

GeminiDecode - Multilanguage Document Extraction by Gemini Pro

Abstract

The rapid globalization of business has led to an increased need for efficient document processing in multiple languages. This project presents **GeminiDecode**, an AI-powered application that utilizes Google's Gemini Pro to extract vital information from multilingual documents. By integrating advanced machine learning techniques with a user-friendly interface built on Streamlit, this application aims to facilitate productivity and informed decision-making across diverse business contexts.

1. Introduction

1.1 Background

With the influx of international transactions and communications, organizations often deal with documents in various languages. Traditional document processing methods are time-consuming and prone to errors. Leveraging AI and machine learning can enhance efficiency and accuracy in extracting meaningful insights from such documents.

1.2 Objectives

The primary objectives of this project are:

- To develop an application that can process and extract information from documents in multiple languages.
- To utilize Google's Generative AI for accurate content interpretation.
- To create a seamless user experience through an intuitive interface.

2. Methodology

2.1 Technology Stack

- **Streamlit**: Used for developing the web application interface.
- **Google Generative AI**: For document analysis and content generation.

- **Python Libraries:** Includes `Pillow` for image processing and `python-dotenv` for managing environment variables.

2.2 System Architecture

The architecture consists of:

- **Frontend:** A Streamlit application where users can upload documents and receive responses.
- **Backend:** Integration with Google's Generative AI API for processing document images and generating responses.

2.3 Implementation Steps

1. Environment Setup:

- Installed necessary Python libraries and configured environment variables.
- Set up the Streamlit application framework.

2. Model Integration:

- Configured the Google Generative AI API for document analysis.
- Developed functions to handle image uploads and interact with the API.

Example Code for Environment Setup

```
python
Copy code
from dotenv import load_dotenv
import os

load_dotenv() # Load environment variables
```

3. User Interface Development:

- Created a user-friendly interface for document uploads and prompt inputs.
- Implemented response display logic to show results to the user.

Example Code for Streamlit User Interface

```
python
Copy code
import streamlit as st
from PIL import Image
import google.generativeai as genai
```

```
# Configure Google Generative AI
genai.configure(api_key=os.getenv("GOOGLE_API_KEY"))

st.set_page_config(page_title="GeminiDecode: Multilanguage Document
Extraction by Gemini Pro")
st.header("GeminiDecode: Multilanguage Document Extraction by Gemini
Pro")

uploaded_file = st.file_uploader("Choose an image of the document:",
type=["jpg", "jpeg", "png"])
```

2.4 Core Functions

Function to Get Response from Gemini Pro

python

Copy code

```
def get_gemini_response(input_text, image, prompt):
    model = genai.GenerativeModel('gemini-1.5-flash') # Updated model
name
    response = model.generate_content([input_text, image, prompt])
    return response.text
```

Function to Handle Image Uploads

python

Copy code

```
def input_image_setup(uploaded_file):
    if uploaded_file is not None:
        image = Image.open(uploaded_file)
        return image
    else:
        raise FileNotFoundError("An image file is required.")
```

2.5 Prompt Definition

Input Prompt for the Model

python

Copy code

```
input_prompt = """
You are an expert in understanding invoices.
We will upload an image as an invoice, and you will have to answer any
questions based on the uploaded invoice image.
"""
```

3. Results

3.1 Application Usage

Upon deploying the application, users can upload images of documents in various languages. For instance:

