

COURSE OUTLINE

LangChain & LangGraph — Build Stateful AI Agents, from Chatbot to Multi-Agent Workflow

An intensive, lab-driven program for developers building a tool-using, multi-agent assistant with LangChain and LangGraph

DURATION

3 days

FORMAT

Instructor-led, lab-driven, continuous capstone project

DELIVERY

Online Instructor Led Workshop · Onsite Program for your Teams

TRAINER

Muhammed Shakir

Why this course exists

Calling an LLM API from Python is straightforward. The work becomes repetitive—and then risky—when an application needs consistent instructions, conversation history, structured outputs, retrieval, tools, branching logic, memory that survives a restart, human approval, and a clear way to see what happened when an answer goes wrong.

This program gives developers a practical, build-first understanding of LangChain and LangGraph: participants grow one AI assistant from a simple LangChain chatbot into a stateful, tool-using, multi-agent LangGraph application that can pause for human approval, resume from saved state, and be traced and evaluated.

By the end of three days, participants will have built a working Learning Companion assistant, seen exactly which problems LangChain solves and which problems LangGraph solves, and used each only where it makes the application easier to read, extend, operate, and explain.

What makes this different

The application is visible from the first session. **This is an engineering room: each idea becomes a small, inspectable part of one working assistant.** You will:

- **Start with the real chat experience** — connect Open WebUI to a LangChain-backed, OpenAI-compatible chat endpoint in the first session and keep using it for three days.
- **Learn LangChain by composing real work** — prompts, messages, structured output, retrieval, and tool calling, each added to the same assistant rather than shown as isolated snippets.

- **Learn LangGraph by needing it** — move to explicit state, nodes, and conditional edges at the moment the application gains branches, loops, tools, and specialist agents.
- **Finish with production habits** — persistence, human-in-the-loop interrupts, streaming, tracing, and small evaluation sets that make an agent's behaviour inspectable rather than magical.

The course favours a clear mental model and one working project over a catalogue of framework features.

Skills You'll Walk Away With

- Explain the difference between a direct LLM API call, a LangChain application flow, and a LangGraph stateful workflow—and choose between them.
- Build reusable chat prompts, message histories, and runnable chains with LangChain Expression Language (LCEL).
- Return validated structured output from a model using schemas instead of parsing free text.
- Add a small retrieval (RAG) capability with document loading, chunking, embeddings, a vector store, and source-backed answers.
- Define tools, bind them to a chat model, and handle tool calls, tool errors, and argument validation safely.
- Model an application as a LangGraph: shared state, nodes, edges, conditional edges, reducers, bounded loops, and **START/END**.
- Build a tool-using ReAct-style agent in LangGraph and explain every step it takes.
- Add persistence with checkpointers, thread-scoped conversation memory, and resumable runs.
- Pause a graph for human approval with interrupts, then resume, edit, or reject the pending action.
- Coordinate specialist agents with a supervisor/router pattern and explicit hand-offs.
- Stream tokens and intermediate steps to a chat client, trace runs, and evaluate the assistant against a small test set.

Who should attend

This program is for developers and technical professionals who want a practical, end-to-end path into LLM application and agent frameworks.

- **Python developers** — moving from one-off LLM API calls to a maintainable AI application.
- **Backend developers** — adding retrieval, tools, and agent workflows to services and products.
- **Frontend and full-stack developers** — understanding what sits behind an AI chat interface and how its flow is composed.
- **AI application engineers** — learning LangGraph's state, persistence, and human-in-the-loop model for real agent work.
- **Technical leads and solution architects** — learning where LangChain and LangGraph help, and where ordinary Python remains sufficient.

Prerequisites: comfort with Python, virtual environments or UV, environment variables, and basic HTTP/API calls. Participants need access to an LLM API key (Grok or an equivalent provider). Prior LangChain, LangGraph, AI-agent, or machine-learning experience is not required.

Course Outline

Day 1 — Build a Clear LangChain Chatbot Foundation

Connect a real chat interface first, then add only the LangChain components that make the application flow clearer, more reusable, and more reliable.

Session 1 — Establish the Chatbot and Its Open WebUI Contract

- LLM application in one flow: Open WebUI ' chat endpoint ' instructions and messages ' model API ' streamed answer
- A direct Python call as a brief comparison, and the responsibilities that accumulate as a chatbot grows
- What LangChain is and is not: composable application abstractions, not a replacement for Python, the provider, or sound design
- The course capstone: one **Learning Companion** assistant that grows across all three days
- *Hands-on:* Create the project, add a FastAPI app with GET /health, GET /v1/models, and an OpenAI-compatible POST /v1/chat/completions; configure the chat model from environment variables; connect Open WebUI and run the first real conversation.

Session 2 — Compose with LangChain Building Blocks

- Chat models, messages, prompt templates, output parsers, and runnable chains (LCEL)
- Separating stable system behaviour from changing user input
- Conversation history as messages: pass a short transcript deliberately with each request
- Streaming responses through the chain to the chat client
- *Hands-on:* Refactor into a prompt ' model ' parser chain with visible message history and streaming; log what is actually sent to the model.

Session 3 — Structured Output and Two Specialist Behaviours

- Structured output with schemas: getting validated objects instead of parsing free text
- Instructions versus user content; consistent answer shapes
- Two specialist behaviours over the same model: general chat and study helper
- Why a growing if/else routing block is the bridge to LangGraph
- *Hands-on:* Add a study-helper chain that returns a structured explanation plus a next learning step, and a simple, visible routing rule between the two behaviours.

