Converting multiple podcasts Audio file into a searchable and queryable chat application

Using Groq Whisper and PineCone

Table of Contents

1. Introduction

The Problem:

Imagine having hours of valuable information locked away in podcast episodes. It's difficult to search for specific topics or quickly get answers from the content. This project aims to solve this by transforming podcasts into a searchable and queryable chat application.

The Solution:

This code leverages the power of two key technologies:

* Groq Whisper: This advanced speech-to-text engine accurately transcribes the audio content of podcast episodes, converting spoken words into searchable text.

* PineCone: This efficient vector database stores and retrieves embeddings (mathematical representations of the transcribed text). This allows for fast and accurate semantic search, enabling users to find relevant information using natural language queries.

Demonstration:

The final product is a user-friendly interface built with Streamlit. This interface empowers users to:

* Ask Questions: Type in any question related to the content of the podcasts.

* Receive Answers: The application leverages the transcribed text and embeddings to quickly identify and retrieve the most relevant information, providing accurate answers to user queries.

2. Step-by-Step Tutorial: Setting Up Your Development Environment

Project Structure

```
podcast_chat_app/
├── audio_processing/
│   ├── __init__.py
│   ├── audio_to_text.py
│   └── split_audio.py
├── data/
│   ├── mp3-files/
│   ├── mp3-chunks/
│   └── episode_metadata.csv
├── embeddings/
│   ├── __init__.py
│   └── vector_store.py
├── transcriptions/
│   ├── __init__.py
│   └── transcribe_podcasts.py
├── ui/
│   ├── __init__.py
│   └── streamlit_app.py
├── utils/
│   ├── __init__.py
│   └── environment.py
├── .env
├── .gitignore
├── README.md
├── requirements.txt
└── setup.py
```

Description of Each Directory and File:

- `audio_processing/`: Contains scripts related to audio file processing.
  - `audio_to_text.py`: Script for converting audio to text using Groq API.
  - `split_audio.py`: Script for splitting large MP3 files into smaller chunks.
  - `__init__.py`: Marks the directory as a Python package.
- `data/`: Contains raw data files.
  - `mp3-files/`: Directory for storing original MP3 files.
  - `mp3-chunks/`: Directory for storing chunks of MP3 files.
  - `episode_metadata.csv`: Metadata for podcast episodes.
- `embeddings/`: Contains scripts related to creating and managing embeddings.
  - `vector_store.py`: Script for storing and querying vector embeddings in Pinecone.
  - `__init__.py`: Marks the directory as a Python package.
- `transcriptions/`: Contains scripts related to transcription of podcasts.
  - `transcribe_podcasts.py`: Script for transcribing podcasts and creating document chunks.
  - `__init__.py`: Marks the directory as a Python package.
- `ui/`: Contains scripts related to the user interface of the application.
  - `streamlit_app.py`: Main script for the Streamlit user interface.
  - `__init__.py`: Marks the directory as a Python package.
- `utils/`: Contains utility scripts.
  - `environment.py`: Script for handling environment variables.
  - `__init__.py`: Marks the directory as a Python package.
- `.env`: File for storing environment variables.
- `.gitignore`: Git ignore file to exclude unnecessary files from version control.
- `README.md`: Documentation file for the project.
- `requirements.txt`: List of Python dependencies.
- `setup.py`: Script for setting up the Python package.

Create the Project Directory

Start by creating a new directory for your project:

```
mkdir podcast_chat_app
cd podcast_chat_app

# Audio Processing
mkdir audio_processing
touch audio_processing/__init__.py
touch audio_processing/audio_to_text.py
touch audio_processing/split_audio.py

# Data
mkdir data
mkdir data/mp3-files
mkdir data/mp3-chunks

touch data/episode_metadata.csv

# Embeddings
mkdir embeddings
touch embeddings/__init__.py
touch embeddings/vector_store.py

# Transcriptions
mkdir transcriptions
touch transcriptions/__init__.py
touch transcriptions/transcribe_podcasts.py

# UI
mkdir ui
touch ui/__init__.py
touch ui/streamlit_app.py

# Utils
mkdir utils
touch utils/__init__.py
touch utils/environment.py

# Root files
touch .env
touch .gitignore
touch README.md
touch requirements.txt
touch setup.py
```

