DOCUMENT BASED Q&A SYSTEM

A STREAMLINED INFORMATION RETRIEVAL

BY

ANSH AYA

Structure of the Report

The report will comprise comprehensive sections covering various facets of the Document Based Q&A System, including:

- Introduction: Description and details of a comprehensive document management system.
- Project Overview: This section provides a brief overview of the project, outlining its purpose and scope.
- Objectives: The objectives of the project are clearly defined to guide its development and measure its success.
- Relevance: The relevance of the project to current challenges or opportunities in its domain is discussed to highlight its importance.
- Problem Formulation: A concise formulation of the problem addressed by the project is presented to provide context for the proposed solution.
- Motivation: The motivations driving the development of the project are articulated, shedding light on the underlying reasons for its undertaking.
- Scope of the Project: The scope of the project is delineated, outlining the boundaries and limitations within which it operates.
- System architecture: Overview of the system's design and components.
- Data modeling: Description of the data structures and models used for document representation and processing.
- Implementation details: Insights into the technical aspects of system development and deployment.
- Results analysis: Examination of the system's performance and efficacy based on experimental findings and user feedback.
- Conclusion: Conclusion of the application that drives productivity and innovation in the digital age

Introduction

I introduce the Document Based Q&A System project, a cutting-edge initiative aimed at revolutionizing the field of information retrieval through advanced natural language processing (NLP) techniques. The project seeks to develop an intelligent system capable of automatically extracting accurate and relevant answers from diverse document sources, thereby streamlining the process of knowledge acquisition and decision-making.

Project Overview

The Document Based Q&A System project represents a significant advancement in the realm of document-centric information retrieval. By harnessing the power of machine learning, semantic analysis, and document parsing algorithms, the system endeavors to provide users with rapid and precise answers to their queries, regardless of the complexity or format of the underlying documents. Through an intuitive user interface, the system empowers users to access actionable insights and glean valuable knowledge from vast repositories of textual data.

Objectives

The primary objectives of the Document Based Q&A System project include:

- Developing a scalable and efficient platform for document-based question answering.
- Implementing state-of-the-art NLP algorithms for text analysis, document parsing, and answer extraction.
- Enhancing the system's performance through continuous optimization and refinement of machine learning models.
- Providing users with a seamless and intuitive interface for querying documents and accessing extracted answers.

Relevance

In today's information-driven world, the ability to quickly access relevant insights from vast amounts of textual data is paramount. The Document Based Q&A System project addresses this need by offering a sophisticated solution that leverages cutting-edge technologies to deliver accurate and timely answers to user queries. From academic research to business intelligence applications, the system has the potential to revolutionize the way organizations and individuals interact with textual information, thereby enhancing productivity and decision-making capabilities across various domains.

Problem Formulation

The Document Based Q&A System project addresses several challenges inherent in traditional information retrieval methodologies, including:

Inefficient manual search processes: Conventional keyword-based search methods often require users to sift through large volumes of documents manually, resulting in time-consuming and labor-intensive information retrieval tasks.

Limited scalability and accuracy: Keyword-based search approaches may struggle to provide accurate and comprehensive answers, particularly when dealing with complex or ambiguous queries.

Interpretation of complex textual data: Extracting meaningful insights from unstructured textual data poses challenges due to the inherent variability and ambiguity of language.

Real-time response capabilities: Traditional search systems may lack the agility to handle dynamic user queries and provide timely responses, especially in scenarios where rapid decision-making is crucial.

Motivation

The Document Based Q&A System project is motivated by the following factors:

Increasing demand for intelligent information retrieval solutions: With the exponential growth of digital data, there is a pressing need for advanced tools and technologies that can facilitate efficient access to relevant information.

Advancements in NLP and machine learning: Recent breakthroughs in NLP and machine learning algorithms have paved the way for the development of sophisticated question answering systems capable of understanding and interpreting textual data with human-like accuracy.

Potential for transformative impact: By enabling users to extract actionable insights from documents rapidly and accurately, the Document Based Q&A System has the potential to drive significant improvements in productivity, decision-making, and knowledge discovery across various domains.

Scope of the Project

The scope of the Document Based Q&A System project encompasses the following areas:

Development of a prototype system capable of extracting answers from diverse document sources, including text documents, PDFs, and web pages.

Implementation of a scalable architecture to accommodate large volumes of documents and user queries, ensuring optimal performance and responsiveness.

Integration of advanced NLP algorithms for text analysis, document indexing, and answer extraction, leveraging cutting-edge techniques such as deep learning and semantic analysis.

Evaluation of the system's performance through rigorous testing and validation procedures, including benchmarking against existing QA systems and user feedback analysis.

Provision of comprehensive documentation and user guides to facilitate the effective utilization of the system by end-users and developers alike.

