LangChain-Powered RAG Chatbot

This report details the AERODYNAMICS Chatbot project, which is designed to create a question-answering system for a PDF document. The project leverages a retrieval-augmented generation (RAG) approach to enable a Large Language Model (LLM) to provide specific, context-aware answers to user queries about the document's content.

1. Project Goal

The primary objective of this project is to build an interactive chatbot that can read and understand a comprehensive PDF document, specifically a book on "Fundamentals of Aerodynamics." The chatbot should be able to answer questions accurately by retrieving relevant information from the document and generating a coherent response.

2. Technology Stack

The project utilizes a robust set of open-source libraries and services, primarily within a Python environment. The key technologies are:

- **Jupyter Notebook:** The entire project workflow is contained within a Jupyter Notebook, allowing for a step-by-step, reproducible process.
- **LangChain:** A framework for developing applications powered by LLMs. It is used for document loading, text splitting, and connecting the various components of the RAG pipeline.
- **Sentence-Transformers:** A library for generating dense vector embeddings from text. The all-MiniLM-L6-v2 model is used for this purpose, which is a lightweight but effective embedding model.
- **FAISS:** A library for efficient similarity search and clustering of dense vectors. It serves as the vector store for the document chunks.
- **Groq:** A fast inference platform for LLMs. The deepseek-r1-distill-llama-70b model is used for generating the final answers.
- **Streamlit:** An open-source app framework for creating

interactive web applications. It is used to build the user-facing chatbot interface.

- **ngrok:** A service that creates a secure tunnel to a localhost server, making the Streamlit application publicly accessible on the internet.

3. Methodology

The project follows a standard RAG pipeline, which can be broken down into the following key steps:

1. **Document Ingestion:** The project starts by ingesting a PDF file. This is done using PyPDFLoader from the langchain-community library.
2. **Text Preprocessing (Chunking):** The large PDF document is split into smaller, manageable chunks. This is a crucial step to ensure that the LLM receives highly relevant and focused context for each query. The RecursiveCharacterTextSplitter is used with a chunk_size of 1500 characters and a chunk_overlap of 100 characters.
3. **Embedding Generation:** Each text chunk is converted into a numerical vector (embedding) using the HuggingFaceEmbeddings model. These vectors capture the semantic meaning of the text.
4. **Vector Store Creation:** The generated embeddings are stored in a FAISS database, which is optimized for fast similarity searches. This database is saved locally to avoid regenerating it for every session.
5. **Chatbot Application:** A Streamlit application is created to handle user interaction.
    - It loads the saved FAISS database.
    - It initializes the Groq LLM model.
    - It defines a custom prompt template to instruct the LLM to use the provided context.
    - When a user submits a query, the application performs a similarity search on the FAISS database to find the most relevant text chunks (documents) from the original PDF.
    - The user's query and the retrieved context are passed to the Groq LLM, which then generates a human-readable answer.

6. **Deployment:** The Streamlit application is launched on a local port (8501) and made accessible to the public internet using an ngrok tunnel.

4. Code Breakdown

The Jupyter Notebook code is divided into two main sections:

Section 1: Data Preparation

This section focuses on processing the PDF document and creating the vector database.

- **API Key Setup:** Groq and ngrok API keys are set up as environment variables.
- **Library Installation:** The necessary Python libraries (sentence-transformers, langchain-community, faiss-cpu, tqdm, pyngrok, streamlit) are installed.
- **Embedding Model Download:** The HuggingFaceEmbeddings model is preloaded to prevent potential errors later.
- **File Upload:** The google.colab.files.upload() function is used to allow the user to upload their PDF file directly in the notebook environment.
- **Document Processing:** The uploaded PDF is loaded, split into chunks, and the chunks are used to create a FAISS database. A progress bar (tqdm) is included to show the embedding generation process.
- **Database Saving:** The created FAISS database is saved to a local directory (vectorstore/database.faiss) for future use.

Section 2: Chatbot Application and Deployment

This section focuses on creating the Streamlit app and making it available.

- **Streamlit App Code:** The full Python code for the Streamlit application is written as a multi-line string. This code defines all the functions for loading the database, retrieving documents, and generating answers. It also sets up the user interface with a title, a text area for queries, and a button to submit questions.
- **File Creation:** The Streamlit app code is written to a new file named CHAT_APP.py.

- **Launch with ngrok:**
  - The ngrok authentication token is set.
  - A shell command is executed to run the CHAT_APP.py file using Streamlit in the background.
  - An ngrok tunnel is created on port 8501, and the public URL is printed to the console, allowing users to access the chatbot.

5. Conclusion

This project successfully demonstrates the creation of a functional and intelligent chatbot tailored to a specific knowledge domain. By combining a PDF document with an LLM via the RAG pipeline, the system can provide accurate, document-grounded answers. The use of Streamlit and ngrok makes the application easy to deploy and share, transforming a static document into an interactive knowledge source.