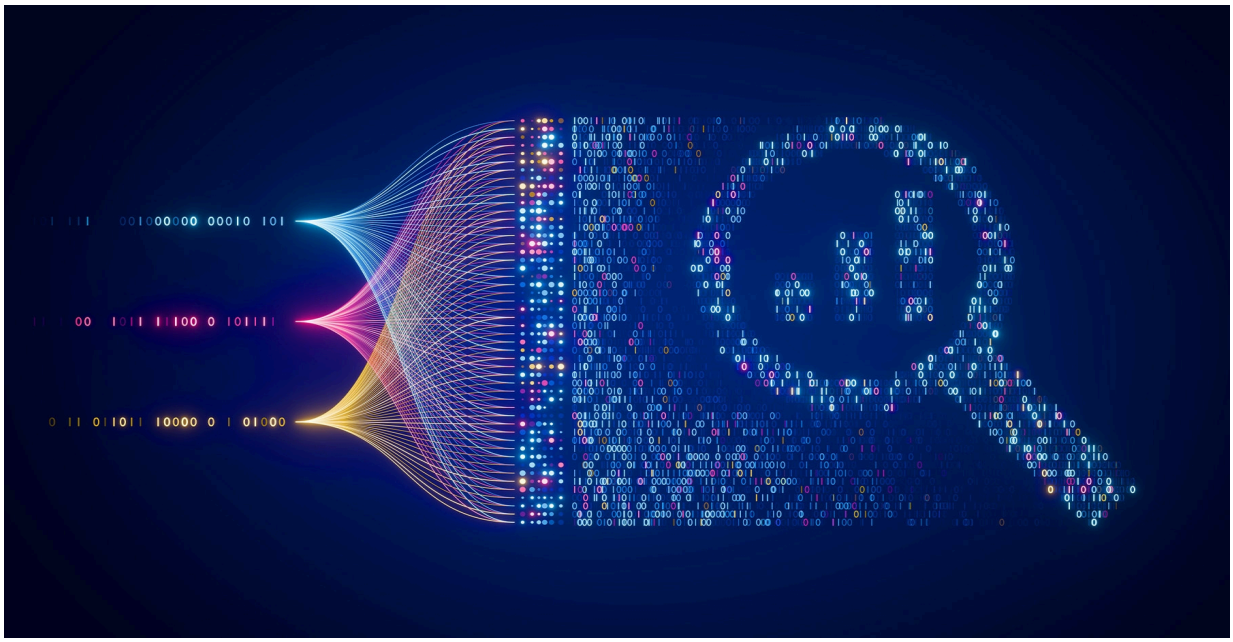
Project Title: Research Paper Summarizer and QA Chatbot

by Ritankar Das



Date: 05/11/2025

Table of Contents

1.0 Project Overview and Objectives

1.1 Introduction

The project addresses the challenge of academic information overload by developing an intelligent system that can efficiently summarize research papers (primarily in PDF format) and provide accurate, context-aware answers to user questions. This is achieved by leveraging the LangChain framework. The final product will serve as a specialized, reliable research assistant.

1.2 Core Objectives

- **Document Ingestion:** Successfully load, parse, and process various research document formats, focusing on **PDF files**.
- **Summarization:** Implement both **concise (abstract-like)** and **detailed (section-by-section)** summarization capabilities.
- **Retrieval-Augmented Generation (RAG):** Build a robust RAG pipeline to enable precise Question-Answering (QA) strictly **grounded in the source documents**.
- **Factual Grounding:** Minimize LLM "hallucinations" by ensuring all QA responses are backed by retrieved document chunks, including optional citation of the source page/chunk.
- **Interactive Interface:** Develop a user-friendly interface using Streamlit .

2.0 Methodology and Architecture

The system is engineered using the RAG architecture, with **LangChain** serving as the primary orchestration framework for connecting the data source, the vector store, and the Large Language Model (LLM).

