GOKARAJU RANGARAJU INSTITUTE OF ENGINEERING AND TECHNOLOGY
Department of Computer Science and Engineering
Major Project Review

Document Assistant using AI

-Ms. B Indu Priya,
Assistant Professor.

K V Chandrahas -20241A05K4
P Avinash Reddy -20241A05M5
Y Satya Narayana Reddy -20241A05N9
N Chethan Goud -21245A0524

[illegible]

Abstract:

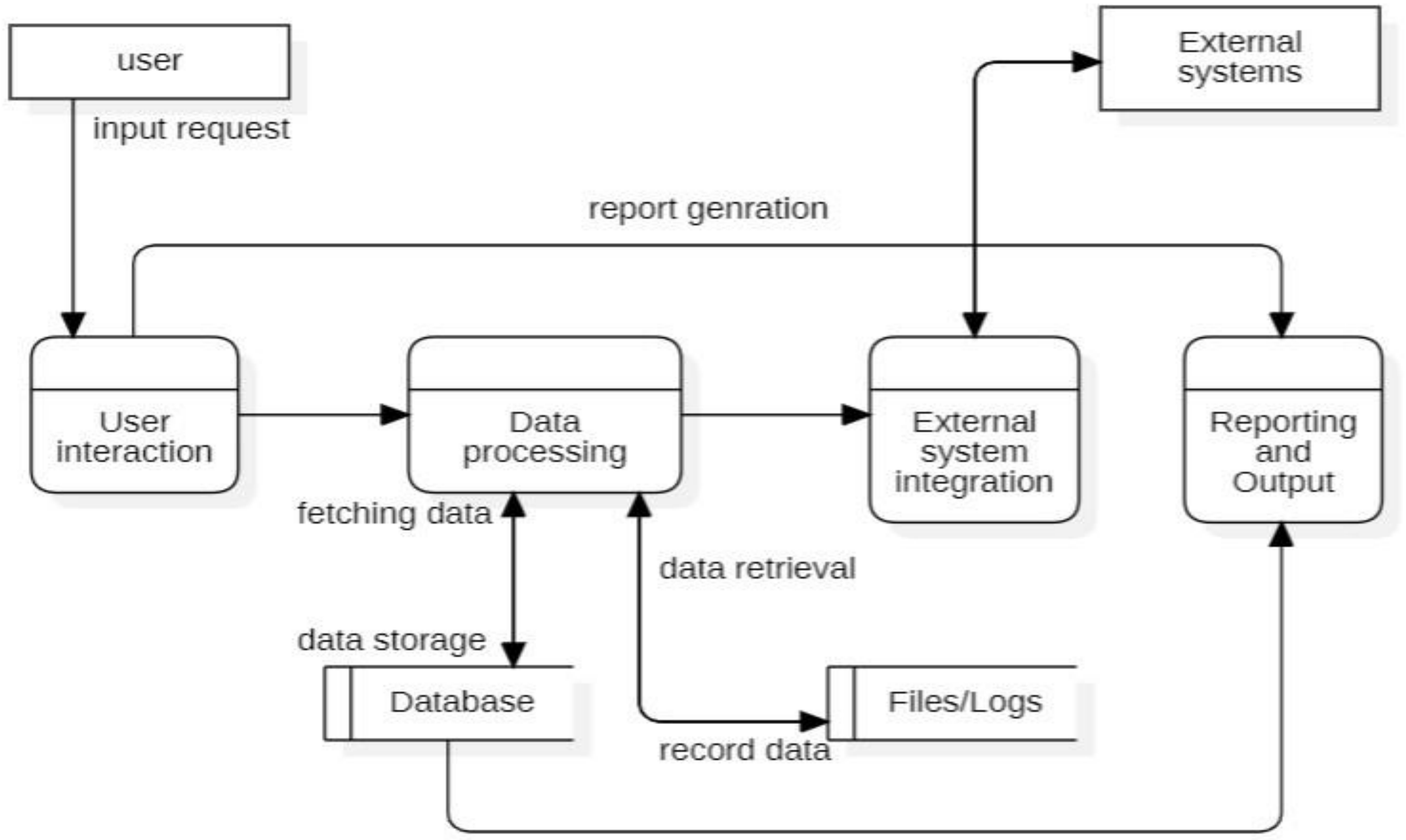
PDFs have pervaded modern life, acting as vital documentation tool in various contexts. Yet, the aspiration for "Data-On-Demand" presents a novel challenge: immediate and accessible data availability. Large Language Models (LLMs) offer a promising resolution. It is a type of machine learning model that can perform a variety of natural language processing (NLP) tasks such as generating and classifying text, answering questions in a conversational manner, and translating text from one language to another. By harnessing potent algorithms, LLMs reconfigure PDF interaction dynamics. Transforming PDFs into interactive conversations augments document processing and enables real-time engagement, effectively transcending the confines of conventional documents. LLMs, trained on extensive textual datasets, simulate human-like interactions. This transformative approach empowers users to interact with PDFs using natural language, seamlessly extracting pertinent information. To realize this concept, a web platform is proposed, allowing users to upload or scan PDFs and make inquiries in natural language. The LLM-powered system extracts and presents information, ushering in a paradigm shift in document interaction and data accessibility.

Proposed System:

The proposed system aims to enhance the accessibility and utility of textual documents, particularly PDFs, by implementing an automated question-answering solution. This system harnesses cutting-edge technologies, including LangChain for document analysis, PyPDF2 for PDF text extraction, and FAISS for text indexing. The process begins with the installation and configuration of essential libraries and tools. PyPDF2 is used to extract text from PDF documents, which is then divided into manageable chunks for efficient indexing. OpenAI's GPT-3 model is employed via LangChain to answer user questions related to the document content. This system offers numerous benefits, catering to students, researchers, professionals, and anyone seeking specific insights from extensive textual resources. Future improvements may involve creating a user-friendly web interface, optimizing performance for larger documents, and ensuring robust error handling and data security. In conclusion, the project demonstrates the potential of automated question answering to make textual documents more accessible, informative, and user-friendly.

