

COURSE OUTLINE

Agentic Workflows & Multi-Agent Systems — Build, Evaluate, and Operate Multi-Agent Systems in Production

An advanced program for AI engineers and technical developers

DURATION

FORMAT

3 days

Instructor-led, hands-on, lab-driven, project-based

DELIVERY

TRAINER

Online Instructor Led Workshop · Onsite Program for your Teams

Muhammed Shakir

Why this course exists

This program turns agentic AI from an impressive demo into an engineered system: learners design, build, evaluate, secure, and operate single-agent and multi-agent workflows in production-oriented conditions.

By the end, participants will understand when an agent is justified, how planning, tools, memory, state, and orchestration fit together, and how to prove that a multi-agent system is useful, safe, observable, and within its cost and latency budgets.

The course culminates in a payment-exception investigation service built from synthetic data: a supervisor coordinates specialist agents, an Agentic RAG system grounds the investigation in policy and runbooks, an MCP-connected ledger provides controlled data access, a critic reviews the result, and a human approval gate protects irreversible actions.

What makes this different

Most agent courses stop at a chatbot with a tool or a successful demo run. **This is a production-minded engineering room: every major design choice is measured against quality, cost, latency, safety, and operational evidence.** You will:

- **Learn agents in plain engineering language** — connect reasoning loops, state, memory, tools, planning, and handoffs to familiar software concepts such as workflows, services, state machines, APIs, retries, logs, and control points.

- **Choose the smallest architecture that works** — use a decision rubric to determine when one agent is enough, when multiple agents earn their coordination cost, and when an agent should not be used at all.
- **Build with inspectable state and controlled actions** — use LangGraph, tool calling, MCP, structured outputs, approval gates, and explicit termination conditions rather than relying on opaque autonomous behavior.
- **Measure the system instead of trusting the demo** — establish a baseline evaluation set early, then compare architecture variants on task quality, trajectory, cost, latency, tool-use accuracy, and safety.
- **Operate what you build** — instrument the workflow, version prompts, protect data, govern tools, deploy the service, and prepare an on-call view of failures and spend.

Every concept is connected to a running system, so participants leave knowing not only how to build an agentic workflow, but also how to decide whether it should exist and how to keep it trustworthy after launch.

Skills You'll Walk Away With

- Explain the difference between an LLM, chatbot, RAG application, agent, and multi-agent system.
- Identify the four core building blocks of an agentic system: model, memory/state, tools, and planning.
- Choose between single-agent, supervisor/worker, peer-to-peer, and handoff architectures using quality, cost, latency, and complexity trade-offs.
- Design bounded planning and reasoning loops with task decomposition, re-planning, retry policies, step budgets, stop conditions, and loop detection.
- Model agent state explicitly with LangGraph, including nodes, edges, cycles, reducers, checkpointing, persistence, and resume-after-failure behavior.
- Build reliable tool-calling workflows with validated schemas, useful tool descriptions, error contracts, permissions, and safe failure handling.
- Connect agents to external capabilities through MCP servers, tools, resources, prompts, transports, and governed client workflows.
- Build Agentic RAG with query processing, multi-hop retrieval, citations, refusal behavior, memory, and separate retrieval and generation evaluation.
- Orchestrate multiple agents through supervision, routing, parallel branches, aggregation, conflict resolution, fallback, and human approval gates.
- Evaluate both outcomes and trajectories using representative, edge, and adversarial cases, traces, spans, feedback, scorecards, and A/B comparisons.
- Apply prompt-injection defenses, input sanitization, output validation, structured-output contracts, least privilege, data boundaries, and red-team testing.
- Deploy and operate an agentic service with FastAPI, LangGraph, containers, Kubernetes, Open-Telemetry, metrics, structured logs, cost dashboards, and alerting.

Who should attend

This program is for experienced technical professionals who want to move from experimenting with agents to engineering multi-step AI systems that can be evaluated and operated responsibly.

- **AI and machine-learning engineers** — building production LLM applications and agentic workflows.
- **Backend and platform developers** — integrating models, tools, services, data, and APIs into dependable systems.
- **Solution and system architects** — making architectural decisions about autonomy, orchestration, governance, and deployment.
- **DevOps, SRE, and platform engineers** — deploying, instrumenting, scaling, and supporting agentic services.
- **Technical leads and engineering managers** — evaluating whether multi-agent complexity is justified for a business workflow.

