DAY 7

Speech-to-Text Applications Using AssemblyAI & Gemini

Today's training was focused entirely on **converting voice to text** using two different projects. Although both involved recording and processing voice input, the main goal in both was to display **textual output only**.

Speech-to-Text Using AssemblyAI

What It Does

This script allows users to record **5 seconds of audio**, and converts their spoken words into **written text** using the **AssemblyAI transcription API**. It's a simple but powerful tool to demonstrate how real-time voice data can be processed and turned into readable output.

How It Works

1. **Audio Recording**

   Uses a sound device to record the user's voice and saves it in .wav format.

2. **Audio Cleaning**

   The raw audio is processed using pydub to ensure it's in a proper format before uploading.

3. **Upload to AssemblyAI**

   The cleaned .wav file is uploaded via AssemblyAI's secure API.

4. **Transcription Request**

   A request is sent to transcribe the uploaded file.

5. **Polling for Result**

   The script checks the status every few seconds until transcription is complete.

6. **Text Output**

   The final transcription is printed directly in the terminal.

7. **Optional Cleanup**

   The user is asked whether they want to delete or save the recorded file.

Technologies Used

- sounddevice, scipy.io.wavfile, pydub – for audio recording & conversion
- requests – for API calls
- dotenv – for securely loading the API key
- AssemblyAI – for speech-to-text processing

Key Learning

- How to structure end-to-end voice input → text output pipelines
- Importance of audio format when working with transcription APIs
- Understanding asynchronous processing and polling logic
- Clean code structure for managing temporary files and user interaction

This tool provided a clean and accurate way to capture spoken language and turn it into usable text - ideal for note-taking, journaling, or command recognition systems.

Gemini-Backed Voice-to-Text Assistant

This project looked like a full voice assistant, but for this use case, it was **intentionally kept as a speech-to-text pipeline only**. Even though it included tools like Gemini and Flask, the final response from the assistant was **only displayed in text form** — no spoken output involved.

Objective

Convert voice input to text, send that text to **Gemini AI** for a smart response, and print Gemini's reply on-screen.

Flow of Execution

1. **Voice Input**

   Records audio for 5 seconds and saves it.

2. **Audio Preparation**

   Formats the audio using pydub to make it ready for upload.

3. **AssemblyAI Transcription**

   Uploads the audio and gets the text output after polling.

4. **Gemini AI Response**

   The transcription is sent to Gemini, which processes it and sends back a natural-language reply.

5. **Text-Only Output**

   The Gemini reply is:

   - Shown in a **Flask web interface**
   - Logged in a local .txt file
   - **Not** converted into speech — all output remains in text form

Tech Stack

Component	Role
AssemblyAI	Converts voice to text
Gemini API	Generates intelligent replies
Flask	Web interface to display responses
Python(pydub,sounddevice,scipy)	Audio handling
.env	Securely stores API keys

Output Example

- **User says:** “What’s the capital of Japan?”
- **AssemblyAI transcribes:** “What’s the capital of Japan?”
- **Gemini replies:** “The capital of Japan is Tokyo.”
- The reply is displayed in the browser and logged in output.txt.

Learning Points

- How to combine two different APIs (AssemblyAI + Gemini)
- Building lightweight web interfaces using Flask

- Maintaining a fully **text-based output system**, even when working with voice input
- Using file logging to save conversations for review or audit

Final Takeaways

Both tools I built today are **pure speech-to-text applications**. While one transcribes and prints the user's own voice, the other interprets the meaning and returns a smart reply using Gemini — **all in text format**.

Even though the second project had the potential for spoken replies, I kept it focused on text output, just as planned. This ensures the applications stay lightweight, focused, and easy to integrate into future projects like chat interfaces, productivity apps, or education bots.

I'm now much more confident in building:

- Voice-to-text tools
- Voice-controlled AI pipelines
- Text-based smart assistants