End-to-End AI Voice Assistance Pipeline

Overview

This pipeline integrates several components to create an AI voice assistant. The system records audio input, transcribes it into text, generates a relevant response using a pre-trained language model, and converts the text response back into speech. Each component is modularized into separate Python files for clarity and maintainability.

Components

1. **Audio Recording and Saving** (transcribe.py)
2. **Text Transcription** (transcribe.py)
3. **Text Response Generation** (text_to_response.py)
4. **Text-to-Speech Conversion** (text_to_speech.py)
5. **Main Execution** (main.py)

Detailed Implementation

1. Audio Recording and Saving (transcribe.py)

- **record_audio(duration=5, fs=16000)**:
  - Records audio from the microphone for a specified duration (5 seconds by default) and sample rate (16000 Hz by default).
  - Uses sounddevice for audio recording.
  - Returns the recorded audio as a flattened NumPy array.
- **save_audio_as_wav(audio, filename="temp.wav", fs=16000)**:
  - Saves the recorded audio data as a WAV file using soundfile.
- **apply_vad(audio, fs=16000, vad_mode=3)**:
  - Applies Voice Activity Detection (VAD) to filter out non-speech parts of the audio.
  - Uses webrtcvad to determine speech frames.

- Converts audio frames to int16 for VAD processing and concatenates voiced frames.
- Returns the VAD-processed audio.

- **transcribe_audio(file_path)**:
  - Transcribes the given audio file to text using the Whisper model (tiny.en).
  - Uses soundfile to read the audio file and whisper for transcription.
  - Returns the transcribed text.

2. Text Response Generation (text_to_response.py)

- **generate_response(query)**:
  - Generates a relevant response using a pre-trained language model (gpt2).
  - Loads the model and tokenizer using transformers.
  - Uses pipeline for text generation.
  - Constructs a prompt with the input query and generates a response.
  - Truncates the response to the first two sentences.
  - Returns the generated response.

3. Text-to-Speech Conversion (text_to_speech.py)

- **text_to_speech(text, output_file="output.wav", pitch=1.0, speed=1.0, gender='female')**:
  - Converts text to speech using SpeechT5Processor and SpeechT5ForTextToSpeech from the transformers library.
  - Uses SpeechT5HifiGan as the vocoder for high-quality speech synthesis.
  - Loads speaker embeddings from Matthijs/cmu-arctic-xvectors.
  - Adjusts pitch and speed of the generated audio.
  - Saves the output audio to a WAV file.

4. Main Execution (main.py)

- **main()**:
  - **Step 1: Record Audio**: Calls record_audio() and save_audio_as_wav() to capture and save audio.
  - **Step 2: Transcribe Audio**: Calls transcribe_audio() to convert the audio to text.

- **Step 3: Generate Response**: Calls generate_response() to produce a relevant text response.
- **Step 4: Convert Text to Speech**: Calls text_to_speech() to generate speech from the text response.
- Handles exceptions throughout the pipeline to ensure robustness.
- Measures and prints the total time taken for execution.

Choice of Models and Libraries

- **Audio Recording and Processing**:
  - sounddevice: For recording audio.
  - soundfile: For saving and loading WAV files.
  - webrtcvad: For voice activity detection.
  - whisper: For transcribing audio to text.
- **Text Generation**:
  - transformers library by Hugging Face.
  - **Model**: gpt2 (smaller variant of GPT-2 for lightweight processing).
- **Text-to-Speech**:
  - transformers library by Hugging Face.
  - **Models**:
    - SpeechT5Processor and SpeechT5ForTextToSpeech for generating speech.
    - SpeechT5HifiGan for high-quality vocoding.
  - **Speaker Embeddings**: Utilizes pre-trained embeddings from Matthijs/cmu-arctic-xvectors.

Parameters Used

- **Audio**:
  - Sample rate: 16000 Hz
  - Duration: 5 seconds (can be adjusted as needed)

- **Text Generation**:
  - max_length=100: Limits the length of generated text.
  - num_beams=5: Uses beam search for better text generation.
  - temperature=0.5: Controls randomness in text generation.
  - truncation=True: Ensures responses are truncated to the max_length.
- **Text-to-Speech**:
  - pitch=1.0: Default pitch (can be adjusted).
  - speed=1.0: Default speed (can be adjusted).
  - gender='female': Selects female speaker embeddings (can be changed).

Conclusion

This E2E pipeline efficiently integrates various models and libraries to create a functional AI voice assistant. The modular approach allows for easy updates and improvements to individual components. Ensure to test each component separately and verify that the models and libraries are correctly installed and configured.

