Technical Summary

Overview

The project is a **PDF-based Q&A web application** that allows users to upload PDF documents, ask questions about their content, and receive answers generated by an AI model. The application is built using a **FastAPI backend** and a **Bootstrap-based frontend**, with an AI-powered retrieval-augmented generation (RAG) system for answering questions. The backend uses **Groq's Llama3-70b model** for natural language processing and **HuggingFace embeddings** for document retrieval.

Architecture

1. Frontend

- **Framework**: Plain HTML, CSS, and JavaScript with Bootstrap for styling.
- **Key Features**:
  - **File Upload**: Users can upload PDF files using a file input field.
  - **Chat Interface**: A simple chat interface allows users to ask questions and view responses in real-time.
  - **Dynamic Tabs**: Uploaded PDFs are displayed as tabs, with summaries shown in the corresponding sections.
  - **Loading Indicators**: A loading popup with a spinner animation is displayed during file uploads and query processing.

2. Backend

- **Framework**: FastAPI for building RESTful APIs.
- **Key Features**:
  - **File Upload**: Accepts PDF files, extracts text using `PyPDF2`, and processes the content.
  - **Text Chunking**: Splits text into smaller chunks using `RecursiveCharacterTextSplitter` for efficient storage and retrieval.
  - **Embeddings**: Uses HuggingFace's `sentence-transformers/all-MiniLM-L6-v2` model to generate embeddings for text chunks.
  - **Vector Storage**: Stores text chunks and their embeddings in a **ChromaDB** collection for semantic search.
  - **Query Processing**: Retrieves relevant text chunks based on user queries and generates answers using the Groq API.
  - **Conversation History**: Maintains a history of user queries and model responses for context-aware answers.

    - **TF-IDF Summarization**: Ranks sentences in the uploaded PDF using TF-IDF to generate concise summaries.

3. AI/ML Component

- **Model**: Groq's Llama3-70b model for generating responses.
- **Retrieval-Augmented Generation (RAG)**:
    - **Embedding-Based Retrieval**: Uses HuggingFace embeddings to find the most relevant text chunks from the uploaded PDFs.
    - **Context-Aware Responses**: Combines retrieved chunks with user queries to generate accurate and context-aware answers.
- **Query Analysis**: Classifies user queries into categories (e.g., greetings, document-related) to handle them appropriately.

Tech Stack

- **Frontend**: HTML, CSS, JavaScript, Bootstrap.
- **Backend**: FastAPI, PyPDF2, ChromaDB, HuggingFace Transformers, Groq API.
- **AI/ML**: HuggingFace embeddings, Groq's Llama3-70b, TF-IDF for summarization.
- **Database**: ChromaDB for vector storage.
- **Deployment**: Local development (can be extended to Docker, Kubernetes, or cloud platforms).

Challenges and Solutions

1. Handling Large PDFs

- **Challenge**: Extracting and processing large PDFs can be resource-intensive and slow.
- **Solution**: Implemented text chunking and TF-IDF-based summarization to reduce the amount of data processed by the AI model.

2. Maintaining Conversation Context

- **Challenge**: Ensuring the AI model remembers previous interactions for context-aware responses.
- **Solution**: Stored conversation history in a list and passed the last few interactions to the model with each query.

3. Semantic Search

- **Challenge**: Retrieving the most relevant text chunks for user queries.

- **Solution**: Used HuggingFace embeddings and ChromaDB for semantic search, ensuring accurate retrieval of relevant content.

4. Edge Cases

- **Challenge**: Handling invalid file uploads
- **Solution**: Added validation checks in the backend and provided user-friendly error messages in the frontend.

Improvements with More Time

1. **Scalability**:
    - Implement **stream-based processing** for large files to avoid memory issues.
    - Use a distributed vector database like **Weaviate** or **Pinecone** for better scalability.
2. **Enhanced Retrieval**:
    - Experiment with **hybrid search** (combining keyword and semantic search) for improved retrieval accuracy.
    - Add **re-ranking** of retrieved chunks based on relevance scores.
3. **UI/UX Enhancements**:
    - Add support for **multiple document uploads** in a single session.
    - Implement **real-time updates** using WebSockets for a smoother chat experience.
4. **Advanced AI Features**:
    - Add **document summarization** after upload for quick insights.
    - Implement **multi-turn conversations** with follow-up questions.
5. **Deployment**:
    - Containerize the application using **Docker** for easy deployment.
    - Deploy on a cloud platform like **AWS**, **GCP**, or **Azure** for production use.

Experience

Building this application was a great learning experience, especially in integrating AI models with a full-stack application. The combination of FastAPI for backend logic, HuggingFace embeddings for semantic search, and Groq's Llama3-70b for response generation provided a robust foundation for the project. The challenges faced during development, such as handling large files and maintaining conversation context, helped deepen my understanding of AI-powered applications and their real-world constraints. Overall, this project was a rewarding exercise in problem-solving, creativity, and technical implementation.

Conclusion

This project demonstrates a functional and scalable approach to building an AI-powered document Q&A system. With further refinements and additional features, it has the potential to become a powerful tool for extracting insights from PDF documents.