Phase 3: Retrieval-Augmented Generation (RAG)

> **Goal of Phase 3**: Teach the model to answer using **your data**, not its memory.

We will move **slowly and safely**. No agents. No tricks. Only deterministic pipelines.

3.0 Big Picture (Before Code)

Why Phase 3 Exists

LLMs have three serious limits: - They **forget** between sessions - They **hallucinate** confidently - They **don't know your private data**

RAG fixes this by adding a **retrieval step** before generation.

Mental Model (Lock This In)

```
User Question
   ↓
Convert to embedding
   ↓
Search vector database
   ↓
Retrieve relevant chunks
   ↓
Inject into prompt
   ↓
LLM answers using retrieved context
```

The LLM is **not allowed to guess**. It answers only from retrieved text.

3.1 Key Components of RAG (Plain English)

1. Documents

Your knowledge source: - PDFs - Text files - Notes - Web pages

2. Text Splitter

Large documents are split into **small chunks** so retrieval is accurate.

3. Embeddings

Text → numbers (vectors) that capture meaning.

Similar meaning → vectors close together.

4. Vector Database (ChromaDB)

Stores vectors and allows **semantic search**.

5. Retriever

Given a question, fetches the **most relevant chunks**.

6. Prompt + LLM

Retrieved chunks are injected into the prompt, then the LLM answers.

3.2 Tools We Will Use (Phase 3)

- **Ollama** (local LLM)
- **LangChain** (pipelines)
- **ChromaDB** (vector database)
- **Local embeddings** (no API keys)

Everything stays **offline and free**.

3.3 First RAG Experiment (Very Small)

What We Will Build FIRST

Not a chatbot.

We will build:

> **Question → Retrieve text → Answer**

No memory yet. No agents yet.

3.4 Step 1: Create Sample Knowledge Base

Create a file called `knowledge.txt`:

```
LangChain is a framework for building applications powered by large language
models.

It helps developers create chains, agents, and retrieval systems in a structured
way.

RAG stands for Retrieval-Augmented Generation. It reduces hallucination by
grounding answers in documents.
```

3.5 Step 2: Load and Split Documents

```python
from langchain.document_loaders import TextLoader
from langchain.text_splitter import RecursiveCharacterTextSplitter

loader = TextLoader("knowledge.txt")
documents = loader.load()

splitter = RecursiveCharacterTextSplitter(
    chunk_size=100,
    chunk_overlap=20
)

chunks = splitter.split_documents(documents)

print(len(chunks))
```

Why splitting matters: - Smaller chunks → better retrieval - Less noise in answers

---

3.6 Step 3: Create Embeddings (Local)

```python
from langchain_ollama import OllamaEmbeddings

embeddings = OllamaEmbeddings(
    model="nomic-embed-text"
)
```

> This converts text into vectors using a **local embedding model**.

3.7 Step 4: Store in ChromaDB

```
from langchain.vectorstores import Chroma

vectorstore = Chroma.from_documents(
    documents=chunks,
    embedding=embeddings,
    persist_directory="./chroma_db"
)
```

Now your **knowledge is searchable**.

3.8 Step 5: Retrieve Relevant Chunks

```
retriever = vectorstore.as_retriever(search_kwargs={"k": 2})

query = "What is RAG?"

results = retriever.get_relevant_documents(query)

for doc in results:
    print(doc.page_content)
```

You should see **only relevant text**, not guesses.

3.9 Step 6: RAG Prompt + LLM

```
from langchain_core.prompts import PromptTemplate
from langchain_ollama import ChatOllama

llm = ChatOllama(model="llama3", temperature=0.1)

rag_prompt = PromptTemplate(
    input_variables=["context", "question"],
    template="""
Answer the question using ONLY the context below.
If the answer is not in the context, say "I don't know".

Context:
{context}
```

```
Question:
{question}
"""
)
```

3.10 Final RAG Chain (Deterministic)

```
context = "\n\n".join([doc.page_content for doc in results])

response = llm.invoke(
    rag_prompt.format(
        context=context,
        question=query
    )
)

print(response.content)
```

The LLM is now **grounded**.

3.11 What You Just Learned

- How RAG actually works
- Why hallucination drops
- Why vector databases matter
- How retrieval controls answers

Phase 3 Checkpoint

You should now understand: - Difference between memory and retrieval - Why RAG is industry-critical - How ChromaDB fits into LangChain

---

Next (Only When Ready)

Phase 3.2: - RAG + Conversation Memory - Query rewriting - Source attribution

Do NOT rush. Master this first.