Approach for Round 2 Assignment

Aryma Labs

Here I have explained How and What for every function I wrote in order to give a quick walkthrough!

TechStack - Langchain , Streamlit , Gemini-pro LLM , FAISS vectorDB, Google GENAI Embeddings,Python

Helper.py Functions

- **Extracting text from pdfs -** The initial phase of the process involved the extraction of textual content from three PDF documents using the PyPDF2 library. Additionally, utilizing web scraping techniques, text data was extracted from three article links. Subsequently, the extracted text from both sources was amalgamated into a unified corpus.

```python
def extract_text_from_pdf(pdf_path):
    with open(pdf_path, "rb") as pdfFile:
        pdfReader = PdfReader(pdfFile)#used pdfReader from pypdf2 to read the pdf
        numPages = len(pdfReader.pages)
        # Extracting text from each page
        all_text = ""
        for page_num in range(numPages):
            page = pdfReader.pages[page_num]
            text = page.extract_text()
            if text:
                all_text += text.encode('ascii', 'ignore').decode('ascii') + "\n" # Written this so as ignor Non- Ascii characters.
    return all_text
```

```python
def extract_text_from_url(url):
    response = requests.get(url)
    # Parsing the HTML content using BeautifulSoup
    soup = BeautifulSoup(response.text, 'html.parser')
    # Extracting the main content
    article_content = soup.find_all('p')
    text = '\n'.join([p.get_text() for p in article_content])#combined all paaragraphs

    # Return the text with non-ASCII characters removed
    return text.encode('ascii', 'ignore').decode('ascii')
```

- **Creating Chunks of the text extracted -** After extracting the text, I created chunks of it to facilitate personalized search functionality. This allows for efficient querying by calculating similarity scores from a vector database. This method enhances the search experience by providing more accurate and relevant results based on the given query.

```python
def get_text_chunks(text):
    #created Chunks of the Text data so as to get a personalized search
    text_splitter = RecursiveCharacterTextSplitter(chunk_size=10000, chunk_overlap=1000)
    chunks = text_splitter.split_text(text)
    return chunks
```

- **Creating embeddings of the chunks of text and stored in vector database** - After extracting the text and creating chunks, I generated embeddings for these text chunks using Google Generative AI Embeddings.Next, I stored the generated embeddings in a FAISS vector database. FAISS (Facebook AI Similarity Search) is a powerful tool designed for efficient similarity search and clustering of dense vectors. By storing the embeddings in FAISS.The vector database in which I saved the embeddings was saved so that we do not have to create the embeddings again and again.

```python
def get_vector_store(text_chunks):
    # Stored vector embeddings in FAISS vector db
    embeddings = GoogleGenerativeAIEmbeddings(model = "models/embedding-001")#model for creating vector embeddings
    vector_store = FAISS.from_texts(text_chunks, embedding=embeddings)
    vector_store.save_local("faiss_index")#I have saved the vector database so that I don't have to create and save embeddings agin and again !
```

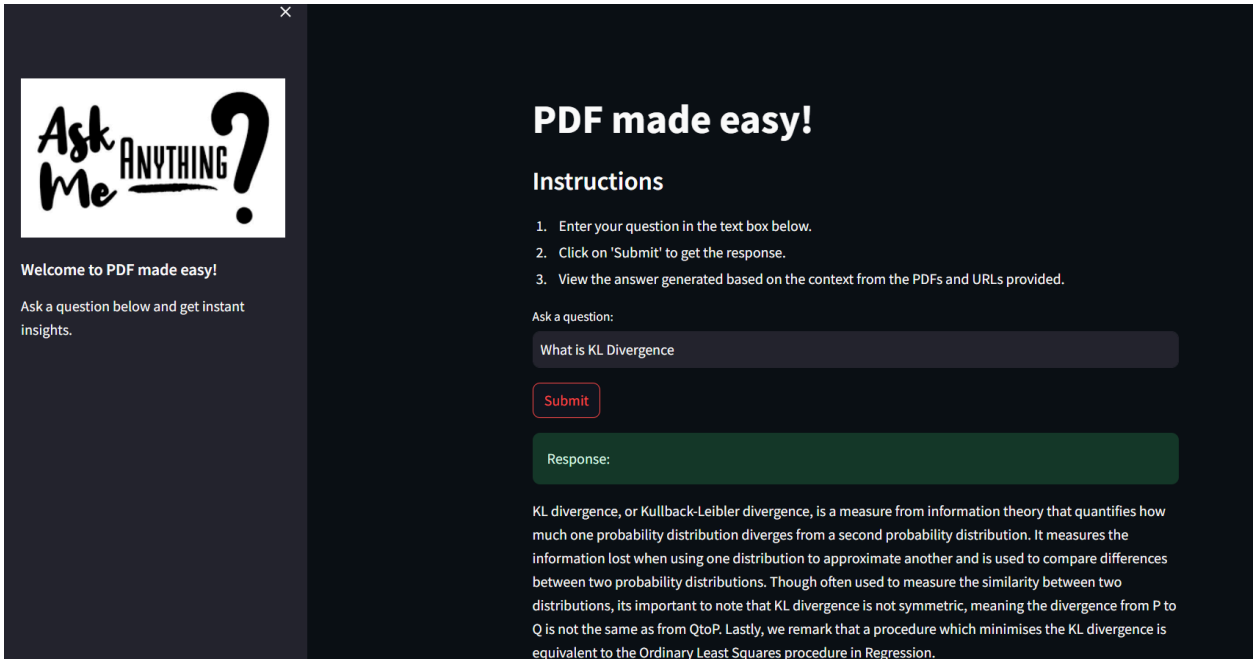
- **Chain Initialization** -The "get_conversational_chain" function establishes a conversational chain utilizing the Gemini-Pro model, integrating a prompt template to guide response generation based on provided context and user questions. Meanwhile, the "user_input" function facilitates user interaction by generating embeddings for queries using the Google Generative AI Embeddings model, conducting similarity searches in the Faiss index to retrieve relevant documents, and employing the conversational chain to produce detailed responses based on input documents and user inquiries.

```python
def get_conversational_chain():
    # It is a popular way to instruct the LLM to Give right answers ,So I used it.
    prompt_template = """
    Answer the question as detailed as possible from the provided context, make sure to provide all the details, if the answer is not in
    provided context just say, "answer is not available in the context", don't provide the wrong answer\n\n
    Context:\n {context}?\n
    Question: \n{question}\n

    Answer:
    """
    # Using gemini-pro model as LLM
    model = ChatGoogleGenerativeAI(model="gemini-pro",
                                   temperature=0.3)

    prompt = PromptTemplate(template = prompt_template, input_variables = ["context", "question"])
    chain = load_qa_chain(model, chain_type="stuff", prompt=prompt)#created a Stuff type chain to get a Q/A model
    return chain
```

```python
def user_input(user_question):

    embeddings = GoogleGenerativeAIEmbeddings(model = "models/embedding-001")#same model for creating embedding for the query...

    new_db = FAISS.load_local("faiss_index", embeddings, allow_dangerous_deserialization=True)#loaded the previously saved vector db
    docs = new_db.similarity_search(user_question)#getting all similiar text present in the Database with the query

    chain = get_conversational_chain()


    response = chain(
        {"input_documents":docs, "question": user_question}
        , return_only_outputs=True)

    return response
```

App.py Functions

- **Creating UI -** Lastly ,I just created the UI for the whole model using streamlit framework.



Hope You will Like the assignment .

Thank You for the awesome task !!!

Shaurya Mishra
