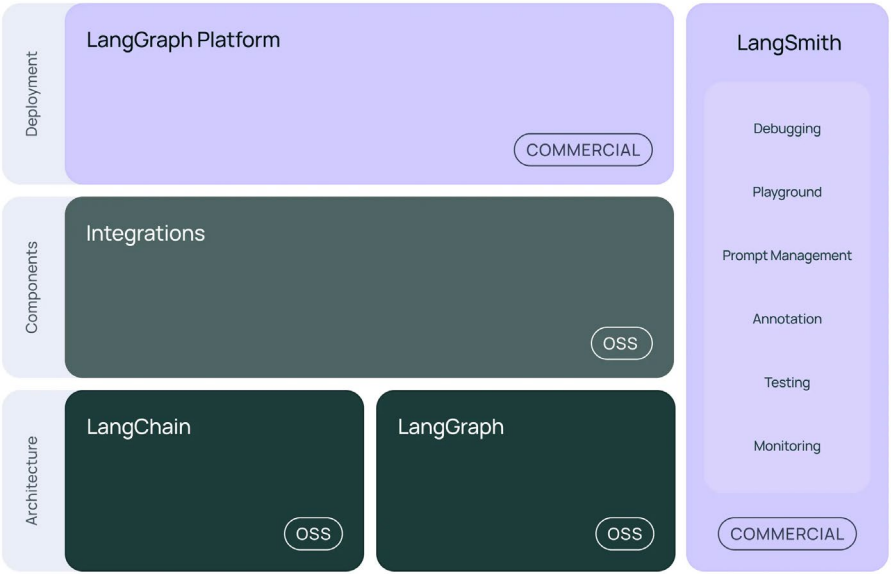
Applied Generative AI

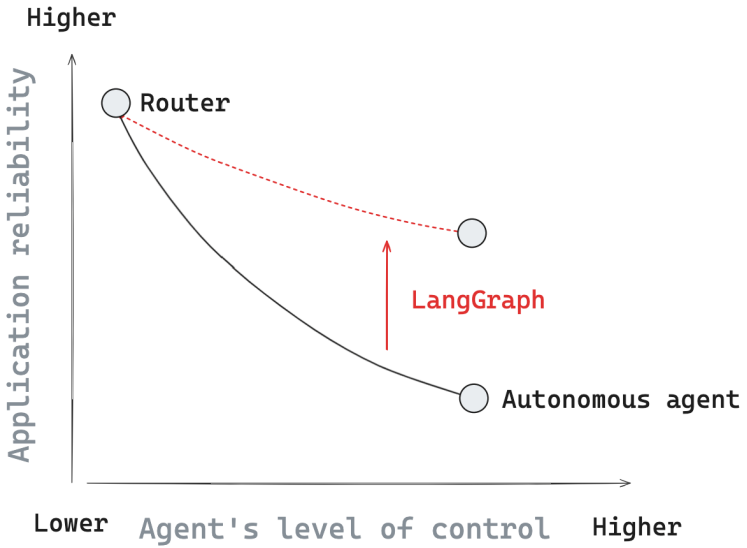
Agents in LangGraph: Introduction

What is LangGraph?



- A flexible framework for orchestrating Generative AI workflows using a graph-based approach.
- Part of the LangChain ecosystem for Agents.

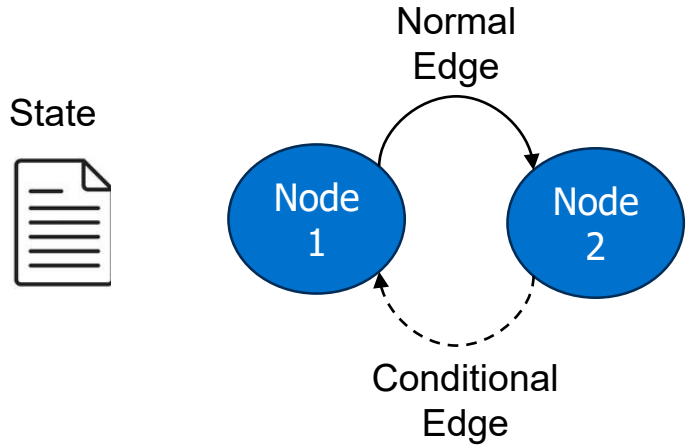
Why use LangGraph?