CODE SNIPPETS

1. Audio Recording and Saving (transcribe.py)

Recording Audio:

```
import sounddevice as sd

import numpy as np

def record_audio(duration=5, fs=16000):

    print("Recording...")

    recording = sd.rec(int(duration * fs), samplerate=fs, channels=1, dtype=np.float32)

    sd.wait()

    print("Recording complete.")
```

```
    return recording.flatten()
```

Saving Audio as WAV:

```
import soundfile as sf

def save_audio_as_wav(audio, filename="temp.wav", fs=16000):
    sf.write(filename, audio, fs)
```

Applying Voice Activity Detection (VAD):

```
import webrtcvad
import numpy as np

def apply_vad(audio, fs=16000, vad_mode=3):
    vad = webrtcvad.Vad(vad_mode)
    audio_int16 = (audio * 32767).astype(np.int16)
    frame_duration_ms = 20
    frame_length = int(fs * frame_duration_ms / 1000)
    voiced_frames = []
    for start in range(0, len(audio_int16), frame_length):
        end = start + frame_length
        frame = audio_int16[start:end]
        if len(frame) < frame_length:
            frame = np.pad(frame, (0, frame_length - len(frame)), mode='constant',
constant_values=0)
        if vad.is_speech(frame.tobytes(), fs):
            voiced_frames.append(frame)
```

```
    if voiced_frames:
        voiced_audio = np.concatenate(voiced_frames)
    else:
        voiced_audio = np.array([], dtype=np.float32)
    voiced_audio = voiced_audio.astype(np.float32) / 32767
    return voiced_audio
```

Transcribing Audio with Whisper:

```
import whisper
import soundfile as sf
model = whisper.load_model("tiny.en")
def transcribe_audio(file_path):
    try:
        audio, _ = sf.read(file_path, dtype='float32')
        audio = apply_vad(audio)
        result = model.transcribe(audio)
        return result["text"]
    except Exception as e:
        print(f"Error transcribing audio: {e}")
        return ""
```

2. Text Response Generation (text_to_response.py)

Generating Response with GPT-2:

```
from transformers import AutoTokenizer, AutoModelForCausalLM, pipeline

def generate_response(query):

    try:

        model_name = "gpt2"

        tokenizer = AutoTokenizer.from_pretrained(model_name)

        model = AutoModelForCausalLM.from_pretrained(model_name)

        generator = pipeline("text-generation", model=model, tokenizer=tokenizer)

        prompt = f"Q: {query}\nA:"

        response = generator(

            prompt,

            max_length=100,

            num_return_sequences=1,

            pad_token_id=tokenizer.eos_token_id,

            no_repeat_ngram_size=2,

            temperature=0.5,

            num_beams=5,

            truncation=True

        )[0]['generated_text']

        sentences = response.split('. ')

        if len(sentences) > 1:

            response = '. '.join(sentences[:2]).strip() + ('.' if response.endswith('.') else '')

        else:
```

```
            response = response.strip()
        return response
    except Exception as e:
        print(f"An error occurred: {e}")
        return ""
```

3. Text-to-Speech Conversion (text_to_speech.py)

Converting Text to Speech:

```
import numpy as np
from transformers import SpeechT5Processor, SpeechT5ForTextToSpeech,
SpeechT5HifiGan
import torch
import soundfile as sf
from datasets import load_dataset
def text_to_speech(text, output_file="output.wav", pitch=1.0, speed=1.0, gender='female'):
    try:
        processor = SpeechT5Processor.from_pretrained("microsoft/speecht5_tts")
        model = SpeechT5ForTextToSpeech.from_pretrained("microsoft/speecht5_tts")
        vocoder = SpeechT5HifiGan.from_pretrained("microsoft/speecht5_hifigan")


        embeddings_dataset = load_dataset("Matthijs/cmu-arctic-xvectors", split="validation")
        speaker_embeddings =
torch.tensor(embeddings_dataset[7306]["xvector"]).unsqueeze(0)
        inputs = processor(text=text, return_tensors="pt")
```

```
        if gender == 'male':
            speaker_embeddings = torch.tensor(embeddings_dataset[0]["xvector"]).unsqueeze(0)
        with torch.no_grad():
            speech = model.generate_speech(inputs["input_ids"],
speaker_embeddings=speaker_embeddings, vocoder=vocoder)
        audio = speech.squeeze().cpu().numpy()
        audio = audio * pitch
        audio = np.interp(np.arange(0, len(audio), speed), np.arange(0, len(audio)), audio)
        sf.write(output_file, audio, 22050)
        print(f"Audio saved to {output_file}")
    except Exception as e:
        print(f"An error occurred during TTS conversion: {e}")
```

4. Main Execution (main.py)

Main Pipeline Execution:

```
import sys
import os
import warnings
import time
warnings.filterwarnings("ignore")
sys.path.append(os.path.dirname(os.path.abspath(__file__)))
from transcribe import record_audio, save_audio_as_wav, transcribe_audio
from text_to_response import generate_response
from text_to_speech import text_to_speech
```

```
def main():
    try:
        audio = record_audio()
        save_audio_as_wav(audio)
        text_output = transcribe_audio("temp.wav")
        print("Transcribed Text:", text_output)
        with open("transcribed_text.txt", "w") as f:
            f.write(text_output)
        response = generate_response(text_output)
        print("Response:", response)
        with open("response_text.txt", "w") as f:
            f.write(response)
        text_to_speech(response, "response.wav")
    except Exception as e:
        print(f"An error occurred in the main pipeline: {e}")
if __name__ == "__main__":
    start_time = time.time()
    main()
    end_time = time.time()
    print(f"Total time taken (latency): {end_time - start_time} seconds")
```