System Architecture:

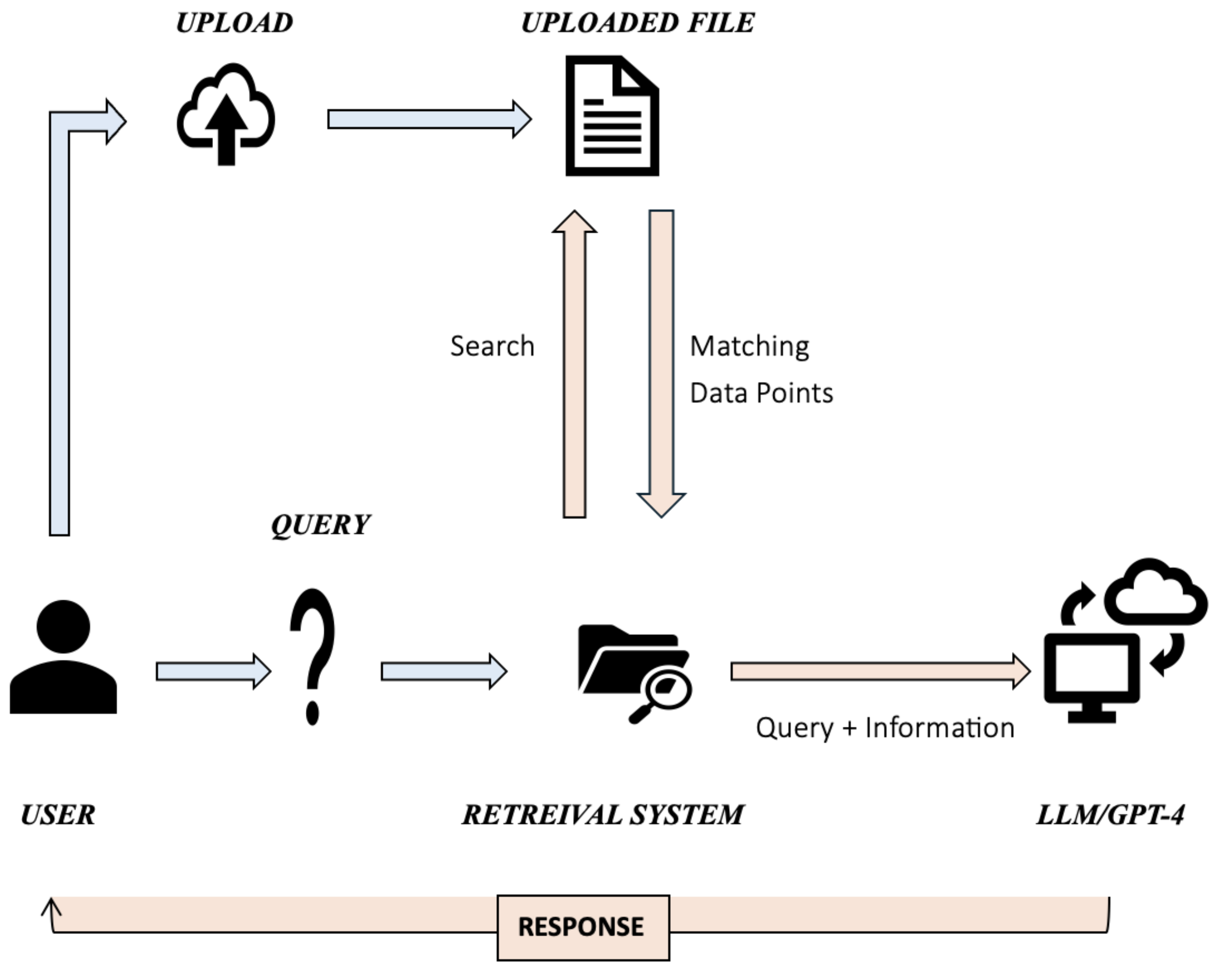

Fig 1: System Architect of the Application

Data Modelling:

The approach involves combining Pinecone, a robust vector database engine, with LLM, known for its natural language understanding capabilities. By integrating these technologies, the aim is to improve the search functionality of this application, enabling it to understand and answer diverse queries accurately. This not only enhances the versatility of the platform but also makes it intelligent in interpreting and responding to user questions.

The deployment of the Document Based Q&A System is performed using the Streamlit Python framework, known for its capability to build attractive and interactive user interfaces. By leveraging Streamlit, the system ensures that users have a seamless and engaging experience while interacting with the platform. The framework's versatility allows for the creation of visually appealing dashboards and intuitive controls, enhancing the overall usability and accessibility of the system.

Implementation:

Implementation Overview

The implementation stage of the Document Based Q&A System project involves the integration of various technologies and APIs to create a robust and efficient system for document-based question answering. This section provides an overview of the key components and processes involved in the implementation of the system.

Technologies and APIs Used

OpenAI API: Utilized for natural language processing (NLP) tasks, including text understanding and answer generation.

Pinecone Vector Database API: Integrated for storing and indexing document vectors to facilitate efficient retrieval of information.

Streamlit Python Framework: Employed for building interactive and visually appealing user interfaces for the system.

Implementation Process

The implementation process of the Document Based Q&A System can be divided into the following stages:

Loading and Processing User's Uploaded Files

The implementation starts with loading the user's uploaded files, which may include text documents, PDFs, or PPTx. These files are processed to extract relevant textual content and convert it into a format suitable for storage and analysis.