Prerequisites: practical programming experience, familiarity with HTTP APIs, Git, Python, JSON, and basic container or cloud concepts. Prior exposure to LLM APIs, prompts, embeddings, or RAG is helpful. Advanced mathematics is not required; the program focuses on system behavior, engineering trade-offs, and measurable outcomes.

Course Outline

Day 1 — The Agent Loop: Reasoning, Memory & State

Build the mental model for agents, compare planning patterns, and make state explicit before adding orchestration complexity.

Module 1 • Agents vs. Multi-Agent Systems

- LLM vs. agent — reasoning loops, action loops, autonomy; agents vs. chatbots vs. RAG
- The four building blocks: LLM (brain) · Memory (state) · Tools (actions) · Planning
- Patterns: single agent, multi-agent, supervisor/worker — and the coordination tax of each extra agent
- The decision rubric you take away: when one agent is enough, when to split, when to build neither
- LLM limitations that create the need for agents in the first place
- Fast on-ramp: LangChain 1.x essentials and `create_agent` — enough to build, not a tour

Module 2 • Agentic Planning & Reasoning

- Architectures compared on identical tasks: Chain-of-Thought, ReAct, Tree-of-Thought, Reflection
- Task decomposition into sub-goals an agent can finish; dependency-aware ordering
- Re-planning vs. blind retry — transient failure or wrong plan?
- Stop conditions, step budgets and loop detection — the most common production failure

- Where each pattern earns its cost: quality vs. tokens vs. latency, measured not asserted

Module 3 · Memory, State & the LangGraph Substrate

- Memory types — short-term (conversation), long-term (persistent), episodic (experience)
- Buffer vs. summary memory; compaction, and when a long session degrades the agent
- Perception: turning raw tool and environment output into observations the model can act on — structured vs. raw, and what the agent cannot see
- LangGraph: StateGraph, nodes, edges, cycles — state as an explicit, inspectable object
- State reducers and checkpointing — persistence, resume after restart, and the audit trail
- Shared vs. private memory across agents, and context poisoning when one agent's error propagates

Day 1 Labs — LLM vs. agent · agent building blocks · Chain-of-Thought · ReAct pattern · Tree-of-Thought · Reflection · conversation memory · memory strategies · session management · first StateGraph · state reducers · checkpointing · build your baseline eval set & pass rate — the number every later milestone is measured against

Day 2 — Tools, MCP, Orchestration & Agentic RAG

Give agents controlled access to the outside world, then measure whether collaboration and retrieval justify their cost.

Module 4 · Tool Calling & MCP [MCP]

- Why tools matter; custom tool development, multi-tool orchestration, function-calling agents
- Tool descriptions are instructions — a measured lab moves accuracy on description quality alone
- MCP architecture: servers, tools, resources, prompts, transports; why a protocol beats bespoke integrations
- MCP client configuration and workflows; bridging MCP servers into LangChain agents
- Enterprise MCP use cases and ecosystem discovery
- Injection arriving through tool results; what a tool must return when it fails

Module 5 · Multi-Agent Collaboration & Orchestration

- Supervisor/worker — rule-based and LLM-routed — plus peer-to-peer and agent handoffs
- Task decomposition and distribution; aggregation and conflict resolution between agents
- Parallel execution, multi-branch graphs, conditional routing, error handling, retry and backoff
- Cost of collaboration: a 5-agent system runs 3–10× the tokens of one agent — per-agent model routing, cheap model routes, strong model reasons
- Human-in-the-loop as an orchestration *mechanism*: interrupt points, approval gates, timeout, escalation
- Measure the graph against your own Day 1 single agent — quality, cost, latency; and when the honest answer is "don't"
- Production patterns: specialisation, fallback, termination and runaway-loop control

Module 6 · Agentic RAG

- From RAG pipeline to retrieval as a tool the agent chooses to call, repeatedly, with intent
- Embeddings, ChromaDB collections and metadata, document loading, chunking, similarity-search strategy
- Query processing, multi-hop retrieval, and re-querying when the first result is inadequate
- Response generation with citations; grounding discipline and refusing when unsupported
- RAG-powered agents with memory — the retriever inside a conversational, tool-using agent
- Evaluating retrieval separately from generation — faithfulness vs. relevance