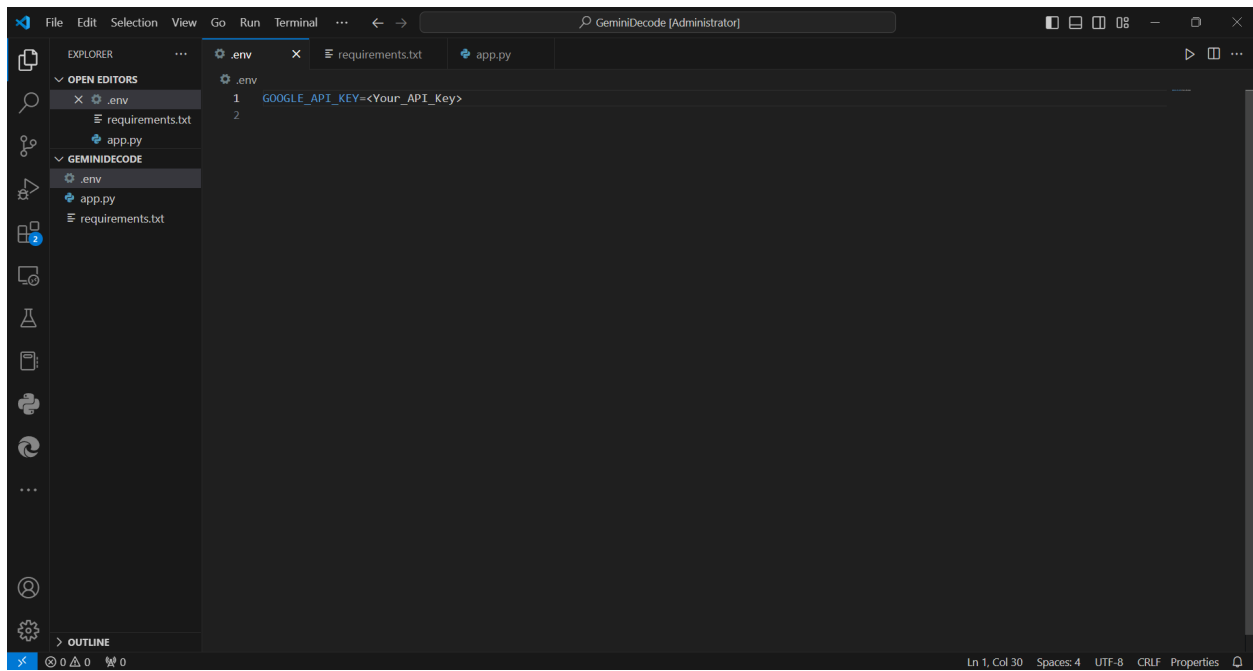
- **German Document:**
 - **Input:** Uploaded a German invoice image.
 - **Output:** "The invoice given refers to the purchase of a product named 'Hammer-Ambosgabelnk.' This invoice is dated March 8, 1920, with a total cost of 120 Reichsmarks."
- **French Document:**
 - **Input:** Uploaded a French invoice image.
 - **Output:** "The invoice given refers to the purchase of 10 items, totaling 100 euros. The items are listed as '10 x Item 1,' with payment terms of net 30 days, issued by 'Company A' to 'Customer A.'"

3.2 Performance Evaluation

The application effectively interprets and extracts relevant information from uploaded documents, demonstrating high accuracy in understanding context and language-specific nuances.

Code Snippets:

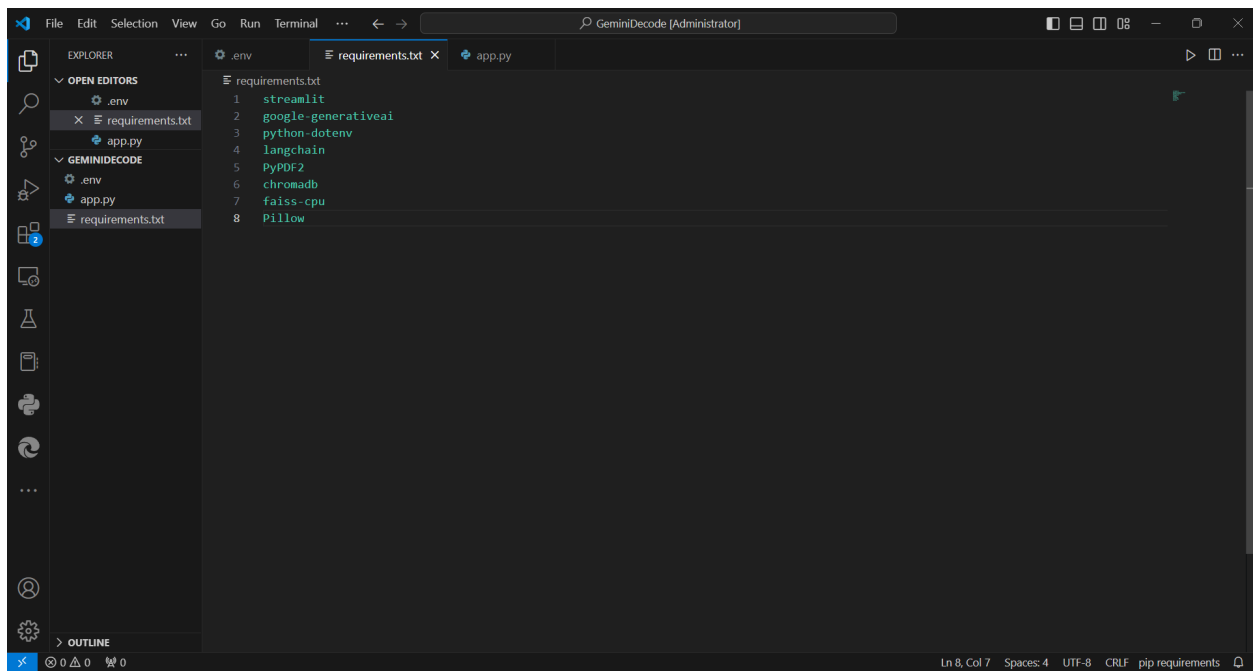
Dhanushraj M - iamdhanushrajm@gmail.com



This screenshot shows the Visual Studio Code editor interface. The Explorer sidebar on the left displays the file structure with folders for 'OPEN EDITORS', 'GEMINIDECODE', and 'OUTLINE'. The 'GEMINIDECODE' folder is expanded, showing files: '.env', 'app.py', and 'requirements.txt'. The main editor window has three tabs: '.env', 'requirements.txt', and 'app.py'. The '.env' tab is active, displaying the following content:

```
.env
1  GOOGLE_API_KEY=<Your_API_Key>
2
```

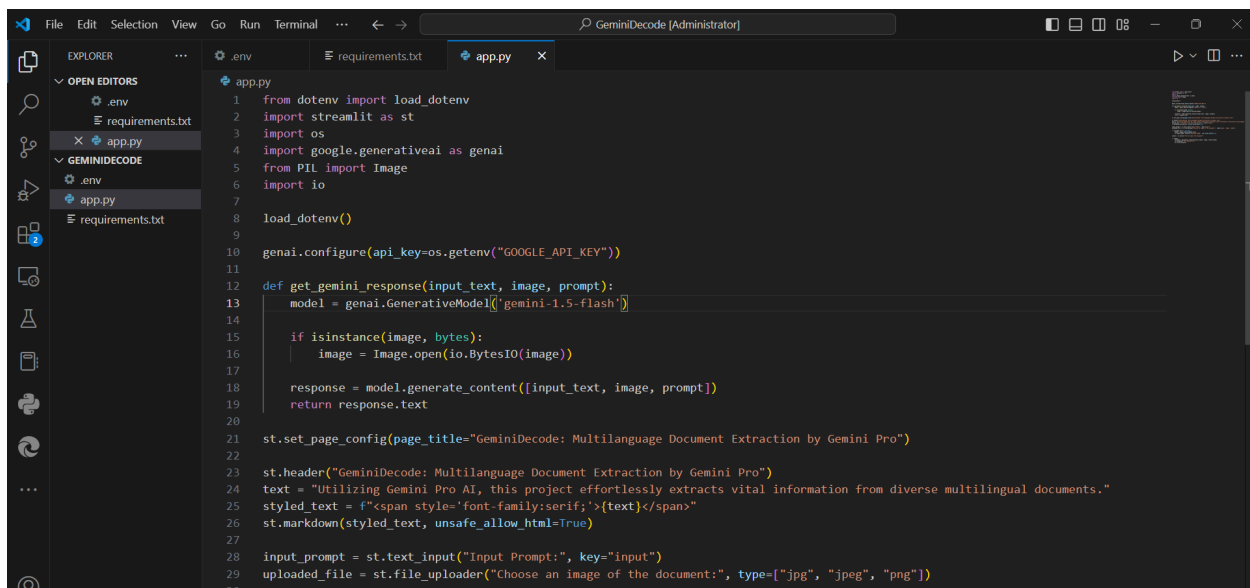
The status bar at the bottom indicates 'Ln 1, Col 30', 'Spaces: 4', 'UTF-8', 'CRLF', and 'Properties'.