Integration of OpenAI API and Pinecone Vector Database API

The system integrates the OpenAI API and Pinecone Vector Database API to perform various actions. These APIs are employed utilizing Streamlit's st.secret function which provides a dictionary-like interface to access secrets stored in a secrets.toml file.

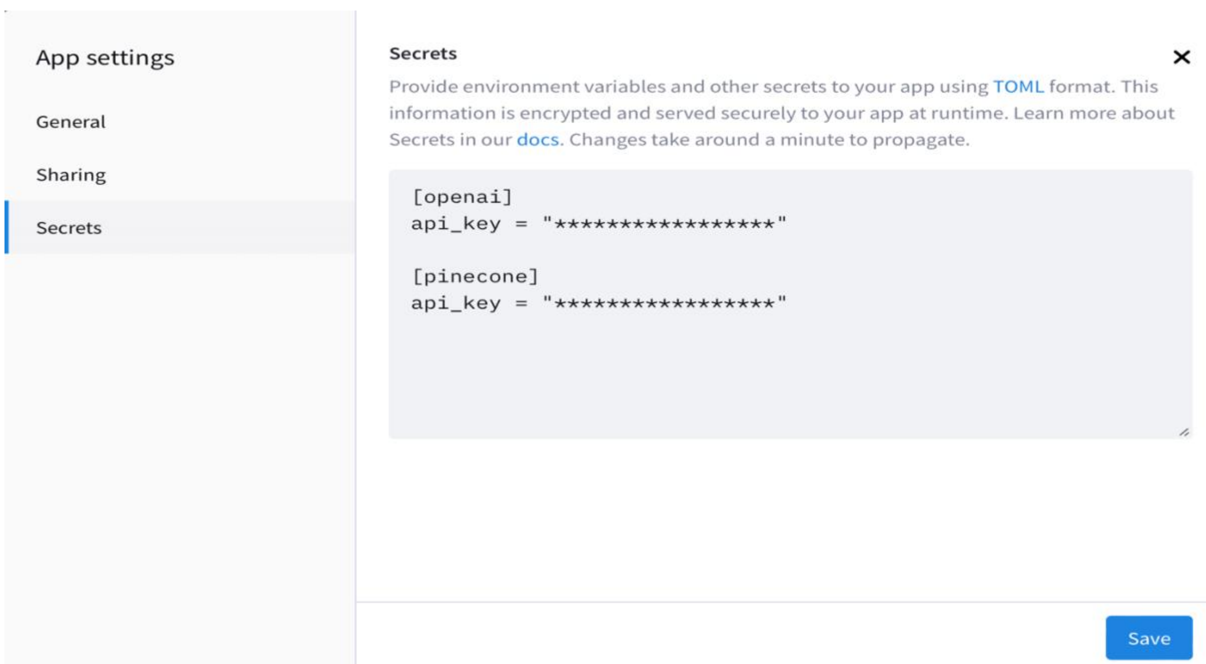

Fig 2.1: App Settings in Streamlit.

```python
# Set up the environment

# Load secret keys
secrets = st.secrets
openai_api_key = secrets["openai"]["api_key"] # Access OpenAI API key
os.environ["OPENAI_API_KEY"] = openai_api_key

pinecone_api_key = secrets["pinecone"]["api_key"] # Access Pinecone API key
os.environ["PINECONE_API_KEY"] = pinecone_api_key
```

Fig 2.2: Environment setup in python.

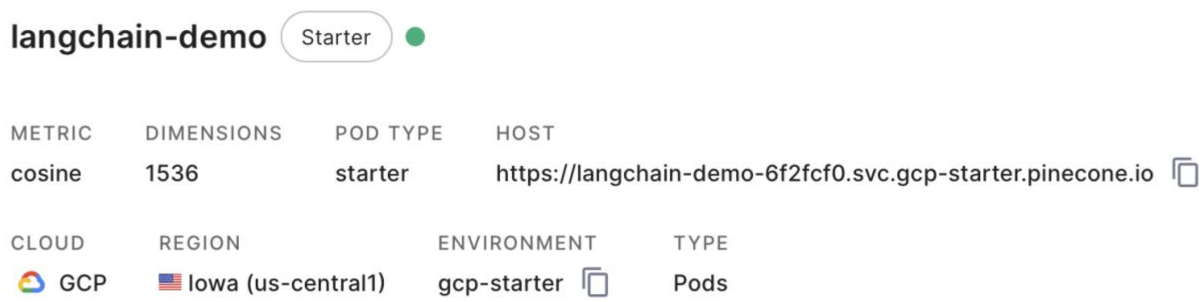

Fig 2.3: Pinecone Database.

Storing document vectors in the Pinecone database for efficient retrieval.

Utilizing advanced NLP models, such as "GPT4" and LangChain's retrieval chain, to understand user questions and retrieve relevant information from the uploaded files.