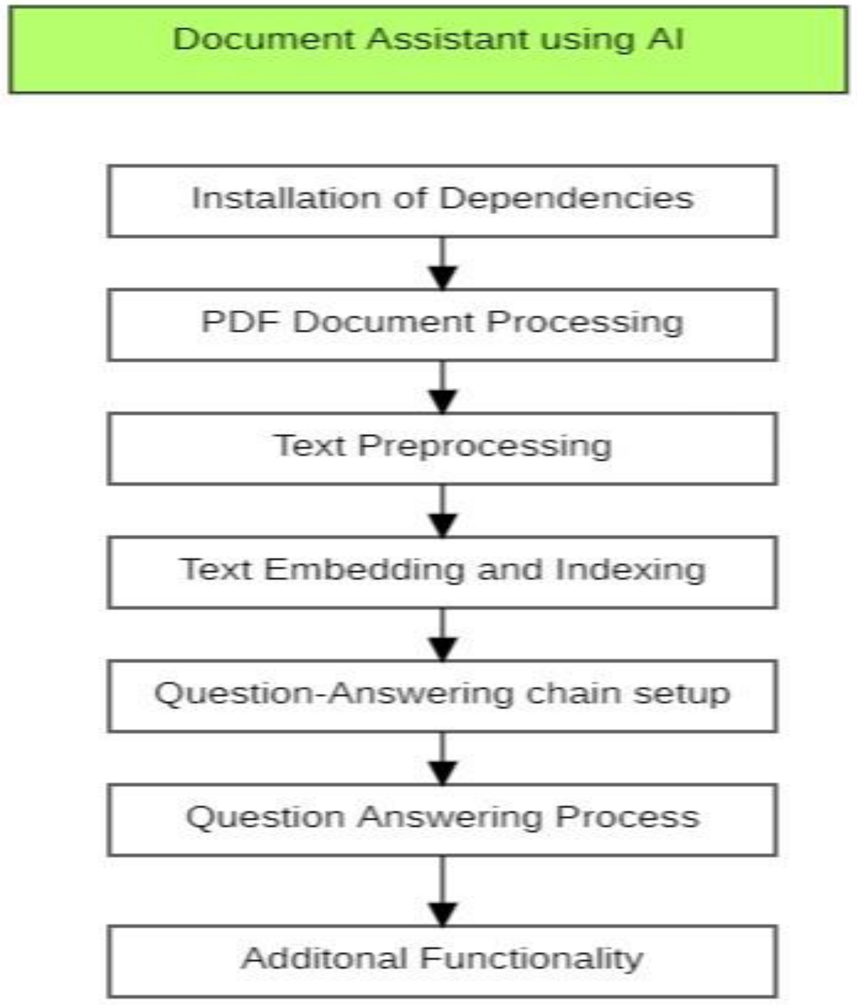
GOKARAJU RANGARAJU INSTITUTE OF ENGINEERING AND TECHNOLOGY
Department of Computer Science and Engineering
Major Project Review - 2

Architecture:



GOKARAJU RANGARAJU INSTITUTE OF ENGINEERING AND TECHNOLOGY
Department of Computer Science and Engineering
Major Project Review - 2

Design Methodology:


Document Assistant using AI
Installation of Dependencies
PDF Document Processing
Text Preprocessing
Text Embedding and Indexing
Question-Answering chain setup
Question Answering Process
Additonal Functionality

Design Methodology:

1. Installation of Dependencies: The code begins by installing necessary Python packages and libraries using pip, including LangChain, OpenAI, PyPDF2, tiktoken, and faiss-cpu. It sets up the OpenAI API key as an environment variable, which is required for authentication with the OpenAI API.

2. PDF Document Processing: The code uses the PyPDF2 library to extract text content from a specified PDF document .It reads each page of the PDF, extracts the text, and concatenates it into a single string variable called raw_text.

3. Text Preprocessing: To prepare the extracted text for analysis and indexing, the code splits it into smaller, manageable chunks using a character-based text splitter. The text is divided into chunks with a specified chunk size and overlap to facilitate efficient indexing.

4. Text Embedding and Indexing: The code uses OpenAIEmbeddings to embed the text data. Embeddings capture semantic relationships within the text. A vector index is created using FAISS, which allows for efficient similarity-based search of text chunks.

5. Question-Answering Chain Setup: The LangChain library is utilized to load a question-answering chain. The chain leverages OpenAI's GPT-3 model, a state-of-the-art language model, for question answering. The chain type "stuff" is loaded, which is designed for question answering.

6. Question-Answering Process: Users can input questions related to the document's content. The code searches for relevant text chunks using the FAISS index based on the user's query. The retrieved text chunks are then fed into the question-answering chain, which generates answers to the user's questions based on the content of the retrieved chunks. The answers are returned as the output of the system.

7. Additional Functionality: The code also demonstrates the use of a map-re rank question-answering chain to provide more comprehensive answers. It sets up FAISS as a generic retriever for similarity-based searches.

Modules in the project:

1. Document Processing Module:

The document processing module is responsible for preparing the document for question answering. It does this by extracting text from the PDF document, dividing the extracted text into tokens, and converting the text tokens into a format conducive to further analysis using OpenAIEmbeddings. Extracting text from the PDF document: The document processing module uses the PyPDF2 library to extract text from the PDF document. This process involves identifying the different pages in the PDF document and then extracting the text from each page. Dividing the extracted text into tokens: The document processing module then divides the extracted text into tokens. Tokens are the basic units of text that are used for NLP tasks such as question answering. Tokens can be words, punctuation marks, or other symbols. Converting the text tokens into a format conducive to further analysis using OpenAIEmbeddings: The document processing module then uses the OpenAIEmbeddings library to convert the text tokens into a format conducive to further analysis. OpenAIEmbeddings creates dense vector representations of text, which can be used for a variety of NLP tasks, including question answering.