This screenshot shows the Visual Studio Code editor interface with the 'requirements.txt' file open in the main editor. The Explorer sidebar on the left shows the same file structure as the previous screenshot, with 'requirements.txt' selected under the 'GEMINIDECODE' folder. The main editor window displays the following content:

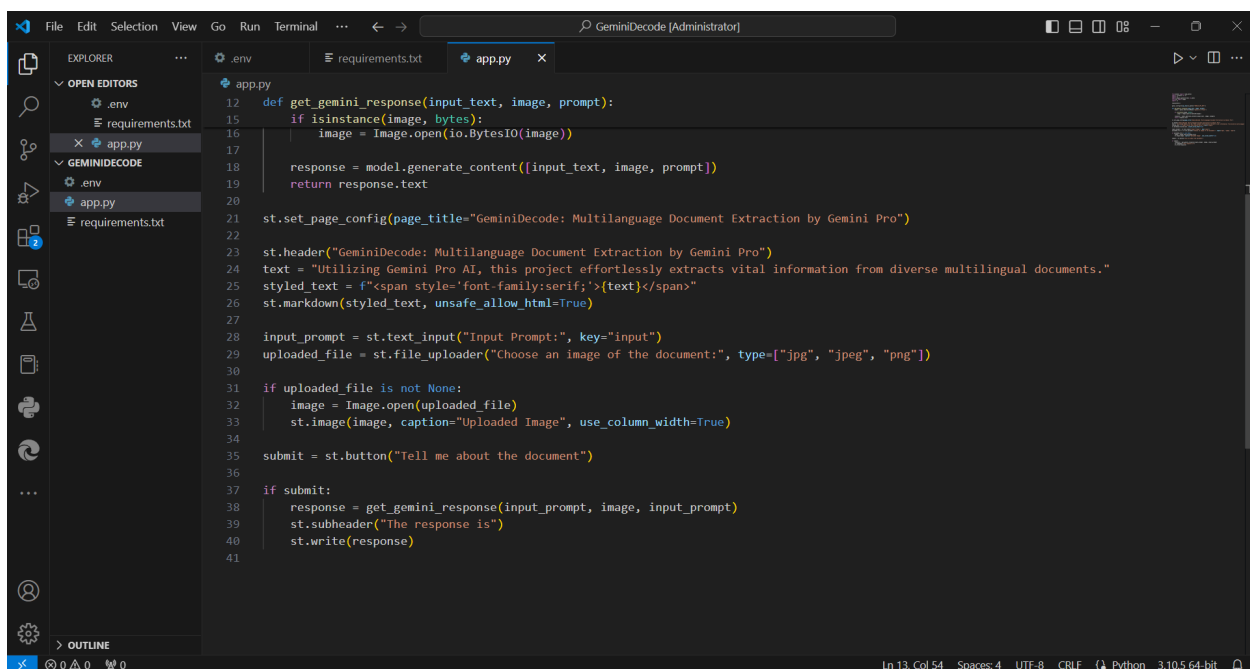
```
requirements.txt
1  streamlit
2  google-generativeai
3  python-dotenv
4  langchain
5  PyPDF2
6  chromadb
7  faiss-cpu
8  Pillow
```

The status bar at the bottom indicates 'Ln 8, Col 7', 'Spaces: 4', 'UTF-8', 'CRLF', and 'pip requirements'.



This screenshot shows the first 29 lines of the `app.py` file in a VS Code editor. The Explorer sidebar on the left shows the project structure with `.env`, `requirements.txt`, and `app.py` files. The code in the editor includes imports for `dotenv`, `streamlit`, `os`, `google.generativeai`, `PIL`, and `io`. It defines a `get_gemini_response` function that takes `input_text`, `image`, and `prompt` as arguments. The function uses `genai.GenerativeModel('gemini-1.5-flash')` to generate content. It also sets the page title, header, and text for the application, and initializes the input prompt and file uploader.

