

LangGraph is a powerful orchestration framework designed to help you build complex, stateful, and multi-step AI workflows. By representing these workflows as a graph, it allows you to handle sophisticated logic like cycles, branches, and parallel execution with ease.

Core Building Blocks-

- **Graphs, Nodes, and Edges:** The structure is central to LangGraph. Nodes represent individual tasks (typically Python functions), while Edges define the flow of execution between these nodes. This setup enables you to visualize your AI workflow as a flowchart.
- **State:** This acts as a shared memory that persists and evolves throughout the entire workflow execution. Every node has access to this state, allowing it to read or make updates as the process moves forward.
- **Reducers:** When multiple nodes update the state, reducers determine *how* those updates are applied (e.g., whether to replace, add, or merge data). This is critical for preventing data loss in complex, multi-agent systems.
- **Execution Model:** LangGraph follows a message-passing approach. When a node executes, it makes a partial update to the state, which is then passed along the edges to the next node. The workflow runs in supersteps, allowing for concurrent parallel processing where multiple nodes might run simultaneously.

Common Workflow Patterns

LangGraph supports several standard design patterns to tackle different AI challenges:

- **Prompt Chaining:** Breaking a complex task into a sequence of smaller, manageable LLM calls.
- **Routing:** Dynamically deciding which path or tool to use based on the input query.
- **Parallelization:** Running multiple sub-tasks at once to improve efficiency, such as multi-angle content moderation.
- **Orchestrator-Worker:** A central LLM (orchestrator) determines tasks for specialized workers based on the user's specific request.
- **Evaluator-Optimizer:** An iterative loop where an LLM generates a solution, which is then critiqued and refined until it meets a specific quality threshold.