Why LangGraph. LangGraph Docs.
https://langchainai.github.io/langgraphjs/concepts/high_level/

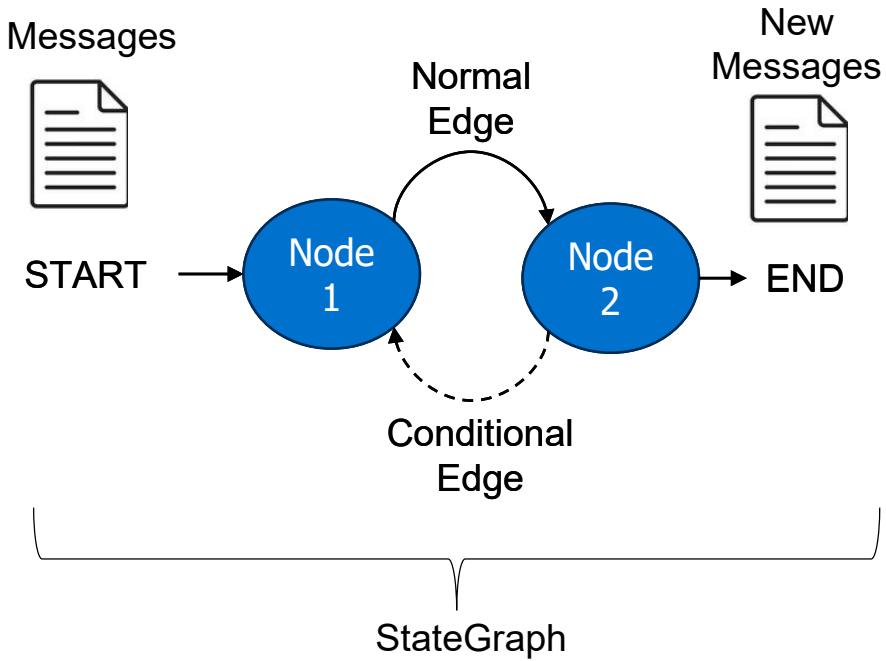
- Lower-level Agent creation.
- Enhances maintainability with clear state sharing and predictable data flows.
- Leverages existing protocols (e.g., LangChain) for integrations.

Fundamental Objects of LangGraph



- **State** is the data structure that represents the application:
  - Shared across the nodes.
- **Nodes** are operations:
  - Typically, Python functions.
- **Edges** send data between nodes:
  - Two main types of edges: Normal and Conditional.

The Graph Structure of LangGraph



- LangGraph creates Agentic workflows in a graph structure consisting of nodes and edges that share the state.
- A StateGraph is used to bring all of the objects comprising a workflow together.
- Use the same runnable protocol as LangChain.

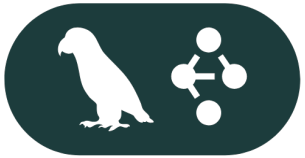
Key Takeaways

- LangGraph is a lower-level framework for implementing AI Agents.
- LangGraph uses a graph structure to create workflows for Agentic solutions.
- LangGraph nodes, which are connected by edges, operate on a state object to accomplish tasks.

