

COURSE OUTLINE

Building Intelligence with RAG — Build a Grounded Knowledge Assistant

An intensive, lab-driven program for developers building a small, inspectable RAG application

DURATION

3 days

FORMAT

Instructor-led, lab-driven, capstone project

DELIVERY

Online Instructor Led Workshop · Onsite Program for your Teams

TRAINER

Muhammed Shakir

Why this course exists

An LLM can produce fluent answers, but a useful knowledge application must first find the right source material and show the user what supports its answer. Retrieval-Augmented Generation (RAG) supplies that missing evidence path.

This program gives developers a practical, build-first understanding of RAG: participants create a small knowledge assistant that ingests documents, retrieves relevant passages, generates grounded streamed answers, and makes its evidence visible.

By the end of three days, participants will have built a working RAG pipeline over a legal-text corpus, inspected it through a diagnostic API and Open WebUI, and compared a focused set of retrieval improvements.

What makes this different

The application is visible from the first session. **This is an engineering room: each idea becomes a small, inspectable part of one working knowledge assistant.** You will:

- **Start with the real user experience** — connect Open WebUI to an OpenAI-completions-style API, learn the streaming contract, and use /v1/query alongside chat to inspect what the system is doing.
- **Build the evidence path in order** — move from documents to chunks, embeddings, retrieval, context, citations, and streamed answers without replacing earlier work.
- **Compare useful retrieval choices** — establish semantic search, then use hybrid search, re-ranking, exact lookup, and query transformations only where they help answer a real question.

The course favours a clear, working RAG mental model over boring presentation slides.

Skills You'll Walk Away With

- Explain the document ' chunk ' embedding ' retrieval ' context ' generated-answer flow, and recognize when a question needs clarification or cannot be supported by the corpus.
- Set up Open WebUI against an OpenAI-completions-style chat endpoint and understand how a streamed answer, confidence signal, and sources reach the user.
- Parse, chunk, embed, and index documents with source metadata in MongoDB.
- Build Semantic RAG with inspectable retrieval results and grounded, source-backed answers.
- Compare semantic retrieval, hybrid search, and re-ranking on questions that reveal their different strengths.
- Use structured exact lookup, query decomposition, HyDE, and a bounded Agentic RAG controller as extensions to the same evidence-first RAG application.

Who should attend

This program is for experienced professionals who want to build and understand document-grounded AI applications.

- **Frontend and backend developers** — adding source-backed document search and answers to applications and APIs.
- **AI application developers** — moving from basic LLM calls to a working retrieval-and-generation pipeline.
- **Data and search engineers** — connecting documents, embeddings, metadata, and search to an LLM application.
- **Technical leads and solution architects** — learning the moving parts and trade-offs of a small RAG application.

Prerequisites: comfort with programming, HTTP APIs, Git, basic database queries, and calling an LLM API. Deep machine-learning expertise or advanced mathematics is not required; embeddings and retrieval are introduced in plain engineering language.

Course Outline

Day 1 — Set the RAG Stage and Make Knowledge Searchable

See the chat experience and diagnostic API first, then build the document and retrieval foundation they will use.

Session 1 — Establish the Application Contract

Create one small application whose chat interface, diagnostic endpoint, and streaming contract remain stable as RAG capability is added.

