MP3 Transcription and Chatbot Application

Sanjay Bhaskar Kashyap

July 12, 2024

1 Objective

The goal of this project is to develop a domain-specific application that combines the strengths of a Large Language Model (LLM) for understanding and processing natural language queries with the efficiency of a vector database for data storage and retrieval. The application focuses on creating a responsive chatbot that provides personalized responses based on transcriptions of MP3 files.

2 Introduction

This application allows users to upload MP3 files, transcribe them using advanced speech-to-text technology, and interact with the resulting transcript through a conversational chatbot interface. The chatbot utilizes Retrieval-Augmented Generation (RAG) to generate responses based on user queries, ensuring that the answers are relevant and accurate to the context provided by the transcript.

3 Key Components

3.1 User Interaction

The application provides a user-friendly interface for uploading MP3 files and interacting with the transcript via a chatbot.

3.2 Transcription Process

The uploaded MP3 files are transcribed using AssemblyAI's transcription API.

3.3 Vector Database Integration

The transcript is split into chunks and stored in the Pinecone vector database for efficient retrieval.

3.4 Chatbot Implementation

The chatbot uses OpenAI's GPT model to process user queries and provide relevant responses based on the transcript stored in the vector database.

3.5 Backend Integration

The backend logic is managed using LangChain, which orchestrates the interaction flow and data retrieval.

4 Implementation Details

4.1 Setting Up Environment Variables

The application begins by setting up environment variables for the necessary API keys.

Listing 1: Setting up environment variables

```
os.environ['PINECONE_API_KEY'] = "your_pinecone_api_key"
os.environ['OPENAI_API_KEY'] = "your_openai_api_key"
os.environ['ASSEMBLYAI_API_KEY'] = "your_assemblyai_api_key"
```

4.2 Transcription of MP3 Files

MP3 files are uploaded through the Streamlit interface and transcribed using AssemblyAI.

Listing 2: Transcription of MP3 files

```
def transcribe_audio(file_path):
    transcriber = aai.Transcriber()
    transcript = transcriber.transcribe(file_path)
    if transcript.error:
        st.error(f"Transcription error: {transcript.error}")
        return ""
    return transcript.text
```

4.3 Storing Transcripts in Pinecone

The transcribed text is split into chunks and stored in the Pinecone vector database for efficient semantic search.

Listing 3: Storing transcripts in Pinecone

```
def push_transcription_to_pinecone(transcript):
    index.delete(delete_all=True, namespace="mp3-transcription-namespace")
    chunks = text_splitter.split_text(transcript)
    vectors = []
    for i, chunk in enumerate(chunks):
        vector = embeddings.embed_documents([chunk])[0]
        vectors.append({
            "id": str(i),
            "values": vector,
            "metadata": {"text": chunk, "source": f"chunk_{i}"}
        })
    index.upsert(vectors=vectors, namespace="mp3-transcription-namespace")
```

4.4 Query Processing with LangChain

The chatbot uses LangChain to process user queries and retrieve relevant chunks from the vector database.

Listing 4: Query processing with LangChain

```
def get_chatbot_response(user_query, transcript):
    if not check_query_relevance(user_query, transcript):
        return "I'm sorry, but the transcript doesn't contain any information re

    llm = OpenAI(temperature=0, max_tokens=500)
    qa_chain = RetrievalQA.from_chain_type(
        llm=llm,
        chain_type="stuff",
        retriever=retriever,
        return_source_documents=True,
        chain_type_kwargs={"prompt": PROMPT}
    )
    result = qa_chain({"query": user_query})
    answer = result['result']
    source_documents = result.get('source_documents', [])

    if "I don't have any information" in answer:
        return "I'm sorry, but I don't have any information about that in the pr

    sources = [doc.metadata.get('source', 'Unknown') for doc in source_documents
    response = f"{answer}\n\nSources: {', '.join(sources) if sources else 'No sp
    return response
```

4.5 User Interface with Streamlit

The user interface allows users to upload MP3 files and interact with the chatbot.

Listing 5: User interface with Streamlit

```
def main():
    st.title("MP3 Transcription and Chatbot")
    if 'transcript' not in st.session_state:
        st.session_state.transcript = None

    if 'messages' not in st.session_state:
        st.session_state.messages = []

    st.subheader("Upload an MP3 file for transcription:")
    uploaded_file = st.file_uploader("Choose an MP3 file", type="mp3")

    if uploaded_file and not st.session_state.transcript:
        with tempfile.NamedTemporaryFile(delete=False, suffix='.mp3') as tmp_fil
            tmp_file.write(uploaded_file.read())
            audio_file_path = tmp_file.name
            transcript = transcribe_audio(audio_file_path)
            st.session_state.transcript = transcript
            st.success("Transcription completed successfully.")
            push_transcription_to_pinecone(transcript)

    if st.session_state.transcript:
        st.subheader("Transcript")
        st.text_area("", st.session_state.transcript, height=250)
        st.subheader("Chat about the transcript:")

        for message in st.session_state.messages:
            with st.chat_message(message["role"]):
                st.markdown(message["content"])

        if prompt := st.chat_input("Ask a question about the transcript"):
            st.session_state.messages.append({"role": "user", "content": prompt}
            with st.chat_message("user"):
                st.markdown(prompt)
            with st.chat_message("assistant"):
                message_placeholder = st.empty()
                full_response = get_chatbot_response(prompt, st.session_state.tr
                message_placeholder.markdown(full_response)
            st.session_state.messages.append({"role": "assistant", "content": fu

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

5 Challenges and Solutions

5.1 Handling Large Transcripts

- **Challenge:** Managing and processing large transcripts efficiently.
- **Solution:** The transcripts are split into manageable chunks using a text splitter to ensure efficient processing and storage.

5.2 Ensuring Relevant Responses

- **Challenge:** Providing relevant responses to user queries.
- **Solution:** Implemented a preprocessing step to check query relevance before generating responses.

6 Conclusion

This project successfully demonstrates the integration of transcription services, vector databases, and LLMs to create a responsive and intelligent chatbot. The application enhances productivity by providing accurate and contextually relevant responses based on user queries and transcriptions of MP3 files.

7 Future Work

Future enhancements could include:

- Extending the application to support additional file formats.
- Improving the user interface for a more seamless user experience.
- Adding more advanced natural language processing capabilities to handle more complex queries.