Applied Generative AI

Agents in LangGraph: Components of LangGraph

The Main Components of LangGraph



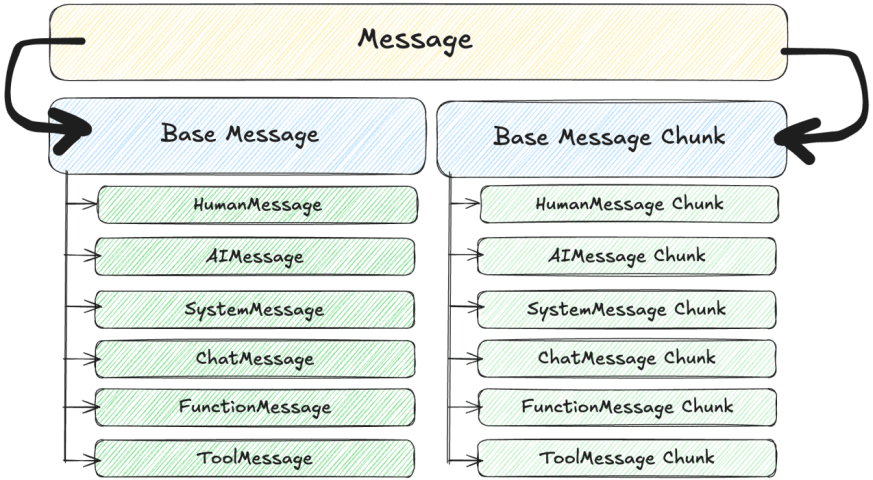
LangGraph

- Models
- Messages
- Tools
- Reducers
- Memory

The (Large Language) Model

- Language models (and other AI models) that generate responses, process prompts, and drive decision-making within the workflow.
- Typically encapsulated as a function node.

Messages



- Structured data (e.g., HumanMessage, AIMessage, ToolMessage) used to pass information between nodes and build conversational history.
- Often form a key part of the State.

Understanding the LangChain HumanMessage class. Lunary Blog.
https://lunary.ai/blog/langchain-humanmessage

Tools

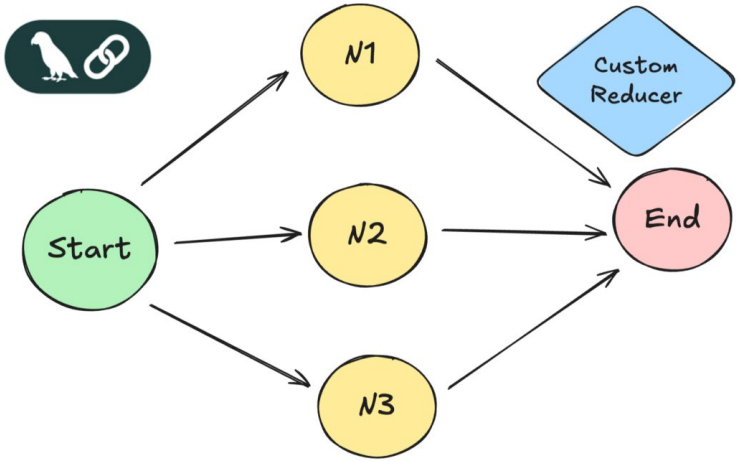
```python
def tool(a: type, b: type) -> type:
    """Description of the tool

    Args:
        a: first argument
        b: second argument
    """
    # Do some things

    return result
```

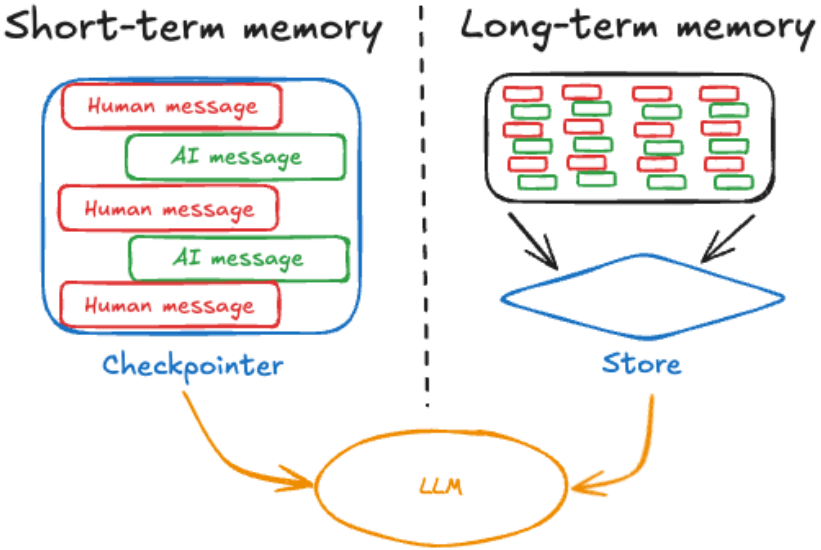
- External functions or API integrations that empower Agents to perform actions.
- Tool integrations are what make Agents useful.
- Many existing tools in LangChain.

