Project Documentation

Web Application for Document Summarization

Objective

The objective of this project is to develop a web application that allows users to upload documents and receive summarized versions using a locally deployed Language Model (LLM).

Approach

Backend

1. **Framework Selection**:
    - Chose Flask for its simplicity and lightweight nature suitable for quick API development.
    - Used the transformers library from Hugging Face for the summarization task.
2. **API Endpoints**:
    - Implemented /upload endpoint to handle file uploads.
    - Implemented /summarize endpoint to process and summarize the uploaded document using a pre-trained LLM.
3. **File Handling**:
    - Created functions to securely handle file uploads.
    - Stored uploaded files in a designated directory.
4. **Summarization**:
    - Loaded a pre-trained model (e.g., GPT-2) and used it to generate summaries.
    - Integrated the model with the backend to handle summarization requests.

Frontend

1. **Framework Selection**:
    - Used React for its component-based architecture and ease of integration with backend services.
2. **UI Components**:
    - Created a file upload component.
    - Displayed the uploaded file’s content and the summarized text.
3. **API Integration**:
    - Implemented API calls to the backend endpoints for file upload and summarization.
    - Managed responses and displayed summaries in a user-friendly manner.

LLM Deployment

1. **Local Environment Setup**:

- Used Docker to create a consistent environment for the LLM.
- Deployed the LLM locally to ensure efficient and secure processing.

2. **Model Integration**:
    - Loaded a pre-trained model and ensured it could generate text summaries.
    - Integrated the model with the backend for handling summarization requests.

Docker Integration

1. **Backend Dockerfile**:
    - Created a Dockerfile to containerize the Flask backend.
    - Installed necessary dependencies and exposed port 5000.
2. **Frontend Dockerfile**:
    - Created a Dockerfile to containerize the React frontend.
    - Installed necessary dependencies, built the app, and exposed port 3000.
3. **Docker Compose**:
    - Created a docker-compose.yml file to manage the backend, frontend, and LLM services.
    - Configured service dependencies and exposed the necessary ports.

Challenges Faced

1. **Model Deployment**:
    - **Challenge**: Deploying a pre-trained LLM locally with limited resources.
    - **Solution**: Chose a smaller model (like GPT-2) to ensure it could run efficiently on local hardware. Used Docker to create a consistent deployment environment.
2. **File Handling**:
    - **Challenge**: Handling various file types (PDF, DOCX, TXT) securely.
    - **Solution**: Implemented robust file validation and error handling to manage different file types and potential upload issues.
3. **Concurrency**:
    - **Challenge**: Ensuring the backend could handle multiple concurrent requests.
    - **Solution**: Used Flask's built-in support for concurrent request handling. Tested the application under load to ensure performance.
4. **API Integration**:
    - **Challenge**: Ensuring seamless integration between the React frontend and Flask backend.
    - **Solution**: Used Axios for API calls and implemented proper error handling and user feedback mechanisms in the frontend.

Conclusion

This project successfully demonstrates the integration of a locally deployed LLM into a web application for document summarization. By leveraging Docker, we ensured a consistent and reproducible environment for both development and deployment. The challenges faced during the project were addressed through careful planning and the use of appropriate technologies and frameworks.