Conversational Interface for Multi-Source Information Retrieval

Abhiyan Sainju
School of Engineering and Applied Science
The George Washington University
Washington, D.C
abhiyan.sainju@gwu.edu

Balaji Senthilkumar
School of Engineering and Applied Science
The George Washington University
Washington, D.C
balaji.senthilkumar@gwu.edu

***Abstract*— This project presents a conversational interface that enables efficient information retrieval from documents, PDFs, YouTube videos, and audio recordings. It leverages natural language processing (NLP) and vector databases to extract text, create vector stores, and allow users to query the content through a chat-like experience. The application also integrates an offline language model alongside the existing online model (GPT-3.5-turbo). This addition will provide users with an alternative for local processing or address privacy concerns. Additionally, our project also incorporates a feature to process YouTube video transcripts which enables users to search video content seamlessly. Furthermore, our project also includes an innovative feature that allows users to record audio and interact with it in a way that is similar to other supported sources. The primary objective of this project is to streamline the information access process by providing a concise and relevant responses to user queries, saving time and effort compared to manual searching across multiple sources. Our project aims to assist a range of industries, including education, research, journalism, and others, where obtaining material from several sources, such as pdf documents and audio recordings, can be time-consuming.**

Keywords— conversational, natural language processing, offline language model, local processing, queries, audio

I. INTRODUCTION

People frequently become overwhelmed by the sheer amount of data that is available in different formats, including documents, PDFs, and multimedia content like audio and videos, in today's information-rich society. While this amount of data might be beneficial, quickly identifying and getting precise, relevant information is a big challenge. Manually searching through these many sources and filtering vast amounts of data takes time and is inefficient, resulting in frustration and lost productivity. While text-based data can be effectively processed and retrieved with existing approaches, audio content is frequently underutilized or inaccessible due to a lack of efficient interaction and retrieval tools. This project intends to address these issues by providing a streamlined and straightforward solution for retrieving information from numerous sources, including audio recordings, thereby reducing time and increasing user productivity in a variety of sectors such as education, journalism, and research. Our project seeks to overcome these issues by developing a user-friendly conversational interface that streamlines the process of retrieving information from various sources. By integrating cutting-edge technologies such as natural language processing (NLP), vector databases, and large language models (LLM), our application allows users to access and interact with information in a more efficient and intuitive way.

The application's core feature relies around extracting text from submitted PDF files, YouTube videos, and audio files, building vector storage, and allowing users to query the content via a chat-like interface. Users can ask questions about the documents, audio files and the videos. The system will analyze the query submitted by the user and gather relevant information from the vector store, delivering brief and contextual responses customized to the queries of the user. To increase versatility and capacities, our project integrates an offline language model (LLM) alongside OpenAI's current online model (GPT-3.5-turbo). The offline approach offers an option for local processing, addressing potential privacy concerns or instances with limited internet access.

Recognizing the expanding importance of multimedia content, our project also includes YouTube video transcript processing. This feature transforms how people consume and engage with video information, opening up new opportunities for research, education, and knowledge acquisition. Our project's primary objective is to streamline and improve the efficiency and usability of the information retrieval process. The goal is to save time and deliver only the necessary information by allowing users to communicate with many sources via a conversational interface, removing the need for manual searching and filtering via multiple sources. This approach will increase the productivity while also providing users with a compelling and intuitive experience, allowing them to fully leverage massive knowledge resources.

The paper is organized as follows. Section I explains the problem the project is trying to solve and the chatbot, which uses Generative AI and Large Language Models to solve this problem. Section II discusses corresponding research on Generative AI, document chatbots, and the framework involved. Section III describes the proposed technique and implementation details. Section IV summarizes the implementation and experimental results. In Section V, we provide some source code snippets to end our paper.

II. Related Works

In today's world, information retrieval is crucial for many practical purposes. Information retrieval aims to locate and recover relevant information for a user's inquiry [1]. To organize results based on relevance to the user's query, many relevant records may be grouped together. This method prioritizes the most relevant information, improving the effectiveness of information retrieval.

There are several approaches for information retrieval, ranging from traditional methods based on term matching between queries and documents to modern strategies that use deep learning algorithms to improve results in Natural Language Processing (NLP) activities [2]. These methods include Convolutional Neural Networks (CNNs) [3], Recurrent Neural Networks (RNNs) [4], Long Short-Term Memory (LSTM) [5], Sequence to Sequence mechanisms using Attention [6], transfer learning, and pre-training strategies [7].