Create and Activate a Virtual Environment

Create and Activate a Virtual Environment

Create a virtual environment to manage your project's dependencies:

```
python -m venv venv
source venv/bin/activate  # On Windows use `venv\Scripts\activate`
```

Create the Required Files

Create the necessary files and Subdirectories for your project:

```
groq
langchain
langchain_community
langchain_pinecone
pydub
tiktoken
sentence-transformers
pinecone_text
langchain_groq
streamlit
python-dotenv
```

Install Dependencies

`requirements.txt`

Install the required packages using **`pip`**:

```
pip install -r requirements.txt
```

Configure Environment Variables

Set up any necessary environment variables.

```
export GROQ_API_KEY='your_groq_api_key'
export PINECONE_API_KEY='your_pinecone_api_key'
```

3. Code for Files:

`audio_processing/audio_to_text.py`

```
import os
from groq import Groq

client = Groq(api_key=os.getenv('GROQ_API_KEY'))
model = 'whisper-large-v3'

def audio_to_text(filepath):
    with open(filepath, "rb") as file:
        translation = client.audio.translations.create(
            file=(filepath, file.read()),
            model=model,
        )
    return translation.text
```

`audio_processing/split_audio.py`

```
from audio_processing.audio_to_text import audio_to_text
from langchain.text_splitter import TokenTextSplitter
from langchain.docstore.document import Document
import pandas as pd
import numpy as np

def transcribe_podcasts(episodes_df, text_splitter):
    documents = []
    cnt = 0

    for index, row in episodes_df.iterrows():
        cnt += 1
        audio_filepath = row['filepath']
        transcript = audio_to_text(audio_filepath)
        chunks = text_splitter.split_text(transcript)
        for chunk in chunks:
            header = f"Date: {row['published_date']}\nEpisode Title: {row['title']}\n\n"
            documents.append(Document(page_content=header + chunk, metadata={"source": "local"}))
        if np.mod(cnt, round(len(episodes_df) / 5)) == 0:
            print(round(cnt / len(episodes_df), 2) * 100, '% of transcripts processed...')

    return documents
```

`embeddings/vector_store.py`

```
from langchain_community.embeddings.sentence_transformer import SentenceTransformerEmbeddings
from langchain_pinecone import PineconeVectorStore

def store_documents_in_pinecone(documents, index_name):
    embedding_function = SentenceTransformerEmbeddings(model_name="all-MiniLM-L6-v2")
    docsearch = PineconeVectorStore.from_documents(documents, embedding_function, index_name=index_name)
    return docsearch
```

transcriptions/transcribe_podcasts.py

```
from audio_processing.audio_to_text import audio_to_text
from langchain.text_splitter import TokenTextSplitter
from langchain.docstore.document import Document
import pandas as pd
import numpy as np

def transcribe_podcasts(episodes_df, text_splitter):
    documents = []
    cnt = 0

    for index, row in episodes_df.iterrows():
        cnt += 1
        audio_filepath = row['filepath']
        transcript = audio_to_text(audio_filepath)
        chunks = text_splitter.split_text(transcript)
        for chunk in chunks:
            header = f"Date: {row['published_date']}\nEpisode Title: {row['title']}\n\n"
            documents.append(Document(page_content=header + chunk, metadata={"source": "local"}))
        if np.mod(cnt, round(len(episodes_df) / 5)) == 0:
            print(round(cnt / len(episodes_df), 2) * 100, '% of transcripts processed...')

    return documents
```

ui/streamlit_app.py

```
import os
import streamlit as st
import pandas as pd
from embeddings.vector_store import store_documents_in_pinecone
from transcriptions.transcribe_podcasts import transcribe_podcasts
from utils.environment import load_env
from langchain_groq.chat_completion import transcript_chat_completion

# Load environment variables
groq_api_key, pinecone_api_key = load_env()

st.title("Podcast Content Query App")
st.write("Ask questions based on the content of the podcast.")

# Load episode metadata
episode_metadata_df = pd.read_csv('data/episode_metadata.csv')
chunk_fps = os.listdir('data/mp3-chunks/')
episode_chunk_df = pd.DataFrame({
    'filepath': [f"data/mp3-chunks/{fp}" for fp in chunk_fps],
    'episode_id': [fp.split('_chunk')[0] for fp in chunk_fps]
})
episodes_df = episode_chunk_df.merge(episode_metadata_df, on='episode_id')

# Transcribe podcasts
text_splitter = TokenTextSplitter(chunk_size=500, chunk_overlap=20)
documents = transcribe_podcasts(episodes_df, text_splitter)

# Store documents in Pinecone
pinecone_index_name = "podcast-transcripts"
docsearch = store_documents_in_pinecone(documents, pinecone_index_name)

# Handle user input
user_question = st.text_input("Enter your question:", "")

if user_question:
    with st.spinner("Querying..."):
        relevent_docs = docsearch.similarity_search(user_question)
        relevant_transcripts = '\n\n------------------------------------------------------
\n\n'.join([doc.page_content for doc in relevent_docs[:3]])
        response = transcript_chat_completion(groq_api_key, relevant_transcripts, user_question)
        st.write("**Answer:**", response)
```

