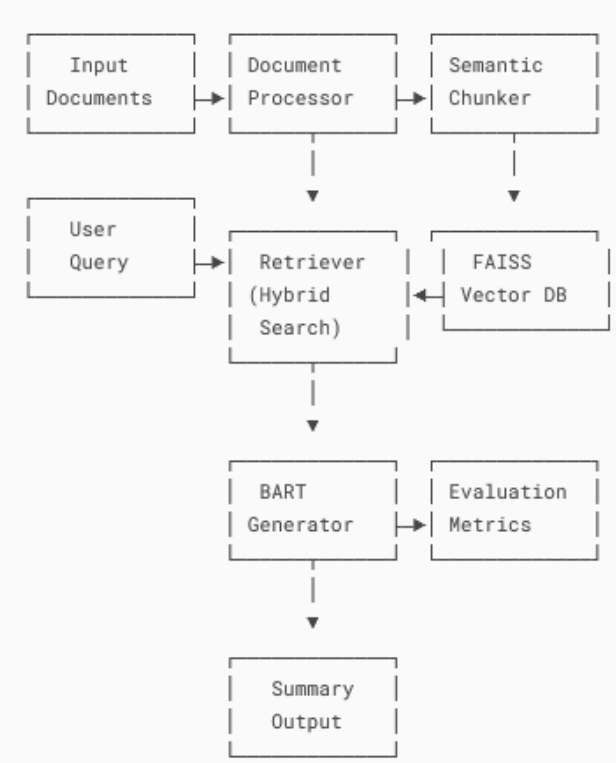
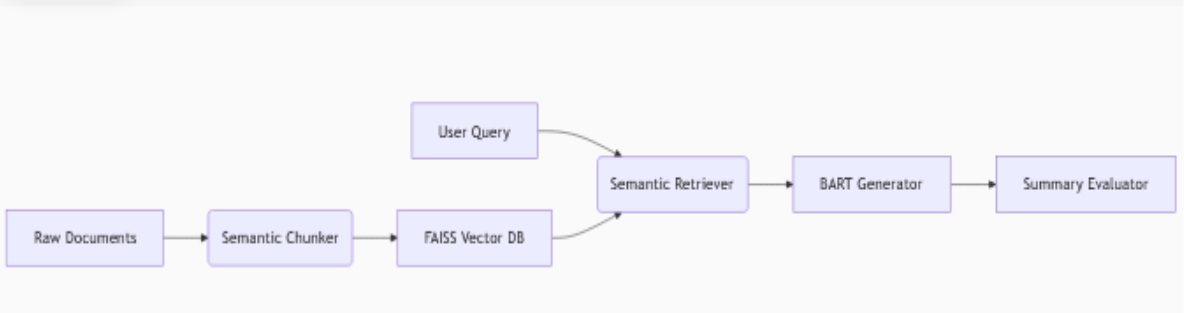
Retrieval Augmented Generation

DOCUMENT SUMMARIZATION





Sofia Raheel

Genesys Research Lab Submission(Internship)

GitHub Link:

https://github.com/sofiaRaheel/Document-Summarization-using-Retrieval-Augmented-Generatio-RAG-

INTRODUCTION TO RETRIEVAL-AUGMENTED GENERATION (RAG)

Retrieval-Augmented Generation (RAG) is an advanced technique in natural language processing that combines retrieval-based methods with generative language models. Unlike traditional models that rely solely on internal parameters, RAG systems fetch relevant external information at inference time, enabling more accurate, current, and explainable outputs.RAG operates in two main steps:

1. **Retrieval**: A retriever searches a knowledge base to find relevant documents or passages.
2. **Generation**: A language model generates responses conditioned on the retrieved content.

This approach enhances factual accuracy and adaptability, making it ideal for tasks like question answering, chatbots, legal research, and enterprise search.

INTRODUCTION TO THE PROJECT

This project aims to build a RAG system for Summary Generation, combining external knowledge retrieval with generative AI to deliver accurate, context-aware responses. Key steps include setting up a vector database, implementing a dense retriever, using or fine-tuning a generative model, and designing a modular query-response pipeline. The project also evaluates performance, scalability, and real-world applications.

CODEBASE EXPLANATION

This project implements a Retrieval-Augmented Generation (RAG) system using a modular pipeline designed to provide accurate, context-aware answers by combining document retrieval with large language models (LLMs). The codebase is structured into three primary stages: **Document Ingestion & Embedding**, **Query-Time Retrieval**, and **Context-Aware Generation**.

ARCHITECTURE:

1. **Document Ingestion & Embedding:**
    a. Documents are first loaded (PDFs, text, etc.) and split into manageable

chunks using LangChain's `RecursiveCharacterTextSplitter`. These chunks are then embedded into numerical vectors using models like `all-MiniLM-L6-v2` or OpenAI's `text-embedding-ada-002`. The resulting vectors are stored in a vector database such as FAISS or ChromaDB for fast similarity search.

   b. Why: Traditional keyword search misses meaning. Embeddings enable semantic understanding, retrieving text based on meaning, not exact words.

2. **Query-Time Retrieval:**
   a. When a user submits a query, it's embedded the same way as the documents and compared with stored vectors to retrieve the most relevant chunks. LangChain's retriever tools handle this efficiently, returning only the most semantically aligned texts.
   b. Why: This allows the system to surface conceptually related documents, even if they don't share exact terms with the query.
3. **Context-Aware Generation:**
   a. The retrieved text chunks and the user query are passed to an LLM (e.g., GPT-4 or Mistral) via a framework like LangChain's `RetrievalQA` or `ConversationalRetrievalChain`. The LLM then generates a grounded, accurate answer using the context provided.
   b. Why: LLMs alone may hallucinate or be outdated; by grounding responses in retrieved facts, the system improves accuracy and traceability.

WHY THIS APPROACH?

- **More accurate**: Retrieval ensures factual grounding.
- **More up-to-date**: Content updates are as simple as adding/removing documents.
- **More trustworthy**: Users can trace answers back to original sources.
- **Highly modular**: Each component can be tuned or swapped independently.

CONCLUSION

This RAG pipeline integrates retrieval and generation in a seamless, scalable way. It overcomes the limitations of standalone LLMs by grounding outputs in real data, making it ideal for applications requiring both precision and explainability.