Reducers



- Custom functions that merge state updates.
- Typically ensure that new messages or data are appended or combined properly rather than simply overwritten.

Memory



- Persistence layers that maintain state across turns and threads, supporting both short-term (session-specific) and long-term memory for richer context management.
- Remember across graph invocations.

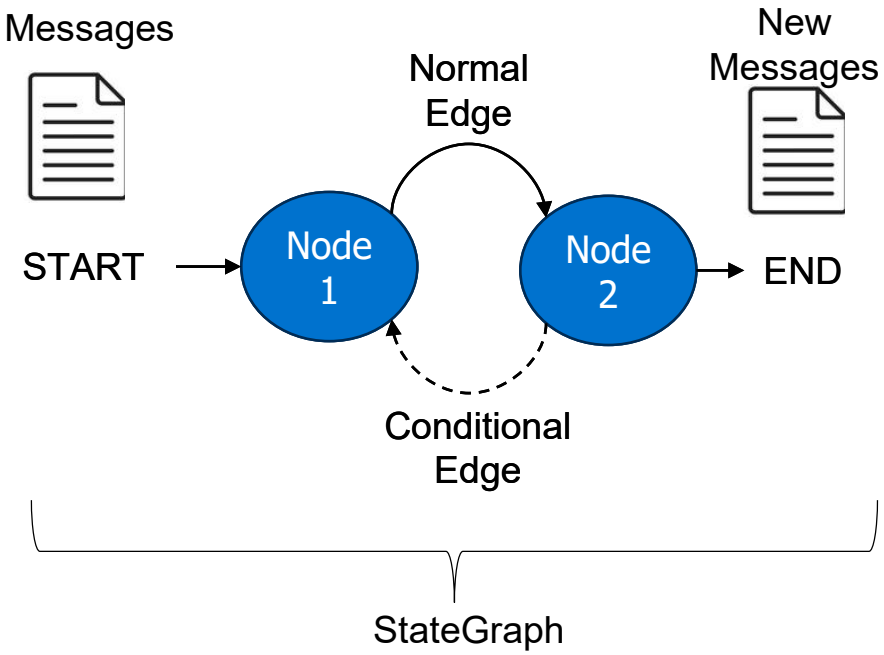
Key Takeaways

- LangGraph's core components (Models, Messages, Tools, Reducers, and Memory) collaborate to drive Agent workflows.
- Within LangGraph there are default and commonly used version of these components as well as the ability to create your own.
- Tools and the LLM are frequently the components that need designer specification. Other components can usually be the default.

Applied Generative AI

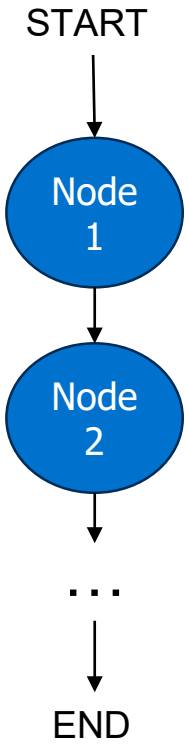
Agents in LangGraph: LangGraph Design Patterns

The Main Design Patterns of LangGraph



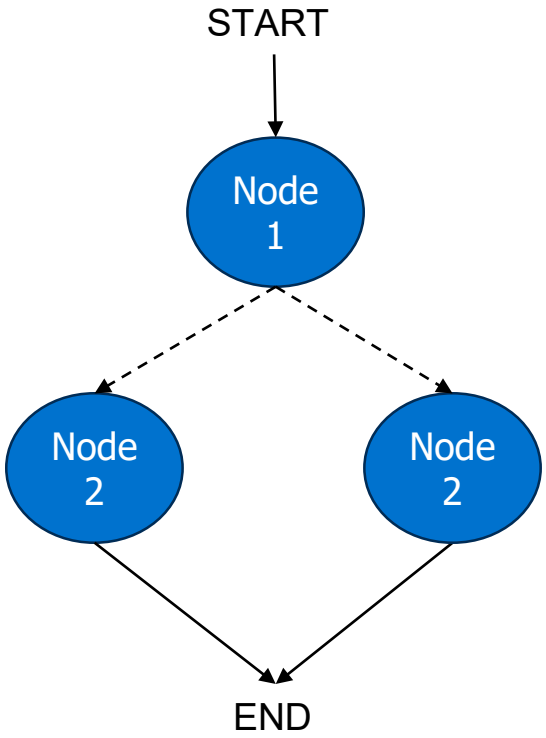
- Chain
- Router
- Agent
- Agent with Memory

