Retrieval Augmented Generation (RAG)

Bridging Frozen Weights with Fresh Data

Iverina Ivanova & Gemini 3 Pro

Goethe Universität Frankfurt

January 12th, 2026

Recap: The Frozen Encyclopedia

Recall from Karpathy (Weeks 7-9):

- Training is like printing a massive encyclopedia.
- It is expensive and takes months.
- **Result:** The model's knowledge is frozen at the training date.

1. The Cutoff Issue
Model trained in 2024 doesn't know the 2025 election results.

2. The Specialization Issue
Model read the internet, but not *your* seminar paper.

The Problem: The Confident Hallucination

We discussed **hallucinations** and the different strategies of mitigating them.

When an LLM doesn't know the answer, for example, because of its cutoff date, it relies on probability to guess the next word.

The Problem: The Confident Hallucination

Scenario: You ask a model trained on data up to Dec. 2024:

"Who is the current Chancellor of Germany?"

The Issue

The model does not know.

The Reaction

It relies on probability.

- It might guess **"Angela Merkel"** (most frequent in training data).
- It might guess **"Olaf Scholz"** (last known state).
- **It cannot know the new reality without access to fresh data.**

The Problem: The Confident Hallucination

How do we fix this without retraining the whole model?

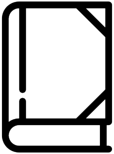
Strategies for mitigating hallucinations

- Tool calling (e.g., web search)
- Code execution

Another strategy: RAG (Retrieval Augmented Generation)

Concept: Don't force the model to rely on its long-term memory. Give it access to external sources to consult before it answers the question.

Standard LLM



Closed-Book Exam
Relies only on memorized facts.

RAG

Open-Book Exam
Allowed to browse the textbook (your PDF) before answering.

Why are we learning this now?

RAG brings the core topics we've covered so far together:

1. **Vector Semantics:** How does the LLM find the right piece of information in a document to answer our question? It uses *embeddings* and *cosine similarity* to retrieve the relevant data.
2. **LLM Generation:** Once the data is retrieved from the external sources, the generative LLM considers it before formulating its answer.
3. **Ethics & Reliability:** RAG helps ground AI in facts, reducing hallucinations and making sources traceable (citations).

Under the Hood: Two Types of Memory

RAG combines two types of knowledge:

1. Parametric Knowledge
(Internal Memory)

- What the model learned during pre-training.
- *Examples:* grammar, vocabulary, general facts (e.g., "Paris is in France").
- **Hard & expensive to update.**

2. Non-Parametric Knowledge
(External Source)

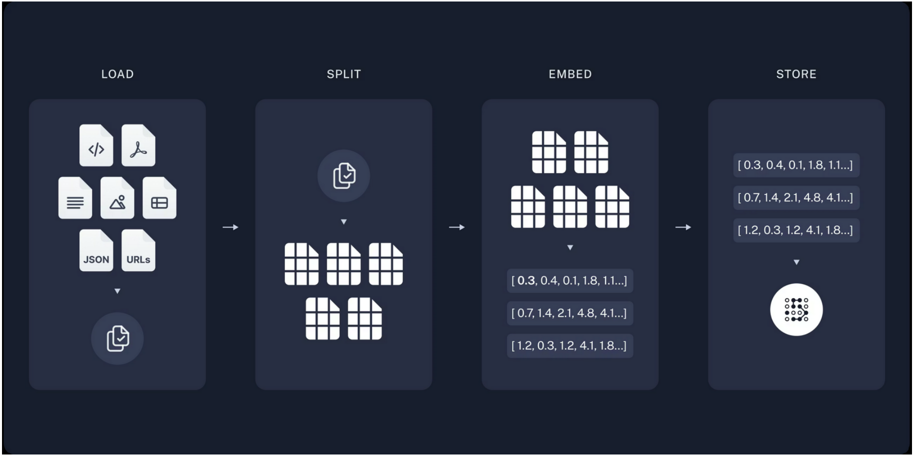
- Your external data (PDFs, databases).
- The model looks this up in real-time.
- **Easy to update** (just add a new PDF).

The Standard RAG Pipeline: 6 Steps

How do we build this system? It is a process of translation and search.
Simple RAG Demo

1. **Load:** Import your documents (PDFs, txt).
2. **Split (Chunk):** Break text into small pieces.
3. **Embed:** Turn text into vectors.
4. **Store:** Save vectors in a database.
5. **Retrieve:** Find the most similar chunks to the user's question.
6. **Generate:** The LLM writes an answer using the retrieved chunks.

How does RAG work?