Crafting Effective Prompts

A crucial aspect of the system's behavior is crafting effective prompts for guiding the model's responses. The prompts are structured and presented in a way that aligns with the desired outcomes, ensuring that the model's responses are concise, understandable, and on point.

Setting Up GitHub Repository and Streamlit Deployment

Once the Python code file is ready with the necessary dependencies, a GitHub repository is created to manage the project files. The repository includes all the required files for the application, including code, data, and documentation. Streamlit is then used to deploy the system, with the GitHub repository connected to facilitate seamless deployment and version control. Streamlit's logging capabilities allow for real-time monitoring and analysis of the model's responses.

Monitoring and Analysis

Throughout the implementation process, continuous monitoring and analysis of the system's performance are essential. Streamlit provides tools for analyzing logs and tracking model responses, allowing developers to identify and address any issues or improvements needed to enhance the system's functionality and accuracy.

Results

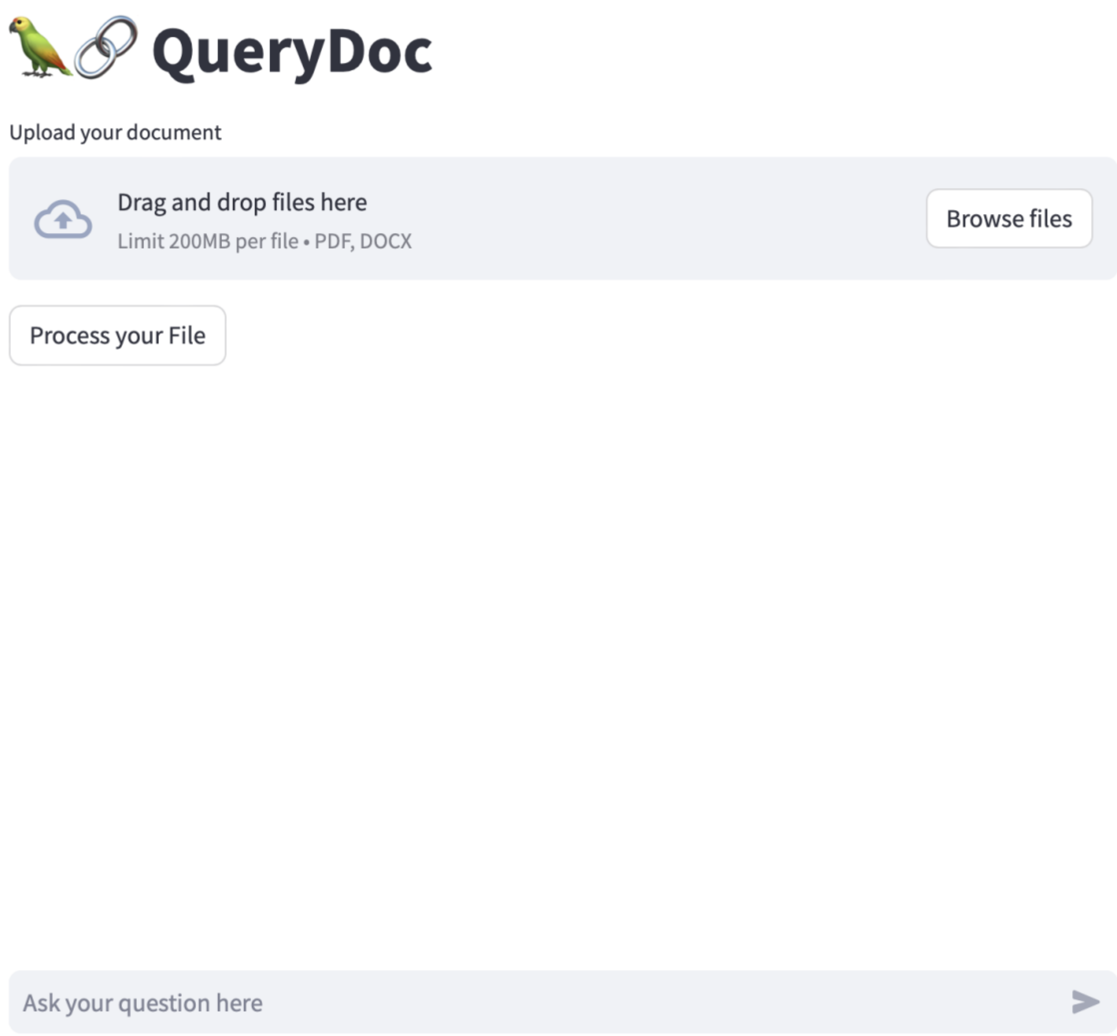

Fig 3.1: Homepage

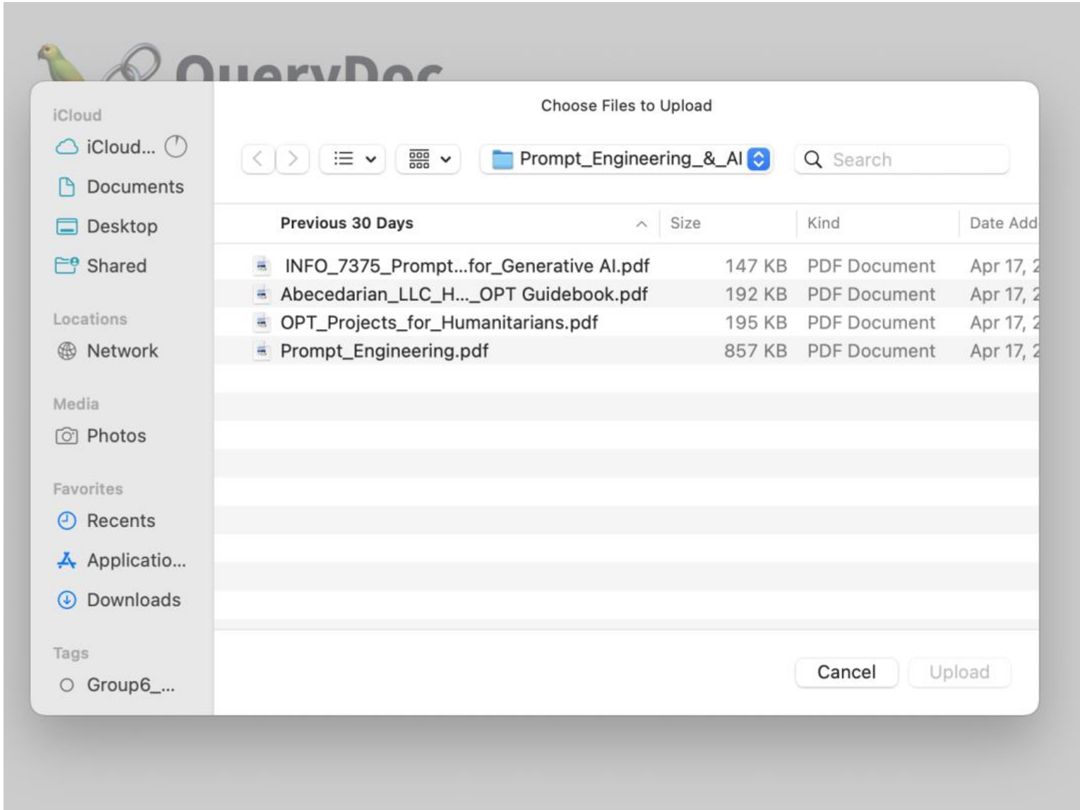

Fig 3.2: Browsing Device.

