Dev Halvawala-devhalvawala15@gmail.com

Detailed Description of the Solution

Overview:

The solution is an AI-powered voice assistance pipeline that records user audio, transcribes it to text, generates a response using a language model, and finally converts the response back to speech. The project involves several steps, including audio recording, speech-to-text conversion, text processing using a language model, and text-to-speech conversion.

Choice of Models, Libraries, and Parameters:

Audio Recording:

1. **JavaScript & HTML**: Used to create a simple audio recorder in the browser. The MediaRecorder API is used for capturing audio streams, and the recorded audio is then processed into a format that can be handled by the backend.
2. **ffmpeg-python**: Employed for audio processing, specifically to convert the audio from the browser-compatible format (WebM) to a WAV file, which is easier to handle in Python.

Speech-to-Text Conversion:

1. **SpeechRecognition**: The Python library `SpeechRecognition` is used to convert the recorded audio into text. This library supports various speech recognition engines and APIs, and in this case, the Google Web Speech API is used.
2. **Pydub**: The `Pydub` library is used to handle audio file manipulations like opening and saving audio files.

Text Processing:

1. **Transformers**: The Hugging Face `transformers` library is used to load and interact with a pre-trained language model. Specifically, the GPT-2 model (`gpt2-medium`) is used to generate responses based on the transcribed text.
2. **Tokenization and Generation Configurations**: Various parameters like `max_new_tokens`, `temperature`, and `top_p` are configured to control the response generation process. Padding is added to the tokenizer to handle input sequences of varying lengths.

Text-to-Speech Conversion:

1. **Edge TTS**: The `edge-tts` library is used for converting the generated text response back to speech. This library leverages Microsoft's Azure Text-to-Speech service, allowing for high-quality speech synthesis with different voice options.

