INTRODUCTION TO RETRIEVAL-AUGMENTED GENERATION (RAG)

Module 5 — Harwell Prompt Engineering

LEARNING OBJECTIVES

By the end of this module you will be able to:

- Explain the concept of RAG: moving beyond training data by connecting LLMs to your own documents/wiki
- Describe how RAG works at a high level: embeddings, vector databases, retrieval, and generation
- Compare when to use RAG vs. fine-tuning vs. long-context windows
- Relate RAG to internal knowledge bases (e.g. searching internal documentation)

BRIDGE FROM MODULE 4

What we learned yesterday:

- **How** to use AI tools effectively
- Sidecar and integrated workflows

The problem:

- ❌ AI doesn’t know your internal policies
- ❌ AI doesn’t know your codebase structure
- ❌ AI can’t access your wiki or documentation
- ❌ You’re pasting docs manually, which is slow and risky

Today: Learn how to connect AI to your own knowledge with RAG.

THE PROBLEM: LLMS DON'T KNOW YOUR KNOWLEDGE

The limitation:

- LLMs are trained on public data up to a cutoff date
- ❌ They don't know your internal documentation
- ❌ They don't know your proprietary codebase
- ❌ They don't know your organization's policies
- ❌ They can't access your wiki or knowledge base

The pain:

- Query: “What’s our company’s policy on data retention?”
- ❌ AI gives generic answer (not your policy)
- ❌ You have to find and paste the policy manually
- ❌ Slow, error-prone, risky (data privacy)

THE SOLUTION: RAG

RAG = Retrieve, Augment, Generate

- **Retrieve** relevant document chunks from your knowledge base
- **Augment** the prompt with those chunks as context
- **Generate** answer based on your actual documents

Result:

WITHOUT RAG VS. WITH RAG

Without RAG:

- Query: “What’s our data retention policy?”
- AI response: Generic answer based on training data
- ❌ Not your actual policy
- ❌ May be outdated or wrong

With RAG:

- Query: “What’s our data retention policy?”
- **Step 1**: Retrieve relevant document chunks
- **Step 2**: Add chunks to prompt as context
- **Step 3**: Generate answer based on your documents
- ✅ Answer grounded in your real policy
- ✅ Accurate and specific

HOW RAG WORKS: THE FLOW

Step 1: Document preparation

- **Chunking**: Break documents into smaller pieces
- Why? Documents too large for context window
- Example: Policy document → 10 chunks of ~500 words each

Step 2: Embeddings

- Convert text chunks to numerical representations (embeddings)
- Why? Allows similarity search
- Example: “data retention policy” embedding is similar to “data storage rules” embedding

Step 3: Vector database

- Store embeddings in a vector database
- Why? Fast similarity search
- Examples: Pinecone, Weaviate, Chroma, PostgreSQL with pgvector

HOW RAG WORKS: RETRIEVAL AND GENERATION

Step 4: Query processing

- User asks: “What’s our data retention policy?”
- **Embed query**: Convert to embedding
- **Retrieve**: Find top 3-5 most similar chunks from vector DB
- **Augment prompt**: Add chunks to prompt as context
- **Generate**: LLM generates answer based on retrieved chunks

The complete flow:

Documents → Chunks → Embeddings → Vector DB → Retrieve → Augment → Generate

SIMPLE ANALOGY

Think of RAG like a librarian:

1. You ask a question
2. Librarian searches the catalog (retrieval)
3. Librarian brings relevant books (chunks)
4. Librarian reads from those books to answer (generation)

RAG does this automatically with your documents.

RAG VS. FINE-TUNING VS. LONG CONTEXT

RAG — use when:

- ✅ You have documents that change frequently
- ✅ You need to cite sources (which document?)
- ✅ Documents are too large for context window
- ✅ You want to add knowledge without retraining

Fine-tuning — use when:

- ✅ You need model to learn specific style or format
- ✅ You have large dataset of examples
- ✅ You want model behavior to change permanently
- ⚠️ Expensive, requires retraining for updates

Long context windows — use when:

- ✅ You have a few large documents
- ✅ Documents don’t change often
- ✅ You need full document context
- ⚠️ Expensive, slower, limited by model’s context window

DECISION FRAMEWORK

Solution	Use when
RAG	Multiple docs, frequent updates, need citations
Fine-tuning	Need style/behavior change, large example dataset
Long context	Few large documents, need full context

Choose based on your needs.

USE CASES: WHEN RAG HELPS

Use case 1: Internal documentation search

- Problem: “Where’s the API documentation for our payment service?”
- RAG: Search internal docs, retrieve relevant sections, answer with citations
- ✅ Faster than manual search
- ✅ Answers grounded in actual docs

Use case 2: Q&A over policies

- Problem: “What’s our policy on remote work?”
- RAG: Retrieve policy document, answer based on actual policy
- ✅ Accurate, cites source
- ✅ No manual lookup needed

Use case 3: Codebase Q&A

- Problem: “How does our authentication system work?”
- RAG: Retrieve relevant code files, explain based on actual code
- ✅ Understands your codebase
- ✅ Can reference specific files

RELEVANCE TO THIS AUDIENCE

RAG helps with:

- Internal knowledge bases
- Governance and compliance
- “Search then answer” workflows
- Codebase understanding
- Policy Q&A

You can use RAG-enhanced AI tools without building RAG yourself.

SUMMARY

1. **Problem**: LLMs don’t know your internal knowledge
2. **Solution**: RAG retrieves relevant docs and adds to prompt
3. **Architecture**: Documents → Chunks → Embeddings → Vector DB → Retrieve → Generate
4. **When to use**: Multiple docs, frequent updates, need citations
5. **Use cases**: Internal docs, policies, codebase Q&A

BRIDGE TO MODULE 6

What we've learned:

- **RAG** connects AI to documents

What's next:

Module 6: MCP (Model Context Protocol) — connects AI to live systems (files, repos, databases).

Both solve the "AI doesn't know your context" problem.

QUESTIONS?

Module 5 — Introduction to Retrieval-Augmented Generation (RAG)