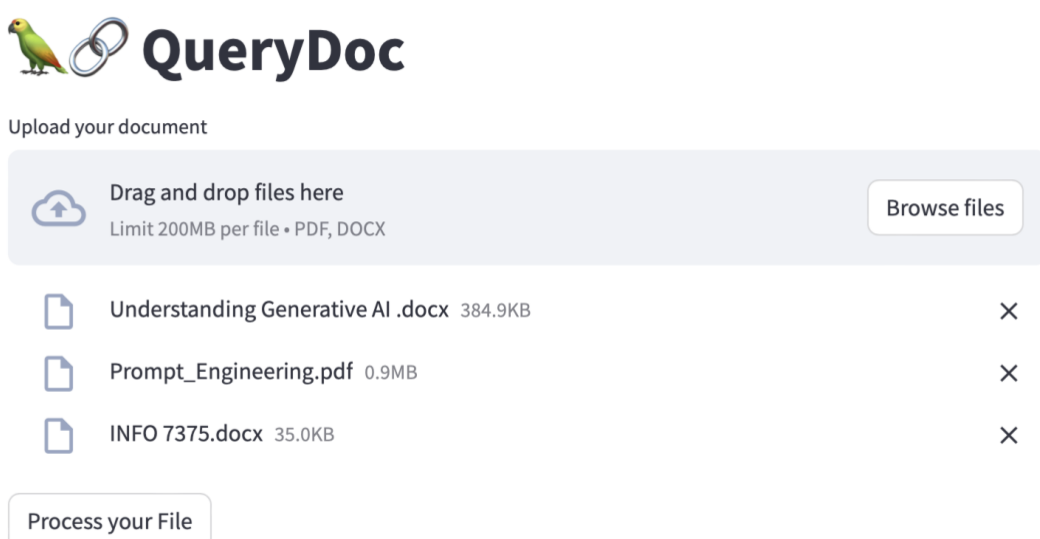

Fig 3.4: Uploaded files

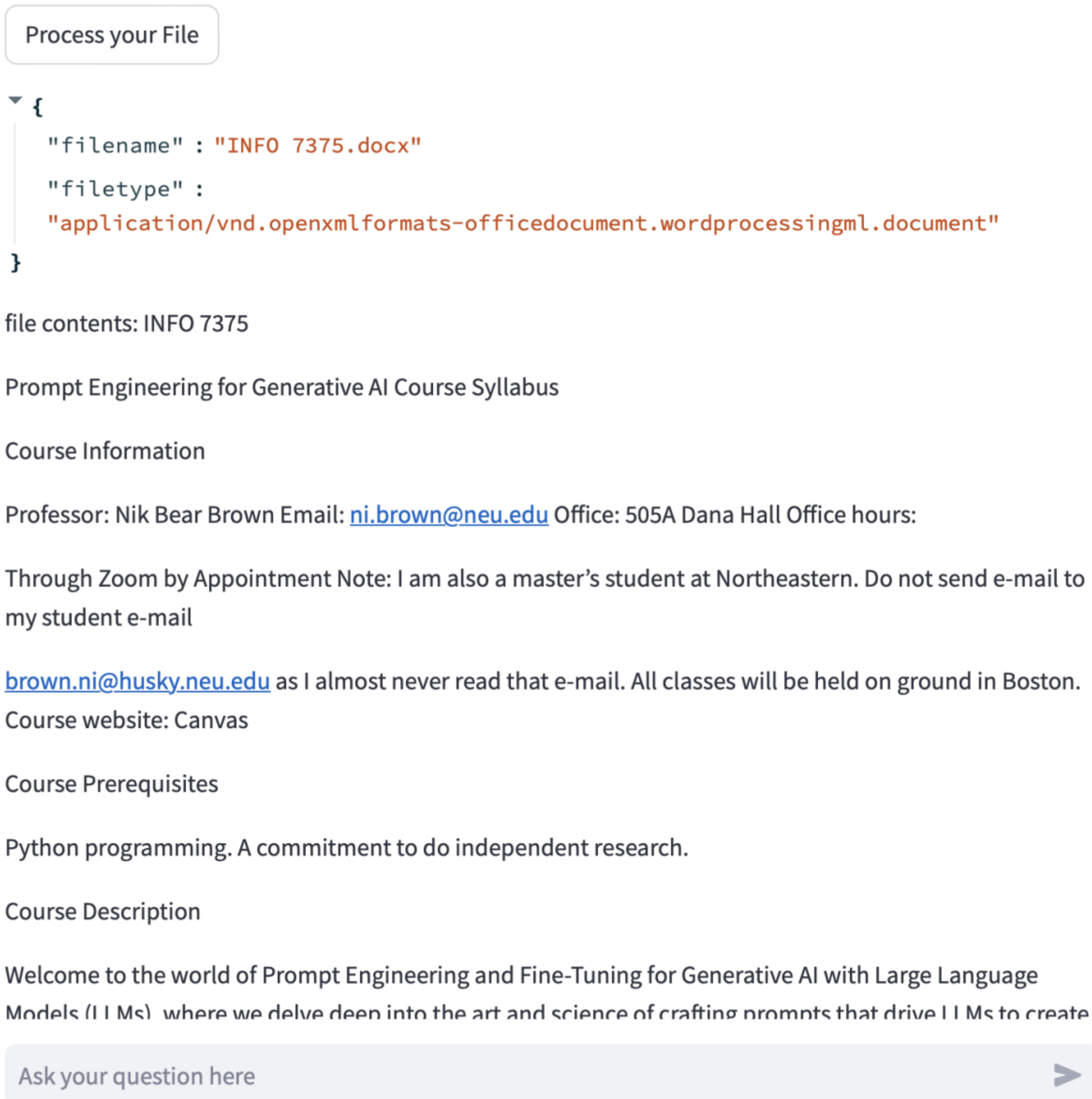

Fig 3.3: Processed file results

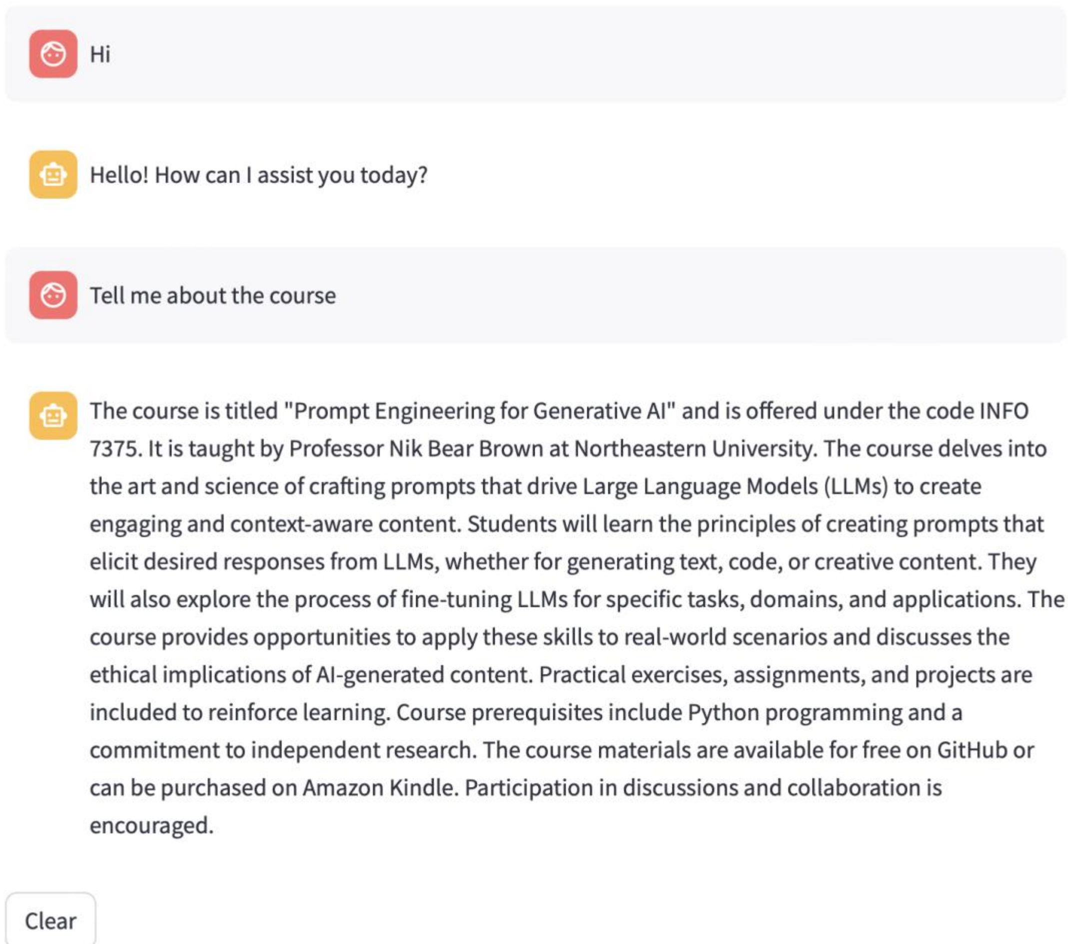

Fig 3.5: Chat Interface 1.

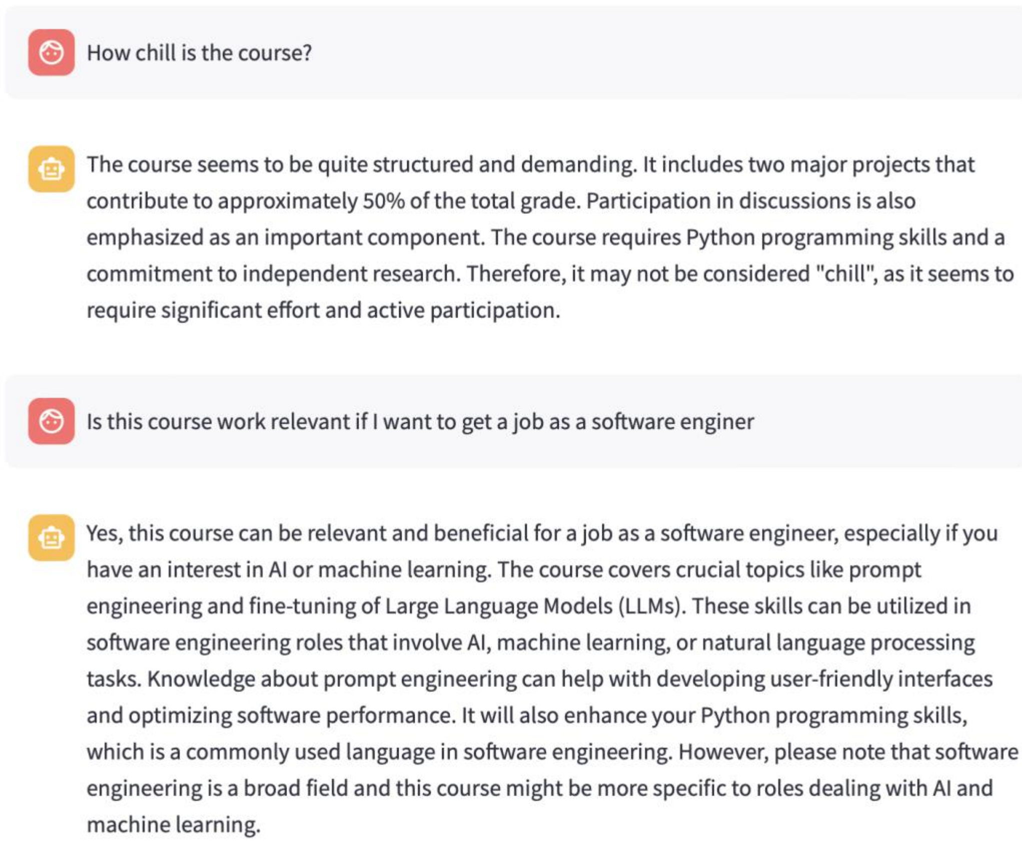

Fig 3.6: Chat Interface 2.

Conclusion:

In conclusion, the project represents a significant step forward in improving the educational experience through technology. By leveraging semantic search and Large Language Models, we aim to empower students with a dynamic platform that simplifies information retrieval and fosters deeper understanding. The commitment to delivering a user-friendly, effective solution underscores the dedication to advancing education in the digital age.

