PDF summarizer and query resolver chatbot:

Objective : This PDF Summarizer and Query Resolver Chatbot is an AI-powered tool designed to help users extract insights from PDF documents effortlessly. It allows users to upload a PDF, generates concise summaries, and answers specific questions based on the document's content. Built with Streamlit , OpenAI's GPT API, and FAISS, the chatbot ensures quick, relevant responses. Ideal for professionals and students, it saves time by streamlining the process of understanding complex documents.

- If you need any further explanation just paste this code in chatgpt it will explain everything.

Modules used : streamlit, langchain, FAISS

To run this code you need OPENAI's API : to know about open ai api
Link: https://youtu.be/nafDyRsVnXU?si=nJieUgIUpApMq6jQ
To use this api key you have to pay money to it. (if can't buy, you can add it your resume and if interviewer ask just say that i borrowed key from somebody, after that they deleted the key)

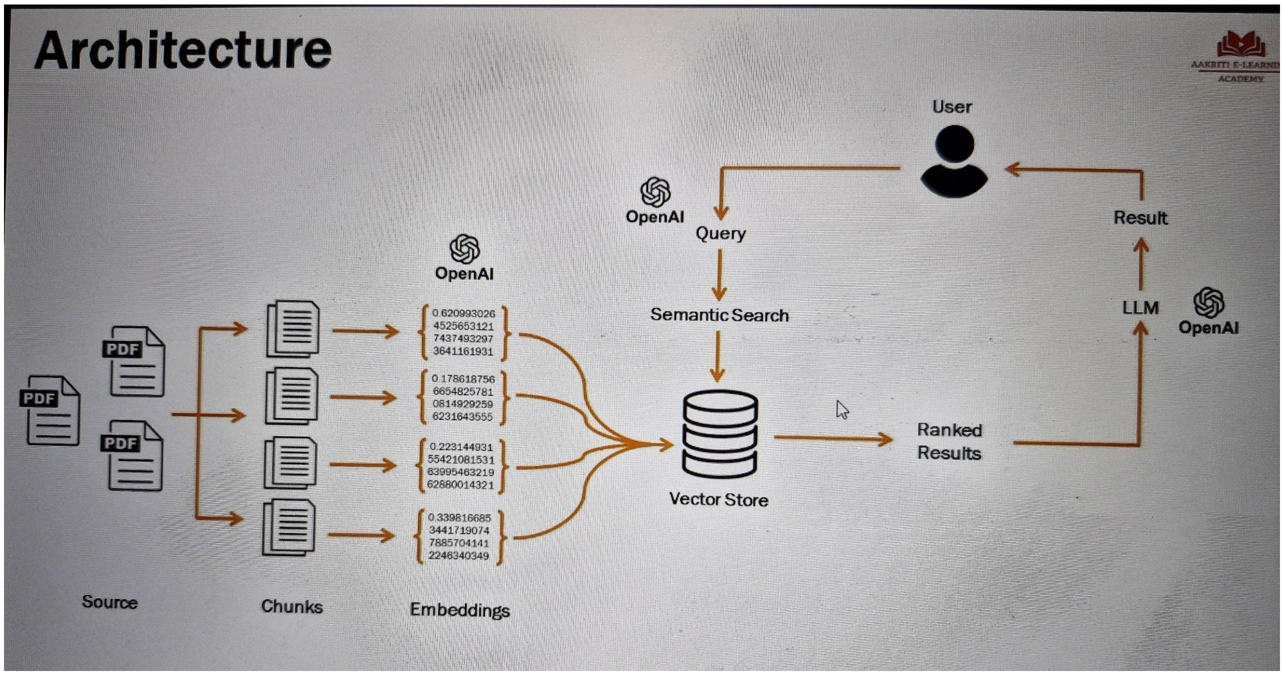
Code:

```
import streamlit as st
from PyPDF2 import PdfReader
from langchain.text_splitter import RecursiveCharacterTextSplitter
from langchain.embeddings import OpenAIEmbeddings
from langchain.vectorstores import FAISS
from langchain.chains.question_answering import load_qa_chain
from langchain_community.chat_models import ChatOpenAI

OPENAI_API_KEY = "enter your api key here"

st.header("My First Chatbot")

with st.sidebar: #for side bar in the website
    st.title("pdf summarizer")
    file= st.file_uploader("upload your pdf document", type = "pdf")

#to read & extract the data
```

```
if file is not None:
    pdf_reader= PdfReader(file)
    text=""
    for page in pdf_reader.pages:
        text += page.extract_text()
        #st.write(text)

#break it into chunks
    text_splitter = RecursiveCharacterTextSplitter(
        separators="\n",
        chunk_size= 1000,
        chunk_overlap = 150,
        length_function = len
    )

    chunks = text_splitter.split_text(text)
    #st.write(chunks)

    #generating embeddings
    embeddings = OpenAIEmbeddings(openai_api_key = OPENAI_API_KEY)

    #Creating vector store
    vector_store = FAISS.from_texts(chunks,embeddings)

    #user question
    user_question = st.text_input("enter your question")

    if user_question:
        match = vector_store.similarity_search(user_question)
        #st.write(match)

        #define the llm
        llm = ChatOpenAI(
            openai_api_key = OPENAI_API_KEY,
            temperature= 0,
            max_tokens=1000,
            model_name = "gpt-3.5-turbo"
        )

        #output results
```

```
chain = load_qa_chain(llm, chain_type= "stuff")
response = chain.run(input_documents = match, question = user_question)
st.write(response)
```

The architecture of the project :



Interviewer may ask you questions based on this project :

Technical difficulties ;

One challenge was managing chunk size, if we give random number of chunk size, the matter it will create may go out of context. So we used recursive character textspillter from langchain. which allows for controlled chunk size with slight overlap to ensure context is preserved between chunks.

Why did you choose Streamlit for this project? Were there any other tools you considered for the front-end interface?"
Answer: Streamlit was chosen for its simplicity and rapid development capability, allowing me to focus more on backend functionalities rather than spending a lot of time on frontend details. Other options I considered were Flask and Django, but Streamlit's support for interactive widgets, like file uploads and text input, made it ideal for this prototype.

Can you explain the role of LangChain in this project, and why it was necessary?"
Answer: LangChain was essential for breaking down text into chunks, generating embeddings, and facilitating question-answering functionality. It streamlined handling large texts and working with the embeddings, which would have been more complex and time-consuming to implement from scratch.

Further Improvements:

- *"What improvements would you consider adding if you had more time or resources?"*
- **Answer**: I'd like to add multi-document support, allowing users to query across multiple PDFs. Additionally, integrating multi-language support would make it more versatile, and optimizing the processing time for even larger documents would improve performance.