2. Question Answering Module:

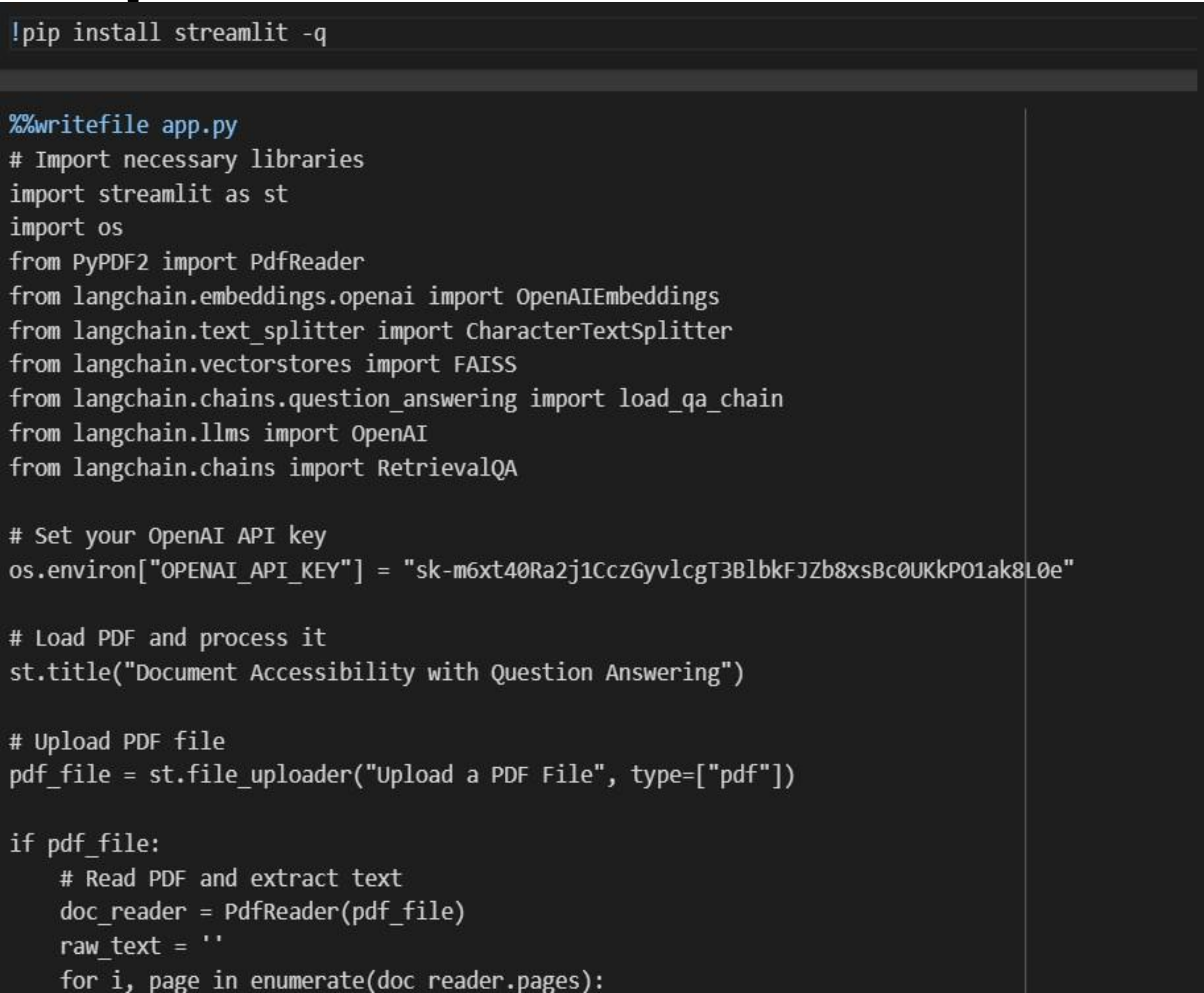
The question answering module is responsible for providing users with answers to their questions about the document. It does this by using the QA chain loaded via LangChain.

Loading the QA chain via LangChain: The question answering module uses the LangChain library to load the QA chain. The QA chain is a pipeline of NLP and ML components that are used to answer questions about a given document.

Providing users with the ability to input their questions related to the document content: The question answering module provides users with the ability to input their questions related to the document content. The QA chain will then endeavor to provide relevant and coherent answers.

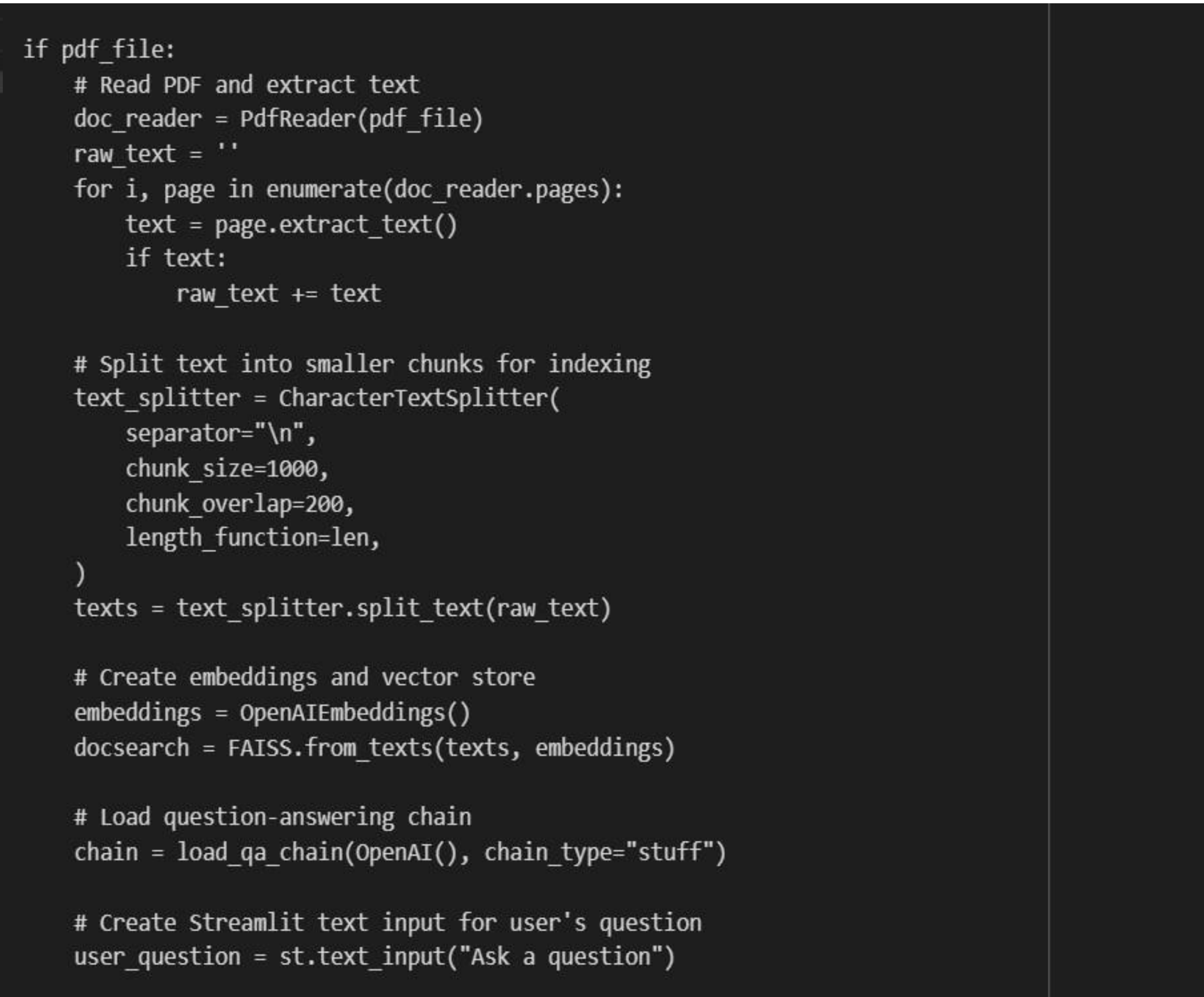
GOKARAJU RANGARAJU INSTITUTE OF ENGINEERING AND TECHNOLOGY
Department of Computer Science and Engineering
Major Project Review

