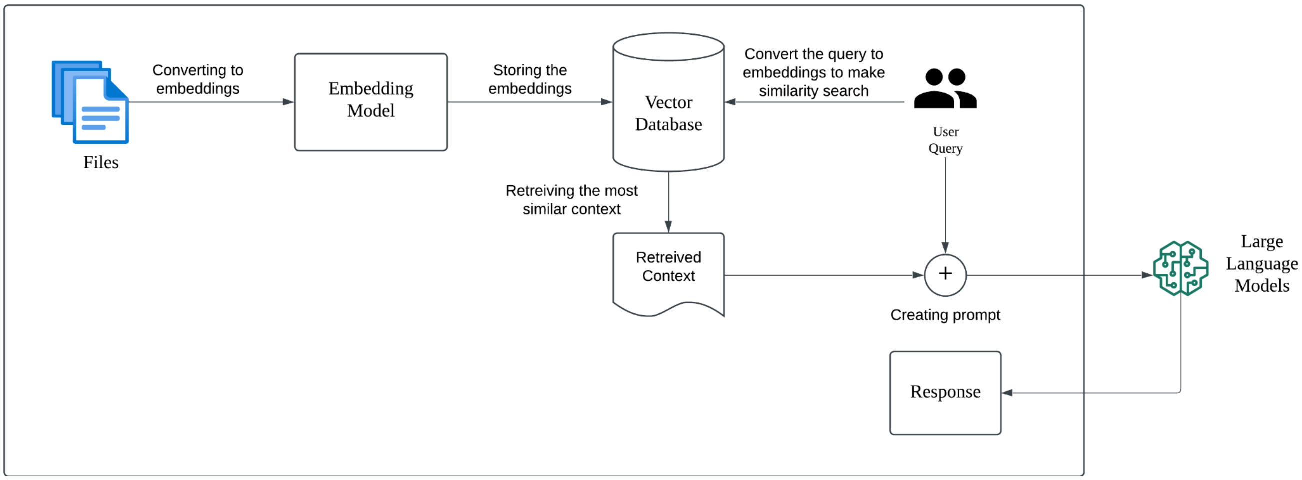
RAG APPLICATION

High Level Architecture:



This system processes various source files, including PDFs, audio and video transcripts, Stack Overflow data, and images, to generate responses using a retrieval-augmented generation (RAG) pipeline. Here's an overview of its workflow:

Key Components

1. **Embedding Creation:**
    - Source files are processed using an embedding model (**all-MiniLM-L6-v2**) to convert text into vector representations.
    - The generated embeddings are stored in a vector database (**FAISS**) for efficient retrieval.
2. **Query Processing:**
    - User queries are transformed into vector embeddings using the same model (**all-MiniLM-L6-v2**).
    - These embeddings are used to perform a similarity search in **FAISS**, retrieving the most relevant context.
3. **Response Generation:**
    - A prompt is constructed by combining the user query with the retrieved context.
    - The prompt is processed by a large language model (**gemini-1.5-flash**) to generate a coherent and accurate response.

Data Sources

1. Jenkins User Manual in PDF format.

2. Video explanations covering Jenkins pipelines.
3. Stack Overflow data for frequently asked questions (FAQs).

Implementation

1. Data Preprocessing

- **PDF Processing:** Text is extracted from PDF files on a page-by-page basis using **PyMuPDF**.
- **Image Extraction:** Images embedded within PDFs are also extracted using **PyMuPDF**.
- **FAQs Extraction:** Relevant FAQ data is scraped from Stack Overflow threads, extracting questions along with only the accepted answers. **(Library used : Selenium, BeautifulSoup)**
- **Video Processing:** Video content is converted into audio using the **MoviePy** library.
- **Audio Transcription:** The extracted audio is transcribed into text using the **SpeechRecognition** library, which leverages **Google Speech Recognition**.

2. Embedding Generation

- This system utilizes the **all-MiniLM-L6-v2** embedding model from **Hugging Face**.
- It is an **open-source** model, making it accessible and adaptable for various applications.

3. Vector Indexing

Why use a vector database?

- A vector database indexes and stores vector embeddings, enabling fast retrieval and efficient similarity searches.
- This system utilizes **Facebook AI Similarity Search (FAISS)** for vector indexing.
- FAISS can also be used to create an **on-premises** database for enhanced control and customization.

