LangChain vs LangGraph vs LlamaIndex: When to Use Each

The short version

All three are frameworks for building LLM applications, but they solve different problems. LangChain is a general-purpose toolkit for chaining LLM calls with tools, memory, and data sources. LangGraph is a lower-level orchestration layer (built by the LangChain team) for controlling complex, stateful, possibly cyclic agent workflows as explicit graphs. LlamaIndex is focused on one thing: getting your data into and out of an LLM efficiently, i.e. retrieval-augmented generation (RAG).

LangChain

LangChain provides pre-built abstractions for prompts, chains, agents, memory, and integrations with hundreds of tools and vector stores. Its value is breadth: instead of writing glue code for every LLM provider, retriever, or tool, you compose existing building blocks.

Use it when: you want to move fast on a fairly linear pipeline (prompt -> LLM -> tool -> output) and you value ecosystem breadth over fine-grained control of execution flow.

Watch out for: the abstraction layers can make debugging harder, and very complex agent logic (loops, conditional branches, human-in-the-loop) gets awkward to express with plain chains.

LangGraph

LangGraph models your application as a graph of nodes and edges, where each node is a step (an LLM call, a tool call, a human approval step) and edges define transitions, including conditional and cyclic ones. State is explicit and persisted, which makes long-running, multi-step agents easier to

reason about, checkpoint, and resume.

Use it when: your workflow has loops, branching logic, needs to pause for human input, or needs durable state across steps (e.g. a multi-day approval workflow or a self-correcting agent that retries until a check passes).

Watch out for: more upfront design work than a simple chain; you're explicitly modeling control flow rather than getting it for free.

LlamaIndex

LlamaIndex specializes in data ingestion, indexing, and retrieval. It has strong tooling for chunking documents, building vector/graph/keyword indexes, and running retrieval strategies (hybrid search, query rewriting, sub-question decomposition) tuned specifically for RAG quality.

Use it when: your core problem is "answer questions accurately over a large private corpus of documents." It generally produces better retrieval quality out of the box than rolling your own with LangChain's retriever abstractions.

Watch out for: it's less suited to general agent orchestration or complex multi-tool workflows outside of retrieval; you'll often pair it with LangChain or LangGraph for the surrounding logic.

How they combine in practice

These tools aren't mutually exclusive. A common pattern: use LlamaIndex for ingesting and indexing documents and producing a high-quality retriever, wrap the overall agent logic and tool orchestration in LangGraph if the flow has loops or needs durable state, and fall back to LangChain's simpler chain abstractions for straightforward, linear steps that don't need graph-level control.

Quick decision guide

- Simple, linear LLM pipeline with lots of integrations needed fast -> LangChain
- Complex agent with branching, loops, or long-running state -> LangGraph
- Core problem is retrieval quality over a document corpus -> LlamaIndex
- Real-world system -> often a combination of the three

Note: this is a test article generated for evaluating the BookStack/n8n integration; treat the technical claims as a reasonable starting point rather than a fully vetted reference.

Revision #1
Created 21 July 2026 10:23:53 by N8N
Updated 21 July 2026 10:23:53 by N8N