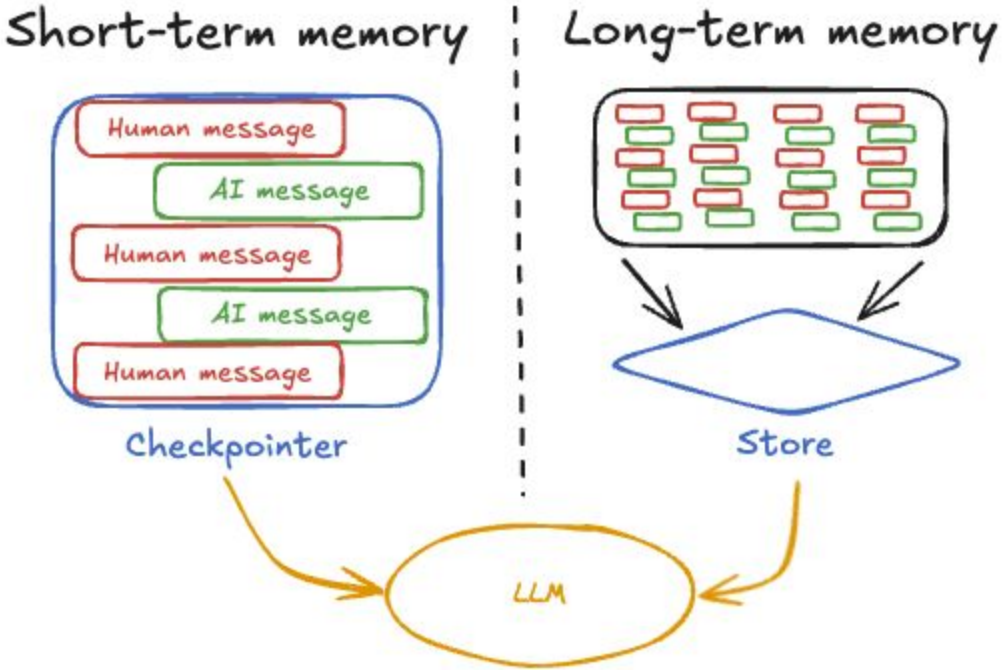
When stopped, we have a persistence layer with the current checkpoint written to it