Source: Langchain. URL: https://python.langchain.com/docs/tutorials/rag/

Steps 1 & 2: Load and Split

1. Loading (Real-world data is messy) Challenge: Computers see a stream of characters. Tables, footnotes, and multi-column layouts often break the reading process. If you feed garbage formatting into the RAG, you get garbage answers.

2. Splitting (The Chunking Dilemma) Challenge: How big should a piece be?

The Trade-Off: Context vs. Precision

- **Too Small (e.g., 1 sentence):** *Risk:* **Loss of Context.** (Example: A chunk says *He disagreed.* → The LLM doesn't know who *He* is or what he disagreed with.)
- **Too Large (e.g., 5 pages):** *Risk:* **Noise.** The retrieval might grab the right page, but the LLM gets distracted by 4 pages of irrelevant info surrounding the answer.

Steps 1 & 2: Load and Split

Goal: Find the *goldilocks* zone: Enough text to make sense, small enough to be precise.

Best practices

- fixed-size chunking (e.g., 512 tokens) with a chunk overlap (e.g., 50 tokens) that ensures context continuity.
- document-based chunking (e.g., using specialized libraries that segment the document intelligently based on its layout and data structures)

Steps 3 & 4: Embed and Store

Recall Session 5 (Vector Semantics): Computers cannot *read* text; they understand numbers.

- We use an **embedding model** to turn our text chunks into lists of numbers (embeddings/vectors).
- We create indexes on the embeddings and store both the chunks and their corresponding embeddings in a **vector database**.

"Friedrich Merz was elected chancellor on May 6th, 2025." $\rightarrow$ $[0.1, -0.5, 0.8, ...]$

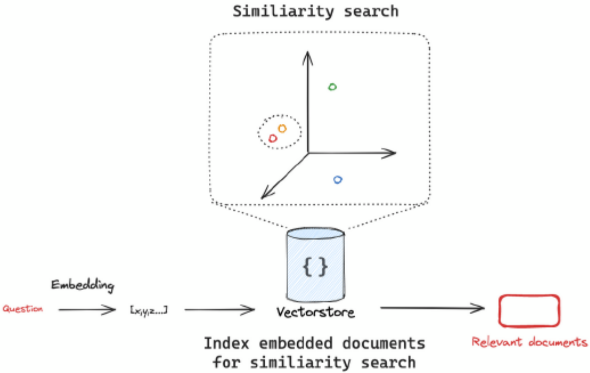
This allows us to search by **meaning**, not just by keyword matching.

Step 5: The Mechanism (Cosine Similarity)

User Question: "Who is the current Chancellor of Germany?"

1. **Vectorize:** Convert the question to a vector.
   $[0.1, -0.9, 0.4...]$
2. **Compare:** Embeddings are represented as points in a vector space. Points which are clustered together are more similar in meaning.
   Small angle = High similarity.
3. **Retrieve Top-K:** Grab the top closest chunk to the question.
   Match found: "Election results of 2025..."

Similarity search



Step 6: Context Injection

We don't just ask the question. We build a **Composite Prompt**.

The Hidden Prompt Structure

1. System Instruction:
You are a helpful assistant. Answer using ONLY the provided context.

2. Injected Context (The Retrieval):
"...on January 15th, Friedrich Merz was elected..."

3. User Question:
"Who is the current Chancellor?"

Result: The LLM generates a factual answer based on the injected context.

Summary & Next Steps

Why RAG matters:

- It allows us to chat with our own datasets without retraining the model.
- It provides citations (we know *where* the answer came from).
- It connects the frozen model to fresh data.

Coming up:

- We'll discuss some RAG-related challenges.
- We'll explore RAG implementations:
  - **NotebookLM** (Google's RAG tool)
  - Code implementations with state-of-the-art Python libraries

References

- Introduction to RAG in AI development. URL: https://learn.microsoft.com/en-us/azure/databricks/generative-ai/tutori
- Chunking Strategies for LLM Applications. URL: https://www.pinecone.io/learn/chunking-strategies/
- ChunkVisualizer. URL: https://chunkviz.up.railway.app/explanation
- Sentence Similarity. URL: https://huggingface.co/tasks/sentence-similarity

Meta-Analysis

This slide deck was developed in a multi-step process involving **Gemini 3 Pro**.

1. Providing the model with the necessary background of the course purpose and the target audience.
2. Asking Gemini to explain RAG in simple terms and to relate its relevance to the previously discussed topics.
3. **Human Verification:** The specific definitions and workflows were cross-referenced with the technical documentations from the source pages to ensure factual accuracy.
4. Refining certain explanations by making them less wordy or by using visual representations.