Several projects have also explored similar concepts of conversational interfaces and information retrieval from diverse sources. One such notable study was "Automating Information Retrieval from Faculty Guidelines" [8]. The authors created an automated information retrieval system using Pinecone vector storage and OpenAI's ChatGPT. The research suggests a generative AI chatbot, a conversational machine powered by AI that responds in natural language. Unlike rule-based chatbots, generative AI chatbots use deep learning architectures like transformers to produce contextually relevant and coherent responses. The study's findings show that chatbots that can answer context-related inquiries within documents are possible [8].

Deepgram's Chrome addon, ChatTube [9], allows users to engage with YouTube videos. The company does not disclose the system's operating specifics. However, this Chrome extension is not accessible to anyone. A user must pay a $9.99 monthly subscription charge to gain access to its services.

While these projects have made significant contributions, they often focus on specific domains or types of content. The proposed project aims to provide a more comprehensive solution by integrating various sources, including PDFs, documents, and multimedia content like YouTube videos, and audio recordings. Additionally, the incorporation of both online and offline language models offers added flexibility. The offline language model also addresses potential privacy concerns.

III. Solutions/Analyses

Our project uses a multifaceted strategy to achieve its goal of offering an effective conversational interface for multi-source information retrieval. Figure 1 shows the overall overview of our project. The user can choose one of the following options in our application: PDF Chat, Audio Chat, Lecture Chat, YouTube Chat or, Offline Chat. Our application will then convert the PDF or YouTube video transcripts, or the audio recordings into their respective embeddings and store then into a vector database. Our project uses ***FAISS*** as the vector storage for the embeddings for PDF chat, Lecture chat, Audio Chat and YouTube Chat. For offline chat, our system uses ***Chroma*** as the vector storage. The process of converting files into embeddings and storing them into vector storage is explained in this section below.

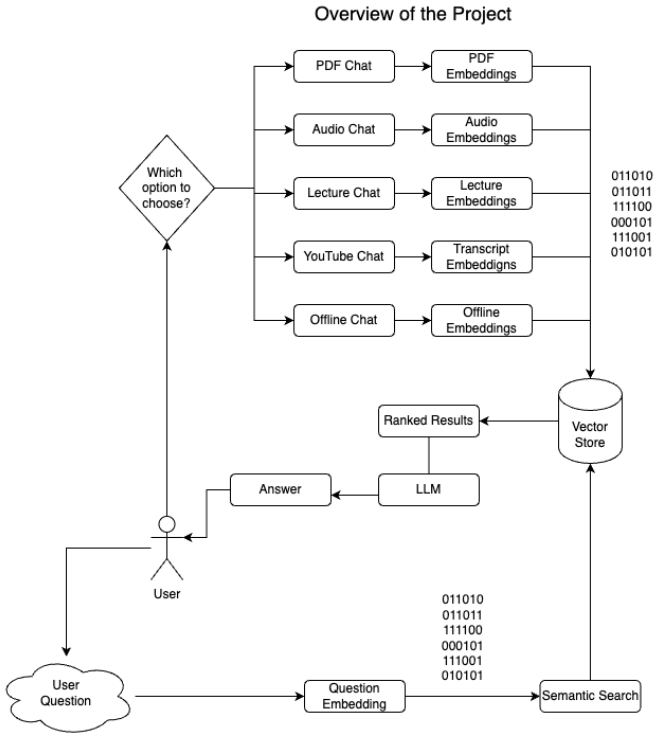

Figure 1: Overview of the Project

Once the various files are converted into their respective embeddings and stored into the vector database, the user can ask questions into our system through a modern, easy to use chat interface. Our system then converts the queries of the user into embeddings and searches the vector database for matching embeddings. The highest ranked result is sent into the LLM (*OpenAI's GPT 3.5 Turbo* or *Mistral* for offline processing) to convert the matched embeddings into a text answer. This text answer to the user query is then displayed in out chat interface.

The core components of out project along with their implementation details are listed below:

A. *Text Extraction and Division into groups:*

- The application extracts text from submitted PDF files using the ***PyPDF2*** library. This library includes capabilities for reading and parsing PDF documents,

allowing the application to access the textual content of each page.

