

RAG Explained: How AI Answers From Your Own Documents

How an AI assistant answers from your own documents: chunking, embeddings, semantic search, building the prompt, citations, where RAG goes wrong, and when to use it instead of fine-tuning.

20 min read · Content last verified 8 October 2026 · Reviewed by Shravya M

Free awareness guide. Not legal or financial advice. <https://www.shravyadigital.com/guides/retrieval-augmented-generation>

CONTENTS

1. Before you start
2. Chapter 1: The problem RAG solves
3. Chapter 2: The idea: an open-book exam
4. Chapter 3: Preparing the documents: chunking
5. Chapter 4: Embeddings and searching by meaning
6. Chapter 5: The full RAG pipeline
7. Chapter 6: When RAG goes wrong
8. Chapter 7: RAG, fine-tuning or a long context?
9. Key words

Before you start

This guide is the reading for **Course 3** of the AI Foundations Pathway. It takes about 20 minutes and builds on Course 2, *Large Language Models*.

RAG (retrieval-augmented generation) is the most common way companies make an AI assistant answer from *their own* documents: a bank's policies, a college's rulebook, a company's product manuals. You'll learn how it works step by step, where it goes wrong, and how to judge one.

All names and examples here are made up.

Chapter 1: The problem RAG solves

An LLM on its own has three big gaps:

1. **It doesn't know your documents.** It was never trained on your company's HR policy or last month's price list.
2. **It has a knowledge cut-off.** Anything newer is missing.
3. **It can hallucinate.** Asked something it doesn't know, it may confidently make up an answer.

Retraining a model every time a document changes would be slow and very expensive. RAG solves this differently: **find the relevant facts first, then give them to the model along with the question.**

Chapter 2: The idea: an open-book exam

Think of an exam:

- A **closed-book** exam is an LLM answering from memory, which may be outdated or wrong.
- An **open-book** exam is RAG: before answering, the student looks up the right pages and answers from them.

RAG has two parts, and the name says them in order:

1. **Retrieval:** search a collection of documents for the passages most relevant to the question.
2. **Augmented generation:** add those passages to the prompt and ask the LLM to answer *using them*.

Ravi asks his company assistant, "How many days of paternity leave do I get?" The system finds the paragraph on paternity leave in the HR policy, hands it to the LLM, and the LLM answers from that paragraph, ideally saying which document it came from.

The rule: In RAG, the answer is only as good as what was retrieved.

Chapter 3: Preparing the documents: chunking

Before anything can be searched, the documents are prepared:

1. **Collect** them: PDFs, web pages, manuals, FAQs.
2. **Clean** them: remove repeated headers and footers, and convert scanned pages to text.
3. **Split** them into smaller pieces called **chunks**, often a few paragraphs each.

Why chunks? A whole 200-page manual won't fit well in the prompt, and most questions need only a small part of it.

Chunk size is a trade-off:

- **Too small:** a chunk loses its meaning ("It is 15 days." *What* is?).
- **Too large:** the useful sentence is buried in irrelevant text, and fewer chunks fit.

Good systems split along natural breaks such as headings and sections, and keep a little overlap so ideas aren't cut in half.

Chapter 4: Embeddings and searching by meaning

How does the system find the *right* chunk? Usually with **embeddings**.

An **embedding** turns a piece of text into a long list of numbers that represents its **meaning**. Texts with similar meanings get similar numbers, even if they use different words. "Leave for new fathers" and "paternity leave" end up close together.

The chunks' embeddings are stored in a **vector database**. When a question comes in:

1. The question is turned into an embedding too.
2. The system finds the chunks whose embeddings are **closest** in meaning.
3. The top few (often 3 to 10) are retrieved.

This is **semantic search**: searching by meaning rather than exact words. Many systems combine it with ordinary **keyword search**, which is better for exact things like product codes, names or section numbers. Some also **re-rank** the results with a second model for better accuracy.

Chapter 5: The full RAG pipeline

Put together, one question goes through these steps:

1. **Question:** "What's the late fee on my credit card?"
2. **Embed** the question.
3. **Retrieve** the most relevant chunks from the vector database.
4. **Build the prompt:** instructions ("Answer only from the passages below. If the answer isn't there, say you don't know."), plus the retrieved passages, plus the question.
5. **Generate:** the LLM writes the answer.
6. **Cite:** the answer shows which documents or sections it used, so a person can check.

That instruction in step 4 matters. Telling the model to answer *only* from the passages, and to admit when the answer isn't there, greatly reduces hallucination.

Chapter 6: When RAG goes wrong

RAG reduces mistakes; it doesn't remove them. Common failures:

- **Wrong chunk retrieved.** The search finds something similar but not right, and the answer is built on it.
- **Missing or out-of-date documents.** If the new policy was never added, the old one is quoted.
- **Bad chunking.** The key sentence was split across two chunks and neither makes sense alone.
- **The model still drifts.** It mixes in its own "memory" instead of the passages.
- **Access leaks.** An employee gets an answer from a document they shouldn't see. Good systems check permissions *before* retrieving.
- **Prompt injection.** A document contains hidden text like "ignore your instructions and...", and the model obeys it. Retrieved text should be treated as information, never as commands.

How teams test a RAG system:

- Build a list of real questions with known correct answers, and check them regularly.
- Measure **retrieval** (was the right chunk found?) separately from **generation** (was the answer faithful to it?).
- Check that citations really support the answer.

Chapter 7: RAG, fine-tuning or a long context?

There are three ways to give a model your knowledge:

Approach	What it is	Best when
RAG	Look up facts at question time	Documents change often; you need citations and access control
Fine-tuning	Train the model further on your examples	You need a consistent style, format or specialised behaviour
Long context	Paste whole documents into one prompt	A few documents, one-off questions

Many real systems use RAG, sometimes with a little fine-tuning. You'll meet RAG in customer support bots, company knowledge assistants, legal and medical research tools, and "chat with your PDF" apps.

When you use one, check the citations it gives. A RAG answer with a source you can open is far more trustworthy than one without.

Key words

- **RAG:** retrieval-augmented generation; find relevant text, then answer from it.
- **Chunk:** a small piece of a document, prepared for search.
- **Embedding:** numbers representing the meaning of a piece of text.
- **Vector database:** a store of embeddings that can be searched by meaning.
- **Semantic search:** finding text by meaning rather than exact words.
- **Re-ranking:** a second pass that puts the best results first.
- **Citation:** the source an answer was based on.
- **Prompt injection:** hidden instructions inside content that try to hijack the model.

Take the course for what you just read

Take the free 10-question assessment and pass with 8 or more. Every course you pass counts towards a Pathway Certificate issued by Shravya Digital, earned when you pass all five courses in the pathway: <https://digital.shravyaworld.org/courses/retrieval-augmented-generation>

Sources

Checked against these primary sources on 8 October 2026. Rules change: if something here looks out of date, trust the source.

- Lewis et al., "Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks" (2020) (<https://arxiv.org/abs/2005.11401>)
- IBM: What is retrieval-augmented generation? (<https://www.ibm.com/think/topics/retrieval-augmented-generation>)
- OWASP Top 10 for LLM Applications: Prompt injection (<https://genai.owasp.org/llmrisk/llm01-prompt-injection/>)