INSTRUCTOR-LED · HANDS-ON · FIVE DAYS

RAG & Agentic AI Engineering

Five days taking working software engineers from their first LLM call to a production-shaped AI agent — with a full day on RAG (Retrieval-Augmented Generation) at the centre, and evaluation, guardrails and cost control at the end. Built on LangGraph or AWS Bedrock.

DURATION	HANDS-ON	CORE	STACK
5 days · 1,950 min	**~75% of contact time**	**RAG · Agents · Evals · MCP**	**LangGraph or AWS Bedrock**

TECHNOLOGIES AND TOPICS COVERED

RAG (Retrieval-Augmented Generation) · **Agentic AI** · **AI agents** · **LangGraph** · **LangChain** · **AWS Bedrock** · **MCP (Model Context Protocol)** · LLM (Large Language Model) · GenAI · Amazon Bedrock AgentCore · Bedrock Knowledge Bases · Vector databases · Embeddings · Semantic search · Hybrid search · Reranking · Chunking strategies · Golden sets · Recall@k · MRR · Tool calling / function calling · Structured output · Prompt engineering · Context engineering · Multi-agent systems · Supervisor and router patterns · Plan-and-execute · Human-in-the-loop (HITL) · Checkpointing and persistence · LLM evaluations (evals) · LLM-as-a-judge · Guardrails · Prompt injection defence · Observability and tracing · Token cost optimisation · Semantic caching · LLMOps · Python · FastAPI · CI/CD for AI

What each engineer can do afterwards

- ✓ Write a tool-using AI agent from first principles, and say what a framework adds.
- ✓ Get typed, validated structured output from an LLM, and handle it when it fails.
- ✓ Run an **eval suite in CI** that fails the build on a quality regression.
- ✓ Find and close a prompt-injection path in their own build.
- ✓ Build a **RAG pipeline and state its retrieval score** against a golden set.
- ✓ Model an agent as a state graph with branching, persistence and HITL approval gates.
- ✓ Trace a run, attribute token cost, and cut latency with evidence.
- ✓ Expose an internal system to any agent as an MCP server.

EACH PARTICIPANT LEAVES WITH

- A **working AI agent** written without a framework, plus the LangGraph or Bedrock version beside it.
- A **RAG pipeline with a measured retrieval score** and the golden set behind it.
- An **eval suite** wired into a build that catches a regression.
- A course manual and reference implementation to rebuild everything independently.

THE ORGANISATION GAINS

- **Capability that stays** — engineers who build and maintain agentic AI systems unaided.
- A shared vocabulary across engineering, architecture and security for reviewing AI work.
- The ability to **evaluate AI quality with numbers**, which makes it reviewable and fundable.
- A team that can tell a workable AI proposal from an unworkable one.

WHO THIS IS FOR

Working **software engineers** who write code day to day. Python is the working language; engineers from Java, C#/.NET or similar backgrounds are well served, with a short primer provided in advance so everyone starts level. Architects and technical leads take the same programme and gain the review vocabulary with it.

Day by day

Timed to an eight-hour day with 90 minutes of breaks — 390 minutes of instruction. One system is built across the week: an AI agent that answers from company documents through RAG, acts through tools, and is then evaluated and hardened. Compresses to four days where scheduling requires it.

DAY 1 Foundations — the LLM as a component

75 + 120 + 120 + 75 = 390 min

75 min — **How a large language model behaves** LAB

Context windows, token budgets, temperature and non-determinism, shown by running things rather than by slides. **Build:** find the point at which output stops being repeatable.

120 min — **Structured output** LAB

Typed, validated objects instead of prose — JSON schemas, structured output modes, and recovery when parsing fails. **Build:** break it with bad enums and truncated responses, then handle each case.

120 min — **Tool calling (function calling)** LAB

How an LLM selects a tool, how arguments arrive, and why tool-schema design is really API design. **Build:** three tools over a stub internal API, then make the model choose wrong on purpose.

75 min — **Consolidation and supported build** SESSION

Open working time with the instructor circulating. Anyone behind gets individual attention; anyone ahead extends their build. Everyone ends Day 1 on the same baseline.

DAY 2 AI agents — the agent loop, built by hand

135 + 105 + 105 + 45 = 390 min

135 min — **Building the agent loop** LAB

Reason, call, observe, repeat — written from first principles with no framework, so every abstraction later has something concrete beneath it. **Build:** a hand-rolled AI agent answering a multi-step question.

105 min — **Termination, iteration limits and errors** LAB

Stop conditions, iteration caps, and how errors travel through a loop that calls an LLM that calls a tool. **Build:** make it fail three ways, then bound each failure.

105 min — **Extending the agent** LAB

More tools, harder questions, and the point where a single prompt stops being enough. **Build:** add tools until selection degrades, then diagnose the description, the schema, or tool count.

45 min — **Review and shared baseline** SESSION

Comparing implementations across the room — engineers solve the agent loop differently, and the comparison teaches more than the build did.

