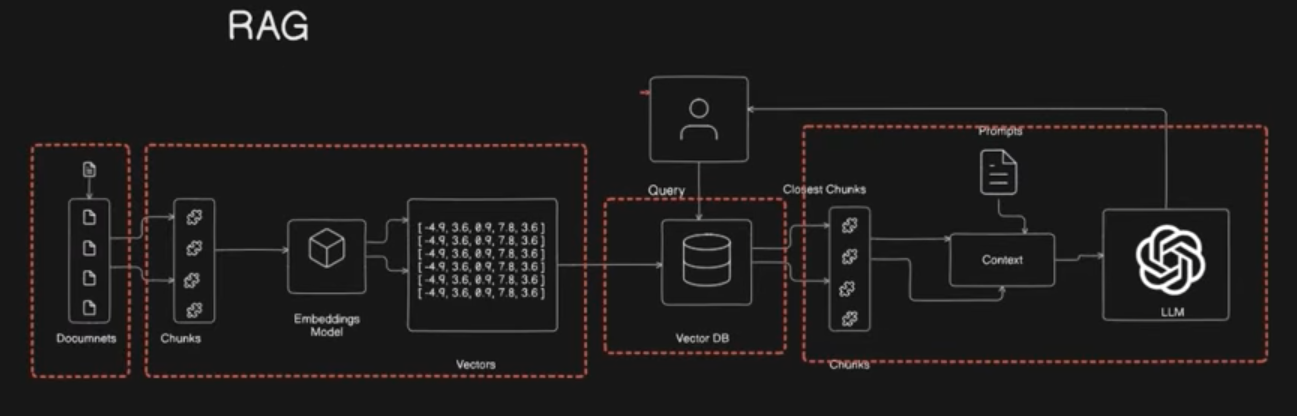
RAG PDF Chatbot - Workflow Progress Report

# 1. Introduction

This document explains the current progress of the RAG (Retrieval Augmented Generation) PDF Chatbot project. The chatbot allows users to upload PDFs and interact with them via a conversational interface built using Streamlit. The application leverages embeddings, a vector database, and an LLM for question answering.

# 2. Workflow Diagram



# 3. Workflow Steps

The chatbot workflow can be divided into the following steps:

* 1. Document Upload: Users upload one or more PDF documents.
* 2. Chunking: The PDFs are split into smaller text chunks for efficient processing.
* 3. Embedding Generation: Each chunk is converted into numerical vectors using HuggingFace embeddings.
* 4. Vector Store: The vectors are stored in a FAISS vector database (local).
* 5. Query Input: The user enters a query/question in the Streamlit UI.
* 6. Retrieval: The query is converted into embeddings and the most relevant chunks are retrieved from FAISS.
* 7. Context Formation: Retrieved chunks are combined as context for the LLM.
* 8. LLM Response: Google Gemini API (via LangChain wrapper) generates an answer based on the query + context.
* 9. Conversational Memory: Previous interactions are stored in session state to allow follow-up questions.
* 10. Output: The chatbot displays the answer in text form along with the conversation history.

# 4. Tools & Technologies

The following tools and libraries are currently used in the chatbot:

* Streamlit - Frontend UI for chatbot
* LangChain - Framework for RAG pipeline
* HuggingFace Embeddings - Embedding model
* FAISS - Vector database for similarity search
* Google Gemini API - LLM for generating answers
* Session State - For conversational memory management

# 5. Features Completed

* PDF upload and text chunking
* Embeddings creation with HuggingFace
* Vector search using FAISS
* Integration with Google Gemini API for responses
* Streamlit UI with chat interface
* Conversation history stored in session state
* Option to download conversation log

# 6. Features Pending (Next Steps)

* Add citations with page numbers and clickable links
* Implement session ID-based conversational memory for multiple users
* Generate synthetic Q&A dataset from PDFs
* Evaluate chatbot accuracy
* Allow the LLM to remember things like names, facts, or follow-ups even if not in the PDF
* **Imports**
* You import standard libraries (os, datetime, pandas, base64) for general use.
* PyPDF2 for reading PDF files.
* LangChain components (text\_splitter, FAISS, HuggingFaceEmbeddings, ChatGoogleGenerativeAI, PromptTemplate, load\_qa\_chain) for building the RAG pipeline.
* streamlit for the UI.
* **Helper Functions**
* get\_pdf\_text(pdf\_docs) → Extracts raw text from uploaded PDFs.
* get\_text\_chunks(text) → Splits raw text into smaller chunks for better retrieval.
* get\_vector\_store(text\_chunks) → Converts chunks into embeddings (using HuggingFace) and stores them in a FAISS vector DB.
* get\_conversational\_chain(api\_key) → Defines how to query Gemini (LLM) with retrieved context and a prompt.
* user\_input(...) → The main function that handles user queries: retrieves relevant chunks, queries Gemini, displays results, and logs the conversation.
* **Main Function (main())**
* Sets up the Streamlit UI (page title, sidebar, inputs).
* Handles PDF upload and processing into a vector store.
* Takes a user question, runs it through the retrieval + generation pipeline, and displays results.
* Manages conversation history via st.session\_state.