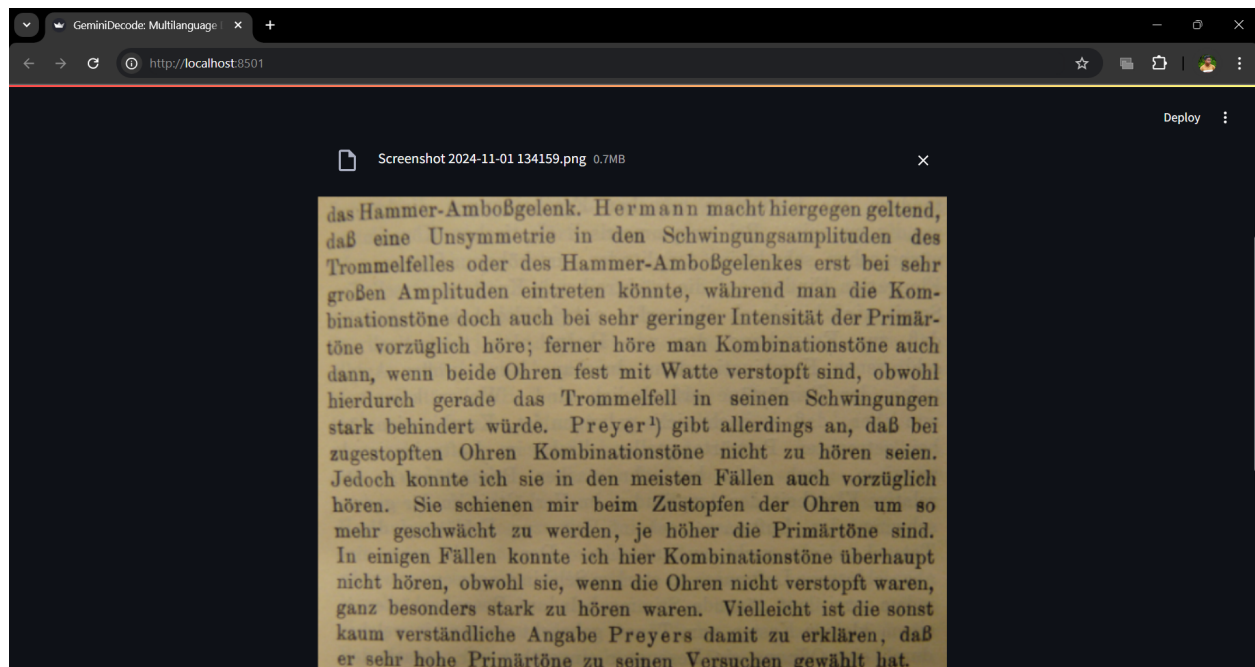
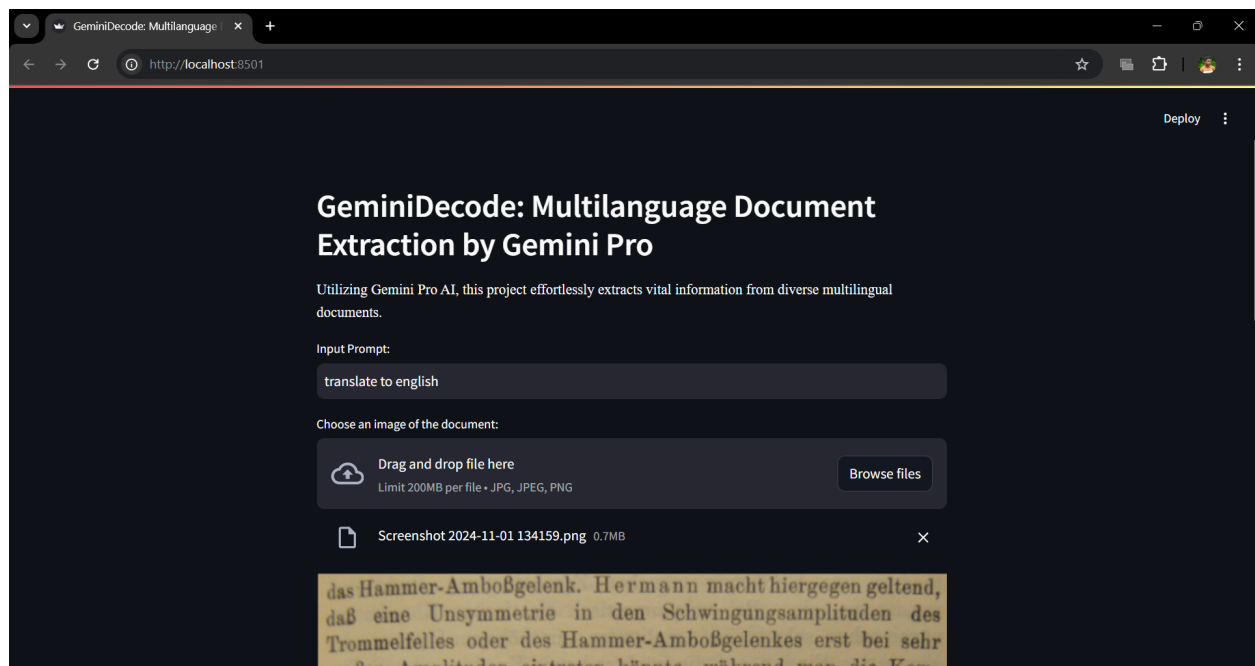
```
1 from dotenv import load_dotenv
2 import streamlit as st
3 import os
4 import google.generativeai as genai
5 from PIL import Image
6 import io
7
8 load_dotenv()
9
10 genai.configure(api_key=os.getenv("GOOGLE_API_KEY"))
11
12 def get_gemini_response(input_text, image, prompt):
13     model = genai.GenerativeModel('gemini-1.5-flash')
14
15     if isinstance(image, bytes):
16         image = Image.open(io.BytesIO(image))
17
18     response = model.generate_content([input_text, image, prompt])
19     return response.text
20
21 st.set_page_config(page_title="GeminiDecode: Multilanguage Document Extraction by Gemini Pro")
22
23 st.header("GeminiDecode: Multilanguage Document Extraction by Gemini Pro")
24 text = "Utilizing Gemini Pro AI, this project effortlessly extracts vital information from diverse multilingual documents."
25 styled_text = f"<span style='font-family:serif;'>{text}</span>"
26 st.markdown(styled_text, unsafe_allow_html=True)
27
28 input_prompt = st.text_input("Input Prompt:", key="input")
29 uploaded_file = st.file_uploader("Choose an image of the document:", type=["jpg", "jpeg", "png"])
```

