MatSoc, IITK

Architecting Intelligence

Capstone 1

Mentors: Basudev Mohapatra
Khushal Wadhwa
Mansi Ghodke

Due Date: January 17th, 2026

End Evaluation

---

1 Introduction

With the rapid growth of academic literature, tools that assist in reading and understanding research papers have become increasingly important. However, a standard language model by itself is not sufficient for reasoning over research papers. In the absence of explicit retrieval and document-level grounding, such models may overlook relevant sections or produce answers that are not directly supported by the source material. To overcome these challenges, more advanced techniques are required that integrate document retrieval with language generation. In this assignment, you will design and implement a **Retrieval-Augmented Generation (RAG) pipeline** that can intelligently answer questions about research papers you provide it with, grounded strictly in the content of the uploaded documents. Your system should be capable of:

- Parsing research papers in PDF format
- Structuring and embedding their contents
- Retrieving relevant sections based on user queries
- Generating faithful, grounded responses
- Producing flowcharts / mind maps based on retrieved content using Mermaid Chart.

2 Problem Statement

Design an AI Research Assistant that allows a user to upload a research paper in PDF format and ask questions such as:

- **"What is the main contribution of this paper?"**
- **"What loss function is used?"**
- **"Summarize the experimental setup."**
- **"What limitations does the paper mention?"**
- **"Create a flowchart of the proposed method."**

The system must retrieve relevant information from the papers and use it as context for generating answers. The LLM must not rely on external knowledge.

3 System Overview

Your system should implement the following pipeline:

- Research Paper PDFs
- Structured PDF Parsing
- Section-Aware Chunking
- Embedding Generation
- Vector Database
- User Query
- Targeted Retrieval
- Context Assembly
- LLM-Based Answer Generation
- Mermaid Diagram Generation

4 System Architecture & Requirements

4.1 Prerequisites API Setup

Goal: Obtain access to a Large Language Model (LLM) service that will power your application.

Baseline: I relied on Google Gemini. You must obtain a free API key to make it work.

Action: Go to Google AI Studio and create a new API key for your project.

Your Task: Secure access to an LLM API. Follow the baseline and use Gemini 2.5 Flash.

4.2 Document Ingestion (PDF Parsing)

Goal: Extract text from PDF files into a format suitable for processing.

Baseline: I used **pypdf** to extract raw text page-by-page.

Your Task: Implement a PDF parser. You may stick to simple text extraction or attempt more complex layout analysis (preserving tables/headers).

Suggested Packages/Libraries:

- PyMuPDF (fitz): Faster and often more accurate text extraction than pypdf.
- pdfplumber: Excellent for extracting data from tables if your papers are data-heavy.

- Unstructured: A powerhouse library that handles cleaning, removing headers/footers, and advanced partitioning.
- LlamaParse: specialized for complex documents with tables and charts (requires API).

4.3 Chunking & Splitting

Goal: Break the long document text into smaller, manageable "chunks" for the embedding model.

Baseline: I used **RecursiveCharacterTextSplitter** from LangChain with a sliding window (chunk size 1000, overlap 200).

Your Task: Implement a chunking strategy. The baseline splits by character count, but you can explore semantic splitting.

Suggested Packages/Libraries:

- SemanticChunker (LangChain): Splits text based on meaning (embedding similarity) rather than fixed character counts.
- NLTK / Spacy TextSplitter: Splits text by actual sentences rather than arbitrary characters, ensuring grammar isn't broken.
- MarkdownHeaderTextSplitter: If you convert the PDF to Markdown first, this splits text by headers (Abstract, Intro, Method).

4.4 Embeddings & Vector Storage

Goal: Convert text chunks into numerical vectors and store them for similarity search.

Baseline: I used **Google GenAI Embeddings (models/text-embedding-004)** and stored them in **ChromaDB**.

Your Task: Select an embedding model and a vector database.

Suggested Packages/Libraries:

Embeddings:

- HuggingFaceEmbeddings (e.g., all-MiniLM-L6-v2): Runs locally, free, good for privacy.
- OpenAIEmbeddings (text-embedding-3-small): Industry standard, paid API.
- Cohere Embed: Specifically optimized for RAG and retrieval tasks.

Vector Databases:

- FAISS (Facebook AI Similarity Search): Extremely fast, runs locally in-memory.
- Pinecone / Weaviate: Managed cloud vector databases (good for scaling).
- Qdrant: High-performance open-source vector search engine.

4.5 Retrieval Strategy

Goal: Find the most relevant chunks for a user's question.

Baseline: I implemented a Hybrid Search combining Dense Retrieval (Vector similarity (**ChromaDB**) for semantic understanding) and Sparse Retrieval (Keyword matching (**BM25**) for exact term matching.)

Your Task: Implement a retrieval mechanism. You typically need the top 3-5 most relevant chunks.

Suggested Packages/Libraries:

- Pure Vector Search: Skip BM25 and rely solely on Cosine Similarity (simpler implementation).
- Re-ranking (Cross-Encoder): Retrieve 20 documents using vector search, then use a Cross-Encoder (like bge-reranker) to sort them by accurate relevance.
- Parent Document Retriever: Retrieve small chunks for accuracy, but feed the LLM the larger "parent" chunk for better context.
- Ensemble Retriever: Combine results from multiple different retrievers using weighted scoring.

4.6 LLM & Answer Generation

Goal: Synthesize the retrieved context into a coherent answer.

Baseline: I used **Gemini 2.5 Flash** with a specific prompt structure.

Your Task: Integrate an LLM to generate answers. The LLM must be instructed to only use the provided context.

Suggested Packages/Libraries:

- OpenAI (GPT-4o / GPT-3.5): The most common standard for RAG.
- Anthropic (Claude 3.5 Sonnet): Known for having a large context window and better reasoning for research papers.
- Groq (Llama 3): Ultra-fast inference, great for real-time chat experiences.
- Ollama (Local LLMs): Run Llama 3 or Mistral locally on your machine (free, private).

4.7 Diagram Generation (Mind Map)

Goal: Visualize the paper's logic (e.g., Methodology steps) as a flowchart.

Baseline: Direct Prompt Engineering. The LLM is asked to output raw **Mermaid JS code (graph TD)** based on the retrieved text.

Your Task: Generate a visualization of the paper's content. Go with the baseline

4.8 User Interface (Frontend)

Goal: A clean interface for uploading PDFs and chatting.

Baseline: I used **Streamlit.** It handles file uploads, chat history, and rendering Mermaid diagrams via HTML components.

Your Task: Build a frontend. It must support PDF upload and a chat window.

Suggested Packages/Libraries:

- Gradio: Very popular for ML demos; easier to set up than Streamlit for simple Input/Output flows.
- Chainlit: Built specifically for Chatbots/RAG. Has built-in "Chain of Thought" display.
- FastAPI + React: If you want to separate the backend and frontend completely (Professional Software Engineering approach).

5 Deliverables

Each group will submit the Colab notebook by Saturday EOD, along with a 30s video to show its deployment. Do try to deviate from the baseline.