Code Snippets

```
# -*- coding: utf-8 -*-
"""Dev Halvawala_Lizomotors

Automatically generated by Colab.
Original file is located at
    https://colab.research.google.com/drive/14uj0fwKbMIOXFrx0wQSA3pYkH2
dF3Pen
```

```
# **Step - 1 Record Audio from Microphone**
"""
```

```
!pip install ffmpeg-python
from google.colab import drive
drive.mount('/content/drive')
```

```
"""
To write this piece of code I took inspiration/code from a lot of
places.

Here are some of the possible references:
https://blog.addpipe.com/recording-audio-in-the-browser-using-pure-
html5-and-minimal-javascript/
https://stackoverflow.com/a/18650249
https://hacks.mozilla.org/2014/06/easy-audio-capture-with-the-
mediarecorder-api/
https://air.ghost.io/recording-to-an-audio-file-using-html5-and-js/
https://stackoverflow.com/a/49019356
"""
from IPython.display import HTML, Audio
from google.colab.output import eval_js
from base64 import b64decode
import numpy as np
from scipy.io.wavfile import read as wav_read
import io
import ffmpeg
AUDIO_HTML = """
<script>
var my_div = document.createElement("DIV");
var my_p = document.createElement("P");
var my_btn = document.createElement("BUTTON");
var t = document.createTextNode("Press to start recording");
my_btn.appendChild(t);
//my_p.appendChild(my_btn);
my_div.appendChild(my_btn);
document.body.appendChild(my_div);
var base64data = 0;
var reader;
```

```
var recorder, gumStream;
var recordButton = my_btn;
var handleSuccess = function(stream) {
  gumStream = stream;
  var options = {
    //bitsPerSecond: 8000, //chrome seems to ignore, always 48k
    mimeType : 'audio/webm;codecs=opus'
    //mimeType : 'audio/webm;codecs=pcm'
  };
  //recorder = new MediaRecorder(stream, options);
  recorder = new MediaRecorder(stream);
  recorder.ondataavailable = function(e) {
    var url = URL.createObjectURL(e.data);
    var preview = document.createElement('audio');
    preview.controls = true;
    preview.src = url;
    document.body.appendChild(preview);
    reader = new FileReader();
    reader.readAsDataURL(e.data);
    reader.onloadend = function() {
      base64data = reader.result;
      //console.log("Inside FileReader:" + base64data);
    }
  };
  recorder.start();
  };
recordButton.innerText = "Recording... press to stop";
navigator.mediaDevices.getUserMedia({audio: true}).then(handleSuccess);
function toggleRecording() {
  if (recorder && recorder.state == "recording") {
      recorder.stop();
      gumStream.getAudioTracks()[0].stop();
      recordButton.innerText = "Saving the recording... pls wait!"
  }
}
// https://stackoverflow.com/a/951057
function sleep(ms) {
  return new Promise(resolve => setTimeout(resolve, ms));
}
var data = new Promise(resolve=>{
//recordButton.addEventListener("click", toggleRecording);
recordButton.onclick = ()=>{
toggleRecording()
sleep(2000).then(() => {
  // wait 2000ms for the data to be available...
  // ideally this should use something like await...
  //console.log("Inside data:" + base64data)
  resolve(base64data.toString())
```

```
});
}
});
</script>
"""
def get_audio():
  display(HTML(AUDIO_HTML))
  data = eval_js("data")
  binary = b64decode(data.split(',')[1])
   # Check if data contains a comma before splitting
  if ',' in data:
    binary = b64decode(data.split(',')[1])
  else:
    # Handle the case where data does not contain a comma
    print("Error: Invalid data format. Base64 encoded string does not
contain a comma.")
    return None, None  # Or raise an exception if appropriate
  process = (ffmpeg
    .input('pipe:0')
    .output('pipe:1', format='wav')
    .run_async(pipe_stdin=True, pipe_stdout=True, pipe_stderr=True,
quiet=True, overwrite_output=True)
  )
  output, err = process.communicate(input=binary)
  riff_chunk_size = len(output) - 8
  # Break up the chunk size into four bytes, held in b.
  q = riff_chunk_size
  b = []
  for i in range(4):
      q, r = divmod(q, 256)
      b.append(r)
  # Replace bytes 4:8 in proc.stdout with the actual size of the RIFF
chunk.
  riff = output[:4] + bytes(b) + output[8:]
  sr, audio = wav_read(io.BytesIO(riff))
  return audio, sr
```

```
audio, sr = get_audio()
```

```
if audio is not None:  # Check if audio was successfully recorded
  import scipy.io.wavfile as wav
  # Specify the filename for the saved audio file
  filename = "/content/drive/MyDrive/Colab Notebooks/Lizomotors-Design
an End-to-End AI Voice Assistance Pipeline/recorded_audio.wav"
  # Save the audio file
  wav.write(filename, sr, audio)
  print(f"Audio file saved as {filename}")
```

```
"""**Audio File to Text**"""
```

```
!pip install SpeechRecognition
!pip install pydub
import os
import speech_recognition as sr
from pydub import AudioSegment
```

```
r = sr.Recognizer()
#Open the audio file
with sr.AudioFile("/content/drive/MyDrive/Colab Notebooks/Lizomotors-
Design an End-to-End AI Voice Assistance Pipeline/recorded_audio.wav")
as source:
  audio_text = r.record(source)
#Recognize the speech in the media
text = r.recognize_google(audio_text, language = 'en-US')
```

```
#Print the transcript
file_name = ("/content/drive/MyDrive/Colab Notebooks/Lizomotors-Design
an End-to-End AI Voice Assistance Pipeline/transcription.txt")
with open(file_name, "w") as file:
  #write to the file
  file.write(text)
