LangChain vs LangGraph: Pros, Cons, and Practical Considerations

Estimated Reading Time: 10 minutes

Introduction

Recent developments in AI have produced powerful frameworks for building applications around large language models (LLMs). LangChain (released 2022) and LangGraph (released 2023) are two such frameworks from LangChain Inc.

They take different approaches:

- LangChain uses a linear "chain" of components (prompts, models, tools).
- LangGraph uses a graph-based orchestration of stateful, multi-agent workflows.

In practice, LangChain is ideal for straightforward, sequential tasks (for example, a simple QA or RAG pipeline), whereas LangGraph is designed for complex, adaptive systems (for example, coordinating multiple AI agents, maintaining long-term context, or human-in-the-loop approval).

What is LangChain?

LangChain is an open-source framework for developing LLM-driven applications. It provides tools and APIs (in Python and JavaScript) to simplify building chatbots, virtual assistants, Retrieval-Augmented Generation (RAG) pipelines, and other LLM-based workflows. At its core, LangChain uses a "chain" or directed graph structure: you define a sequence of steps (prompts → model calls → outputs) that execute in order.

For example, a RAG workflow might chain: (1) retrieve relevant documents, (2) summarize, (3) generate an answer. The LangChain ecosystem includes prebuilt components for prompts, memory buffers, tools (such as search or calculator), and agents that can pick actions. It also integrates with dozens of LLM providers. LangChain's flexible, modular design has made it popular for quickly prototyping AI apps with minimal coding.

The LangChain framework is layered: a core of abstractions (chat models, vectors, tools), surrounded by higher-level chains and agents that implement workflows.

LangChain's architecture is **modular**. The base `langchain-core` defines interfaces for models, prompts, memory and tools. The main `langchain` package adds chains and agents that form the "cognitive architecture". Popular LLM providers (OpenAI, Google, etc.) have their own integration packages, making it easy to switch models. There is also a `langchain-community` package for third-party extensions. Overall, LangChain serves as a high-level orchestration layer for LLMs and data. It handles inputs/outputs and connects components, but it remains mostly stateless by default (conversation histories can be passed, but long-term memory must be explicitly managed).

What is LangGraph?

LangGraph is a newer framework (built by the same team) focused on stateful multi-agent orchestration. LangGraph is an extension of LangChain aimed at building robust and stateful multi-actor applications with LLMs by modeling steps as edges and nodes. In simple terms, while LangChain chains operations, LangGraph lets you build a graph of agents and tools. Each node in the graph can be an LLM call or an agent (with its own prompt, model, and tools), and edges define how data and control flow between them. This architecture natively supports loops, branches, and complex control flows.

LangGraph is explicitly designed for long-running, complex workflows. Its core benefits include durable execution (agents can pause and resume after failures), explicit human-in-the-loop controls, and persistent memory across sessions. For instance, LangGraph provides "comprehensive memory" that records both short-term reasoning steps and long-term facts. It even offers built-in inspection and rollback features so developers can "time-travel" through an agent's state to debug or adjust its course. In practice, LangGraph is used to build applications where many agents work together – for example, one agent might retrieve documents, another summarizes them, a third plans next steps, etc. These agents communicate using shared "state" (a kind of global memory) and can be arranged hierarchically or in parallel. Major companies (LinkedIn, Uber, Klarna, and so on) have begun using LangGraph to build sophisticated agentic applications in production.

Key Architectural Differences

Feature	LangChain	LangGraph
Type	LLM orchestration framework based on chains and agents.	AI agent orchestration framework based on stateful graphs.
Workflow Structure	Linear/DAG workflows (sequence of steps with no cycles). Good for "prompt → model → output" flows	Graph-based workflows (nodes and edges allow loops, branches, and dynamic transitions). Suited for complex flows.
State Management	Implicit/pass-through data. Chains carry inputs forward, but long-term state is limited by default	Explicit global state ("memory bank") that all agents access. State is persistently stored and updated at each step.
Task Complexity	Best for simple to medium tasks: chatbots, RAG pipelines, sequential reasoning.	Designed for complex, multi-step tasks and workflows that evolve over time (for example, multi-agent assistants).
Agents and Collaboration	Typically single-agent or linear chain; agents operate independently without inter-communication.	Multi-agent. Agents (nodes) can call each other using the graph, share memory, or be arranged hierarchically.

The above table summarizes the LangChain vs. LangGraph trade-offs.

Pros and Cons

Framework	Pros	Cons
LangChain	- Easy and quick to set up for common LLM tasks - Extensive community and prebuilt components (for example, QA chains, map-reduce, memory buffers)	- Not well-suited for long-running or highly interactive processes - Lacks built-in persistent memory and multi-agent

Framework	Pros	Cons
	- Excellent for RAG workflows and chatbots - Implicit chaining model requires minimal boilerplate code	orchestration - Workflows cannot natively loop or branch dynamically - Debugging is harder due to opaque state passing between steps
LangGraph	- Built for complexity and scale - Agents can run concurrently or sequentially with shared context - Supports durable execution (resume from point of failure) - Deep visibility into internal state and execution path (using LangSmith) - Human-in-the-loop support is first class - Ideal for orchestrating multi-step business processes	- More complex to learn and set up - Requires explicit definition of states, nodes, and edges - Slower to develop simple use cases compared to LangChain - Ecosystem is newer with fewer templates and extensions - Overhead may be unnecessary for simple tasks

When to use which framework?

Use Case	Use LangChain When…	Use LangGraph When…
Workflow Complexity	You have a clearly-defined, linear workflow.	You need complex workflows with branching logic or conditional steps.
Development Speed	You want to build something quickly—ideal for prototyping and MVPs.	You're building a production-grade system where reliability, traceability, and durability are essential.
Memory Requirements	Stateless or light memory needs (for example, current conversation only).	Long-term memory is needed across interactions or agents (for example, remembering context across sessions).
Interaction Style	Simple LLM tool use (for example, retrieval, transformation, response).	Multi-turn or human-in-the-loop interactions requiring persistent state and coordination.
System Design	Linear pipelines such as document Q&A, summarization, or format conversion.	Multi-agent architectures, process automation, or workflows with retries, dependencies, or approvals.
Team Collaboration	Individual developer exploring LLM capabilities quickly.	Teams designing modular, orchestrated systems with accountability and version control.

Conclusion

The LangChain and LangGraph frameworks represent two evolving approaches to building with LLMs. LangChain offers a simple, powerful abstraction for chaining prompts and tools in sequence, while LangGraph offers a flexible, stateful architecture for orchestrating complex agent workflows. Developers should choose between them based on the complexity and requirements of their project: use LangChain for straightforward pipelines and experimentation, and adopt LangGraph when you need durable, multi-agent orchestration and fine-grained control. As both frameworks grow, they will likely continue to influence each other. LangChain is already integrating more stateful features (for example, LangGraph memory), and LangGraph can leverage LangChain's components. The right choice depends on the use case at hand, and understanding the trade-offs above will help you select the best tool for your LLM application.

Note

LangChain is deprecating its legacy agent framework in favor of LangGraph, which offers enhanced flexibility and control for building intelligent agents. In LangGraph, the graph manages the agent's iterative cycles and tracks the scratchpad as messages within its state. The LangChain "agent" corresponds to the prompt and LLM setup you provide. Refer to LangChain and LangGraph's latest documentation for latest updates.

Author

Karan Goswami

Other Contributor(s)

Faranak Heidari