**TECHNICAL WRITE UP**

**Tech Stack Choices**

**Points:**

* Python, Streamlit for UI
* PyMuPDF, docx2txt for file handling
* LangChain for chaining and QA
* OpenAI/Hugging Face for embeddings
* FAISS for vector storage
* Hugging Face Spaces for deployment

**Response Structuring Approach**

* In order to ensure that the chatbot provides accurate and contextually relevant answers from uploaded documents, a layered approach to response structuring was adopted. The core idea is to convert the unstructured data (PDFs, DOCX files) into meaningful, queryable information using vector embeddings. First, each document is preprocessed and broken down into manageable text chunks using LangChain’s text splitter. This step is critical because large documents often exceed the token limit of most language models. Chunking ensures that information remains coherent while fitting within the model's processing constraints.
* Once chunked, each segment is embedded into a high-dimensional vector space using models from either Hugging Face or OpenAI. These embeddings represent the semantic meaning of each chunk. The entire collection of these vectorized chunks is then stored using FAISS, a fast and efficient similarity search engine. This allows the system to perform a similarity search when a user inputs a query, retrieving only the most relevant parts of the document.
* At runtime, when a user submits a question through the Streamlit interface, the chatbot performs a similarity search against the FAISS index to find the most contextually relevant chunks. These retrieved chunks are compiled and passed along with the user’s query to the language model, which then generates a response grounded in the actual content of the uploaded documents. This layered pipeline—from chunking to embedding to semantic search—ensures that answers remain both accurate and context-aware, giving users meaningful insights from their own files.

**Challenges Faced & Solutions Implemented (Detailed – 3 Paragraphs)**

* One of the first challenges encountered during development was dealing with the token limit of large language models. Documents, especially PDFs and DOCX files, can contain thousands of words. Feeding the entire content to the model is neither feasible nor efficient. This was resolved by using a recursive chunking method through LangChain’s text splitter, allowing the content to be divided intelligently at logical breakpoints like sentences or paragraphs. Each chunk was then processed individually for embeddings, helping the model work within its limits while maintaining context relevance.
* Another issue was loss of context across multiple chunks. Since queries may relate to concepts or answers spread across different parts of a document, retrieving just one segment might miss essential information. To address this, a strategy was implemented to retrieve multiple top-ranked chunks from FAISS and combine them intelligently. These combined chunks provided broader context to the model while answering, significantly improving the quality and depth of the responses. It also allowed the chatbot to better understand and reference information that might be spread throughout the document.
* A major deployment hurdle was faced when trying to expose the app to the internet using Ngrok. Ngrok's latest versions now require authentication tokens and a verified account, which caused frequent authentication failures during public deployment. As a solution, the entire project was migrated to Hugging Face Spaces, which supports hosting Streamlit apps natively and offers a more robust, stable, and free alternative. This also simplified collaboration and sharing, as users could access the chatbot from any device without requiring local setup or tunneling.