- Once the entire text has been extracted, the application uses LangChain's ***CharacterTextSplitter*** to separate it into smaller, more digestible chunks or groups. This division of the whole text into smaller groups is required for effective vector embedding and storage in the vector database.
- The ***CharacterTextSplitter*** function allows changing the size of the chunks, the overlap between them, and the separator used to split the text. Our project uses a chunk size of 1000 words and an overlap of 200 words between the chunks. The overlap between word chunks helps to keep context intact between various chunks.

B. Vector Store Creation

- The chunked text segments are turned into vector representations using language model-generated embeddings. The online model operates on OpenAI's *OpenAIEmbeddings*, which is accessed via the *langchain_openai* library.
- For offline model integration, the project makes use of *OllamaEmbeddings* This model may be imported and used to generate embeddings without relying on cloud-based services, which addresses any privacy concerns or scenarios involving limited internet connectivity.
- The resulting embeddings are then saved in a high-performance vector database via the ***FAISS*** library from the *langchain_community.vectorstores* module. ***FAISS*** (Facebook AI Similarity Search) is a library that does efficient similarity search and grouping of dense vectors, making it ideal for storing and retrieving embedded data.
- The ***FAISS*** vector store is loaded with text chunks and their accompanying embeddings, allowing for rapid retrieval and similarity comparisons during conversational exchanges.

C. Conversational Interface

- Our application uses ***ConversationalRetrievalChain*** from *LangChain* to maintain the conversation context between the user and our system interface.
- To support contextual and conversational interactions, this component combines a language model (either *GPT-3.5-turbo* or *Mistral*, depending on the user's choice), a retriever (the *FAISS* vector store or, *Chroma* for offline model), and a memory component (***ConversationBufferMemory***).
- The *ConversationalRetrievalChain* processes user queries, retrieves relevant information from the vector store based on vector similarity, and generates responses using the language model while taking into account the conversation history.
- The memory component keeps track of the conversation's history, allowing the language model to respond contextually depending on past chat interaction history.

D. YouTube Video Transcript Processing:

- In order to be able to use the transcriptions/subtitles that are embedded or auto-generated in a youtube video, we use an api called 'youtube-transcript-api' that requires a video id as an argument to fetch the corresponding video's data.
- To get this video id, we require the user to provide us the link to the youtube video and then use pattern matching by regular expressions to extract the video id from the link.

E. Offline Language Model Integration

- The offline LLM used in our implementation is *Mistral AI's 7B model* that is accessed through Ollama's framework. *Ollama* is a lightweight, extensible framework for building and running language models on your local machine.
- We found the Mistral 7B model to be the best fit for our requirements. It is available under the *Apache 2.0* license, making it freely accessible. It can run 100% offline and locally, keeping your data private from the internet. The size of the model is around 4GB but can vary depending on the performance requirements and available hardware.
- The *Mistral* model allows a larger token input of up to 8K tokens and excels in natural language tasks in comparison to other offline models. It can also be used on the GPU to improve inference speed. In our implementation, the model runs solely on the CPU.

F. Audio Processing

- We use Streamlit libraries to capture audio data from the user. The audio is then converted into byte streams and stored as a temporary file.
- We utilize OpenAI's '*whisper-1*' model to aid in converting the captured audio data into usable transcripts for further processing. The converted data can be used to generate pdfs that the user can download and refer to or the user can directly query the data.

The application's user interface is created with *Streamlit*, a Python-based framework for developing interactive web applications. *Streamlit* makes it easier to create a user-friendly interface, allowing users to submit files, start conversations, maintain conversation context, and view responses in a fluid manner.

Diffuculties Faced:

Some of the difficulties faced during the implementation of this project was with: maintaining session states for user inputs and other variables and preventing conflicts with stored context data, dealing with deprecated libraries and code, improving the inference speed.

Everytime an action is executed in our application, it essentially re-executes the application causing a variety of problems such as losing uploaded/cached user data, loss of contextualized data for LLM models, and longer execution and processing time.

Some of the dependencies or imported libraries upon installation seemed to have a lot of deprecated code. Some of the default versions of packages installed through pip were old and not the most up to date, resulting in a lot of unresolved bugs. This was eventually resolved through research and updating to the most up to date packages/libraries available through github or other repositories. In one such instance, when trying to create a pdf for the user to download, the pdf turned out to be blank every time. The cause of this was as aforementioned earlier.

We attempted to improve the inference speed of the offline model by enabling gpu acceleration but there is no proper support for it with the libraries and frameworks we were using for the project. We also attempted to improve the overall processing speed for the application through caching some frequently used data but while some attempts were successful others resulted in unresolvable conflicts and errors.