2.1 Key LangChain Components

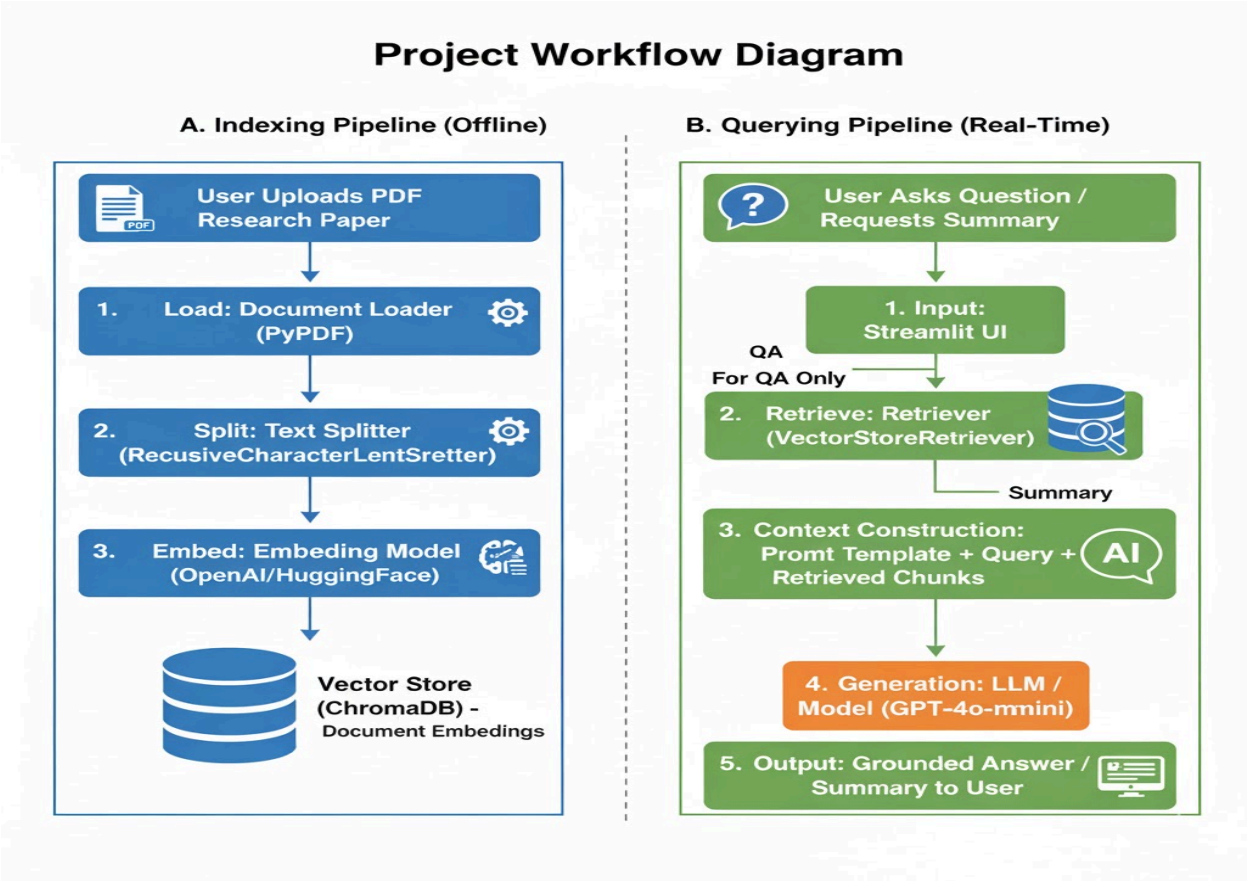
The following components are essential for the RAG pipeline's functionality:

Component	Purpose	Examples/Libraries
Document Loaders	To ingest data from various sources (e.g., PDF files).	PyPDFLoader
Text Splitters	To break large documents into smaller, semantic chunks suitable for embedding and retrieval.	RecursiveCharacterTextSplitter
Embedding Models	To convert text chunks and user queries into dense **vector representations**.	OpenAIEmbeddings

Vector Stores	To efficiently store and enable fast similarity search over document embeddings.	ChromaDB
Retrievers	To fetch the most **relevant document chunks** based on the vectorized user query.	VectorStoreRetriever
LLMs/Chat Models	The core model for generating summaries and answering questions based on the retrieved context.	OpenRouter ,gpt-4o-mini
Chains/Agents	To define and manage the sequence of steps for summarization and QA workflows.	summarize_, QA

2.2 Workflow Diagram

The project workflow is divided into two distinct phases: Indexing (Data Preparation) and Querying (Real-time Interaction).