Day 2 Labs — tool basics · tool-calling accuracy measurement · multi-tool agent · MCP client config · MCP workflows · LangChain–MCP bridge · supervisor/worker · LLM-routed supervisor · peer-to-peer · agent handoffs · task decomposition · aggregation & conflict resolution · parallel execution · human-in-the-loop · multi-agent vs. single-agent scorecard (quality · cost · latency) · RAG chain · RAG with citations

Day 3 — Evaluation, Safety & AgentOps

Turn the workflow into a service that can be tested, defended, deployed, observed, and supported in production.

Module 7 · Multi-Agent System Evaluation

- Why demo-driven development fails here: same input, twice, two different runs
- Extending the Day 1 eval set — representative, edge and adversarial cases; outcome vs. trajectory assertions
- Tracing agent reasoning — traces, spans, generations; tool-call visibility; latency breakdown
- Scoring in LangFuse: feedback collection, scores, A/B testing
- Prompt management and versioning as an evaluated, rolled-out artifact, not an edit in place
- Cost analysis: tokens and cost per step, per agent, per run — and cost/latency as release gates, not reports
- Locating the failure: which agent, tool or retrieval step actually caused it

Module 8 · Safety & Guardrails

- Prompt-injection detection and input sanitisation; jailbreak resistance; red-teaming your own system
- Output validation and structured-output contracts enforced between every hop
- Guardrails integration into the agent graph — where a check belongs and what it costs
- MCP tool governance — scopes, least privilege, blast radius; what an agent may call unattended
- Human-in-the-loop as a safety *control*: approval gates before any write or irreversible action
- Safety monitoring: what to alarm on for a system that can be wrong without erroring
- Data boundaries — PII, secrets, and what must never enter a prompt, a trace or a vector store

Module 9 · Deployment & AgentOps

- FastAPI + LangGraph as a service: async handling, streaming responses, error handling, validation
- Health checks and readiness probes; secrets management; the production readiness checklist
- Containerise and deploy to Kubernetes: Deployments, Services, Ingress, Secrets, HPA, resource limits
- OpenTelemetry — the three pillars, custom spans, OTLP and the collector
- Production observability end to end: traces, metrics, structured logging, cost dashboards, alerting
- Scaling, debugging and the on-call runbook for an agentic service

Day 3 Labs — tracing agents · feedback & evaluation · prompt management · cost analysis · prompt-injection detection · output validation · guardrails integration · red-team testing · MCP security & governance · FastAPI + LangGraph · health probes · OTel setup & instrumentation · production observability

Capstone — Payment-Exception Investigation Service

Participants build and defend a production-minded investigation service using synthetic systems and data:

- Supervisor agent coordinating specialist agents
- Agentic RAG over payment policy and operational runbooks
- Ledger queried through a governed MCP server
- Critic agent reviewing the investigation result
- Human approval gate before any write or irreversible action
- FastAPI endpoint with validation and streaming or asynchronous handling
- Containerized deployment to Kubernetes
- Langfuse instrumentation for traces, feedback, and evaluation
- Fixed cost and latency budget
- Acceptance based on clearing the Day 1 evaluation set, not on a successful demo run

What participants take away

- A one-agent-or-many decision rubric
- A reference supervisor/worker implementation
- Their own MCP server and tool-governance scopes
- An Agentive RAG retriever that cites evidence and refuses unsupported answers
- A reusable evaluation harness and quality/cost/latency scorecard
- Guardrail, red-team, and approval-gate patterns
- Kubernetes deployment and observability manifests
- A production-oriented capstone that can be inspected, measured, defended, and extended

A note on delivery

This is a build-and-measure program, not a slide-only overview of autonomous AI. Each day combines focused instruction, guided demonstrations, individual or pair labs, review, troubleshooting, and evidence-based comparison. The through-line is a payment-exception investigation service that evolves from a bounded agent into a supervised, tool-using, retrieval-grounded, evaluated, guarded, observable service.

Participants leave with working system patterns and reusable engineering artifacts—not just a list of agent frameworks or a demo that works once.