

Building a RAG Pipeline with LangChain

Objective

Build a functional RAG (Retrieval-Augmented Generation) pipeline that demonstrates how to retrieve relevant information from a knowledge base and use it to generate accurate, contextual responses with an LLM.

1. Project Setup & Requirements

Technical Requirements

- **Python 3.9+**
- **Required Libraries** (requirements.txt is provided) :

API Keys

- Obtain an OpenAI API key (or use local alternatives like Ollama with Llama 2/3)
- Store securely in .env file:
OPENAI_API_KEY=your_key_here

2. Core Components to Implement

A. Document Loading & Processing

- **Choose at least 3 document type:** PDFs (research papers), web articles, or textbook chapters, videos, or images
- **Implement document loading** using LangChain's document loaders
- **Split documents** into chunks (experiment with different chunk sizes and overlap)

B. Vector Database Setup

- **Create embeddings** using OpenAI or open-source models (all-MiniLM-L6-v2)
- **Store in ChromaDB** with metadata (source, page number, etc.)
- **Implement similarity search** with configurable k-value

C. RAG Pipeline Construction

- **Build retrieval chain** that:
 1. Takes user query (a simple UI)
 2. Retrieves relevant chunks
 3. Formats context for LLM
 4. Generates response with citations
- **Implement prompt engineering** with:
 - Contextual instructions
 - Citation requirements
 - "Don't know" fallbacks

D. Evaluation & Testing

- **Create test queries** (some within knowledge base, some outside)
- **Compare outputs:** RAG vs. base LLM responses
- **Measure accuracy** and hallucination reduction

E. Create evaluation metrics (Good to have but not required for next week's presentation)

- **Faithfulness to source**
- **Relevance to query**
- **Hallucination detection**

Presentation Requirements (15-Minute Group Presentation Must Include):

1. **Demo (5 minutes)**
 - Live demonstration of your RAG system
 - Show before/after comparison
2. **Architecture Diagram**
 - Visual representation of your pipeline
 - Key design decisions explained
3. **Challenges & Solutions**
 - Document chunking strategies tried
 - Retrieval quality issues addressed
 - Prompt engineering iterations
4. **Quantitative Results**
 - Accuracy metrics (if implemented)
 - Response time measurements
 - Success/failure cases analysis
5. **Code Submission**
 - Well-documented Jupyter notebook or Python scripts

Learning Resources (Essential Reading)

1. [LangChain RAG Documentation](#)
2. [Retrieval-Augmented Generation Paper](#)
3. [ChromaDB Documentation](#)

Remember: The goal is to understand the RAG components deeply, not just make it work. Be prepared to explain WHY you made each design choice!