Gemini Pro & LangChain based ChatBot Interaction with Multiple PDF’s

This project utilizes Streamlit to create an interactive web application that allows users to interact with multiple PDFs using Gemini-Pro and LangChain for question answering.

Table of Contents

Introduction

This application leverages Streamlit, a Python framework for building web applications, to create a tool that facilitates interaction with multiple PDFs. It integrates Gemini-Pro, Google's Generative AI, and LangChain for processing and generating responses based on the contents of uploaded PDF files.

Key functionalities include:

- Extracting text from multiple PDF documents.
- Using LangChain for text chunking and FAISS for efficient similarity search.
- Utilizing Gemini-Pro for generating responses to user questions based on the context from PDF contents.

Setup

Requirements

Make sure you have the following installed:

1.**Langchain**: Used for tasks related to language processing, potentially including natural language understanding or generation tasks.

2. **streamlit**: A framework for building interactive web applications with Python.

3. **google-generativeai**: Provides access to Google's Generative AI models for generating content based on prompts.

4. **python-dotenv**: Loads environment variables from a .env file into os.environ, making it easy to manage configurations.

5. **pathlib**: Provides an object-oriented interface for filesystem paths, making it simpler to work with file and directory paths.

6. **PyPDF2**: Enables reading, merging, splitting, and extracting text or data from PDF files programmatically.

7**. Chromadb**: Likely used for interacting with Chrome or Chromium-based browser databases or profiles programmatically.

8. **langchain_google_genai**: Integrates Google's Generative AI models into LangChain for generating content based on prompts.

9. **faiss-cpu**: Provides CPU-based similarity search capabilities for efficient vector indexing and retrieval in LangChain applications.

Installation

1**.** First we need to create a environment specific to the project folder and it can be done in this way
-> **conda create -p venv python==3.10 -y**

2. Then we need to activate the environment before writing the code
-> **conda activate venv/**

3. After this we need to load all the required libraries into our project this can be done individually through command prompt by running it as administrator using the pip command or we can follow this simple way
->**pip install -r requirements.txt**

Input

1. Upload multiple PDF files using the file uploader in the sidebar.

2. Enter your question related to the contents of the uploaded PDF files in the text input field.

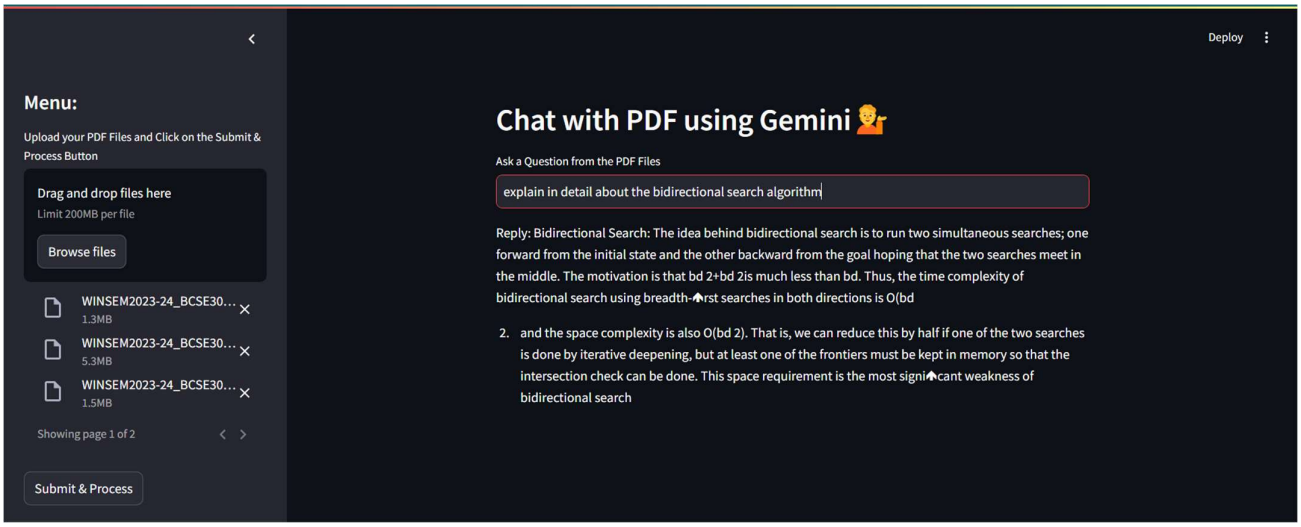
Output

1. The application displays a chat interface where it processes the uploaded PDFs to generate responses to the user's questions.

2. Responses are generated based on the context extracted from the PDFs using Gemini-Pro and LangChain.

3. Summary of processing status and results are shown using Streamlit's components.

Website Link Accessible: https://gemini-llm-invoice-extractor-8xgmjpcgyockrv8uaqhxo2.streamlit.app/


Menu:
Upload your PDF Files and Click on the Submit & Process Button
Drag and drop files here
Limit 200MB per file
Browse files
WINSEM2023-24_BCSE30...
1.3MB
WINSEM2023-24_BCSE30...
5.3MB
WINSEM2023-24_BCSE30...
1.5MB
Showing page 1 of 2
Submit & Process
Deploy
Chat with PDF using Gemini
Ask a Question from the PDF Files
explain in detail about the bidirectional search algorithm
Reply: Bidirectional Search: The idea behind bidirectional search is to run two simultaneous searches; one forward from the initial state and the other backward from the goal hoping that the two searches meet in the middle. The motivation is that bd 2+bd 2is much less than bd. Thus, the time complexity of bidirectional search using breadth-rst searches in both directions is O(bd
2. and the space complexity is also O(bd 2). That is, we can reduce this by half if one of the two searches is done by iterative deepening, but at least one of the frontiers must be kept in memory so that the intersection check can be done. This space requirement is the most signi cant weakness of bidirectional search