Automated PDF Text Extraction, Translation, and Conversational AI Integration Using Streamlit and Google Generative AI

Overview:

This project provides a comprehensive solution for extracting text from PDF files, translating the text to English (if necessary), and enabling conversational interactions with the extracted content using Google Generative AI. The application leverages various technologies such as Streamlit for the web interface, PyMuPDF and Tesseract for text extraction, Google Translate for translation, and LangChain for conversational AI.

Technologies Used:

- Streamlit: For creating a web-based user interface.
- PyMuPDF (fitz): For converting PDF pages into images.
- Tesseract OCR: For extracting text from images.
- Google Translate API: For translating text into English.
- LangChain: For managing the conversational AI workflow.
- FAISS: For creating and managing a vector store of text embeddings.
- Google Generative AI: For generating embeddings and conversational responses.

Project Structure:

1. Streamlit Interface: Handles user interactions, such as uploading PDFs and asking questions.
2. PDF to Image Conversion: Converts PDF pages to images using PyMuPDF.
3. Optical Character Recognition (OCR): Extracts text from images using Tesseract.
4. Text Translation: Translates extracted text to English using Google Translate.
5. Text Preprocessing: Cleans and prepares the text for embedding and AI processing.
6. Text Chunking: Splits the text into manageable chunks.
7. Embedding Generation and Vector Store: Generates embeddings using Google Generative AI and stores them in FAISS.
8. Conversational AI Integration: Sets up a conversational chain using LangChain and Google Generative AI.

PDF Upload and Processing:

User Upload:

Users can upload one or multiple PDF files through the Streamlit interface. The application reads each file and processes it page by page.

Image Generation from PDF:

For each page of the uploaded PDFs, PyMuPDF generates an image. This ensures that even non-text PDFs (like scanned documents) can be processed.

OCR and Text Extraction

Text Extraction Using Tesseract:

The images are fed into Tesseract OCR, which extracts the text content. This step converts visual information into machine-readable text.

Text Translation

Translation to English:

The extracted text, which might be in various languages, is translated to English using Google Translate. This ensures that the subsequent AI processing is done on text in a consistent language.

Text Cleaning and Preprocessing

Cleaning Text:

The raw extracted text undergoes cleaning to remove non-ASCII characters and extra whitespace. This prepares the text for efficient tokenization and chunking.

Preprocessing Text:

The cleaned text is then processed to ensure it is ready for AI embedding and conversational analysis.

Text Chunking

Splitting Text:

Using the RecursiveCharacterTextSplitter from LangChain, the cleaned and translated text is split into smaller chunks. This is necessary for efficient processing and embedding generation.

Embedding Generation and Vector Store:

Generating Embeddings:

Google Generative AI embeddings are used to convert text chunks into vector representations. These embeddings capture the semantic meaning of the text, enabling advanced AI processing.

Creating a Vector Store:

The embeddings are stored in a FAISS vector store, which allows for efficient similarity search and retrieval of relevant text chunks.

Conversational AI Integration

Setting Up the Conversational Chain:

The project uses LangChain's integration with Google Generative AI to create a conversational chain. This chain uses a custom prompt template to generate detailed and contextually accurate responses to user queries. The Google Generative AI is called by an .env file which contains the API key it, It will be varied for each users.

User Interaction

Handling User Queries:

Users can input questions via the Streamlit interface. The application retrieves relevant text chunks from the vector store, passes them to the conversational chain, and generates a detailed response.