IV. Summary And Future Works

Our project "*Conversational Interface for Multi-Source Information Retrieval*" provides a novel and comprehensive solution for simplifying information retrieval from many sources, such as documents, PDFs, YouTube videos, and audio recordings. By integrating cutting-edge technology such as natural language processing, vector databases, and both online and offline language models, our project provides users with a robust and user-friendly conversational interface for searching and retrieving information.

The goal of our project was to create a system that allows users to get information from the vast sea of available information sources in an efficient and speedy manner. Our project's most significant accomplishments and contributions include:

1) Efficient Information Retrieval: Our application allows users to access, retrieve data, and query information from a variety of sources using a simple conversational interface, removing the need for manual searching and filtering.

2) Multi-Source Integration: Our project enables information retrieval from documents, PDFs, YouTube videos, and audio recordings, offering a comprehensive solution for a wide range of content types.

3) Online and Offline Language Model Support: The combination of both online (GPT-3.5-turbo) and offline (*Mistral*) language models provides flexibility while also addressing any privacy concerns or circumstances with restricted internet connectivity.

4) Contextual and Conversational Interactions: Our application uses LangChain's ConversationalRetrievalChain and memory components which allows contextual and conversational interactions, resulting in more natural and meaningful responses.

5) User-Friendly Interface: Our application uses a Streamlit-based user interface to provide a smooth and straightforward experience for users, allowing for simple file uploads, conversation start, and response display.

6) Time Saving and Productivity Enhancements: By automating the information retrieval process and giving brief and appropriate responses, the program saves time and effort, resulting in greater productivity.

While our project provides a potent solution to providing an easy to use, multi-source information conversational system, there are a few potential areas for future additions and enhancements.

1) Expanded Source Support: Adding support for new file formats and information sources, such as websites, databases, or knowledge bases, would expand the application's capabilities.

2) Multilingual Audio Support: Extending the audio processing and natural language processing skills to handle multiple languages could broaden the application’s reach and appeal to a wider range of users.

3) Personalized Query Sugesstions and Result Ranking: Using machine learning algorithms to assess user behavior and preferences could lead to tailored query suggestions and result ranking, hence improving the user experience.

4) Implementing GPU Accceleration: Implementing GPU acceleration for offline models would result in much faster inference and processing time. Allowing users to be able to integrate any offline model with our application and running it with GPU acceleration would be a great user experience.

Overall, our project "*Conversational Interface for Multi-Source Information Retrieval*" demonstrates the power of merging cutting-edge technology to improve user experience and information accessibility. This project has the potential to transform the way people and organizations engage with and exploit the immense knowledge available from multiple sources, hence promoting innovation and productivity in a variety of sectors.

V. Source Code

The following code snippet is the main function for our Streamlit python project. For specific module code, please refer the attached source code zip file.

```
   # main function
   def main():
       # loading config files (contains API key) to
the main file.
       config = load_config()
       client = baseOpenAI()
       # using streamlit to set the application page
title
```

```
    st.set_page_config(page_title="Ask questions
to PDFs", page_icon="")
    st.write(css, unsafe_allow_html=True)

    # check if the current session contains pre-
existing conversation
    if "conversation" not in st.session_state:
        st.session_state.conversation = None
    # check if the current session contains pre-
existing chat history
    if "chat_history" not in st.session_state:
        st.session_state.chat_history = None
    if 'flag' not in st.session_state:
        st.session_state.flag=0

    st.header("Ask questions to PDFs")
    st.write(header_template,
unsafe_allow_html=True)  # Render the header
template
    audio_bytes =
audio_recorder(energy_threshold=(5.0))
    user_question = st.text_input("Enter your
questions to the document(s) here: ")
    if st.button("reset audio"):
        st.session_state.flag=0
        audio_bytes=None
        user_question=None
    # is user asks a question, generate response
    if audio_bytes and st.session_state.flag!=1:
        st.audio(audio_bytes, format="audio/wav")
        txt=get_handle_audio(audio_bytes,client)
        st.session_state.flag=1
        if st.session_state.conversation:
            response =
st.session_state.conversation({'question': txt})
            st.session_state.chat_history =
response['chat_history']
    elif user_question:
        if st.session_state.conversation:
            response =
st.session_state.conversation({'question':
user_question})
            st.session_state.chat_history =
response['chat_history']


    # session for the chat between the user and
the PDF(s)
    st.write('<div class="chat-messages">',
unsafe_allow_html=True)
    if st.session_state.chat_history:
        # check if it is a user question or a
response to the question
        for i, message in
enumerate(st.session_state.chat_history):
            if i % 2 == 0:

st.write(bot_template.replace("{{MSG}}",
message.content), unsafe_allow_html=True)
            else:

st.write(user_template.replace("{{MSG}}",
message.content), unsafe_allow_html=True)
    st.write('</div>', unsafe_allow_html=True)


    # sidebar to upload PDF(s)
    with st.sidebar:
        st.write(file_upload_template,
unsafe_allow_html=True)  # Render the file upload
instructions
        pdf_docs = st.file_uploader("",
accept_multiple_files=True, key="file_uploader")
        if st.button("Process",
key="process_button"):
            with st.spinner("Processing your
document"):
                raw_text = get_pdf_text(pdf_docs)
# use get_pdf_text function to extract text
information from the pdf
                text_group =
get_text_chunks(raw_text) # use get_text_chunks
function to group the raw text into multiple groups
                vectorstore =
get_vectorstore(text_group, config) # use
get_vectorstore function to store the vector values
for these text groups

                if vectorstore is not None:
                    st.session_state.conversation
= get_conversation_chain(vectorstore, config)
                else:
                    st.warning("No text found in
the uploaded PDF documents.")

if __name__ == '__main__':
    main()
```

