Retrieval-Augmented Generation (RAG)

Mia Mohammad Imran

Outline

- What is RAG
- Build a baseline RAG pipeline over local documents
- Write grounded prompts that require RAG
- Diagnose common RAG failures and apply basic improvements

Why RAG?

- LLMs do not reliably “know” your repo, tickets, ADRs, or runbooks
- RAG grounds answers in **project artifacts** and enables **traceability**
- Typical SE artifacts for RAG:
  - README, architecture docs, ADRs, API specs
  - Jira issues, incident postmortems, runbooks
  - Coding standards, contribution guides, release notes

Example Prompt: *“What is our branching strategy and required PR checks?” (answer must cite CONTRIBUTING.md or policy doc)*

What is RAG: Key Idea

- RAG is “search + answer”:
  - **Retrieval**: searching for the most relevant information
  - **Generation**: using a language model to generate an answer based on that information
- Think like:
  - Without RAG: model answers from parameters and prompt
  - With RAG: model answers from prompt plus retrieved project evidence

Example: *“Which service owns endpoint /v2/payments/refunds and what are the retry rules?” (cite API spec + service ownership doc)*

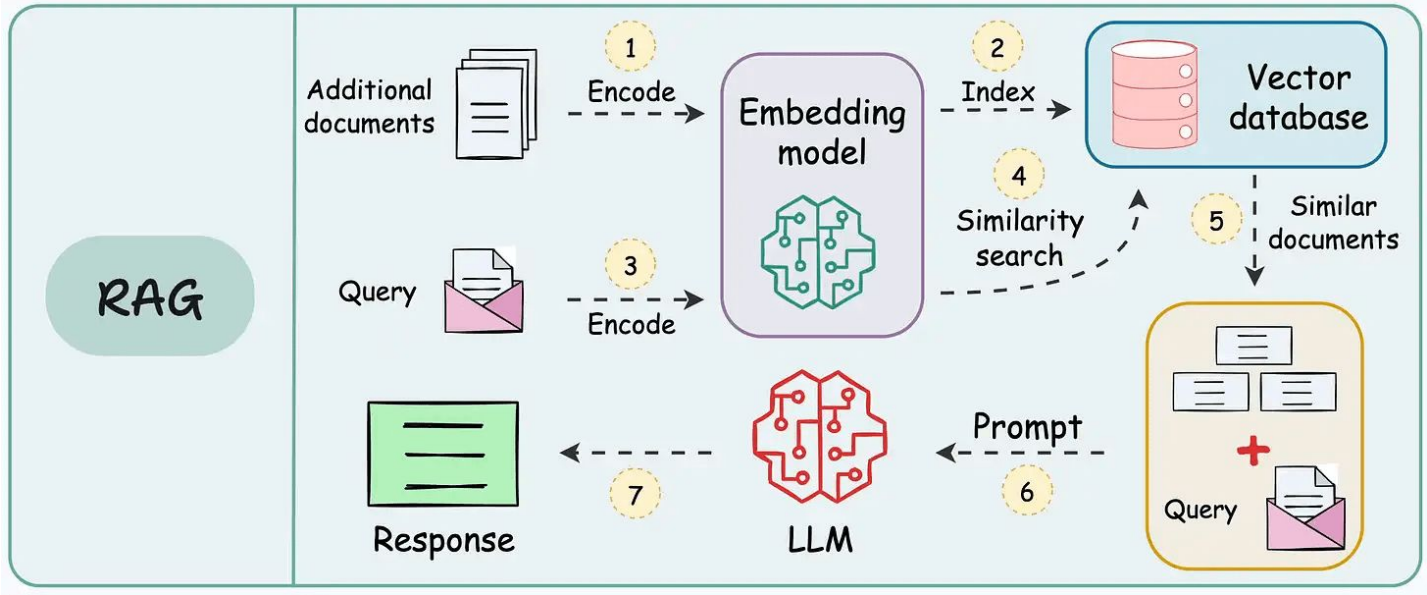
- **Goal: grounded answers with traceable sources**

When to use RAG?