4. RAG Workflow

Source Processing

Various source files (e.g., text, PDFs, transcripts, images, and FAQs) are processed using an embedding model to convert their content into vector representations. These embeddings are stored in a vector database (e.g., **FAISS**) for efficient similarity-based retrieval.

In this system, two separate vector databases are maintained:

1. **User Manual vector store :** Contains embeddings of the **Jenkins User Manual** for retrieving relevant documentation.
2. **Stack Overflow vector store (FAQs):** Stores embeddings of **Stack Overflow threads**, including questions and accepted answers, for enhanced contextual retrieval.

- **Query Embedding:** User queries are transformed into embeddings using the same model applied to the source data.
- **Similarity Search:**
  - The vector database performs a similarity search to retrieve the most relevant embeddings (context) based on the user query.
  - **Image Retrieval :** Once the similarity search is complete, the metadata (which includes the page numbers of the **user manual**) is used to identify the corresponding page. This page number is then used to extract the relevant images stored in the directory.
- **Prompt Construction:** A prompt is generated by combining the user query with the retrieved context from two vector stores and retrieved images ensuring the model has the necessary background information.
- **Response Generation:** The **gemini-1.5-flash** language model processes the prompt to generate an informative and contextually accurate response.

5. Evaluation

- **Evaluation Dataset:**
  A dataset was created containing:
  - User queries and retrieval keywords for **retrieval evaluation**.
  - The same set of user queries along with actual responses for **generation evaluation**.
- **Retrieval Evaluation:**
  - **Metric:** Keyword matching.

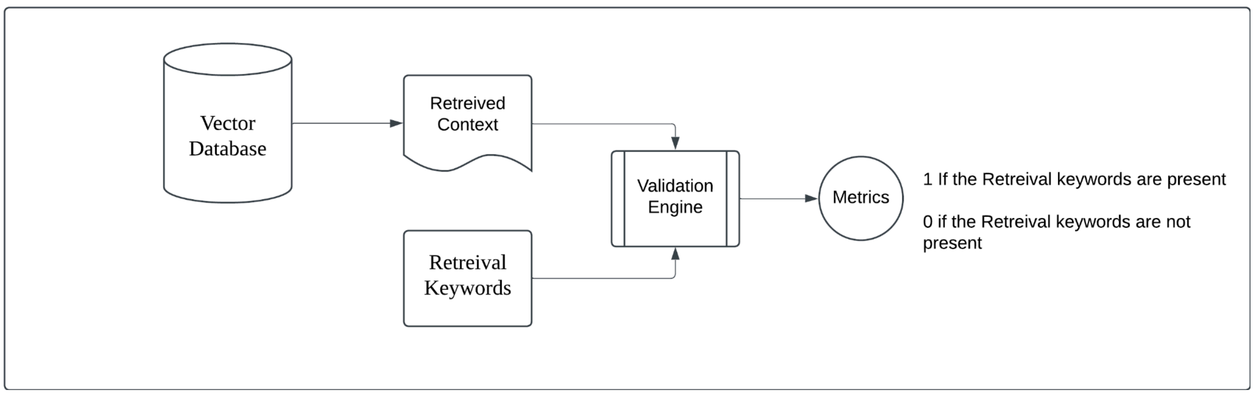
    - **Rationale:** Accurate content retrieval is essential for high-quality response generation. If the retrieved context contains the expected keywords, it is assigned a score of **1**, otherwise **0**.

Retrieval Evaluation: 90.00%

- **Generation Evaluation:**
    - **Metric:** Cosine similarity between the actual response and the generated response.

Generation Evaluation: 83.66%

Retrieval Evaluation Architecture:



Handling Multimedia

- **Image Processing:** Images embedded in PDFs are extracted using **PyMuPDF**.
- **Video Transcription:**
    - Videos are processed using **MoviePy** to extract audio.
    - The extracted audio is transcribed into text using **SpeechRecognition**, which leverages **Google Speech Recognition**.

Scaling

To scale RAG systems effectively:

1. **Distributed Vector Databases:** Use shared and replicated vector databases to handle large-scale retrieval efficiently.
2. **Load Balancing:** Deploy multiple instances of the embedding model and LLM behind a load balancer to distribute query processing evenly.

3. **Asynchronous Processing:** Implement asynchronous APIs and batch processing to handle multiple requests concurrently, reducing response latency.

GitHub Repository: https://github.com/Manokarthi2412/RAG

Video demo: **Link**

Tech stack : Python**,** Html, css, Flask, Faiss, Langchain