DAY 3 RAG — Retrieval-Augmented Generation that works

75 + 105 + 135 + 75 = 390 min

75 min

A naive RAG pipeline, deliberately mediocre LAB

Load, chunk, embed, store in a vector database, retrieve, generate — built fast and imperfectly on purpose. **Build:** ask ten questions and collect the bad answers; they are the day's material.

105 min

Chunking, embeddings and representation LAB

Fixed, recursive, semantic and structure-aware chunking; chunk size and overlap; embedding choice; metadata worth carrying; why tables and scanned pages break retrieval. **Build:** re-chunk three ways and compare.

135 min

Measuring retrieval quality LAB

Golden sets, hit rate, recall@k and mean reciprocal rank. Hybrid search (keyword plus vector), reranking and metadata filters — each kept only if the number moves. **Build:** a 20-question golden set, a baseline, three scored interventions.

75 min

Grounding, citation and refusal LAB

Prompting for citations, forcing an honest “not in the sources”, and handling contradictory documents. **Build:** wire RAG into the Day 2 agent as a tool, with correct refusal behaviour.

DAY 4 Agent frameworks — state, control, coordination

105 + 90 + 105 + 90 = 390 min

CHOICE OF STACK — ONE, SELECTED AT BOOKING

LangGraph + LangChain · default

Graph nodes with typed state, conditional edges and bounded retries, checkpointers with interrupt and resume, and supervisor / router / plan-and-execute patterns.

AWS Bedrock

Amazon Bedrock Agents and AgentCore with session state, action groups on AWS Lambda, Bedrock Knowledge Bases alongside the Day 3 RAG pipeline, and Step Functions for orchestration.

105 min

Port the hand-rolled loop onto the framework LAB

The same agent rebuilt as a state graph. Because the loop was written by hand, every abstraction is recognisable. **Build:** diff the two and name exactly what the framework bought.

90 min

Branching, routing and recovery LAB

Conditional paths on state, fallbacks, bounded retries and cycles that terminate. **Build:** a routing node choosing between RAG retrieval, a tool call, and answering directly.

105 min

Persistence and human-in-the-loop (HITL) LAB

Durable state across turns, resuming an interrupted run, and approval gates before anything destructive — the pattern that makes agentic AI acceptable to a risk function. **Build:** survive a restart mid-run.

90 min

Multi-agent systems, and MCP LAB

Supervisor, router and plan-and-execute patterns, with the honest counter-case that most multi-agent problems are one agent with better tools. **Build:** a two-agent handoff plus an internal service exposed as an MCP (Model Context Protocol) tool.

DAY 5 Production — evals, guardrails, LLMOps

105 + 90 + 90 + 105 = 390 min

105 min — **LLM evaluations and the regression suite** LAB

Assertion-based checks against LLM-as-a-judge, and the ways a judge misleads you. Evaluating retrieval and generation separately. **Build:** wire evals into CI, break something, watch the build go red.

90 min — **Guardrails, verifiers and blast radius** LAB

Output validation, verifier patterns, prompt injection arriving through retrieved documents and tool results, and allowlists. **Build:** attack your own agent through a poisoned source document, then close the hole.

90 min — **Observability, cost and latency** LAB

Tracing, what to log and what never to log, token accounting per request and per feature, semantic caching, model routing, streaming and timeouts. **Build:** cut cost and latency without moving the eval score.

105 min — **Deployment, failure modes, your own system** REVIEW

Packaging, concurrency, rollout, and model-version changes that shift behaviour underneath you. Agent incidents degrade quietly rather than page anyone. **Closing:** each team presents the system it intends to build, with live architecture review and named risks.

How it is delivered

GROUP SIZE

Hands-on work in **groups of five to six**, with a second facilitator for larger cohorts, so labs stay practical rather than becoming a demonstration.

VIRTUAL DELIVERY

Ten half-days across three to four weeks — same content and labs, with better attendance and retention than consecutive full days.

LANGUAGE

Delivered in English, with pacing and written materials adjusted for cohorts working in a second language.

GETTING READY

A short technical readiness call and setup checklist are provided ahead of delivery, so the first morning starts with teaching rather than configuration.

Arjun Thakur

Principal AI Engineer · AI Architect · Corporate AI Trainer

- **15+ years of production software engineering** across payments, travel, fintech and education technology — senior engineering, engineering management and director-level roles at global product companies.
- **Builds and operates a production agentic AI product** — LangGraph agent orchestration, RAG on a vector database, LLM evaluations and guardrails, running live with paying customers. This material is taught by someone who runs it.
- **AWS Certified Solutions Architect.** M.Tech in Computer Science and Engineering. Experienced with multi-day enterprise cohorts spanning developers, QA, DevOps and architects, delivered at around 80% hands-on.

EMAIL arjun@arjunthakur.dev

WEBSITE arjunthakur.dev

LINKEDIN linkedin.com/in/thakurarjun247