The Chain Pattern



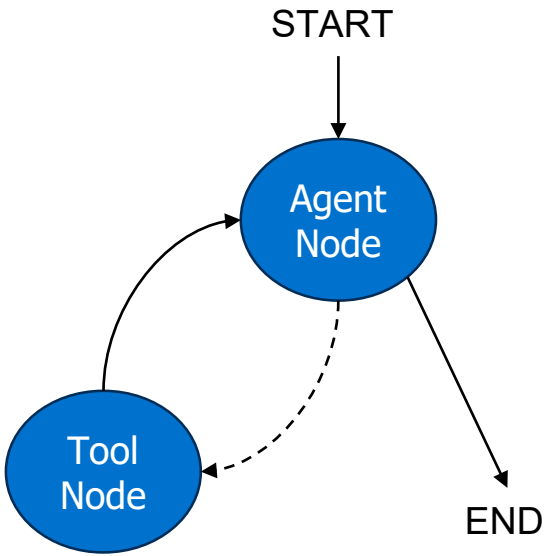
- A linear sequence of nodes where each node's output becomes the next node's input.
- Ideal for fixed workflows where tasks are executed in a predetermined order.
- Simplifies process design by isolating individual steps in the overall operation.

The Router Pattern



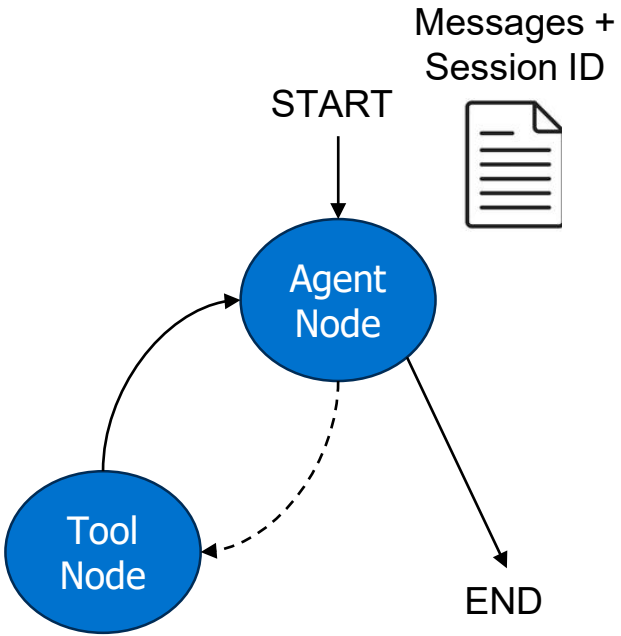
- Implements dynamic, conditional branching based on custom routing logic.
- Directs state to different nodes depending on input conditions or decision functions.
- Useful for complex decision-making workflows that require adaptable execution paths.

The Agent Pattern



- Encapsulates **autonomous behavior** by integrating LLM and tools into a single unit.
- Acts as an intelligent actor that processes inputs, executes actions, and responds dynamically.
- Typically implemented as a function node that orchestrates various tasks within a graph.

The Agent with Memory Pattern



- Extends the basic Agent pattern by incorporating persistent memory management.
- Retains context across multiple graph invocations (short-term and long-term).
- Ideal for conversational or iterative applications where past interactions influence future decisions.

Key Takeaways

- The Agent design patterns are the primary patterns for tool use. They allow for the execution of tools and the Agent to continue to use tools as necessary to answer the user prompt.
- Graphs can often be combinations of design patterns. For example, using a router with a planner Agent to different execution Agents that are Agent patterns.
- Increasing the amount of autonomy decreases the predictability of outcomes.