`utils/environment.py`

```
from dotenv import load_dotenv
import os

def load_env():
    load_dotenv()
    groq_api_key = os.getenv('GROQ_API_KEY')
    pinecone_api_key = os.getenv('PINECONE_API_KEY')
    return groq_api_key, pinecone_api_key
```

`setup.py`

```
from setuptools import setup, find_packages

setup(
    name='podcast_chat_app',
    version='0.1',
    packages=find_packages(),
    install_requires=[
        'groq',
        'langchain',
        'langchain_community',
        'langchain_pinecone',
        'pydub',
        'tiktoken',
        'sentence-transformers',
        'pinecone_text',
        'langchain_groq',
        'streamlit',
        'python-dotenv',
    ],
)
```

This structure ensures that the project is modular, easy to navigate, and scalable. Each component has its own directory and scripts, making it easier to manage and maintain the code.

4. Running the Code in VSCode Terminal

1. **Open the Project in VSCode**
   - Open VSCode and select `File > Open Folder...` to open the `podcast_chat_app` directory.
2. **Open the Terminal**
   - Open the terminal in VSCode by selecting `View > Terminal` or using the shortcut `Ctrl+``.
3. **Run the Streamlit App**
   - Finally, run the main script: streamlit_app.py

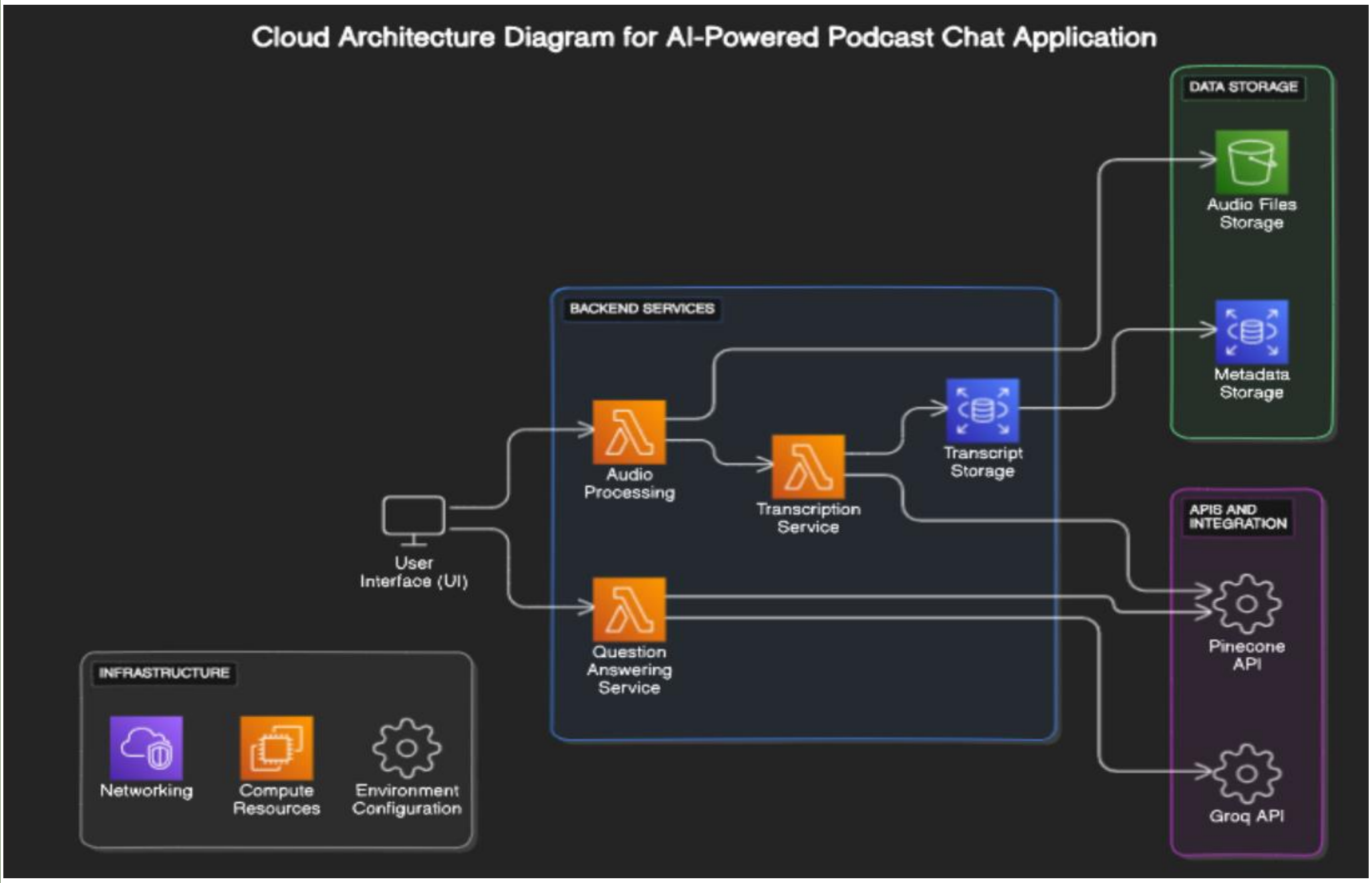
```
streamlit run ui/streamlit_app.py
```