- RAG in one flow: what comes from model knowledge, what comes from retrieved evidence, and why an answer must show its sources
- The two learning views of the same application: `/v1/query` for diagnostic inspection and Open WebUI for the end-user chat experience
- OpenAI-completions-style chat APIs and server-sent streaming: request shape, streamed text, normal completion, and the final plain-language confidence/source summary
- Open WebUI course companion: use the supplied bundle and connection settings instead of building a separate chat frontend
- A light architectural baseline: user ' API ' retrieval ' MongoDB ' LLM ' user, and the distinction between application instructions and retrieved evidence
- *Hands-on:* Agree the small architecture, then seed the capstone and prove the interface contracts before retrieval is implemented.
 - First write and review `docs/architecture.md`. It records the single application flow, source and answer boundaries, the supplied BNS and IPC corpus, the project layout, and what is deliberately deferred. The approved architecture is the guide for the seed; it is not a separate exercise that participants must later replace.
 - Seed a Python 3.12 and UV project with FastAPI, Pydantic settings, PyMongo for the later database connection, Ruff, and a minimal test runner. Create a `src/` application package, `tests/`, `.env.example`, `.gitignore`, `pyproject.toml`, `uv.lock`, and the supplied `data/raw/` location. The shell must start without database or model credentials.
 - Add a safe health endpoint plus a small, shared pattern registry for the six initial course modes: `semantic`, `hybrid`, `hybrid-reranked`, `structured`, `decomposition`, and `hyde`. Each mode returns an honest not-yet-implemented placeholder until its story adds the real behavior; Session 6 adds `agentic` additively.
 - The trainer shares the supplied Open WebUI bundle over the local LAN. Participants extract it and run its setup script once: Windows (the primary classroom path) runs `powershell -Execution-Policy Bypass -File .\open_webui_functions\setup_open_webui.ps1`; macOS/Linux runs `sh open_webui_functions/setup_open_webui.sh`. The script installs, configures, starts, and provisions Open WebUI. Participants do not hand-install it, create accounts, edit the admin panel, or modify its Functions. Keep Open WebUI separate from the capstone; it is the chat client, not the home of retrieval or generation logic.
 - Add `/v1/query` as the diagnostic contract, `GET /v1/models` for mode discovery, and `/v1/chat/completions` as the Open WebUI contract. Both interfaces use the same registry; chat supports a clearly marked placeholder in JSON and server-sent-event streaming until RAG is added.
 - Define the small shared request/result and SSE shapes now: selected mode, retrieved passages, diagnostics, normal OpenAI-compatible text frames, stop frame, and `[DONE]`. Later stories add

answer, confidence, and citations within the established result and readable streamed text rather than changing the API or rebuilding the UI.

- Run the application, perform a health check, and use Open WebUI to select a placeholder mode and receive its streamed response. These are smoke checks for the seed, not an extensive test suite.
- **Session checkpoint:** an approved architecture, a seeded and runnable project, a connected Open WebUI chat, and stable diagnostic and streaming API contracts ready for the RAG pipeline.

Session 2 — Make Information Searchable

Turn the supplied documents into a small, inspectable semantic-search corpus.

- Document ingestion: parsing source files, keeping section boundaries, and preserving a source reference
- Chunking, embeddings, and vector search: what each step contributes to semantic retrieval
- Metadata that makes results understandable: act, chapter, section, title, chunk, and source reference
- Semantic search baseline: top-k results, similarity scores, and inspecting a relevant and an irrelevant result
- *Hands-on:* Build the repeatable corpus and Semantic RAG retrieval baseline in MongoDB.
 - Parse the supplied BNS and IPC PDFs into section-level records with act-qualified section identifiers and source references.
 - Chunk section text while retaining its parent section and source metadata.
 - Store sections, chunks, and embeddings in MongoDB and create the vector-search index.
 - Implement semantic retrieval behind `/v1/query`, returning the retrieved passages, source details, and scores for inspection.
 - Demonstrate natural-language retrieval with a few prepared questions and adjust only the chunking or top-k settings needed to make the behaviour understandable.
 - **Session checkpoint:** a repeatable document-to-MongoDB flow and a diagnostic semantic-search endpoint that makes retrieved evidence visible.

Day 2 — Turn Retrieval into Grounded, Streamed Answers

Assemble retrieved evidence into a cited answer, then improve the baseline for natural-language and exact-term questions.

Session 3 — Build the Grounded Answer Experience

Replace the placeholder chat response with the same evidence-backed answer flow in both Open WebUI and `/v1/query`.