Day 1 checkpoint — a working LangChain chatbot in Open WebUI with reusable prompts, short conversation history, streaming, structured output, and two inspectable behaviours.

Day 2 — Retrieval, Tools, and Your First LangGraph

Give the assistant knowledge and actions with LangChain, then move its control flow into an explicit LangGraph.

Session 4 — Add Retrieval (RAG) the LangChain Way

- Document loaders, text splitters, embeddings, and vector stores—concepts before platform choice
- Retrievers and the retrieval chain: question ' relevant passages ' grounded answer with sources
- When retrieval helps and when it does not; handling unanswerable questions
- *Hands-on:* Load a small set of course notes, build a retriever, and add a source-backed “ask my notes” capability to the assistant.

Session 5 — Tools and Tool Calling

- Defining tools with typed arguments; binding tools to a chat model
- The tool-calling loop: model proposes ' application validates and executes ' result returns to the model
- Tool errors, timeouts, argument validation, and least-privilege tool design
- *Hands-on:* Add two safe tools—such as a quiz generator and a read-only progress lookup—and inspect every tool call and result.

Session 6 — Model the Assistant as a LangGraph

- Why LangGraph: explicit state and routing once a workflow gains branches, tools, retries, and review steps
- Graph vocabulary in one working example: state, reducers, nodes, edges, conditional edges, START, and END
- Nodes are focused work, not automatically agents; routers can be ordinary application logic
- A bounded review loop: reviewer approves or sends a draft once to an improver
- *Hands-on:* Rebuild the assistant as a graph: router ' general / study / RAG specialist ' reviewer ' (one improvement) ' end; print the route, review decision, and final state.

Day 2 checkpoint — the same assistant now answers from its notes, uses tools safely, and runs as an explicit LangGraph whose path can be drawn and explained.

Day 3 — Agents, Memory, Human-in-the-Loop, and Production Habits

Turn the graph into a dependable agent application: tool-using agents, durable state, human approval, multi-agent coordination, and evidence it works.

Session 7 — Tool-Using Agents in LangGraph

- The ReAct pattern as a graph: an agent node, a tool node, and a conditional edge that loops until done
- Prebuilt agent helpers versus a hand-built agent graph—what each hides and when to use which

- Step limits, recursion limits, and stop conditions
- *Hands-on:* Build a study-planning agent that calls tools in a bounded loop; compare a hand-built graph with the prebuilt version.

Session 8 — Persistence, Memory, and Human-in-the-Loop

- Checkpointers and threads: conversation memory that survives restarts; resuming and replaying runs
- Short-term (thread) memory versus long-term (cross-thread) memory, and what should not be remembered
- Interrupts for human approval: pause before an action, then approve, edit, or reject and resume
- *Hands-on:* Add a checkpointer, per-user threads, and an approval step before the agent saves a study plan; resume the run after approval from Open WebUI.

Session 9 — Multi-Agent Coordination

- Supervisor/router patterns, specialist agents, and explicit hand-offs
- Subgraphs for reusable specialist workflows; shared versus private state
- When multiple agents help—and when a single agent or a fixed workflow is better
- *Hands-on:* Introduce a supervisor that delegates to tutor, quiz, and planner agents, each a subgraph with its own tools.

Session 10 — Stream, Trace, Evaluate, and Ship

- Streaming tokens and intermediate steps to the chat client
- Tracing graph runs (e.g. with LangSmith) to see prompts, tool calls, state, latency, and cost
- Small evaluation sets: routing accuracy, answer quality, groundedness, and tool-use correctness
- Packaging and serving a graph behind the existing API; configuration and secrets
- *Hands-on:* Trace a full multi-agent run, write a 10-case evaluation set, fix one failing case, and present the final assistant.

Day 3 checkpoint — a working, explainable multi-agent assistant with retrieval, tools, durable memory, human approval, streaming, tracing, and a small evaluation set.

Capstone project

Participants incrementally build a **Learning Companion** assistant over three days. It starts as a LangChain chatbot in Open WebUI and ends as a LangGraph multi-agent application:

- **Day 1 — LangChain foundation:** an OpenAI-compatible chat endpoint, reusable prompts and messages, streaming, structured output, and two specialist behaviours.
- **Day 2 — Knowledge, actions, and a graph:** source-backed retrieval over course notes, safe tool calling, and an explicit LangGraph with routing and a bounded review loop.
-

Day 3 — Agent application: a tool-using agent loop, checkpointed memory, human approval through interrupts, a supervisor coordinating specialist subgraphs, tracing, and an evaluation set.

The application builds in one direction. Each session extends the same endpoint, Open WebUI connection, configuration, and architecture note; participants never throw their project away for a new demo. The distinction stays clear throughout: **LangChain composes model work; LangGraph controls the state and route through the work.**

Out of scope: model training or fine-tuning, production-scale vector database operations, enterprise authentication and authorization, and open-ended autonomous agents with unrestricted tools. These are natural follow-on topics once the foundations are solid.

A note on delivery

Each day balances explanation, demonstration, guided implementation, inspection, and troubleshooting, with two tea breaks and one lunch break. Participants use Open WebUI as the chat client from the first session, so every capability is visible as a real interaction rather than only a terminal demonstration.

The sequence is deliberate: first make a LangChain application work end to end; then give it knowledge and tools; then make its control flow explicit as a graph; finally add the agent, memory, approval, coordination, and evaluation discipline that real applications need. Participants leave able to explain not only how to use LangChain and LangGraph, but why their assistant uses each one where it does.

Ready to build stateful AI agents with LangChain and LangGraph?

Enquire about a private cohort or a tailored delivery for your developer team.