(3) Continue

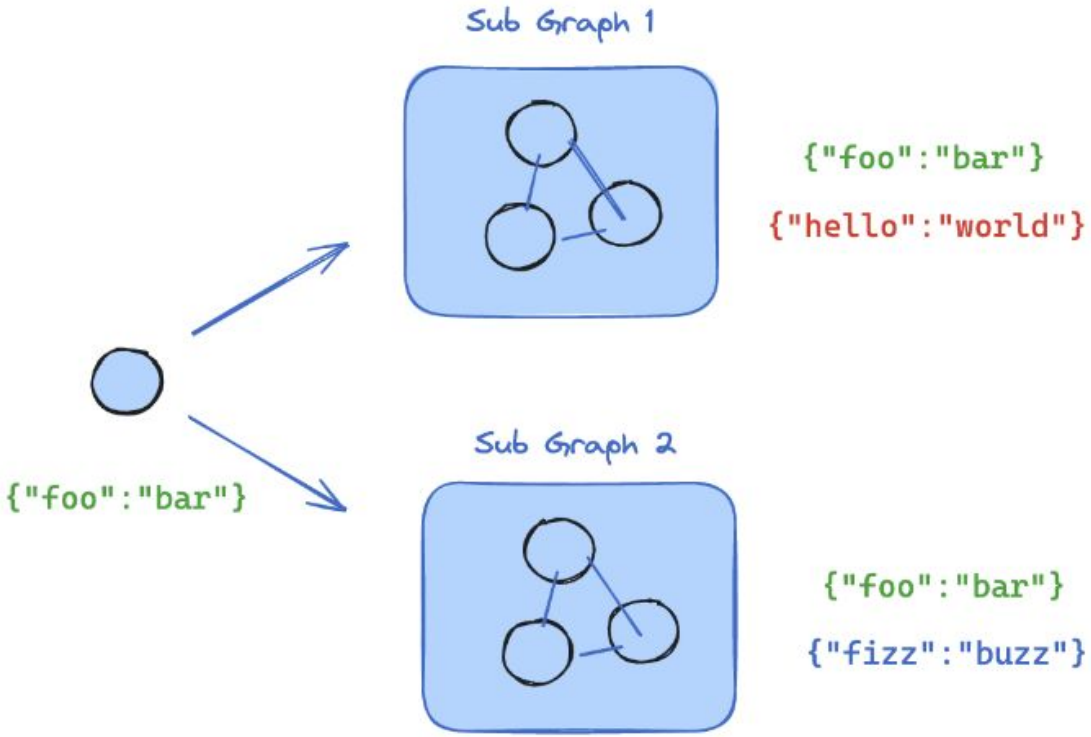


When a human approves the next step (Node 2), graph proceeds from the current checkpoint

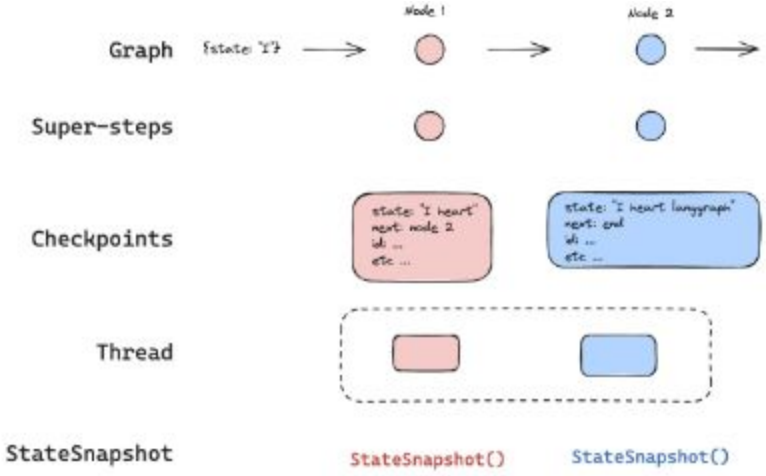
Memory


Short-term memory
Long-term memory
Human message
AI message
Human message
AI message
Human message
Checkpointer
Store
LLM

Multi-agent system - Agents as nodes

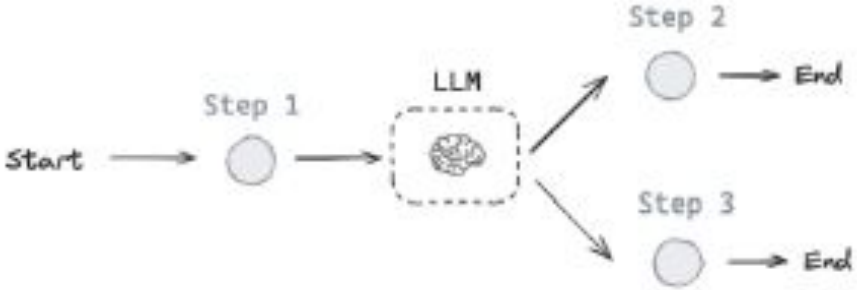


Persistence



Flow

- Initialize the model and tools.
- Initialize graph with state.
- Define graph nodes.
- Define entry point and graph edges.
- Compile the graph.
- Execute the graph.



Demos

1. Very Basic like HelloWorld
2. Plan-and-Execute example from LangGraph

   (https://langchain-ai.github.io/langgraph/tutorials/plan-and-execute/plan-and-execute/#define-our-execution-agent)

What Next?

- Open source project - Via GitHub repo - All Welcome!

?

Thanks