- Context assembly: selecting retrieved passages, preserving their source details, and keeping context within a practical budget
-

Grounded generation: instructing the model to use supplied evidence, cite the act and section, and state when evidence is insufficient

- Streaming an answer with confidence: emit answer text as it is generated, then append a readable confidence and citations summary without changing the client contract established on Day 1
- Reading a RAG interaction in two ways: diagnose retrieved context through `/v1/query`, then experience the same answer through Open WebUI
- *Hands-on:* Implement cited answer generation and connect it to the existing streaming chat route.
 - Assemble context from semantic-retrieval results and retain each passage's source details.
 - Generate an answer that cites only retrieved BNS or IPC sections and declines unsupported questions.
 - Replace the placeholder stream with the answer stream, followed by a readable confidence and citation summary in normal OpenAI-compatible text frames.
 - Make `/v1/query` return the answer together with its retrieved evidence so participants can explain why the chat answer appeared.
 - Use Postman for one answerable and one unsupported question, then run the same questions through Open WebUI.
 - **Session checkpoint:** one working RAG answer path with inspectable evidence, citations, streaming, and confidence in the chat interface.

Day 3 — Improve Retrieval Deliberately

Compare a small set of RAG extensions, retain the evidence path, and finish with a clear demonstration of what each mode is for.

Session 4 — Add Hybrid Search and Re-ranking

Improve the semantic baseline when exact legal terms and candidate ordering matter.

- Hybrid search: combining keyword and semantic retrieval for exact terms, identifiers, and paraphrased questions
- Rank fusion and re-ranking: candidate retrieval versus final context order, with the trade-off of an additional processing step
- Comparison by inspection: use the same question in more than one mode and look at retrieved passages before judging the answer
- *Hands-on:* Add independently selectable retrieval improvements to the existing diagnostic and chat paths.
 - Add keyword retrieval and combine it with vector results to create a **hybrid** mode.
 - Add a small re-ranking step to create a **hybrid-reranked** mode, retaining the before-and-after candidate order for `/v1/query`.
 - Pass the selected mode through the existing context, answer, citation, streaming, and Open WebUI paths.
 -

Compare semantic, hybrid, and hybrid-reranked retrieval using prepared natural-language and exact-section questions.

- **Session checkpoint:** selectable semantic, hybrid, and hybrid-reranked modes whose evidence and answers can be compared without a separate application.

Session 5 — Use Focused Retrieval Extensions

Add only the bounded extensions that clarify a different RAG question shape.

- Structured exact lookup: use a validated act-and-section lookup when the user already knows the precise record they need
- Query decomposition: split a compound comparison question, retrieve evidence for each part, then synthesize from that evidence
- HyDE: use a hypothetical retrieval text to find real passages for a vocabulary-mismatch question; hypothetical text is never cited as evidence
- Choosing the smallest useful path: semantic retrieval by default, hybrid or re-ranking for retrieval quality, structured lookup for exact records, and transformations for the few questions that need them
- *Hands-on:* Add lightweight, inspectable extensions and complete the course demonstration.
 - Implement structured lookup for an explicit act-and-section request and return the same source shape used by the semantic path.
 - Implement bounded query decomposition and HyDE for prepared compound or vocabulary-mismatch questions; retain the subquestions or hypothetical retrieval text only as diagnostics, never as citations.
 - Route each extension through the existing context, answer, /v1/query, and streaming Open WebUI contracts.
 - Demonstrate one suitable question for each added mode, inspect its evidence in Postman, and show the corresponding streamed answer in Open WebUI.
 - **Session checkpoint:** structured lookup, decomposition, and HyDE remain small, inspectable extensions of the shared evidence path.

Session 6 — Add a Bounded Agentic RAG Controller and Demonstrate the Assistant

Use fixed application rules to choose a short, inspectable sequence of existing retrieval capabilities; demonstrate orchestration without autonomy.

- Bounded Agentic RAG: a selected **agentic** mode whose controller validates the request, retrieves, inspects the returned evidence, and either refines once or stops
- A fixed read-only allowlist: structured exact lookup, semantic retrieval, and HyDE retrieval; no planner, free-form tools, writes, browsing, or background work
- Evidence and trace boundaries: real retrieved corpus passages remain the only answer and citation evidence; the trace records the controller's non-sensitive decisions without exposing a HyDE hypothetical text
- *Hands-on:* Add one additive controller to the existing diagnostic and chat paths.

