End-to-End AI Voice Assistance

Introduction

The Speech-to-Speech AI Assistant project aims to create an end-to-end pipeline capable of understanding spoken queries, processing them, and responding with synthesized speech. This assistant is designed to be highly interactive, allowing users to engage with it naturally through voice commands. The system leverages advanced speech recognition, natural language processing (NLP), and text-to-speech (TTS) technologies to achieve seamless communication.

The pipeline's workflow starts with voice activity detection (VAD), which determines when a user is speaking. The detected voice input is then converted to text using automatic speech recognition (ASR). A large language model (LLM) processes the transcribed text to generate an appropriate response. Finally, the system converts this response back into speech using TTS technology, delivering an audible answer to the user.

The project utilizes a variety of state-of-the-art models and libraries, including models for ASR and LLMs such as Mistral and NVIDIA-based solutions, as well as custom modules for voice synthesis, ensuring high performance and flexibility.

Pipeline Architecture

The AI assistant pipeline consists of the following major components:

1. **Voice Activity Detection (VAD):** Detects when the user starts and stops speaking.
2. **Speech-to-Text (STT):** Converts the user’s spoken input into text using a pre-trained model.
3. **Language Model (LLM):** Processes the text input and generates a response using a pre-trained language model.
4. **Text-to-Speech (TTS):** Converts the generated text response back into speech for output.

Libraries and Tools Used

The project leverages various state-of-the-art machine learning models and tools to implement a voice-to-voice AI assistant. The major components include:

1. **Speech-to-Text (ASR)**: Converts spoken input into text using Google's Speech Recognition API.

2. **Language Model (LLM)**: Processes the transcribed text to generate a coherent response using the google/gemma-2-2b-it model.

3. **Text-to-Speech (TTS)**: Converts the generated response back into speech using parler-tts/parler-tts-large-v1 for natural-sounding voice synthesis with custom voice parameters.

ASR Implementation

```python
import speech_recognition as sr
import sounddevice as sd
import scipy.io.wavfile as wav

# Recording parameters
DURATION = 5  # seconds
SAMPLE_RATE = 16000

def record_audio(filename):
    print("Recording...")
    audio = sd.rec(int(DURATION * SAMPLE_RATE), samplerate=SAMPLE_RATE, channels=1, dtype='int16')
    sd.wait()
    print("Recording finished.")
    wav.write(filename, SAMPLE_RATE, audio)
    print(f"Audio saved as {filename}")

def transcribe_audio(filename):
    recognizer = sr.Recognizer()
    with sr.AudioFile(filename) as source:
        audio = recognizer.record(source)
        print("Transcribing...")
        try:
            text = recognizer.recognize_google(audio)
            print("Transcription complete.")
            return text
        except sr.UnknownValueError:
            return "Google Speech Recognition could not understand audio"
        except sr.RequestError as e:
            return f"Could not request results from Google Speech Recognition service; {e}"

def main():
    audio_file = 'recorded_audio.wav'
    record_audio(audio_file)
    transcription = transcribe_audio(audio_file)
    print(f"Transcribed text: {transcription}")

if __name__ == "__main__":
    main()
```

LLM Response Generation

```python
import torch
from transformers import pipeline

pipe = pipeline(
    "text-generation",
    model="google/gemma-2-2b-it",
    model_kwargs={"torch_dtype": torch.bfloat16},
    device="cuda",
)

messages = [
    {"role": "user", "content": "what is the capital of India."},
]

outputs = pipe(messages, max_new_tokens=256)
assistant_response = outputs[0]["generated_text"][-1]["content"].strip()
print(assistant_response)
```

TTS Implementation (using Parler TTS)

```python
import torch
from parler_tts import ParlerTTSForConditionalGeneration
from transformers import AutoTokenizer
import soundfile as sf

device = "cuda:0" if torch.cuda.is_available() else "cpu"

model = ParlerTTSForConditionalGeneration.from_pretrained("parler-tts/parler-tts-large-v1").to(device)
tokenizer = AutoTokenizer.from_pretrained("parler-tts/parler-tts-large-v1")

prompt = "Hey, how are you doing today?"
description = "A female speaker delivers a slightly expressive and animated speech with a moderate speed and pitch."

input_ids = tokenizer(description, return_tensors="pt").input_ids.to(device)
prompt_input_ids = tokenizer(prompt, return_tensors="pt").input_ids.to(device)

generation = model.generate(input_ids=input_ids, prompt_input_ids=prompt_input_ids)
audio_arr = generation.cpu().numpy().squeeze()
sf.write("parler_tts_out.wav", audio_arr, model.config.sampling_rate)

```

Test audio

recorded_audio.wav

```
(venv) D:\AIVA2>python -u "d:\AIVA2\record_and_transcribe.py"
Recording...
Recording finished.
Audio saved as recorded_audio.wav
Transcribing...
Transcription complete.
Transcribed text: what is the capital of France
```