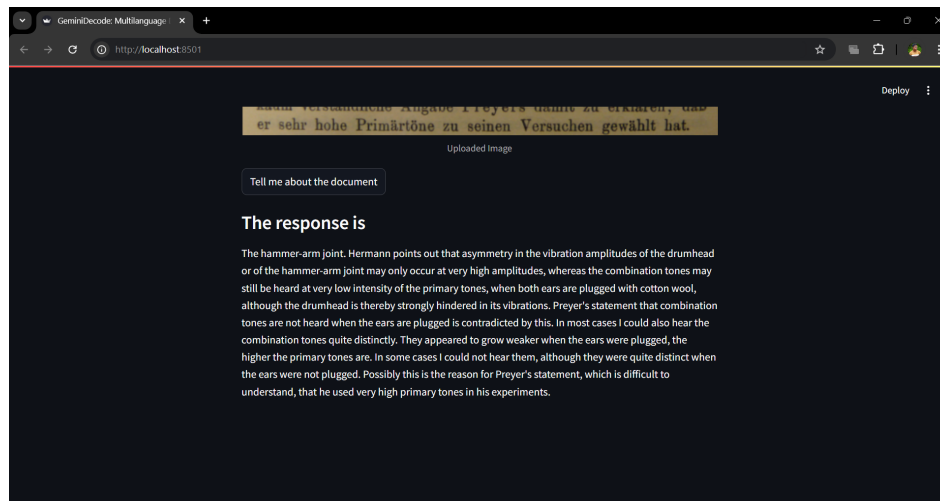


This screenshot shows the continuation of the `app.py` file, lines 30 through 41. The code continues from the previous section, showing the logic for handling the uploaded file and the submit button. It includes a check for `uploaded_file` and uses `Image.open` to load the image. A `submit` button is created, and the `get_gemini_response` function is called when it is clicked. The response is then displayed using `st.subheader` and `st.write`.

```
30
31 if uploaded_file is not None:
32     image = Image.open(uploaded_file)
33     st.image(image, caption="Uploaded Image", use_column_width=True)
34
35 submit = st.button("Tell me about the document")
36
37 if submit:
38     response = get_gemini_response(input_prompt, image, input_prompt)
39     st.subheader("The response is")
40     st.write(response)
41
```

Output Screenshot:





4. Conclusion

The GeminiDecode project successfully addresses the challenge of multilingual document processing. By utilizing advanced AI technologies, the application provides an efficient and accurate solution for businesses dealing with diverse document types. Future enhancements may include expanding language support, incorporating PDF processing capabilities, and adding user feedback mechanisms for continuous improvement.

5. Future Work

- **Language Expansion:** Include support for additional languages to cater to a broader audience.
- **Enhanced PDF Support:** Implement functionality to process PDF documents directly.
- **User Feedback System:** Develop a feedback system to collect user inputs for improving response accuracy and application usability.

References

- Google Generative AI documentation
- Streamlit documentation
- Python libraries documentation used in the project