APPLIED AND GENERATIVE AI

Building Multi-Agent Systems

Level: Advanced • 2 days (expandable to 3) • Virtual, In-person

Overview

Multiple agents working together can tackle problems a single agent cannot. They can also turn a simple task into a slow, expensive, hard-to-debug tangle. The difference is design, and that is what this course is about: building multi-agent systems that are worth the added complexity.

This is a hands-on, advanced course. It opens with the most important skill, judgment, because the honest starting question is whether you need multiple agents at all, and many problems are better served by one good agent. Once we can tell the difference, we build up deliberately: the anatomy of a multi-agent system, then the orchestrator-worker pattern that carries most real designs, then the harder problems of communication, shared state, and running agents in parallel. We finish on reliability, the failure modes that are unique to multi-agent systems and how to contain them. Rather than survey every framework, we build a working system and go deep on the patterns and the discipline that make it dependable. Every module ends with hands-on work and builds on the one before.

Who Should Attend

- Developers who already build single agents and want to coordinate several
- Engineers designing agentic systems for non-trivial, multi-step problems
- Technical leads deciding where multi-agent designs are and are not worth it

Prerequisites

- Comfortable building a single AI agent, for example from *Building Agentic AI with the Model Context Protocol* or equivalent experience
- Working proficiency in a programming language such as Python or TypeScript
- Familiar with LLM APIs and tool or function calling

What You Will Learn

- Judge honestly when a problem needs multiple agents rather than one
- Describe the anatomy of a multi-agent system: roles, communication, and topology
- Build an orchestrator-worker system with a lead agent delegating to workers
- Pass context between agents with shared state and clean handoffs
- Run agents in parallel and aggregate their results
- Contain the failure modes unique to multi-agent systems and observe the whole

Course Outline

Day one: foundations and the core pattern

- When to Use Multiple Agents
  - A single strong agent versus a team: the honest tradeoff
  - What multi-agent adds: latency, cost, complexity, and failure surface

  - The signs a problem genuinely needs multiple agents
  - Lab: take a problem and decide, with reasons, single versus multi-agent
- Anatomy of a Multi-Agent System
  - Roles and responsibilities: one clear job per agent
  - Communication: how agents exchange information
  - Topologies: orchestrator-worker, pipeline, and others
  - Lab: map a multi-agent design on paper before building it
- The Orchestrator-Worker Pattern
  - A lead agent that plans and delegates to workers
  - Defining worker agents and the contracts between them
  - Lab: build an orchestrator that delegates to two worker agents

Day two: coordination, parallelism, and reliability

- Communication and Shared State
  - Passing context between agents without losing or duplicating it
  - Shared memory and handoffs
  - Keeping agents from working at cross-purposes
  - Lab: add shared state and clean handoffs to your system
- Parallelism and Aggregation
  - Running agents concurrently
  - Splitting work and combining results
  - Lab: parallelize part of your system and aggregate the output
- Making Multi-Agent Systems Reliable
  - Failure modes unique to multi-agent: cascades, loops, and runaway cost
  - Guardrails, timeouts, and budgets
  - Observability, and evaluating a multi-agent system as a whole
  - Lab: instrument your system, then inject and recover from a failure

Extended Version

The three-day version keeps the same gradient and adds range and rigor:

- Additional patterns, such as debate and critique, and hierarchical teams
- Framework options, and when to use one rather than roll your own
- Human-in-the-loop checkpoints in multi-agent workflows
- A capstone that designs and builds a multi-agent system for a realistic task, with reliability and observability built in