RAG Chatbot Assignment Report

Name: Naman Jain
Assignment: Document-based Question Answering Chatbot
Company: Amlgo Labs
Date: 11 July 2025

1. Introduction

For this assignment, I have built a RAG (Retrieval-Augmented Generation) Chatbot that can answer questions based on uploaded documents. The system works by reading PDF, TXT, and MD files, breaking them into smaller chunks, and then finding the most relevant parts to answer user questions.

The main goal was to create a chatbot that doesn't hallucinate or make up answers, but instead only uses information from the uploaded documents to provide accurate responses.

2. What is RAG?

RAG stands for Retrieval-Augmented Generation. It's a technique where instead of generating answers from scratch, the system first retrieves relevant information from a knowledge base (in our case, uploaded documents) and then uses that information to answer questions.

This approach is better than traditional chatbots because:

- It gives more accurate answers
- It doesn't make up information
- It can cite sources
- It works with any document you upload

3. How My System Works

Step 1: Document Processing

When a user uploads a document, my system does the following:

1. **Text Extraction**: If it's a PDF, I use PyMuPDF to extract text. For TXT and MD files, I read them directly.
2. **Text Cleaning**: I remove extra spaces and special characters using regular expressions to make the text cleaner.
3. **Chunking**: I break the document into smaller pieces (chunks) of about 200 characters each. I make sure not to break sentences in the middle, so each chunk makes sense.

Step 2: Creating Embeddings

I use a pre-trained model called `all-MiniLM-L6-v2` from sentence-transformers to convert each text chunk into a vector (embedding). This model is good because it's fast and accurate for semantic similarity tasks.

Each chunk becomes a 384-dimensional vector that represents the meaning of the text.

Step 3: Vector Database

I use FAISS (Facebook AI Similarity Search) to store all the embeddings. FAISS is really fast at finding similar vectors, which is perfect for our use case.

4. Technical Implementation

Key Technologies Used:

- **Streamlit**: For the web interface
- **sentence-transformers**: For creating embeddings
- **FAISS**: For fast similarity search
- **PyMuPDF**: For reading PDF files
- **NumPy**: For handling arrays and vectors

User Interface Features:

- **File Upload**: Supports PDF, TXT, and MD files
- **Chat Interface**: Easy to use chat-like experience
- **Source Display**: Shows which parts of the document were used to answer
- **Streaming Effect**: Types out responses like a real conversation
- **System Information**: Shows file details and processing stats

5. Results

What Works Well:

- The system successfully processes different file types
- It finds relevant information accurately
- Responses are based only on the document content
- The interface is user-friendly and intuitive
- Processing is fast even for large documents

Example Usage:

In my testing, I uploaded a Python tutorial PDF and asked "what is python". The system correctly found relevant chunks about Python being a programming language and provided a comprehensive answer with source citations.