This setup ensures that the code runs smoothly in a local VSCode environment, with clear instructions for setting up the project, organizing files, and executing the code.

5. Cloud Architecture Diagram for AI-Powered Podcast Chat Application

Project Overview:

A detailed cloud architecture diagram for an AI-powered podcast chat application. The application transcribes podcast audio files, stores the transcripts in a vector database, and provides a web interface for users to query the content of the podcasts. The architecture has included the following components and details:



Components:

1. **Client Side:**
    - **User Interface (UI):**
        - A web application built using Streamlit for user interaction.
        - Users can input questions about podcast content and receive responses.
2. **Backend Services:**
    - **Transcription Service:**
        - Uses Groq's Whisper model to transcribe audio files into text.
    - **Audio Processing:**
        - Splits large audio files into smaller chunks using Pydub.
    - **Transcript Storage:**
        - Stores transcribed text and metadata in a Pinecone vector database for efficient querying.

    - **Question Answering Service:**
        - Uses Groq's LLaMA model to generate responses based on user queries and transcript content.

3. **Data Storage:**
    - **Audio Files Storage:**
        - A cloud storage solution (e.g., AWS S3 or Google Cloud Storage) to store the original and chunked audio files.
    - **Metadata Storage:**
        - A CSV file or a cloud database (e.g., AWS RDS or Google Cloud SQL) to store podcast metadata such as titles, publication dates, and episode IDs.
4. **APIs and Integration:**
    - **Groq API:**
        - For audio transcription and chat completion services.
    - **Pinecone API:**
        - For storing and querying vector embeddings of the transcriptions.
5. **Infrastructure:**
    - **Compute Resources:**
        - Cloud-based virtual machines or containers (e.g., AWS EC2, Google Compute Engine, or Docker on Kubernetes) for running the transcription and question answering services.
    - **Networking:**
        - Secure communication between UI, backend services, and APIs.
    - **Environment Configuration:**
        - Use of environment variables for API keys and configuration settings.

Workflow:

1. **Audio Upload:**
    - Users upload podcast audio files to the web application.
2. **Audio Processing:**
    - The application splits the audio files into chunks using Pydub.
    - The chunks are stored in cloud storage.
3. **Transcription:**
    - The application uses Groq's Whisper model to transcribe each audio chunk.
    - Transcripts are stored along with metadata in a cloud database and indexed in the Pinecone vector database.
4. **Query Handling:**
    - Users enter queries in the Streamlit web application.
    - The application retrieves relevant transcript chunks from Pinecone based on the query.
    - The Groq LLaMA model generates responses using the retrieved transcripts.
5. **Response Delivery:**
    - The application displays the generated response to the user in the web interface.

Constructive comments and feedback are welcomed