Technical Report

Approach

The application uses FastAPI to provide a RESTful API for uploading and querying PDFs. The PDFs are processed using langchain to create a vector database with FAISS, enabling efficient searches. The frontend is built with a simple Streamlit app, which interacts with the backend API to provide a user-friendly interface for uploading PDFs and asking questions.

Challenges

1. PDF Processing and Text Splitting:

- Challenge: Extracting text from PDFs and splitting it into manageable chunks for indexing.
- **Solution**: Utilized `langchain_community.document_loaders.PyPDFLoader` for robust PDF loading and `RecursiveCharacterTextSplitter` for splitting text into chunks.

2. Vector Database:

- **Challenge :** Managing the FAISS database for vector storage, especially handling updates with new documents. Other options for VDB could be using a cloud hosted or local Qdrant or any other VDB.
- **Solution**: Implemented functionality to initialize and update the FAISS database, ensuring it can handle new documents without reinitializing.

3. Question Relevance and Appropriateness Check:

- **Challenge**: Ensuring that the questions asked are relevant to the PDF content and do not contain inappropriate content.
- **Solution :** Created a guardrails prompt using `ChatPromptTemplate` to analyze and filter questions based on relevance and appropriateness.

4. API and Frontend Integration :

- **Challenge**: Seamlessly integrating the backend API with the frontend to provide a smooth user experience.
- **Solution :** Used `requests` library in Streamlit to interact with the FastAPI backend, providing clear feedback to users based on API responses.

Solutions Implemented

- **PDF Upload and Processing :** Handled using temporary files and robust PDF loaders.
- **Database Management :** FAISS vector store for efficient document retrieval, with mechanisms for database creation and updates.
- **Relevance and Appropriateness Checks :** Guardrails prompt for filtering out inappropriate or irrelevant questions.
- **User Interaction**: Streamlit frontend for a simple and intuitive user experience.

Future Improvements

- **Enhanced Security**: Implement authentication and authorization mechanisms for the API.
- **Scalability**: Optimize the system for handling larger PDFs and more concurrent users.

- **Advanced Querying :** Implement more advanced querying capabilities, such as keyword highlighting and context expansion.

Conclusion

This solution provides a robust and user-friendly application for uploading PDFs, creating a searchable database, and querying the content. It follows best practices for API design, frontend development, and containerization, ensuring it is production-ready and easy to deploy.

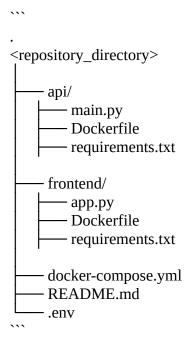
Authors

- Jawad Haider

License

This project is licensed under the MIT License - see the LICENSE file for details.

Final Directory Structure



Instructions to Run the Solution

```
Clone the repository:
"sh
clone <repository_url>
cd <repository_directory>
```

Build and run the containers :
 ```sh
 docker-compose up --build

3. Access the applications:

FastAPI (Backend): http://localhost:8000Streamlit (Frontend): http://localhost:8501

## **Summary**

This solution integrates a FastAPI backend with a Streamlit frontend for a seamless PDF question-answering application. The solution is containerized using Docker, ensuring it is production-ready and easy to deploy.