APPLIED AND GENERATIVE AI

Retrieval-Augmented Generation (RAG) with Vector Databases

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

Overview

Large language models are powerful but limited: they hallucinate, they cannot see your private or current data, and their knowledge stops at a training cutoff. Retrieval-Augmented Generation (RAG) solves this by retrieving relevant information from your own sources at query time and giving it to the model as grounded context.

This is a hands-on, practitioner course. You will build a working RAG system end to end and learn the decisions that separate a demo from a dependable production system: how to prepare and chunk documents, how to choose and operate a vector database, how to construct a retrieval pipeline that returns the right context, how to evaluate and improve answer quality, and how to handle the security, cost, and latency realities of running RAG for real users. Rather than survey every retrieval technique, it goes deep on the pipeline that covers most real systems. Every module ends with a lab and builds on the one before.

Who Should Attend

- Developers building applications on top of large language models
- AI and machine learning engineers adding private or current data to LLM apps
- Solution architects designing enterprise generative AI systems

Prerequisites

- Working proficiency in Python
- Basic familiarity with large language models and calling an LLM API
- Comfort with REST and JSON

What You Will Learn

- Explain the RAG architecture and when to choose it over fine-tuning or long context
- Prepare, chunk, and embed documents for effective retrieval
- Select, load, and query a vector database
- Build a retrieval pipeline with hybrid search and reranking, and assemble retrieved context into grounded, citable prompts
- Evaluate retrieval and answer quality, and diagnose common failure modes
- Run RAG in production: security, access control, cost, latency, and monitoring

Course Outline

Day one: from documents to retrieval

- Why RAG
  - The limits of standalone LLMs: hallucination, stale knowledge, private data

  - RAG versus fine-tuning versus long-context prompting
  - A reference architecture for RAG
  - Lab: call an LLM with and without retrieved context and compare the answers
- Embeddings and Chunking
  - How embeddings capture meaning
  - Chunking strategies and their tradeoffs
  - Document preparation, cleaning, and metadata
  - Lab: chunk and embed a document set and inspect what similarity search returns
- Vector Databases
  - What a vector database does and how similarity search works
  - Survey of options, from pgvector to managed services
  - Indexing, upserts, and metadata filtering
  - Lab: load embeddings into a vector database and query it with metadata filters

Day two: from retrieval to production

- Building the Retrieval Pipeline
  - Query embedding and top-k retrieval
  - Hybrid search and reranking for relevance
  - Assembling context and designing prompt templates
  - Lab: build a retrieval pipeline with hybrid search and reranking
- Generation, Grounding, and Evaluation
  - Calling the LLM with retrieved context; citations and grounded answers
  - When to use a framework (LangChain, LlamaIndex) and when to skip it
  - Retrieval and answer-quality metrics, test sets, and common failure modes
  - Lab: generate cited, grounded answers and score them against a small test set
- Production Concerns
  - Security, access control, and handling sensitive data
  - Cost, caching, and latency
  - Monitoring, feedback loops, and continuous improvement
  - Lab: add access control, caching, and monitoring to the RAG system

Extended Version

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

- Agentic RAG: letting an agent decide what and when to retrieve
- Advanced retrieval: multi-vector, query rewriting, and structured retrieval
- Fine-tuning embedding models for a specific domain
- A capstone: a fuller evaluation harness and an automated quality pipeline for your RAG system