- Use RAG when tasks depend on project artifacts that may change or are not reliably in a base model (policies, design docs, repo conventions).
- Avoid RAG when the task is pure reasoning or general knowledge and the corpus adds little value

Baseline Architecture

- **Indexing**: Corpus → chunk with small overlap→ embed → index
- **Online query**: Query → embed → retrieve top-k → prompt with sources → LLM answer + citations



What counts as documents?

- Anything that has structured/unstructured information
  - README, architecture notes, ADRs, API specs
  - Issues, postmortems, runbooks
  - Coding standards, contribution guide, release notes

Chunking

- **Chunking goals**
  - Make the retriever’s job easy by creating semantically coherent units
  - Avoid chunks that are too long (dilution) or too short (missing context)
- Aim for “one idea per chunk”
- Use overlap to avoid losing boundary context (10-20%)
- Common failure: chunks too large (dilution) or too small (missing definitions)

Embeddings and Similarity Search

- **Embeddings:** An embedding model maps text into a numeric vector where similar meanings are close
- **Similarity metrics:**
  - Cosine similarity (common in text retrieval)
  - Inner product on normalized vectors (equivalent to cosine)
- **Vector stores**
  - In-memory or local: FAISS, Chroma
  - Database-backed: pgvector

Retrieval

- **Top-k retrieval:** Retrieve the k most similar chunks. Typical k is 3 to 8 for baseline systems
- If answers cite irrelevant text, decrease k or improve chunking
- If the right info exists but is not retrieved, increase k slightly and refine chunk boundaries

Example: *Query “timeouts in payments” retrieves general “SRE guidelines” but misses “payments-client config” section → Wrong answer*

Why this is helpful?

- **More accurate answers**: reduces “hallucinations” from the LLM
- **Up-to-date information**: you can add your own latest info to the system
- **Contextual responses**: answers that reflect your own data

Prompt pattern for grounded answers using RAG

Minimal requirements:

- “Use only SOURCES.”
- “If not supported, say: I don’t know.”
- “Cite [S1], [S2].”

Output should include:

- Answer
- Citations

Example prompt: *“Explain the deployment rollback steps for Service A and cite the runbook section.”*

Evaluation

Create a mini test set (10-15 questions):

- Most are answerable with clear sources
- A few are intentionally unanswerable (expected: “I don’t know”)

Grade quickly on:

- Correctness
- Citation quality (relevant, sufficient)
- Refusal correctness (when evidence is missing)

Examples:
Answerable: “What are required checks before merge?”
Unanswerable: “What is My Team Lead’s preferred code style?” (should refuse)

Common failure modes and quick fixes

1. **Wrong answer with citations:** Fix: better chunking, smaller k, or stronger “use sources only” prompt
2. **Misses known info:** Fix: add overlap, adjust chunk size, try k + 2
3. **Too much context:** Fix: reduce k, keep only top evidence, or compress retrieved text

Basic upgrades

- **Query rewriting**: rewrite the question into better retrieval queries
- **Reranking**: reorder retrieved chunks to keep the best evidence
- **Hybrid search**: combine embeddings with keyword search for better recall

RAG in prompting and RAG in agents

- **Single-shot RAG (prompting):** one retrieve, one answer. Best first step
  - “Where is the password rotation policy described?”
- **Agentic RAG:** the model can call retrieval multiple times, refine queries, and verify coverage. Add iteration limits and evidence rules
  - “Investigate failing CI: find policy, locate relevant logs section, propose fix with citations.”

Possible Software Engineering Use Cases

Onboarding Q&A

- “How do I run tests locally and where are env vars documented?”

Policy assistant

- “What is the definition of Sev-1 and who is on call escalation?”

Release assistant

- “Summarize user-impacting changes in last release notes with citations.”