Implementation



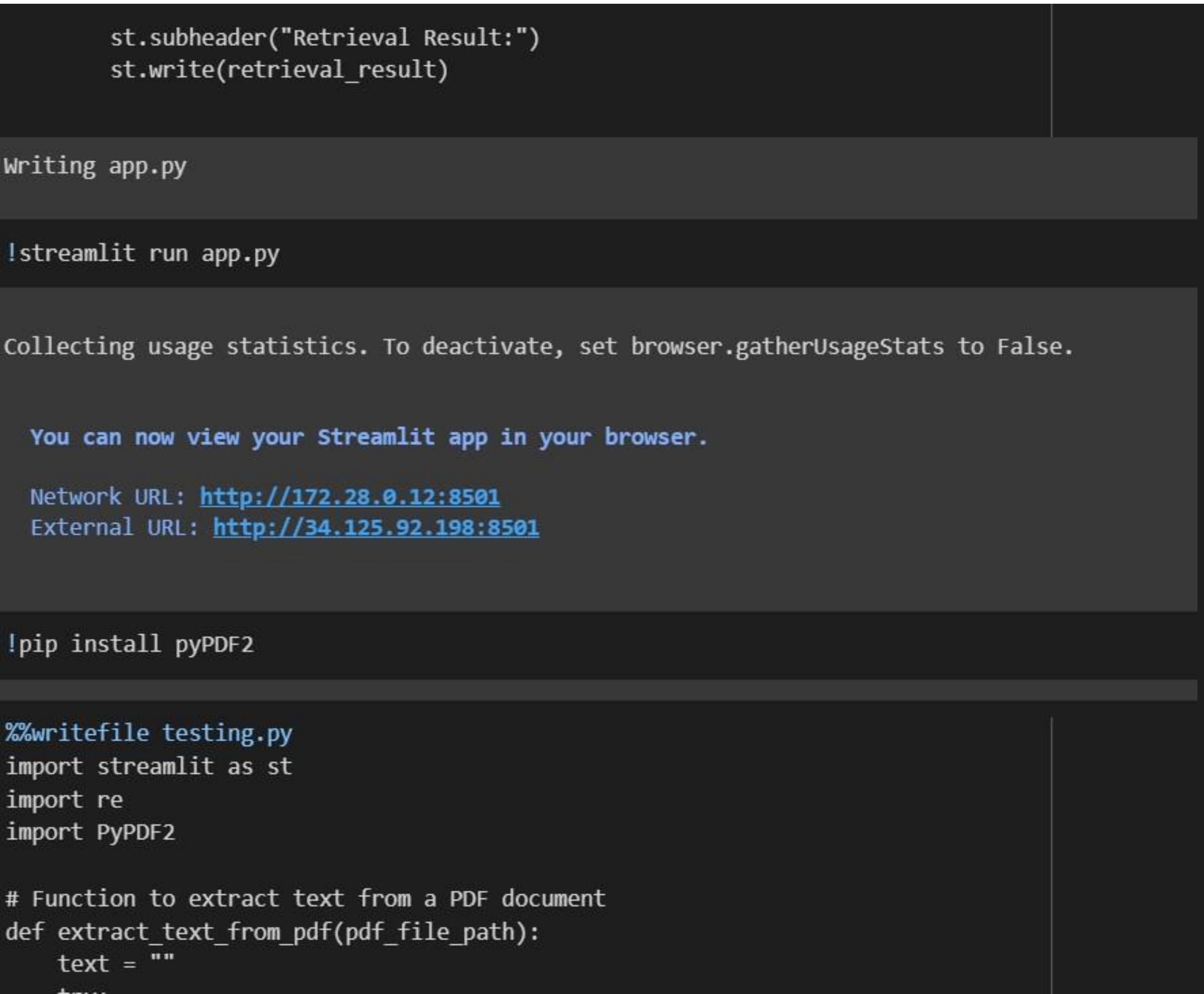
```
!pip install streamlit -q
```

```
%%writefile app.py
# Import necessary libraries
import streamlit as st
import os
from PyPDF2 import PdfReader
from langchain.embeddings.openai import OpenAIEmbeddings
from langchain.text_splitter import CharacterTextSplitter
from langchain.vectorstores import FAISS
from langchain.chains.question_answering import load_qa_chain
from langchain.llms import OpenAI
from langchain.chains import RetrievalQA

# Set your OpenAI API key
os.environ["OPENAI_API_KEY"] = "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"

# Load PDF and process it
st.title("Document Accessibility with Question Answering")

# Upload PDF file
pdf_file = st.file_uploader("Upload a PDF File", type=["pdf"])

if pdf_file:
    # Read PDF and extract text
    doc_reader = PdfReader(pdf_file)
    raw_text = ''
    for i, page in enumerate(doc_reader.pages):
```



```
if pdf_file:
    # Read PDF and extract text
    doc_reader = PdfReader(pdf_file)
    raw_text = ''
    for i, page in enumerate(doc_reader.pages):
        text = page.extract_text()
        if text:
            raw_text += text

    # Split text into smaller chunks for indexing
    text_splitter = CharacterTextSplitter(
        separator="\n",
        chunk_size=1000,
        chunk_overlap=200,
        length_function=len,
    )
    texts = text_splitter.split_text(raw_text)

    # Create embeddings and vector store
    embeddings = OpenAIEmbeddings()
    docsearch = FAISS.from_texts(texts, embeddings)

    # Load question-answering chain
    chain = load_qa_chain(OpenAI(), chain_type="stuff")

    # Create Streamlit text input for user's question
    user_question = st.text_input("Ask a question")
```

GOKARAJU RANGARAJU INSTITUTE OF ENGINEERING AND TECHNOLOGY
Department of Computer Science and Engineering
Major Project Review



```
        st.subheader("Retrieval Result:")
        st.write(retrieval_result)
```

