### ****Internship Task - 2: Speech Recognition System****

Build a basic **Speech-to-Text system** using pre-trained models .

To build a basic speech-to-text system, you can follow these steps:

**Step 1: Install Required Libraries**

Make sure to install `speech\_recognition` and any other necessary libraries. You can do this using pip:

Pip install SpeechRecognition

Pip install pydub # For audio file handling if needed

Pip install wav2vec2 #If using wav2vec models

**Step 2: Import Libraries**

Import the necessary libraries into your script:

Import speech\_recognition as sr

**Step 3: Initialize the Recognizer**

Create an instance of the `Recognizer` class:

Recognizer = sr.Recognizer()

**Step 4: Load the Audio File**

Load your audio file. You can either use an audio file or record from a microphone.

Using an audio file:

Audio\_file = “your\_audio\_file.wav” #Replace with your file path

With sr.AudioFile(audio\_file) as source:

Audio\_data = recognizer.record(source)

**Step 5: Recognize Speech**

Now, use the recognizer to convert speech to text:

Try:

Text = recognizer.recognize\_google(audio\_data) Google Web Speech API

Print(“Transcription: “, text)

Except sr.UnknownValueError:

Print(“Sorry, I could not understand the audio.”)

Except sr.RequestError as e:

Print(“Could not request results; {0}”.format(e))

**Step 6: Run the Program**

Run your script, and it should print the transcription of the audio clip.

Import torch

Import torchaudio

From transformers import Wav2Vec2ForCTC, Wav2Vec2Tokenizer

Import os

# Load pretrained model and tokenizer

Tokenizer = Wav2Vec2Tokenizer.from\_pretrained(“facebook/wav2vec2-base-960h”)

Model = Wav2Vec2ForCTC.from\_pretrained(“facebook/wav2vec2-base-960h”)

# Load the audio file

Def transcribe(audio\_path):

# Load audio file (must be WAV, 16kHz)

Speech, rate = torchaudio.load(audio\_path)

# If stereo, convert to mono

If speech.shape[0] > 1:

Speech = speech.mean(dim=0).unsqueeze(0)

# Resample if not 16kHz

If rate != 16000:

Resampler = torchaudio.transforms.Resample(orig\_freq=rate, new\_freq=16000)

Speech = resampler(speech)

# Tokenize and transcribe

Input\_values = tokenizer(speech.squeeze().numpy(), return\_tensors=”pt”).input\_values

With torch.no\_grad():

Logits = model(input\_values).logits

Predicted\_ids = torch.argmax(logits, dim=-1)

Transcription = tokenizer.decode(predicted\_ids[0])

Return transcription

# Example usage

Audio\_file = “your\_audio.wav” # Provide your audio path here

Print(“Transcription:”, transcribe(audio\_file))

Let’s explore some modifications and applications:

Modifications

1. \*Save Transcription to File\*: Save the transcribed text to a file instead of printing it.

2. \*Continuous Speech Recognition\*: Continuously listen for speech and transcribe it in real-time.

3. \*Specific Keyword Detection\*: Detect specific keywords or phrases in the speech.

Applications

1. \*Virtual Assistant\*: Integrate speech recognition with a virtual assistant to perform tasks.

2. \*Transcription Service\*: Offer a transcription service for podcasts, lectures, or interviews.

3. \*Accessibility Tool\*: Develop an accessibility tool for individuals with disabilities.

**Code Examples**

Save Transcription to File

With open(“transcription.txt”, “w”) as f:

f.write(text)

**Continuous Speech Recognition**

While True:

With sr.Microphone() as source:

Audio = r.listen(source)

Try:

Text = r.recognize\_google(audio)

Print(text)

Except sr.UnknownValueError:

Print(“Sorry, I didn’t catch that.”)

**Deliverable:**

**A functional system that can transcribe short audio clips.**

The deliverable is a working Speech-to-Text system that can take a short audio clip as input and accurately convert the spoken words into written text. This system uses Python and a pre-trained speech recognition library such as speech\_recognition or Wav2Vec. It processes the audio input (in formats like .wav), interprets the speech using AI models, and outputs the corresponding text transcription.This ensures the system demonstrates the basic functionality of automatic speech recognition (ASR), making it suitable for real-world applications like voice assistants, transcription tools, and accessibility services.

To create a functional system capable of transcribing short audio clips, you can follow these steps:

**1.Define Requirements**

Functional Requirements:

* + Accept audio input (e.g., MP3, WAV).
  + Transcribe the audio to text.
  + Output the transcribed text.

Non-Functional Requirements:

* + User-friendly interface.
  + High accuracy of transcription.
  + Fast processing time.

**2.Choose a Technology Stack**

Languages & Frameworks:

* + Python: For processing audio and transcription.
  + Flask/Django: If you want to create a web interface.

Libraries and Tools:

* + SpeechRecognition: For converting speech to text.
  + pydub: For audio file manipulation.
  + audio-transcription API: Such as Google Speech-to-Text, IBM Watson, or Azure Speech Service for better accuracy (optional).

3. **Setup the Environment**

* + Install Python and create a virtual environment.
  + Install necessary libraries:

Pip install SpeechRecognition pydub Flask

**4.Create the Transcription Functionality**

Here’s a sample code using `SpeechRecognition` and Flask for building a simple transcription web service:

From flask import Flask, request, jsonify

Import speech\_recognition as sr

From pydub import AudioSegment

Import os

App = Flask(\_\_name\_\_)

Def transcribe\_audio(file\_path):

Initialize recognizer

Recognizer = sr.Recognizer()

Audio\_file = sr.AudioFile(file\_path)

With audio\_file as source:

Audio\_data = recognizer.record(source)

Recognizing speech using Google Web Speech API

Try:

Text = recognizer.recognize\_google(audio\_data)

Return text

Except sr.UnknownValueError:

Return “Audio Unintelligible”

Except sr.RequestError as e:

Return f”Could not request results; {e}”

@app.route(‘/transcribe’, methods=[‘POST’])

Def transcribe():

If ‘file’ not in request.files:

Return jsonify({“error”: “No file part”})

File = request.files[‘file’]

If file.filename == ‘’:

Return jsonify({“error”: “No selected file”})

Save the file temporarily

File\_path = os.path.join(“uploads”, file.filename)

File.save(file\_path)

Transcribe the audio

Transcription = transcribe\_audio(file\_path)

Remove the file after processing

Os.remove(file\_path)

Return jsonify({“transcription”: transcription})

If \_\_name\_\_ == ‘\_\_main\_\_’:

App.run(debug=True)

```

**5.Create a Simple Frontend Interface (Optional)**

You may create a simple HTML form to upload audio files.

```html

<!DOCTYPE html>

<html>

<head>

<title>Audio Transcription</title>

</head>

<body>

<h1>Upload Audio for Transcription</h1>

<form action=”/transcribe” method=”post” enctype=”multipart/form-data”>

<input type=”file” name=”file” accept=”audio/\*” required>

<input type=”submit” value=”Transcribe”>

</form>

</body>

</html>

**6. Testing the System**

* + Test with various audio clips.
  + Measure transcription accuracy and adjust the recognizer settings if necessary.

**7. Deployment**

* + Deploy the application on platforms like Heroku, AWS, or any suitable cloud platform.

**8. Enhance the System (Optional)**

* + Add user authentication for privacy.
  + Support various audio formats.
  + Improve UI/UX for better usability.

**Conclusion**

This setup provides a basic, functional audio transcription system. Depending on your requirements, you can further enhance the system by integrating more advanced speech recognition services for increased accuracy and functionality.