Working of Algorithms (Appendix):

```
# Import necessary libraries and modules

import os

from openai import OpenAI

from langchain_openai import ChatOpenAI

import pinecone

from pinecone import Pinecone, PodSpec

from langchain.embeddings.openai import OpenAIEmbeddings

from langchain_pinecone import PineconeVectorStore

import streamlit as st

from langchain.text_splitter import Language, RecursiveCharacterTextSplitter

from langchain_core.prompts import ChatPromptTemplate

from langchain.chains import ConversationalRetrievalChain, RetrievalQA

from langchain_community.document_loaders import PyPDFLoader,Docx2txtLoader,
UnstructuredPowerPointLoader

import os.path

import pathlib

import tempfile



## Set up the environment

# Load secret keys



secrets = st.secrets  # Accessing secrets (API keys) stored securely
```

```
openai_api_key = secrets["openai"]["api_key"]  # Accessing OpenAI API key from secrets
os.environ["OPENAI_API_KEY"] = openai_api_key  # Setting environment variable for OpenAI API key

pinecone_api_key = secrets["pinecone"]["api_key"]  # Accessing Pinecone API key from secrets
os.environ["PINECONE_API_KEY"] = pinecone_api_key  # Setting environment variable for Pinecone API key

# Initializing Pinecone with API key
pc = Pinecone(pinecone_api_key=pinecone_api_key)

# Initializing OpenAI embeddings model with API key
embeddings_model = OpenAIEmbeddings(openai_api_key=openai_api_key)

# Name for the index
index_name = "langchain-demo"

# Embed the documents
def vector_db():
  for file in uploaded_files:
    file.seek(0)  # Reset file pointer to beginning

    # Display file details
    file_details = {"filename": file.name, "filetype": file.type}
    st.write(file_details)

    # Create temporary directory and save file there
    temp_dir = tempfile.mkdtemp()
```

```
    path = os.path.join(temp_dir, file.name)
    with open(path, "wb") as f:
        f.write(file.getvalue())

    # Determine file extension
    file_extension = file.name.split(".")[-1].lower()

    # Load document based on its extension
    if file_extension == "pdf":
        loader = PyPDFLoader(path)
    elif file_extension == "docx":
        loader = Docx2txtLoader(path)

    # Load documents and split text
    docs = loader.load()
    for doc in docs:
        text = doc.page_content
        st.write("file contents:", text)

    # Split documents into chunks and create indexes
    text_splitter = RecursiveCharacterTextSplitter(chunk_size=1000, chunk_overlap=50)
    split_data = text_splitter.split_documents(docs)
    indexes = PineconeVectorStore.from_documents(split_data, embeddings_model,
index_name=index_name)

  return indexes
```

```
def get_retrieval_chain(result):
  # Define system prompt for chat interaction
  system_prompt = (
    """
    You are a helpful assistant who helps users answer their question based on the documents they upload.
    Answer the question in your own words from the context given to you.
    If questions are asked where there is no relevant context available, please answer from what you know and say please upload your documents first for better response.

    Context: {context}
    """
  )
  prompt = ChatPromptTemplate.from_messages(
    [("system", system_prompt), ("human", "{question}")]
  )

  # Assigning the OPENAI model and Retrieval chain
  model_name = "gpt-4"
  llm = ChatOpenAI(model_name=model_name)

  # Define the Retrieval chain
  retrieval_chain = RetrievalQA.from_chain_type(
    llm, retriever=result.as_retriever(), chain_type_kwargs={"prompt": prompt}
  )
  st.session_state.chat_active = True
```

```
    return retrieval_chain


# Define Response Function
def get_answer(query):
    # Get retrieval chain
    retrieval_chain = get_retrieval_chain(st.session_state.vector_store)
    # Get answer from retrieval chain
    answer = retrieval_chain({"query": query})
    return answer


# Title for the web app
st.title("🦜📎 QueryDoc")
# File uploader for user to upload a document
uploaded_files = st.file_uploader(
    "Upload your document", type=["pdf", "docx"], accept_multiple_files=True
)
# Button to process uploaded file
if st.button("Process your File"):
    if uploaded_files is None:
        st.write("Please upload a file first.")
    elif uploaded_files is not None:
        if "vector_store" not in st.session_state:
            # Initialize vector store
            st.session_state.vector_store = vector_db()
```

```
# Initialize chat history
if "messages" not in st.session_state:
    st.session_state.messages = []

# Display chat messages from history on app rerun
for message in st.session_state.messages:
    with st.chat_message(message["role"]):
        st.markdown(message["content"])

# React to user input
if query := st.chat_input("Ask your question here"):
    # Display user message in chat message container
    with st.chat_message("user"):
        st.markdown(query)
    # Add user message to chat history
    st.session_state.messages.append({"role": "user", "content": query})

    # Get answer from retrieval chain
    answer = get_answer(query)
    result = answer["result"]

    # Display assistant response in chat message container
    with st.chat_message("assistant"):
        st.markdown(result)
        # Add assistant response to chat history
```

```
    st.session_state.messages.append({"role": "assistant", "content": result})

# Button to clear chat messages
def clear_messages():
    st.session_state.messages = []
st.button("Clear", on_click=clear_messages)
```