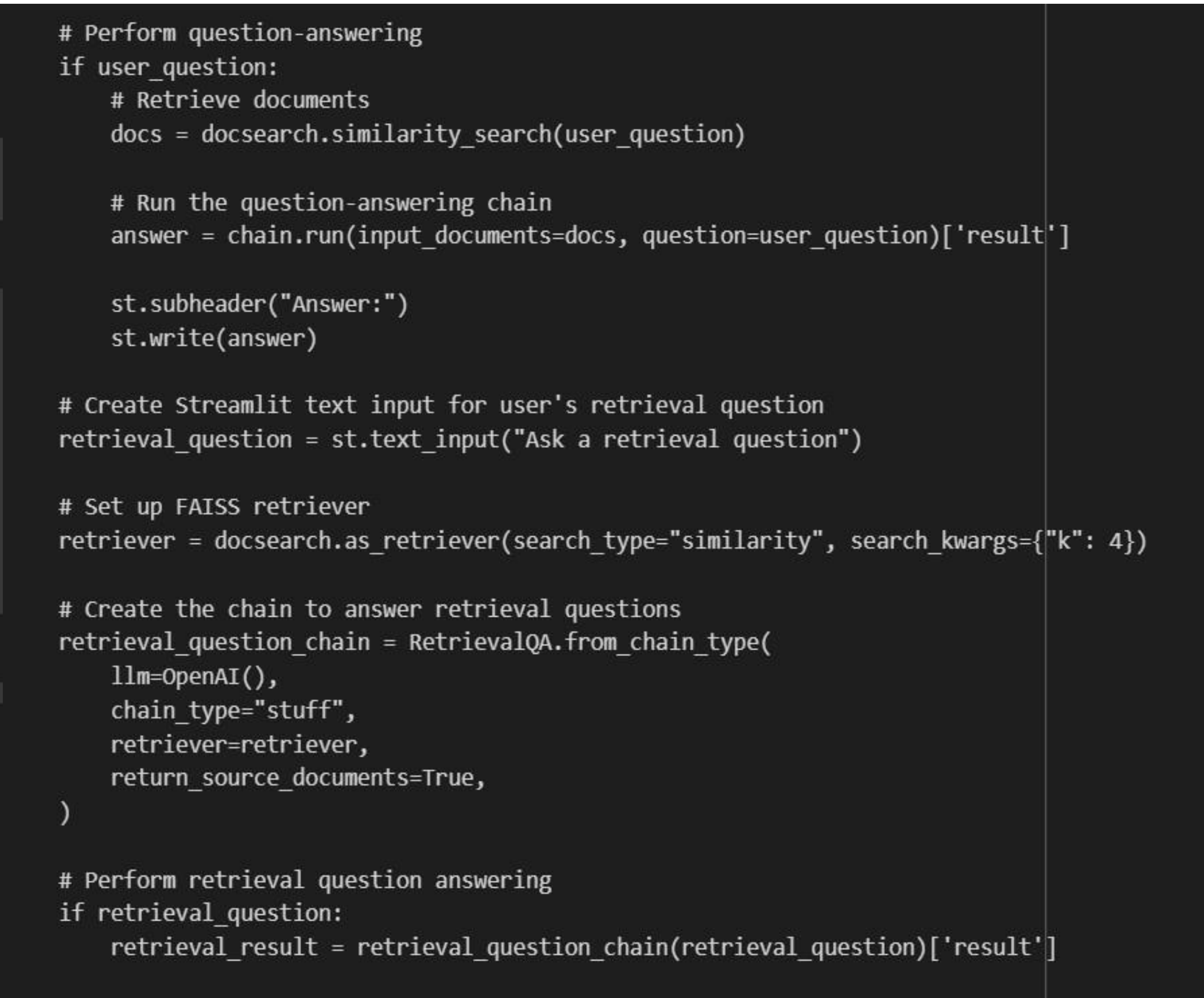
```
Writing app.py
```

```
!streamlit run app.py
```

```
Collecting usage statistics. To deactivate, set browser.gatherUsageStats to False.


  You can now view your Streamlit app in your browser.

  Network URL: http://172.28.0.12:8501
  External URL: http://34.125.92.198:8501
```

```
!pip install pyPDF2
```

```
%%writefile testing.py
import streamlit as st
import re
import PyPDF2

# Function to extract text from a PDF document
def extract_text_from_pdf(pdf_file_path):
    text = ""
```

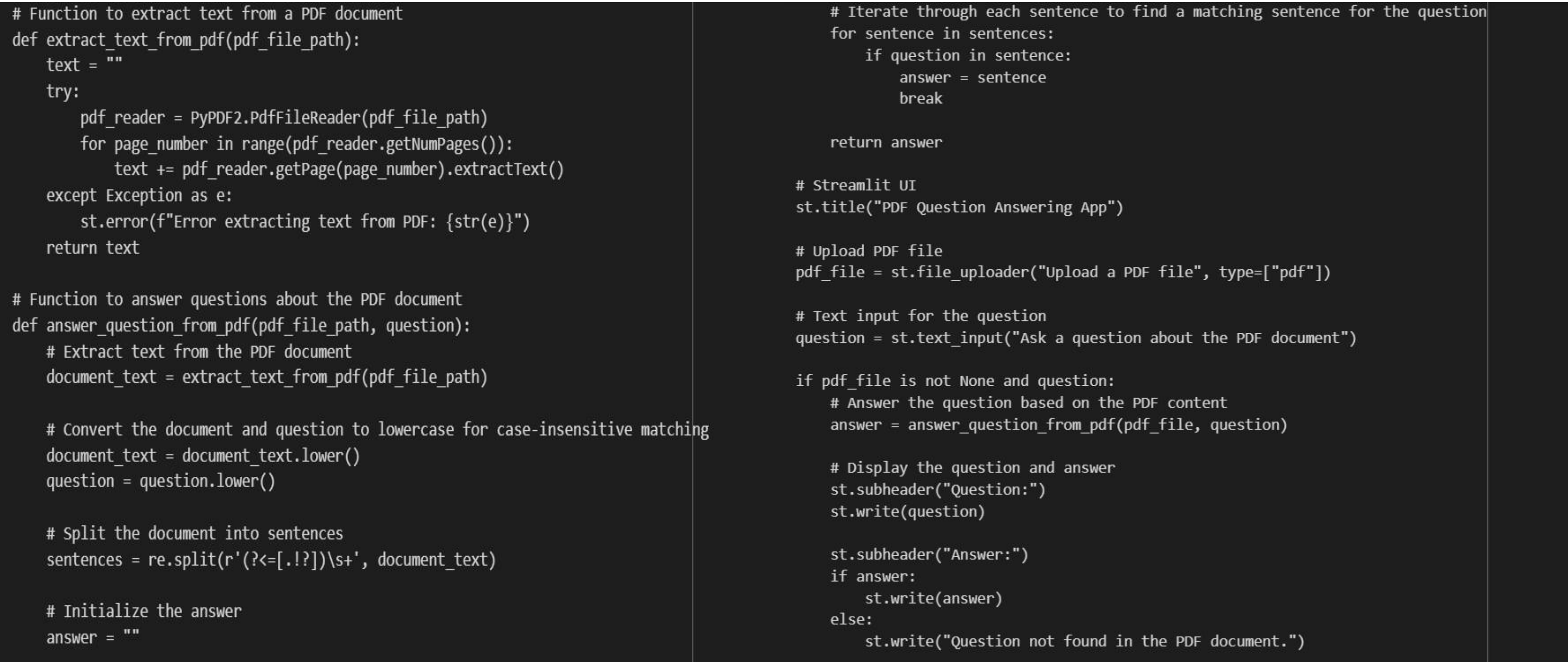


```
# Perform question-answering
if user_question:
    # Retrieve documents
    docs = docsearch.similarity_search(user_question)

    # Run the question-answering chain
    answer = chain.run(input_documents=docs, question=user_question)['result']

    st.subheader("Answer:")
    st.write(answer)

# Create Streamlit text input for user's retrieval question
retrieval_question = st.text_input("Ask a retrieval question")

# Set up FAISS retriever
retriever = docsearch.as_retriever(search_type="similarity", search_kwargs={"k": 4})

# Create the chain to answer retrieval questions
retrieval_question_chain = RetrievalQA.from_chain_type(
    llm=OpenAI(),
    chain_type="stuff",
    retriever=retriever,
    return_source_documents=True,
)

# Perform retrieval question answering
if retrieval_question:
    retrieval_result = retrieval_question_chain(retrieval_question)['result']
```