Saving Processed Text

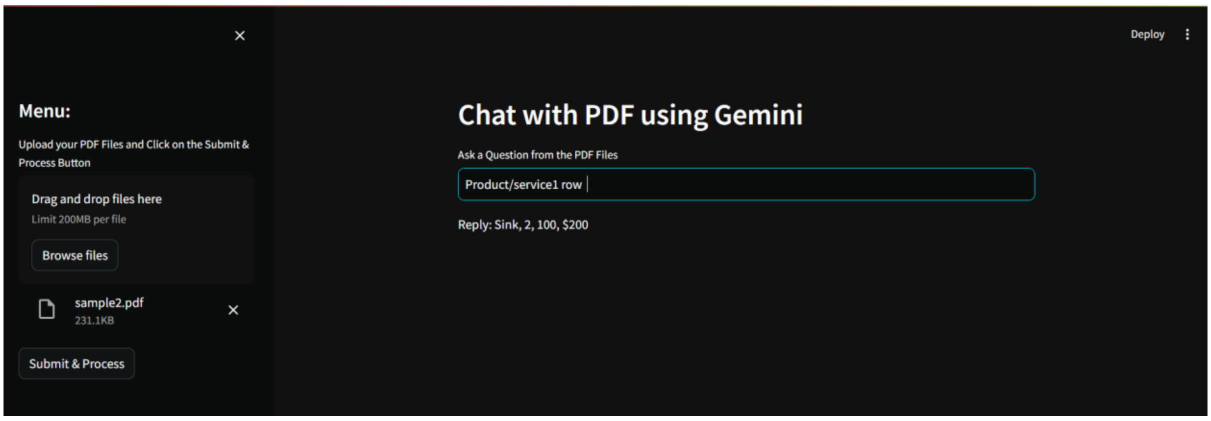
Saving Translated Text:

The preprocessed and translated text is saved locally as a text file. This file can be reviewed or used for further analysis or documentation purposes.

Result

Expected Outcome: Web Server Deployment

Upon running the code, it launches a web server accessible through localhost. Users can interact with the application via a web browser, providing input questions and uploading PDF files for processing. The user interface allows seamless interaction, enhancing accessibility and usability.



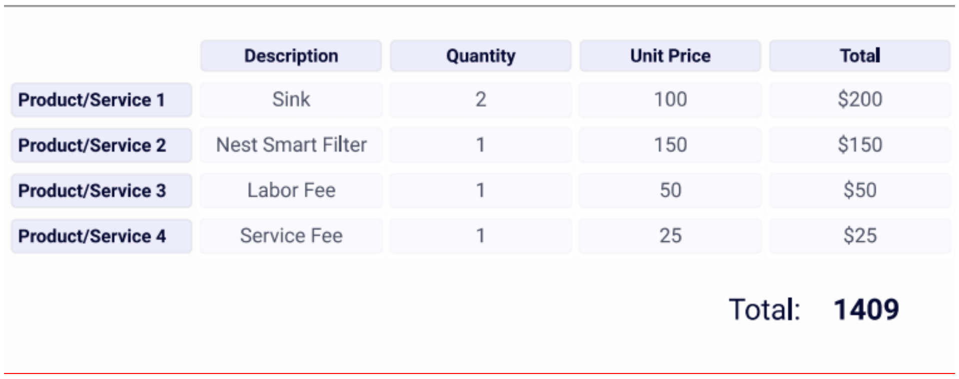
Chat with PDF using Gemini

Ask a Question from the PDF Files

Product/service1 row

Reply: Sink, 2, 100, $200

Fig 1: Chatbot interface with Generated output



	Description	Quantity	Unit Price	Total
Product/Service 1	Sink	2	100	$200
Product/Service 2	Nest Smart Filter	1	150	$150
Product/Service 3	Labor Fee	1	50	$50
Product/Service 4	Service Fee	1	25	$25

Total: **1409**

Fig 2: Snippet from the sample2.pdf file

From comparing fig1,2 we can able to know how the chatbot generates the output according to the questions asked from the query. The test documents are the invoice data in different languages.

Conclusion:

This project demonstrates the seamless integration of several technologies to provide a robust solution for PDF text extraction, translation, and conversational AI. It highlights the power of combining OCR, language translation, and advanced AI models to create an interactive and user-friendly application. The use of Streamlit ensures accessibility and ease of use, while the integration with Google Generative AI and LangChain ensures that the responses are detailed and contextually accurate.