REFERENCES

[1] I. Thalib, Widyawan, and I. Soesanti, "A Review on Question Analysis, Document Retrieval and Answer Extraction Method in Question Answering System," in *Proceeding - ICoSTA 2020: 2020 International Conference on Smart Technology and Applications: Empowering Industrial IoT by Implementing Green Technology for Sustainable Development*, Institute of Electrical and Electronics Engineers Inc., Feb. 2020. doi: 10.1109/ICoSTA48221.2020.1570614175.

[2] K. A. Hambarde and H. Proenca, "Information Retrieval: Recent Advances and Beyond," *IEEE Access*, vol. 11, pp. 76581–76604, Jan. 2023, doi: 10.1109/ACCESS.2023.3295776.

[3] "What are Convolutional Neural Networks? | IBM." Accessed: Apr. 12, 2024. [Online]. Available: https://www.ibm.com/topics/convolutional-neural-networks

[4] P. Muangkammuen, N. Intiruk, and K. R. Saikaew, "Automated Thai-FAQ chatbot using RNN-LSTM," *2018 22nd International Computer Science and Engineering Conference, ICSEC 2018*, Jul. 2018, doi: 10.1109/ICSEC.2018.8712781.

[5] E. Wang, W. L. Putera, H. Lucky, and A. Chowanda, "Chatbot Application to Automate Services in FnB Business Using Seq2Seq LSTM," *Proceedings - 4th International Conference on Informatics, Multimedia, Cyber and Information System, ICIMCIS 2022*, pp. 413–417, 2022, doi: 10.1109/ICIMCIS56303.2022.10017854.

[6] M. A. Khadija, W. Nurharjadmo, and Widyawan, "Deep Learning Generative Indonesian Response Model Chatbot for JKN-KIS," *APICS 2022 - 2022 1st International Conference on Smart Technology, Applied Informatics, and Engineering, Proceedings*, pp. 70–74, 2022, doi: 10.1109/APICS56469.2022.9918686.

[7] N. Shi, Q. Zeng, and R. Lee, "Language Chatbot-The Design and Implementation of English Language Transfer Learning Agent Apps," *2020 IEEE 3rd International Conference on Automation, Electronics and Electrical Engineering, AUTEEE 2020*, pp. 403–407, Nov. 2020, doi: 10.1109/AUTEEE50969.2020.9315567.

[8] M. A. Khadija, A. Aziz, and W. Nurharjadmo, "Automating Information Retrieval from Faculty Guidelines: Designing a PDF-Driven Chatbot powered by OpenAI ChatGPT," in *Proceedings - 2023 10th International Conference on Computer, Control, Informatics and its Applications: Exploring the Power of Data: Leveraging Information to Drive Digital Innovation, IC3INA 2023*, Institute of Electrical and Electronics Engineers Inc., 2023, pp. 394–399. doi: 10.1109/IC3INA60834.2023.10285808.

[9] "ChatTube: Converse with YouTube Videos Using AI | Deepgram." Accessed: Apr. 12, 2024. [Online]. Available: https://deepgram.com/ai-apps/chattube