GOKARAJU RANGARAJU INSTITUTE OF ENGINEERING AND TECHNOLOGY

Department of Computer Science and Engineering

Major Project Review

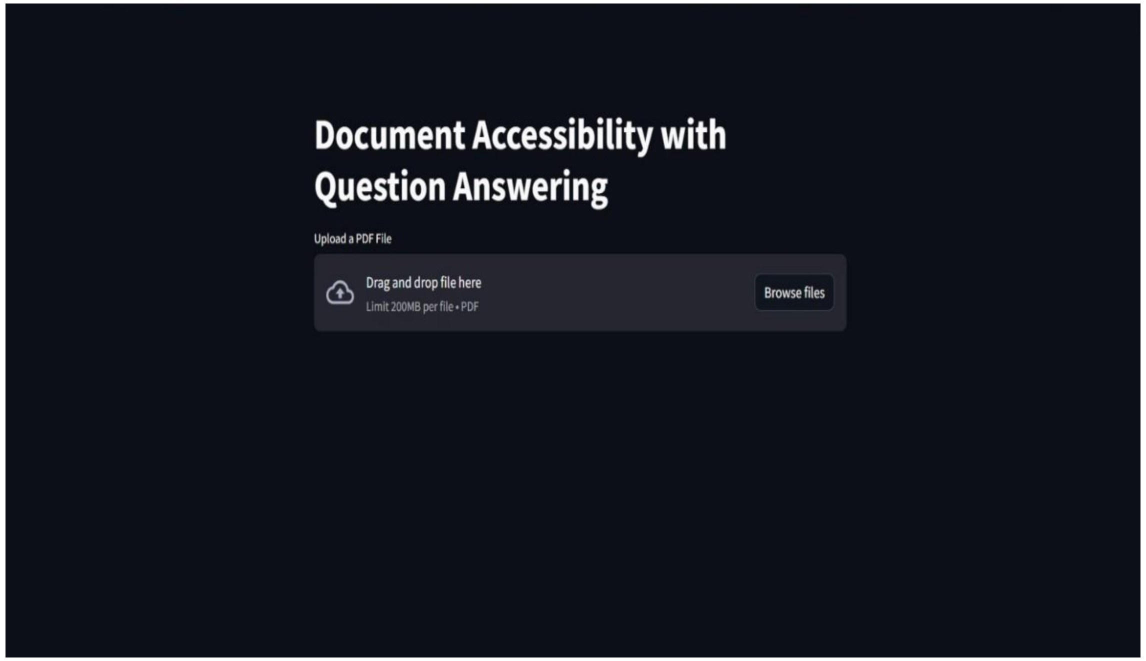


```python
# Function to extract text from a PDF document
def extract_text_from_pdf(pdf_file_path):
    text = ""
    try:
        pdf_reader = PyPDF2.PdfFileReader(pdf_file_path)
        for page_number in range(pdf_reader.getNumPages()):
            text += pdf_reader.getPage(page_number).extractText()
    except Exception as e:
        st.error(f"Error extracting text from PDF: {str(e)}")
    return text

# Function to answer questions about the PDF document
def answer_question_from_pdf(pdf_file_path, question):
    # Extract text from the PDF document
    document_text = extract_text_from_pdf(pdf_file_path)

    # Convert the document and question to lowercase for case-insensitive matching
    document_text = document_text.lower()
    question = question.lower()

    # Split the document into sentences
    sentences = re.split(r'(?<=[.!?])\s+', document_text)

    # Initialize the answer
    answer = ""
```

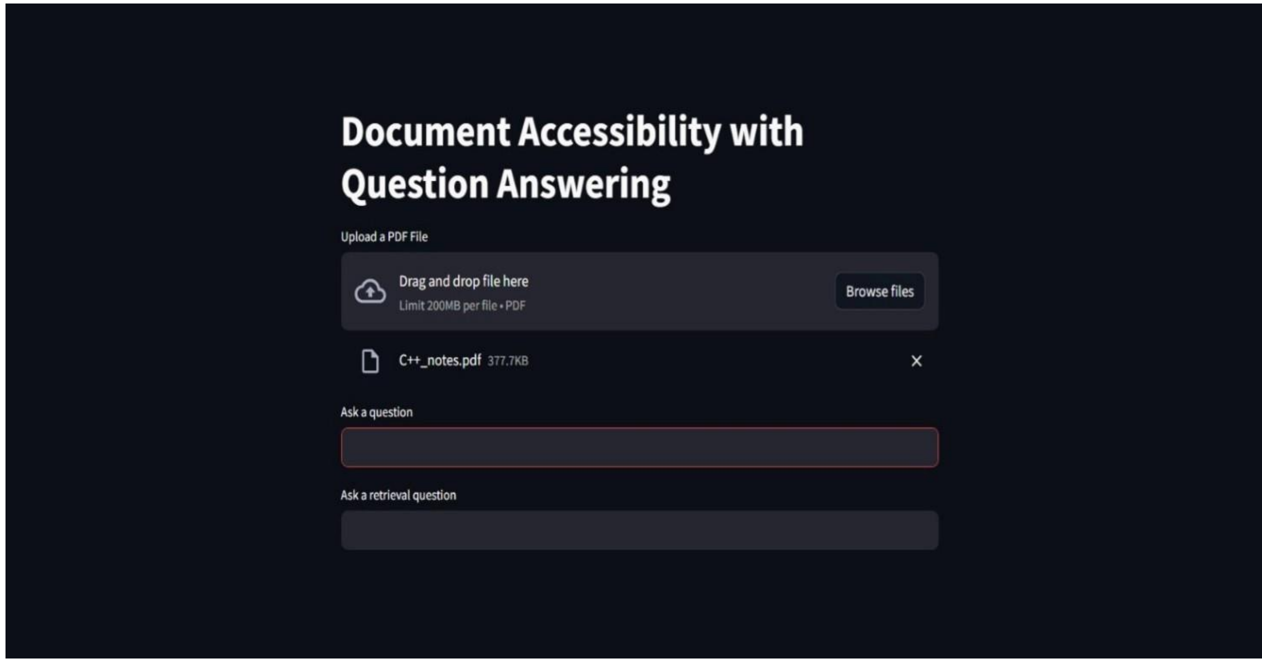
```python
    # Iterate through each sentence to find a matching sentence for the question
    for sentence in sentences:
        if question in sentence:
            answer = sentence
            break

    return answer

# Streamlit UI
st.title("PDF Question Answering App")

# Upload PDF file
pdf_file = st.file_uploader("Upload a PDF file", type=["pdf"])

# Text input for the question
question = st.text_input("Ask a question about the PDF document")

if pdf_file is not None and question:
    # Answer the question based on the PDF content
    answer = answer_question_from_pdf(pdf_file, question)

    # Display the question and answer
    st.subheader("Question:")
    st.write(question)

    st.subheader("Answer:")
    if answer:
        st.write(answer)
    else:
        st.write("Question not found in the PDF document.")
```