  #open the file for editing
os.system(f"start {file_name}")
```

```
with open('/content/drive/MyDrive/Colab Notebooks/Lizomotors-Design an
End-to-End AI Voice Assistance Pipeline/transcription.txt','r')as f:
  text = f.read()
print(text)
```

```
"""# **Step-2 Text input to LLM**"""
```

```
!pip install transformers torch sentencepiece
"""**HF_TOKEN-hf_pNcpCsAaRSzZsSWnBYOAYDOvvUXPXSM**"""
```

```
from transformers import GPT2Tokenizer, GPT2LMHeadModel
from transformers import OpenAIGPTTokenizer, OpenAIGPTModel
MODEL_NAME = 'gpt2-medium'#'openai-gpt'#'distilgpt2'  'distilgpt2' #
tokenizer = GPT2Tokenizer.from_pretrained(MODEL_NAME)
model = GPT2LMHeadModel.from_pretrained(MODEL_NAME)
#----------------------------------------------------------------
# Load model directly
"""from transformers import AutoTokenizer, AutoModelForCausalLM
tokenizer = AutoTokenizer.from_pretrained("meta-llama/Llama-2-7b-hf")
model = AutoModelForCausalLM.from_pretrained("meta-llama/Llama-2-7b-
hf")"""
#--------------------------------------------------------------------
"""from transformers import pipeline
```

```
messages = [
    {"role": "system", "content": "You are a pirate chatbot who always
responds in pirate speak!"},
    {"role": "user", "content": "Who are you?"},
]
chatbot = pipeline("text-generation", model="mistralai/Mistral-7B-
Instruct-v0.3")
chatbot(messages)"""
```

```
transcribed_text = "what is today's weather in New York?"
```

```
"""from transformers import GenerationConfig
# Define generation configuration
generation_config = GenerationConfig(
    max_new_tokens=50,       # Limits the response length
    temperature=0.7,         # Controls randomness
    top_p=0.9,               # Controls diversity
    do_sample=True,          # Enables sampling
    eos_token_id=tokenizer.eos_token_id
)
# Prepare input tokens
input_ids = tokenizer.encode(transcribed_text, return_tensors="pt")
# Generate response
outputs = model.generate(
    input_ids=input_ids,
    generation_config=generation_config
)
# Decode and process the output
generated_text = tokenizer.decode(outputs[0], skip_special_tokens=True)
"""
#-----------------------------------------------------------------------
--------
```

```
from transformers import GenerationConfig, AutoTokenizer,
AutoModelForCausalLM
# Define generation configuration
generation_config = GenerationConfig(
    max_new_tokens=50,       # Limits the response length
    temperature=0.7,         # Controls randomness
    top_p=0.9,               # Controls diversity
    do_sample=True,          # Enables sampling
    eos_token_id=tokenizer.eos_token_id
)
# Add a padding token to the tokenizer
tokenizer.add_special_tokens({'pad_token': '[PAD]'})
# Resize the model embeddings to accommodate the new token
model.resize_token_embeddings(len(tokenizer))
# Prepare input tokens
input_data = tokenizer.encode_plus(
```

```
    transcribed_text,
    return_tensors="pt",
    padding='max_length',  # Padding strategy
    max_length=512,  # Adjust this according to your input size
    truncation=True  # Ensure input is within max_length
)
# Get input_ids and attention_mask
input_ids = input_data['input_ids']
attention_mask = input_data['attention_mask']
# Generate response
outputs = model.generate(
    input_ids=input_ids,
    attention_mask=attention_mask,  # Provide attention mask
    generation_config=generation_config
)
# Decode and process the output
generated_text = tokenizer.decode(outputs[0], skip_special_tokens=True)
print(generated_text)
```

```
import re
def limit_to_two_sentences(text):
    # Use regex to split text into sentences
    sentences = re.split(r'(?<=[.!?]) +', text)
    # Return first two sentences joined together
    return ' '.join(sentences[:2])
# Apply the function to generated text
final_response = limit_to_two_sentences(generated_text)
print("LLM Response:", final_response)
```

```
"""# **Step-3 Generated Text to Speech**"""
```

```
!pip install edge-tts
import edge_tts
import asyncio
```

```
async def text_to_speech(text, output_file, voice="en-US-JennyNeural"):
    """
    Convert text to speech and save it as an audio file.
    Parameters:
    - text: The text to be converted to speech.
    - output_file: The output file path (e.g., 'output_audio.mp3').
    - voice: The voice to use (e.g., "en-US-JennyNeural").
    - pitch: The pitch of the voice (e.g., "0%", "-20%", "10%").
    - rate: The speed of the speech (e.g., "0%", "-20%", "10%").
    """
    # Create an Edge TTS instance
    communicate = edge_tts.Communicate(text, voice)
    # Set pitch and rate
    await communicate.save(output_file)
```

```
# Example usage
generated_text = "Today in New York, the weather is sunny with a high
of 75 degrees Fahrenheit."
# Set the voice parameters
voice = "en-US-JennyNeural"  # You can choose different voices like
"en-US-GuyNeural" for male voice
#pitch = "0%"  # Adjust the pitch (e.g., "-20%", "10%")
#rate = "0%"   # Adjust the speed (e.g., "-20%", "10%")
# Run the text-to-speech conversion
output_file = "/content/drive/MyDrive/Colab Notebooks/Lizomotors-Design
an End-to-End AI Voice Assistance Pipeline/output_audio.mp3"
await text_to_speech(generated_text, output_file, voice)
print(f"Generated speech has been saved to {output_file}")
```