- Validate ambiguous or malformed requests before any tool call; return `clarify` with an inspectable validation trace.
- For an act-qualified section request, use one structured lookup. For an ordinary question, use semantic retrieval and stop when its documented adequacy rule is met. For a prepared vocabulary-mismatch question with inadequate direct semantic evidence, use one HyDE retrieval after semantic retrieval.
- Propagate the existing effective demo/public scope to every call, inspect each result, and enforce at most two calls, one HyDE call, and a finite deadline. Stop explicitly for unavailable tools, exhausted budget, inadequate evidence, or clarification.
- Route only real returned passages through the established context, answer, citation, `/v1/query`, and Open WebUI streaming paths. Keep the controller trace inspectable and non-sensitive.
- Demonstrate an exact lookup, an adequate explanation, a vocabulary mismatch, and an ambiguous request; inspect the diagnostic trace before showing the chat answer.
- **Session checkpoint:** a small, coherent knowledge assistant that participants can explain from document ingestion through retrieval, bounded orchestration, evidence, streamed answer, confidence, and citations.

Capstone project

Participants incrementally build a **BNS and IPC legal-text knowledge assistant** using the supplied source documents. The capstone is deliberately small: it teaches the core RAG evidence path, not a production-ready legal system. It supports natural-language questions, exact act-and-section lookups, and a small set of prepared comparison questions. It does not provide legal advice; when the corpus cannot support an answer, it says so.

The project has two complementary interfaces throughout the course:

- `/v1/query` is the diagnostic learning interface. It exposes the selected retrieval mode, retrieved passages, scores or ordering where relevant, answer, confidence, and citations so participants can inspect the pipeline.
- **Open WebUI through** `/v1/chat/completions` is the end-user interface. It uses the OpenAI-completions-style streaming contract established in the first session, so participants experience the same grounded answer as a chat interaction.

The application builds in one direction. The initial scaffold establishes the Open WebUI and streaming contracts. Ingestion and semantic retrieval fill the diagnostic path with real evidence. Context assembly and answer generation replace the placeholder chat stream. Hybrid search, re-ranking, structured lookup, query decomposition, HyDE, and the bounded Agentic RAG controller extend the same contracts and evidence format; they do not introduce replacement applications or require rework of the user interface.

In scope: PDF ingestion, chunking, embeddings, MongoDB vector search, semantic retrieval, context assembly, grounded citations, streaming answers with confidence, hybrid search, re-ranking, exact struc-

tured lookup, query decomposition, HyDE, and a deterministic, read-only Agentic RAG controller over those existing retrieval paths.

Out of scope for this course: comprehensive security controls, multi-user authorization, production observability and deployment, an evaluation harness, aggressive automated testing, GraphRAG, open-ended agentic systems, and broad platform functionality. These can be introduced later as production concerns or advanced follow-on topics.

The required outcome is a working, explainable classroom application: participants can ask a question in Open WebUI, see a streamed grounded answer with confidence and citations, then use `/v1/query` to inspect the retrieval evidence behind it.

A note on delivery

This three-day course is intentionally tapered from the comprehensive RAG program. It keeps the concepts needed to understand and build a useful document-grounded assistant, including one bounded controller over existing retrieval paths, while removing production hardening, broad security work, evaluation infrastructure, GraphRAG, open-ended agentic orchestration, and other work that would distract from the core learning path.

Each full day balances instructor-led explanation, demonstration, guided labs, review, and troubleshooting, with two tea breaks and one lunch break. The supplied documents, Open WebUI course bundle, and prepared questions keep the work centred on RAG rather than setup or frontend construction.

The sequence is deliberate: first make the chat and API contracts visible; then build real evidence retrieval; then turn that evidence into streamed, cited answers; compare focused retrieval improvements; and finally use fixed controller rules to inspect and refine one retrieval path. Participants leave with a working application and a clear account of how its answer is supported.

Ready to build grounded knowledge applications?

Enquire about a private cohort or a tailored delivery for your developer team.