Lab Exercise: Building a Multi-Stage Audio Transcription and Processing Application with OpenAI

Summary of the Lab

This lab is designed to teach students how to build a Python application that performs audio file conversion, transcription, summarization, and script creation using OpenAI's APIs and external libraries like `ffmpeg`. The focus is on practical experience with:

1. **File Handling**: Students will learn how to work with audio files, convert them from `.m4a` to `.wav` using `ffmpeg`, and manipulate files in Python.
2. **Using OpenAI's APIs**: Students will integrate OpenAI's Whisper model for transcription and the GPT model for summarization and script generation. This will give students hands-on experience with AI models for natural language processing tasks.
3. **Error Handling and Automation**: The lab teaches students how to handle errors gracefully, such as API failures or missing files, and how to automate the entire workflow (from audio conversion to script generation).
4. **Practical Application**: By the end of this lab, students will have a fully functional script that can take an audio file, transcribe it, summarize the transcript, and even create a script from the transcript. This mirrors real-world applications in fields like content creation, journalism, and data processing.

Overall, the lab combines both the technical aspects of working with external APIs and file manipulation, as well as applying those skills to solve a realistic problem in the context of media processing.

Overview:

In this exercise, you will develop a Python application that transcribes audio from an `.m4a` file, summarizes the transcript, and creates a script from the transcript using OpenAI's APIs. Along the way, you will gain hands-on experience with file manipulation, external libraries (like `ffmpeg`), OpenAI's API, and the overall process of creating a multi-step application.

The code provided involves multiple stages:

1. Converting audio from `.m4a` to `.wav` format.
2. Transcribing the audio to text using OpenAI's Whisper model.
3. Summarizing the transcript.
4. Creating a script from the transcript with speaker differentiation.

You will build this application step-by-step, starting from the basics and adding complexity as you go.

Preparation: Setting up Your Environment

Before you begin the coding exercises, ensure that your environment is properly set up to avoid unnecessary troubleshooting.

1. **Install Python 3.7+** Make sure you are using Python 3.7 or later. You can download it from the official Python website.

2. **Create a Virtual Environment** To isolate dependencies, create a new Python virtual environment by running:

   ```
   python -m venv audio_transcribe_env
   ```

3. **Activate the Virtual Environment:**

   - On **Windows**:

```
.\audio_transcribe_env\Scripts\activate
```

- On **macOS/Linux**:

  ```
  source audio_transcribe_env/bin/activate
  ```

4. **Install Required Packages** Run the following command to install necessary dependencies:

   ```
   pip install openai ffmpeg-python
   ```

5. **Set up the OpenAI API Key:**

   - To use OpenAI's models (Whisper, GPT, etc.), you'll need an API key.
   - Get your OpenAI API key and set it in your environment variables:
     - On **Windows**:

       ```
       set OPENAI_API_KEY=your_api_key
       ```

     - On **macOS/Linux**:

       ```
       export OPENAI_API_KEY=your_api_key
       ```

6. **Install FFmpeg**
   FFmpeg is used to convert `.m4a` files into `.wav`. You can download and install FFmpeg from FFmpeg.org.
   After installation, verify it's working by running:

   ```
   ffmpeg -version
   ```

Step 1: Basic Audio File Conversion

In this first step, we'll focus on converting an `.m4a` audio file to `.wav` format using `ffmpeg`.

Objective:

- Use `subprocess` to run an external command (`ffmpeg`) to convert `.m4a` to `.wav` .
- Ensure the audio is in mono format with a standard sample rate (e.g., 16 kHz).

Instructions:

1. Create a new Python file called `audio_converter.py` .
2. Write a function `convert_audio(file_path: str) -> Optional[str]` that takes the file path of an `.m4a` file and converts it to `.wav` using `ffmpeg` . The output `.wav` file should be mono and use a sample rate of 16 kHz.
    - Check if the file ends with `.m4a` . If it doesn't, print an error and return `None` .
    - Use `subprocess.run()` to call `ffmpeg` .
    - Ensure the `.wav` file is written to disk after conversion.
3. Test the conversion function with a sample `.m4a` file.

Guiding Hints:

- Use the `subprocess.run()` function to run the `ffmpeg` command. Example command:

  ```
  ffmpeg -i input.m4a -ar 16000 -ac 1 -c:a pcm_s16le output.wav
  ```

  This will convert the audio to 16 kHz, mono, and PCM format.

Step 2: Transcription of Audio

Once you can convert the audio file, the next step is transcribing the audio using OpenAI's Whisper API.

Objective:

- Use OpenAI's `Whisper` model to transcribe the audio to text.

Instructions:

1. Write a function `transcribe_audio(file_path: str)` that:
    - First calls `convert_audio()` to get a `.wav` file.

    - Then, uses the OpenAI API to transcribe the `.wav` file.

2. Save the transcript to a file called `transcript.txt`.

3. Test the transcription function with a sample `.m4a` file and check if the result is stored in `transcript.txt`.

Guiding Hints:

- Use the `openai.Audio.transcriptions.create()` method to send the `.wav` file to OpenAI for transcription.
- Remember to open the `.wav` file in binary mode (`rb`) when passing it to the API.

Step 3: Summarizing the Transcript

Once you have a transcript, the next step is to summarize the text to make it more concise and actionable.

Objective:

- Use OpenAI's GPT model to summarize the transcript.

Instructions:

1. Write a function `summarize_text(text: str) -> str` that:

    - Takes the transcript text as input and uses the OpenAI GPT model to generate a summary.

2. Save the summary to a file called `summary.txt`.

3. Test the summarization function to ensure the output is concise and contains the key points.

Guiding Hints:

- Use `openai.Completion.create()` with a prompt that requests a summary.
- Example prompt: `"Summarize the following text by providing key points and action items:"`

Step 4: Creating a Script from the Transcript

The final step is to create a formatted script from the transcript. The script should attempt to differentiate speakers and format the text as a dialogue.

Objective:

- Use OpenAI's GPT model to create a script from the transcript with speaker names.

Instructions:

1. Write a function `create_script_from_transcript(text: str) -> str` that:
    - Takes the transcript text and formats it into a script format.
    - The script should differentiate between speakers and format the dialogue neatly.
2. Save the script to a file called `script.txt`.
3. Test the script creation function to ensure the output is properly formatted.

Guiding Hints:

- In the prompt, ask GPT to differentiate between speakers and format the transcript as a script. Example prompt: `"Create a script from the following transcript by distinguishing the likely speakers and formatting the text into a tight script format."`

Step 5: Final Integration

Now that all individual components are working, integrate the functions into a complete program that:

1. Takes an `.m4a` file as input.
2. Converts the audio to `.wav`.
3. Transcribes the audio.
4. Summarizes the transcript.
5. Creates a script from the transcript.

You should check if the files already exist (e.g., transcript, summary, script) and only perform the respective operations if the files are not found.

Final Challenge: Automation and Error Handling

- Add error handling to ensure the program doesn't crash if an error occurs during any step (e.g., invalid file format, OpenAI API failure).
- Automate the process so that the program can be run with just one command (e.g., `python main.py my_audio.m4a`).

Bonus Challenge:
Optimize the audio conversion process by adjusting the compression levels or applying better error handling when the `.m4a` file size exceeds the target limits.

Assessment Criteria:

- Code correctness and functionality (does it convert, transcribe, summarize, and script as expected?).
- Clean code with clear function separation and error handling.
- Proper use of OpenAI API and external libraries.
- Documentation and comments explaining key steps.

By the end of this lab, you should have a robust Python application capable of audio file conversion, transcription, summarization, and script generation!