GOKARAJU RANGARAJU INSTITUTE OF ENGINEERING AND TECHNOLOGY
Department of Computer Science and Engineering
Major Project Review

Output


Document Accessibility with Question Answering

Upload a PDF File


Document Accessibility with Question Answering

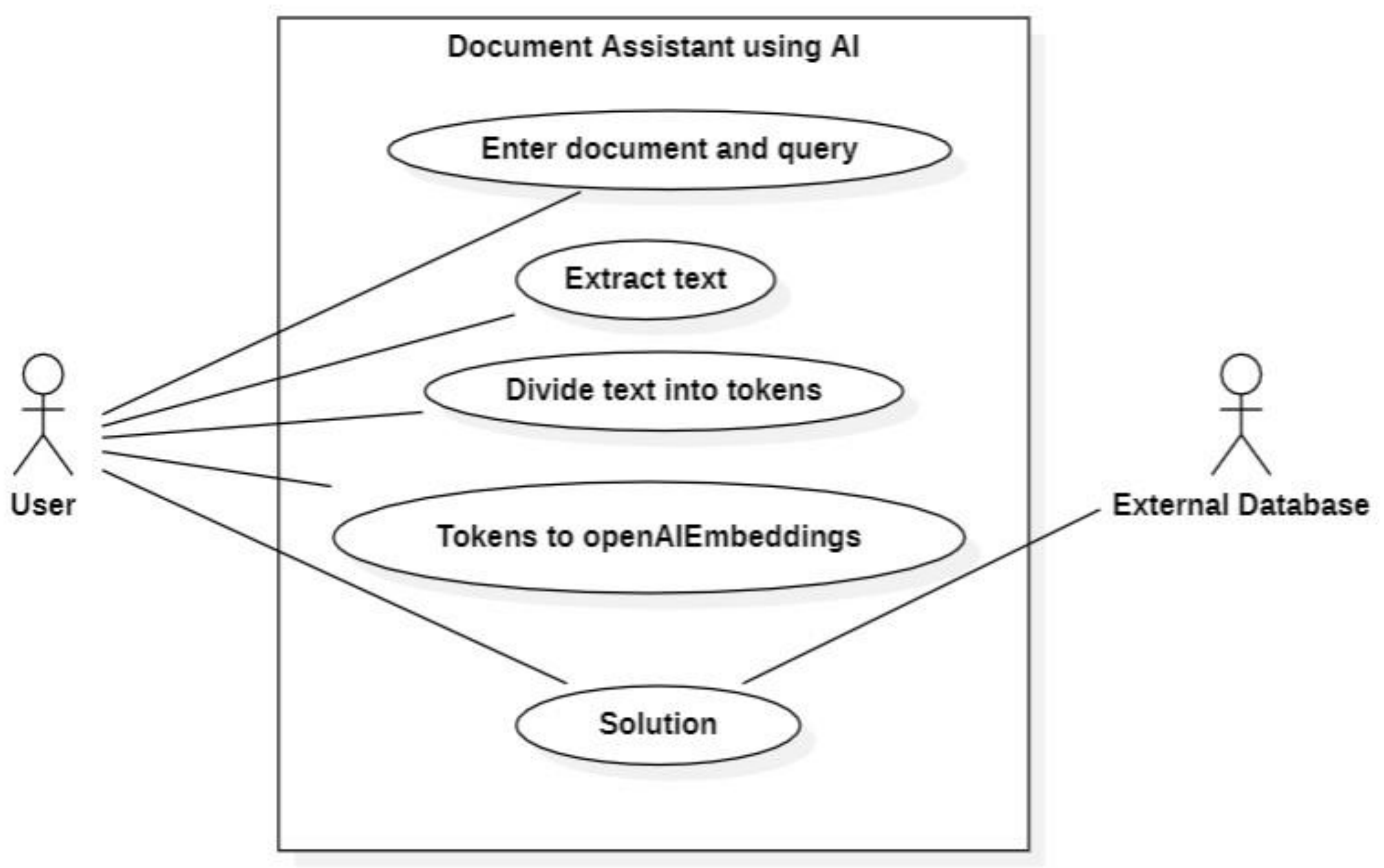
Upload a PDF File

C++_notes.pdf 377.7KB

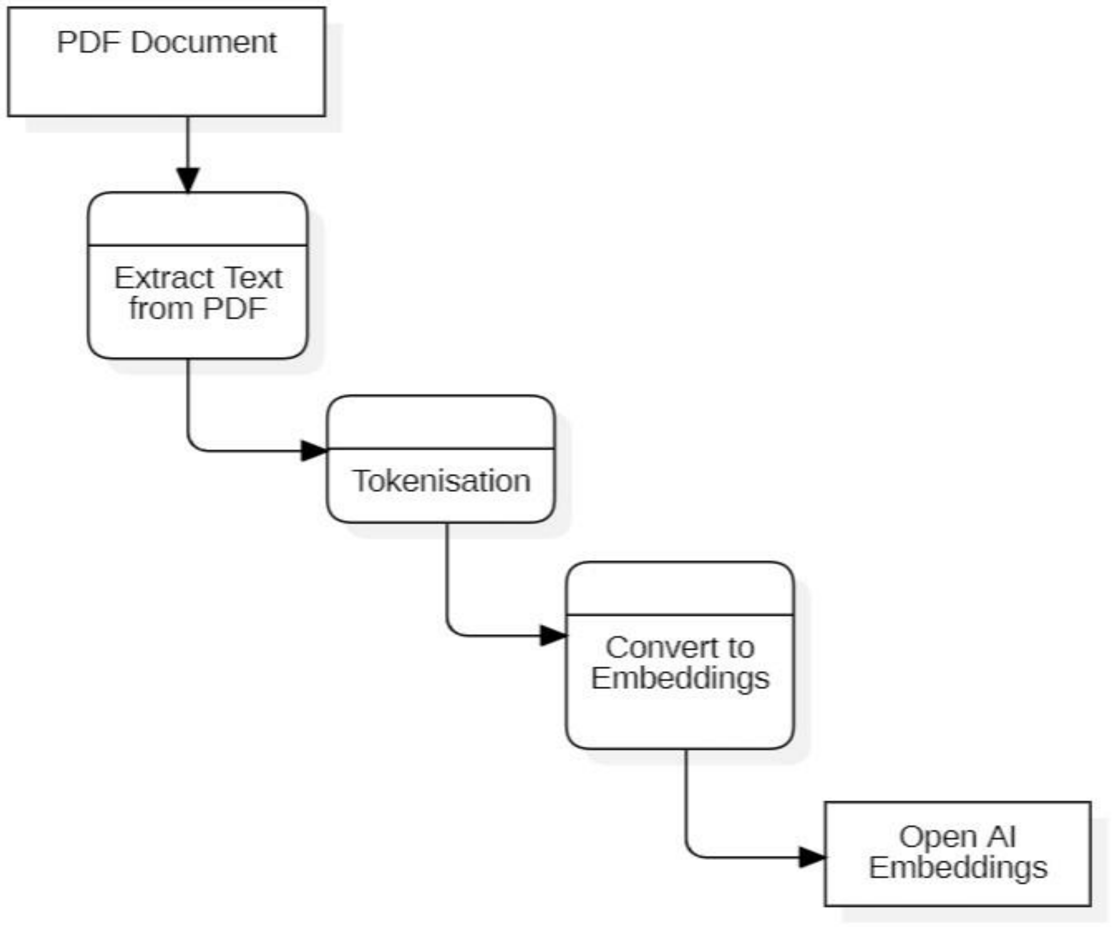
Ask a question

Ask a retrieval question

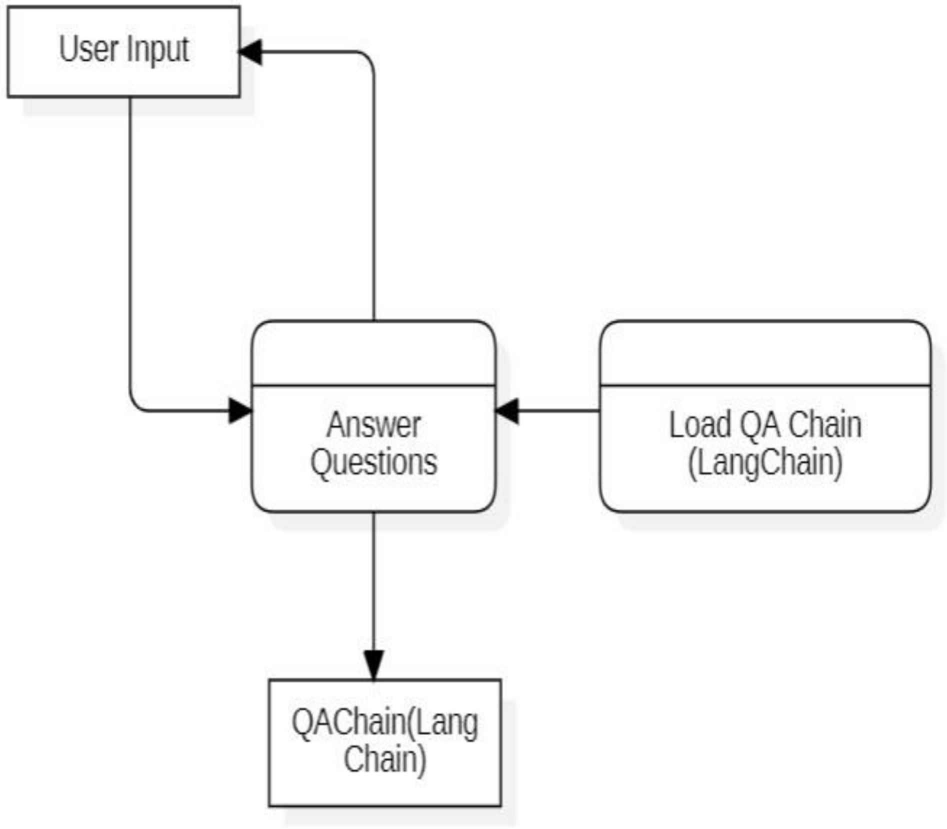
UML Diagrams:



Document Processing Module:



Question – Answering Module:



Scope Limitations:

1. **Scalability:** Document assistants need to be able to handle large volumes of documents and requests. This can be challenging because document assistants need to be able to process and understand documents in real time.

2. **User Experience:** In the previous projects chatbot performance was only considered for evaluation. User experience and interactive user interface must also be considered.

3. **Ethics:** Document assistants need to be developed and used in an ethical way. This is important because document assistants could be used to spread misinformation or to manipulate people.

References:

Base paper link 1: Paper 1

Base paper link 2: Paper 2

Base paper link 3: Paper 3

Base paper link 4: Paper 4

GOKARAJU RANGARAJU INSTITUTE OF ENGINEERING AND TECHNOLOGY
Department of Computer Science and Engineering
Major Project Review

Review Paper

Introduction:

In the contemporary landscape of documentation, Portable Document Format (PDFs) has become an indispensable tool. However, the desire for "Data-On-Demand" introduces a novel challenge: ensuring immediate and accessible data availability. Addressing this challenge, Large Language Models (LLMs) emerge as a potent solution. Proficient in diverse natural language processing (NLP) tasks, LLMs, such as those used in this proposed system, have the potential to redefine how we interact with PDFs. This review explores the transformative role of LLMs in enabling real-time engagement and overcoming the limitations of traditional document structures. By simulating human-like interactions through extensive training on textual datasets, LLMs empower users to interact with PDFs using natural language, ushering in a paradigm shift in document interaction dynamics. The envisaged web platform offers a tangible realization of this concept, allowing users to effortlessly upload or scan PDFs and pose inquiries in natural language. The LLM-powered system then extracts and presents information, promising a revolution in document interaction and data accessibility.

