Contents

[Foundation of Agentic AI 1](#_Toc206446431)

[LangGraph Fundamentals 1](#_Toc206446432)

[Advanced LangGraph 2](#_Toc206446433)

[AI Agents 2](#_Toc206446434)

[Agentic RAG 2](#_Toc206446435)

[Productionization 2](#_Toc206446436)

# Foundation of Agentic AI

Agentic AI vs. AI Agents

Agentic AI vs. Generative AI

Agentic RAG

Traditional RAG vs. Agentic RAG

Top Frameworks for building Agentic AI

**Generative AI** refers to a class of artificial intelligence models that can create new content – such as text, images, audio, code, or video – that resembles human-created data.

Few Generative Model Examples with types.

LLMs based apps like Chat GPT

Diffusion models for images

Code generating LLMs like CodeLLama

TTS models like ElevenLabs

Video gen models like Sora

**Application Areas**

Creative and Business Writing

software Development

Customer Support

Education

Designing

# LangGraph Fundamentals

Building Graphs

State

Nodes

Edges

Conditional Edges

Looping

AI Workflows

**LangGraph**

**LangGraph** is an orchestration framework that enables you to build **stateful, multi-step,** and **event-driven** workflows using large language models (LLMs). It’s ideal for designing both **single-agent** and **multi-agent** agentic AI applications.

Think of LangGraph as **flowchart engine for LLMs** – you define the steps (nodes), how they’re connected(edges), and the logic the governs the transitions. LangGraph takes care of **state management, conditional branching, looping, pausing/resuming,** and **fault recovery** – features essential for building robust, production-grade AI systems.

* LangGraph is an orchestration framework for building intelligent, stateful, and multi-step LLM workflows.
* It enables advanced features like parallelism, loops, branching, memory, and resumability – making it ideal for agentic and production-grade AI applications.
* It models your logic as a graph of nodes (tasks) and edges (routing) instead of a linear chain.

**LangChain or LangGraph?**

* Use **LangChain** when you’re building **simple, linear workflows** – like a prompt chain, summarizer, or a basic retrieval system.
* Use **LangGraph** when your use case involves **complex, non-linear workflows** that need:
  + Conditional paths
  + Loops
  + Human-in-the-loop steps
  + Multi-agent coordination
  + Asynchronous or event-driven execution

**Should we Still use LangChain?**

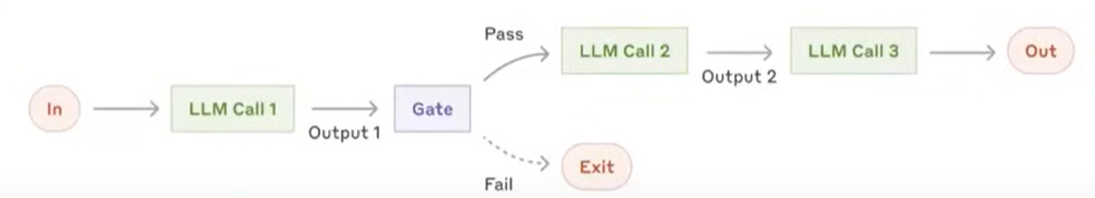
**Yes.** LangGraph is built **on top of LangChain** – it doesn’t replace it.

You’ll still use **LangChain components** like:

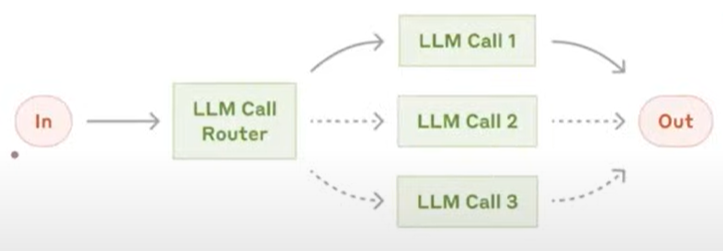
* ChatOpenAI (LLMs)
* PromptTemplate
* Retrievers
* DocumentLoaders
* Tools, etc.

LangGraph handles **workflow orchestration**, while LangChain provides the **building blocks** for each step in that workflow.

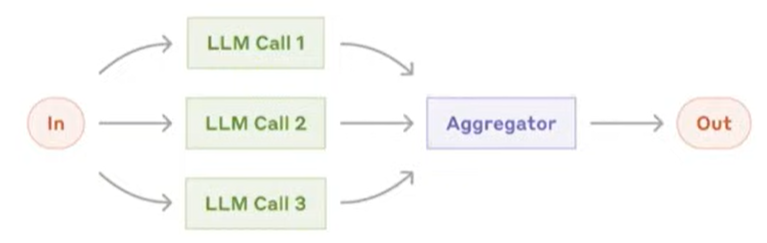
1. LLM Workflows
   1. Prompt Chaining



* 1. Routing



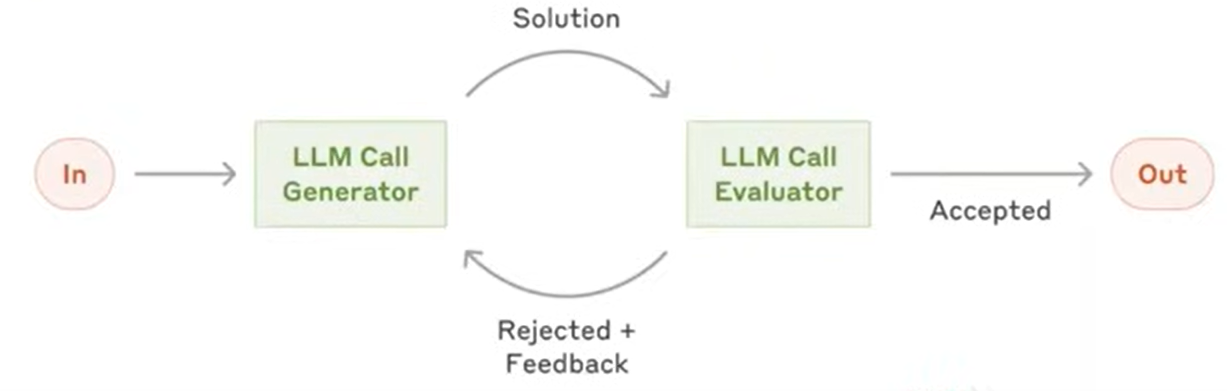
* 1. Parallelization



* 1. Orchestrator Workers



* 1. Evaluator Optimizer



1. Graphs, Nodes and Edges
2. State

In LangGraph, state is the shared memory that flows through your workflow – it holds all the data being passed between nodes are your graph runs.

1. Reducers

Reducers in LangGraph define how updates from nodes are applied to the shared state. Each key in the state can have its own reducer, which determines whether new data replaces, merges, or adds to the existing value.

1. LangGraph Execution Model (Inspired by Google Pragel)
   1. Graph Definition

You define:

* The state schema
* Nodes (functions that perform tasks)
* Edges (which node connects to which)
  1. Compilation

You call .compile on the StateGraph.

This checks the graph structure and prepares it for execution.

* 1. Invocation

You run the graph with .invoke(initial\_state).

LangGraph sends the initial state as a message to the entry node(s).

* 1. Super-Steps Begin

Execution proceeds in rounds.

# Advanced LangGraph

Persistence

memory

Human in the loop

Break point

Check pointers

Time travel

# AI Agents

Popular design patterns

Types of AI Agents:

React Pattern,

Reflection Design Pattern

Self ask with help

Planning

Multi agent systems

# Agentic RAG

C RAG

Self RAG

# Productionization

UI

Debugging

observability

LangSmith

Deploy