

Applied LLM Systems: RAG, Fine-Tuning and Production Architecture

A focused engineering workshop for experienced Python developers

FA-0584 · 2 days

Agentic & Generative AI · Software Development · Data & Analytics · Cybersecurity

Build a bounded RAG prototype and compare fine-tuning approaches, evaluation methods and engineering controls for larger LLM systems.

Introduction

This two-day advanced workshop examines practical architecture decisions for Python LLM systems under latency, cost, data and security constraints. It compares retrieval-augmented generation, fine-tuning and hybrid designs rather than prescribing one approach for every problem.

The hands-on focus is a small retrieval-grounded prototype built from prepared components and data. LoRA/QLoRA workflows are reviewed through a prepared example or demonstration suited to available hardware. Production architecture, tenant isolation, observability and failure handling are design-review topics; completing the workshop is not a guarantee of production readiness, security or model-training proficiency.

Learning outcomes

The course teaches participants to:

- Design a Python model-client wrapper with bounded timeouts, retries and secure configuration.
- Assemble and inspect context and retrieval pipelines for a small LLM application.
- Evaluate retrieval separately from answer quality using a small reference dataset.

- Compare RAG, fine-tuning, LoRA/QLoRA and hybrid approaches against stated task and resource needs.
- Explain data-quality, overfitting, versioning and rollback considerations for model adaptation.
- Identify prompt-injection, data-leakage, tenant-isolation and tool-use risks and propose layered controls.

Prerequisites

- Strong Python engineering skills, including classes, dataclasses, typing, asynchronous programming, exceptions and logging.
- Experience with REST APIs, JSON schemas, pagination, retries and rate limits.
- Software design fundamentals, modularity and separation of concerns.
- Practical Git, command-line and Linux/macOS experience, and comfort reading SDK code and technical documentation.
- Docker, databases, cloud-compute concepts and GPU/memory awareness are strongly recommended.
- Access to a prepared compatible learning environment; training demonstrations depend on model licences, hardware and resource availability.

Training outline

1. Day 1 — Client Architecture and Retrieval Foundations

Module 1 — System Boundaries and Failure Modes

- Map the model, prompts, context, data, tools and guardrails.
- Distinguish deterministic code from probabilistic outputs; make failures observable.
- Choose whether the task needs prompting, external retrieval, behavioural adaptation or a hybrid.

Module 2 — Python Model Clients and Output Contracts

- Compare synchronous/asynchronous calls and streaming/blocking responses.
- Budget tokens and handle bounded retries, timeouts and circuit breakers.
- Centralise client wrappers and keep secrets outside code.
- Compose and version prompts; validate supported structured-output contracts without treating prompts as deterministic executable specifications.

Module 3 — Context and Ingestion

- Assemble retrieved data, metadata, tool outputs and conversation state within a context budget.
- Prioritise and truncate context while detecting contamination and untrusted instructions.
- Compare PDF, HTML, Markdown, API and database extraction issues.
- Clean and normalise sample material; compare fixed, semantic and structure-aware chunking.
- Design retrieval metadata and review incremental ingestion and re-indexing.

Module 4 — Indexing and Retrieval Prototype

- Generate embeddings and compare dimensions, similarity measures and index trade-offs.
- Discuss batching, persistence, rebuilds and tenant boundaries.
- Use metadata filters; compare dense/sparse retrieval, query rewriting, multiple queries and re-ranking.
- Build a selected retrieval path and inspect its results before adding generation.

2. Day 2 — Generation, Evaluation and Adaptation Decisions

Module 5 — Grounded Answers and Evaluation

- Compose stateless or conversational RAG responses with source traceability and insufficient-evidence handling.
- Check citations and multi-document synthesis; model confidence statements are not calibrated certainty.
- Separate retrieval failures from generation failures; compare relevance and faithfulness.
- Use a small reference dataset for regression checks across prompt, model and index changes.
- Review latency, cost, observability and maintenance trade-offs; grounding cannot eliminate hallucination.

Module 6 — Fine-Tuning, LoRA/QLoRA and Hybrids

- Distinguish behaviour/style/structure adaptation from maintaining access to current external knowledge.
- Review dataset construction, data quality, held-out evaluation and overfitting.
- Compare full fine-tuning with parameter-efficient LoRA adapters.
- Explain QLoRA as training LoRA adapters on a quantized base model, not generic quantization-aware training.
- Inspect a prepared training example, GPU/memory constraints, model versioning, rollback and deployment considerations.

- Compare RAG-only, adaptation-only and hybrid designs against task, cost, latency and organisational constraints.

Module 7 — Security and Production Architecture Review

- Analyse direct and retrieved-content prompt injection, data-leakage vectors and tool abuse.
- Review access controls, tenant isolation, data minimisation and logging/audit requirements.
- Compare fail-closed and fail-open behaviour and identify decisions requiring human approval.
- Review the prototype, its failure evidence and remaining operational/security validation before deployment.