

warning "Backend update (October 2026)" GitHub Models inference was retired on July 30, 2026. Any references below are retained only for historical context while this lab is migrated. Use Lab 013 ? Free LLM backends to configure Foundry Local or Ollama before continuing.

Lab 029: LangChain & LangGraph Basics

Level: L200 Path: ? Pro Code Time: ~60 min ? Cost: Free ? GitHub Models free tier

What You'll Learn

Build a conversational agent with LangChain and a tool-calling loop Model multi-step agent logic as a LangGraph state graph Understand the difference between LangChain chains and LangGraph graphs Add conditional routing: when to call a tool vs. return an answer Persist conversation state with LangGraph checkpoints

Introduction

LangChain is one of the most popular open-source frameworks for building LLM-powered applications. LangGraph extends it with explicit state machines ? graphs where nodes are functions and edges are transitions.

When to use each:

| LangChain | LangGraph | Best for | Linear pipelines, RAG chains, simple agents | Complex multi-step agents, branching logic, cycles | State | Implicit (passed through chain) | Explicit (typed state dict) | Loops | Not native | First-class support | Visibility | Chain logs | Graph execution traces |

In this lab we build the same OutdoorGear shopping assistant two ways: first with LangChain (simpler), then with LangGraph (more explicit control).

Prerequisites

[diagram or code sample]

No Azure subscription needed ? we use GitHub Models' OpenAI-compatible endpoint:

[diagram or code sample]

Part 1: LangChain Agent

Step 1: Tools

[diagram or code sample]

Step 2: LangChain Agent with Tool Calling

[diagram or code sample]

Run it: [diagram or code sample]

You should see the agent call `search_products`, inspect a result, then provide a recommendation.

Part 2: LangGraph Agent

LangGraph models the agent as a state machine. This makes the logic explicit and testable.

Step 3: Define the graph state

[diagram or code sample]

Step 4: Define graph nodes

[diagram or code sample]

Step 5: Build and run the graph

[diagram or code sample]

Part 3: Add Persistent Memory (Checkpoint)

LangGraph can persist state between runs using a checkpoint ? this is how you build multi-turn agents that remember conversations:

[diagram or code sample]

? Knowledge Check

question "1. What is the main advantage of LangGraph over a simple LangChain agent?"

LangGraph uses an explicit state machine (graph with nodes and edges) to model agent logic. This makes branching, looping, and conditional routing first-class citizens ? visible, testable, and debuggable. A LangChain agent hides the control flow inside the framework.

question "2. What does the `add_messages` reducer do in LangGraph state?" `add_messages` is a reducer function that tells LangGraph how to update the `messages` field: it appends new messages instead of replacing the whole list. Without it, each node return would overwrite the message history rather than adding to it.

question "3. How does LangGraph checkpointing enable multi-turn conversations?" A checkpointer persists the graph state (all messages) to storage (memory, Redis, PostgreSQL) keyed by a `thread_id`. When you invoke the agent with the same `thread_id`, LangGraph loads the previous state and continues from where it left off ? the agent "remembers" prior turns without you managing history manually.

Summary

Concept | LangChain | LangGraph | Structure | Linear chain | Directed graph (nodes + edges) | Loops | Not native | `graph.add_edge("tools", "llm")` | Branching | Limited | `add_conditional_edges()` | State | Implicit | Explicit TypedDict | Memory | Manual | MemorySaver / PostgresSaver | Debugging | Chain logs | Full graph execution trace |

Next Steps

Full function-calling deep dive: ? Lab 018 ? Function Calling & Tool Use Multi-agent with Semantic Kernel: ? Lab 034 ? Multi-Agent Systems Production AutoGen agents: ? Lab 040 ? Production Multi-Agent

Supporting kit

The site build generates stable PDF and ZIP downloads. The ZIP contains the lab guide and every supporting file declared for this lab.

[Download generated PDF](#) [Download generated ZIP](#)

Supporting file: [CHEATSHEET.md](#)

Supporting files: [Lab 029: LangChain & LangGraph Basics](#)

Start with the documented local or free path. Record assumptions and validation evidence.

Keep hosted services optional unless the lab marks them required.

Supporting file: [README.md](#)

[Lab 029: LangChain & LangGraph Basics](#)

Use these files with the lab guide and generated kit.