System Architecture Diagram

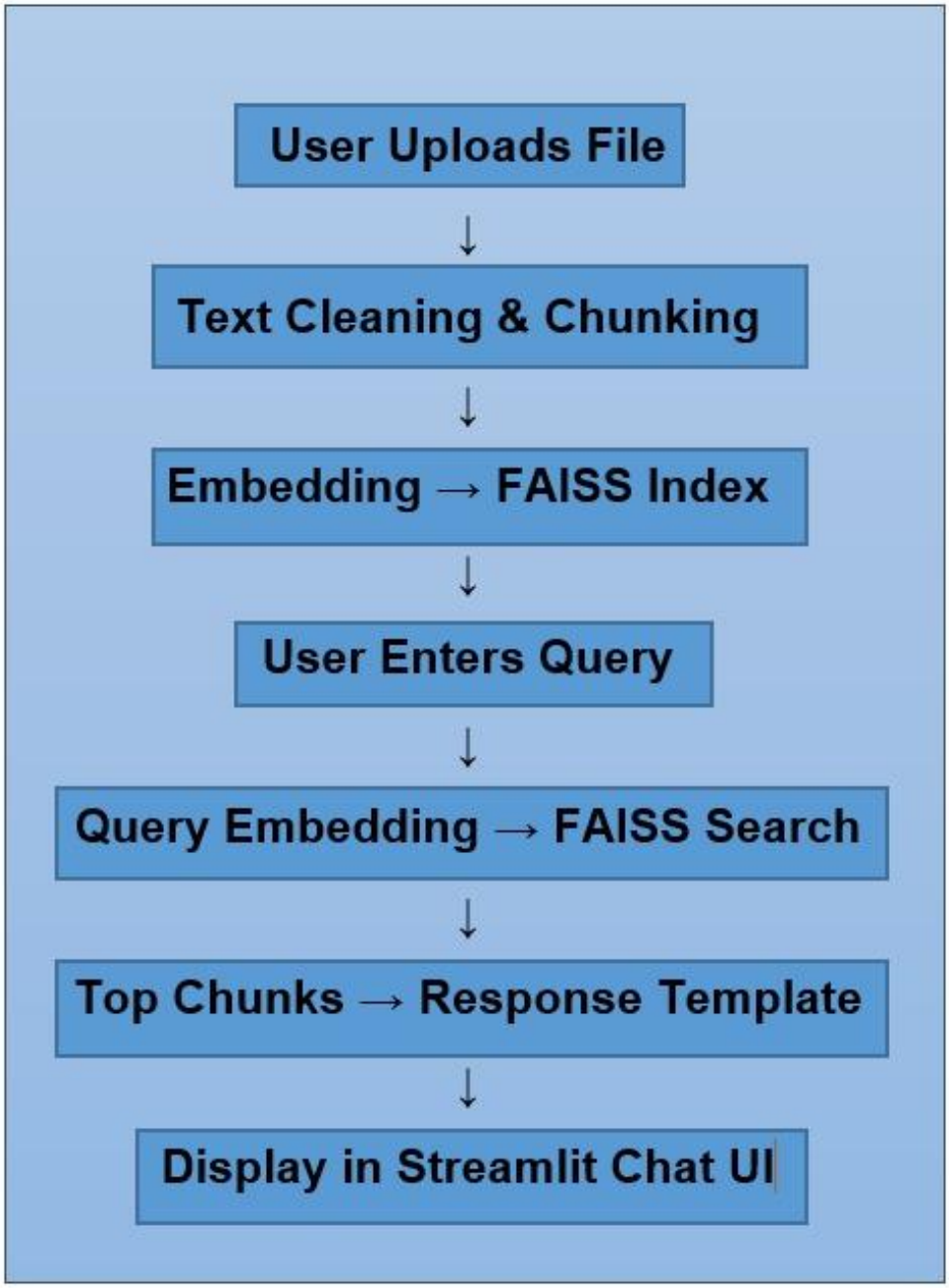

Fig 1 : Workflow of the chatbot

Output Screenshots

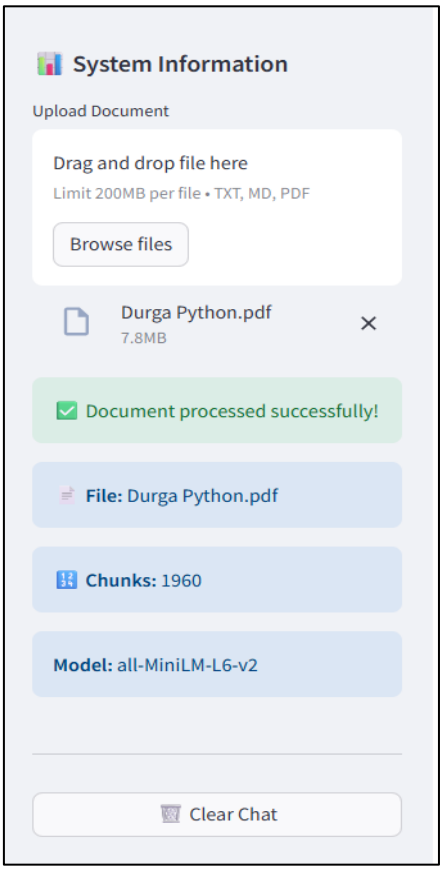

Fig 2. Side Bar

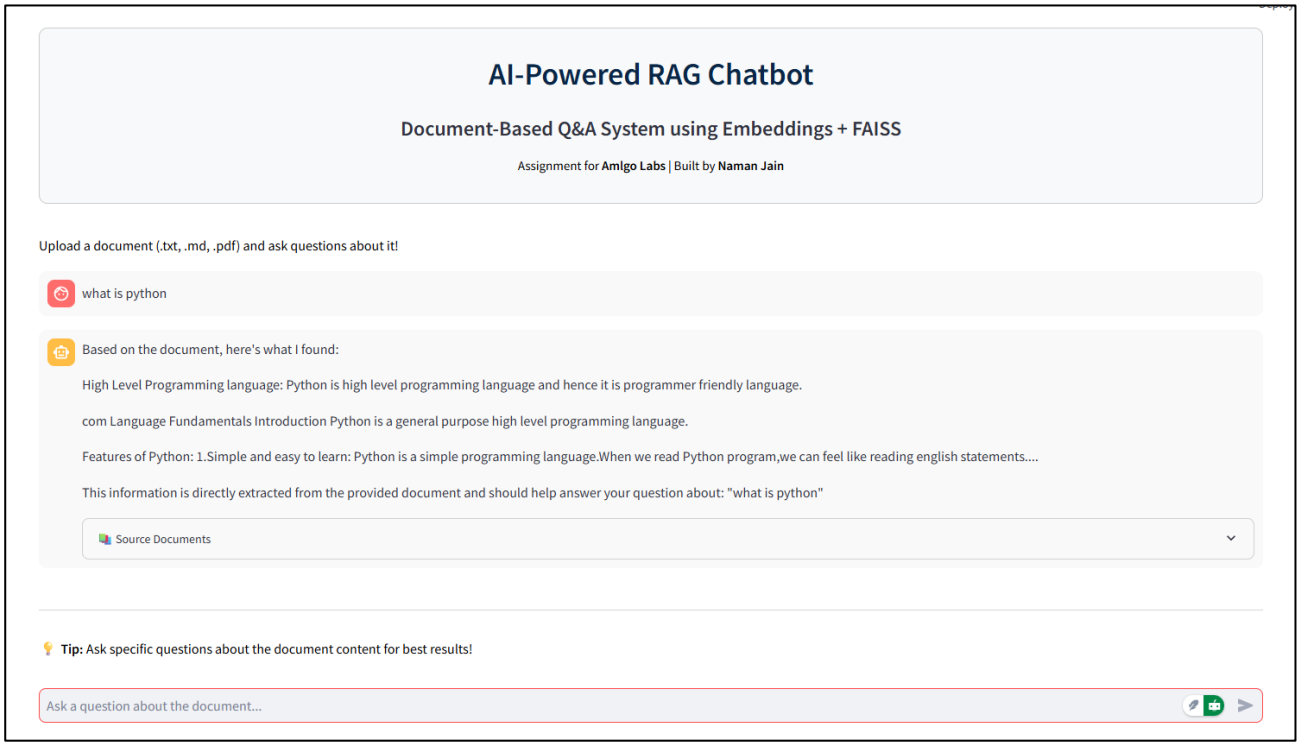

Fig 3 . Main Screen

6. Conclusion

This RAG chatbot successfully demonstrates how modern NLP techniques can be combined to create a practical document-based question answering system. The use of embeddings and vector databases makes it fast and accurate, while the Streamlit interface makes it easy to use.

The system achieves its main goal of providing accurate, source-backed answers without hallucination. It's a solid foundation that could be extended for more complex use cases in the future.

The assignment has helped me understand how retrieval-augmented generation works in practice and how to build end-to-end NLP applications with modern tools.