GOKARAJU RANGARAJU INSTITUTE OF ENGINEERING AND TECHNOLOGY

Department of Computer Science and Engineering

Major Project Review

Review Paper

Theory:

At the heart of LLM-powered document interaction lies a convergence of advanced machine learning and natural language processing (NLP) methodologies. Large Language Models, characterized by their ability to discern intricate patterns in vast textual datasets, operate on deep learning principles. These models, grounded in NLP, leverage techniques such as tokenization, named entity recognition, and language modeling to comprehend and generate human-like text. The document processing module further solidifies the theoretical framework. Using the PyPDF2 library, the system extracts text from PDFs by identifying different pages, which is then tokenized into basic units suitable for NLP tasks. The OpenAIEmbeddings library plays a pivotal role by converting these text tokens into dense vector representations, enhancing the document's suitability for further NLP analysis. This intricate theoretical foundation transforms static PDFs into dynamic, interactive conversations, redefining the boundaries of document processing. The synthesis of advanced theories in this LLM-powered system propels document exploration into a new era where users can engage with PDFs in a manner that mirrors natural language interactions, marking a significant advancement in document processing and data accessibility.

Conclusion:

- Chatting with PDFs using LLMs is a promising new technology with the potential to revolutionize the way we interact with documents. It has the potential to improve information retrieval, creativity, learning, decision-making, and more. However, it is still in its early stages of development, and there are some challenges that need to be addressed before it can be widely adopted.

- Despite these challenges, the potential benefits of chatting with PDFs using LLMs are significant. This technology has the potential to make information more accessible and useful, to help us learn more effectively, and to make better decisions. As the technology continues to develop, we can expect to see even more benefits from chatting with PDFs using LLMs.

GOKARAJU RANGARAJU INSTITUTE OF ENGINEERING AND TECHNOLOGY
Department of Computer Science and Engineering
Major Project Review

Thank You

Any Queries Please?