A. Indexing Pipeline (Data Preprocessing)

This is an offline process performed when a new paper is uploaded.

1. **Load:** User uploads PDF research paper -> **Document Loader**.
2. **Split:** Document is chunked with controlled overlap -> **Text Splitter**.
3. **Embed:** Chunks are converted into vectors ->**Embedding Model**.
4. **Store:** Vectors are persisted into the database -> **Vector Store** .

B. Querying Pipeline (Summarization and QA)

This is the real-time interaction loop.

1. **Input:** User asks a question or requests a summary.

2. **Retrieve (for QA):** User query is embedded, and the most similar document chunks are fetched from the Vector Store ->**Retriever**.
3. **Context Construction:** Retrieved chunks are combined with the user query into a single, comprehensive **Prompt**.
4. **Generation:** The LLM uses the prompt/context to generate a final, grounded answer or a summary -> **LLM/Chat Model**.
5. **Output:** The generated, grounded response is presented to the user.

3.0 Project Components and Implementation Details

3.1 Summarization Module

The module must handle research papers that significantly exceed the standard LLM context window.

- **Strategy:** Implement the **'refine'** summarization technique via the LangChain `summarize_service`. This chain processes the paper iteratively:
    - *Step 1:* Summarizes the first chunk.
    - *Step 2:* Uses the previous summary and the next chunk to **refine** the summary.
    - *Step 3:* Repeats the refinement process until the entire document is covered.
- **Output Types:**
    - **Executive Summary:** A short, paragraph-length summary (abstract replacement).

3.2 Question-Answering Chatbot Module

This module is the core application of the RAG pipeline, ensuring all answers are strictly factual.

- **Primary Chain:** Use the `QA_service` chain for seamless retrieval and generation.
- **Vector Store: Chroma** will be used for rapid, local vector indexing and similarity search.
- **Prompt Engineering:** Custom system prompts will be crucial to control LLM behavior:
    - **Instruction:** Explicitly instruct the LLM to **use only the provided context** for answering.
    - **Guardrail:** Instruct the LLM to state: "I cannot find the answer in the provided documents" if the information is unavailable.
    - **Citation:** Integrate a mechanism to include the source document chunk ID or page number in the final output.

3.3 Technology Stack (Page 5)

The project relies on modern, well-maintained libraries within the Python ecosystem.

Category	Technology/Library	Purpose
Framework	**LangChain**	Orchestration, chain management, and component integration.
Frontend/UI	**Streamlit**	Creating a clean, interactive web interface for user interaction.
Backend	**FastAPI**	Primary development framework for logic and scripting.
LLM	GPT-4o-mini	Generative AI models for text generation tasks.
Vector DB	**Chroma**	Storing and searching document embeddings efficiently.
Parsing	PyPDF	Loading and extracting text content from PDF files.

4.0 Evaluation and Future Enhancements

4.1 Evaluation Metrics

To ensure the system's effectiveness and reliability, performance will be measured across three key areas:

- **Summarization Quality:** Measured using **ROUGE scores** (R-1, R-2, R-L) by comparing generated summaries against a small corpus of human-written abstracts.
- **QA Accuracy (Groundedness):** A custom **Groundedness Score** will be used to measure the percentage of chatbot answers that are verifiable, word-for-word, within the retrieved source context.
- **System Latency:** Measure the time taken from query input to final response generation, aiming for sub-5 second P95 latency.

4.2 Future Enhancements

The following features are planned for future development beyond the scope of the initial version:

- **Multi-Document QA:** Capability to query and summarize information across **multiple** uploaded research papers simultaneously.
- **Advanced Citation:** Automatically generate clickable, inline citations (e.g., within the chatbot's response text.
- **Hybrid Search:** Integrate a combination of vector similarity search and keyword search (**BM25**) for improved, resilient retrieval.
- **User Management:** Implement secure user authentication and persistent storage of uploaded documents and chat history.