Training Day 7 Report

Speech To Text Generation:

Purpose

This Python script captures a 5-second audio recording, sends it to AssemblyAI (a speech-to-text API service), transcribes the audio into text, and saves the result in a file named output.txt.

It involves:

- **Recording audio** using your microphone
- **Uploading** that audio to AssemblyAI
- **Polling** (repeated checking) the API until the transcription is complete
- **Saving** and displaying the final transcribed text
- **Technologies Used**

Module	Purpose
sounddevice	: Captures audio from the microphone
soundfile	: Saves the recorded audio to a .wav file
requests	: Sends HTTP requests to AssemblyAI's REST API
time	: Adds delay between polling attempts
AssemblyAI	: Performs speech-to-text transcription using cloud-based AI

Why Use This Approach?

- Local recording gives user control over data capture.
- AssemblyAI offers highly accurate AI-powered transcription.
- The script runs entirely in terminal, making it lightweight and beginner-friendly.

📥 Input

- No user input is needed — the script records audio for 5 seconds automatically.

📤 Output

- recorded.wav: A saved recording of the user's voice.
- output.txt: A file containing the transcribed text of the spoken audio.

Step-by-Step Explanation of the Code

Step 1: Record Audio

python
CopyEdit

duration: How many seconds to record (5 seconds here).

fs (sampling rate): 44,100 samples per second (CD-quality audio).

filename: Name of the file where audio will be saved.

- **sd.rec(...)**: Starts recording audio into a NumPy array.
- **sd.wait()**: Waits until the recording is finished.
- **sf.write(...)**: Saves the array into a .wav file.
-

WAV Format is used because AssemblyAI expects standard formats like .wav

Step 2: Upload Audio to AssemblyAI

Prepares the API key and endpoint to upload the audio file.

- Sends the audio file to AssemblyAI's /upload endpoint.
- Gets a secure audio_url from the response, which will be used for transcription.

Step 3: Start Transcription Request

- Sends a transcription request to the AssemblyAI API.
- The request includes the audio_url of the uploaded file.
- API returns a unique transcript_id used to track the progress.

Polling:

Polling is a **core strategy** when dealing with async APIs that need processing time. It:

- Enables your script to wait smartly.
- Gives you control over pacing (with sleep).
- Is safer and more predictable than using callbacks in APIs that don’t support them.

Polling and Saving Transcription:

- This continuously checks (polls) the status of the transcription.
- AssemblyAI usually takes a few seconds to process the request.
- When status is "completed", the text result is extracted and written to output.txt.
- The transcription is printed to the terminal as well.
- Prints the error if transcription failed. Waits for 3 seconds before polling again (to avoid overloading the API).

Why is Polling Used in This Script?

AssemblyAI **doesn't immediately return** the transcription. Instead, it:

1. Accepts the audio for transcription.
2. Starts processing it.
3. Asks the client to poll a specific **/transcript/{id}** endpoint to know when it's done.

Theory Behind the Components & Future Enhancements:

1. Audio Recording

- Capturing analog sound via microphone and converting it into a digital signal using a sample rate.
- Audio is stored in a .wav file in PCM format.

---

2. REST API

- A REST API allows a client (Python script) to communicate with a server (AssemblyAI) using standard HTTP methods:
    - POST to send data
    - GET to retrieve status
- JSON format is used for sending/receiving structured data.

---

3. Speech-to-Text (STT)

- AssemblyAI uses deep learning models (likely transformers or CNN-RNN hybrids) trained on thousands of hours of audio to:
    - Detect spoken words
    - Convert them into accurate transcriptions