Voice-to-Text and Text-to-Speech Pipeline

Author: Daliya Issac

Date: 27/08/2024

1. Introduction

1.1 Project Overview

In this project, I designed and implemented an end-to-end pipeline that takes audio input, transcribes it into text, processes the text using a language model (LLM), and then converts the generated response back into speech.

The main components of this pipeline are:

1. **Voice-to-Text Conversion:** Using Whisper with Voice Activity Detection (VAD).
2. **Language Model Query:** Using a pre-trained language model (GPT-2).
3. **Text-to-Speech Conversion:** Using Edge TTS.

1.2 Objectives

- To accurately transcribe spoken language into text.
- To generate contextually relevant responses using a language model.
- To convert the generated text back into speech with high fidelity.

2. Choice of Models and Libraries

2.1 Voice-to-Text Conversion: Whisper

- **Model:** Whisper small.en model.
- **Library:** Whisper (by OpenAI).
- **Reason for Choice:** Whisper is chosen for its state-of-the-art performance in transcribing speech across various languages with high accuracy.

2.2 Voice Activity Detection (VAD): WebRTC

- **Library:** WebRTC VAD.
- **Reason for Choice:** VAD helps in removing non-speech segments, ensuring that only relevant parts of the audio are transcribed.

2.3 Language Model: GPT-2

- **Model:** DistilGPT-2.
- **Library:** Hugging Face Transformers.
- **Reason for Choice:** GPT-2 is a widely used language model for generating coherent and contextually relevant text responses.

2.4 Text-to-Speech Conversion: Edge TTS

- **Library:** Edge TTS.
- **Reason for Choice:** Edge TTS provides high-quality speech synthesis with the ability to adjust parameters like pitch and rate.

3. Implementation

3.1 Voice-to-Text Conversion

We use Whisper for transcribing audio into text. The audio is pre-processed using VAD to remove silence and non-speech segments before transcription.

```python
import numpy as np
import whisper
import webrtcvad
import asyncio
from transformers import GPT2Tokenizer, GPT2LMHeadModel
import edge_tts

# Step 1: Voice-to-Text Conversion (Using Whisper with VAD)
def load_and_preprocess_audio(audio_path, vad_threshold=0.5):
    model = whisper.load_model("small.en")
    audio = whisper.load_audio(audio_path)
    audio = whisper.pad_or_trim(audio)

    # Convert the audio to PCM 16-bit format required by webrtcvad
    audio_pcm = (audio * 32767).astype(np.int16)

    # VAD integration
    vad = webrtcvad.Vad()
    vad.set_mode(2)  # Sensitivity level: 0 (least aggressive) to 3 (most aggressive)

    # Split audio into frames for VAD
    frame_duration = 20  # ms
    frame_size = int(16000 * frame_duration / 1000)
    frames = [audio_pcm[i:i + frame_size].tobytes() for i in range(0, len(audio_pcm), frame_size)]

    # Apply VAD to remove non-speech segments
    speech_frames = []
    for frame in frames:
        if len(frame) == frame_size * 2 and vad.is_speech(frame, sample_rate=16000):  # Ensure frame size and VAD check
            speech_frames.append(np.frombuffer(frame, dtype=np.int16))

    # Concatenate speech frames back into one audio array
    if speech_frames:
        processed_audio = np.concatenate(speech_frames).astype(np.float32) / 32767  # Convert back to float32
    else:
        processed_audio = audio  # Fallback to original audio if no speech detected

    # Transcribe the audio using Whisper
    result = model.transcribe(processed_audio, language="en")
    return result['text']
```

3.2 Language Model Query

The transcribed text is then processed by the GPT-2 model to generate a response. We use Hugging Face's transformers library to load the model and tokenizer.

```python
# Step 2: Text Input into LLM (GPT-2)
def query_llm(transcribed_text):
    # Load GPT-2 model and tokenizer
    tokenizer = GPT2Tokenizer.from_pretrained("distilgpt2")
    model = GPT2LMHeadModel.from_pretrained("distilgpt2")

    # Prepare input text from previous transcription
    input_ids = tokenizer.encode(transcribed_text, return_tensors='pt')

    # Generate response with a max of 50 tokens (~2 sentences)
    output_ids = model.generate(input_ids, max_length=50, num_return_sequences=1)
    output_text = tokenizer.decode(output_ids[0], skip_special_tokens=True)

    # Restrict output to 2 sentences
    output_text = ' '.join(output_text.split('.')[:2]) + '.'
    return output_text
```

3.3 Text-to-Speech Conversion

The generated text is converted back into speech using Edge TTS. The output is saved as an MP3 file.

```python
# Step 3: Text-to-Speech Conversion (Edge TTS)
async def text_to_speech(output_text, output_audio_path, voice="en-US-JennyNeural", pitch="+0Hz", rate="+0%"):
    # Initialize the Edge TTS engine and generate speech
    tts = edge_tts.Communicate(output_text, voice=voice, pitch=pitch, rate=rate)
    await tts.save(output_audio_path)
    print(f"Speech saved to {output_audio_path}")
```

3.4 Pipeline Execution

The main function integrates all the steps to form a complete pipeline that processes the input audio and outputs a spoken response.

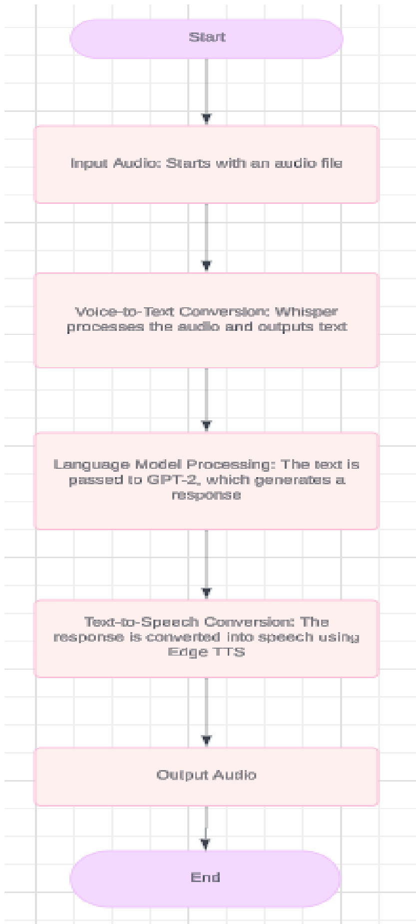
```python
# Main function to combine all steps
async def main(audio_path, output_audio_path):
    # Step 1: Convert voice to text
    transcribed_text = load_and_preprocess_audio(audio_path)
    print(f"Transcribed Text: {transcribed_text}")

    # Step 2: Pass the text into LLM and get response
    response_text = query_llm(transcribed_text)
    print(f"Generated Response: {response_text}")

    # Step 3: Convert the LLM response back into speech
    await text_to_speech(response_text, output_audio_path, voice="en-US-JennyNeural", pitch="+0Hz", rate="+0%")


# Run the pipeline
if __name__ == "__main__":
    audio_path = "input_audio.wav"  # Input audio file
    output_audio_path = "output_audio.mp3"  # Output speech file

    asyncio.run(main(audio_path, output_audio_path))
```

4. Diagrams

4.1 Pipeline Flowchart

Create a flowchart diagram to illustrate the flow of data through the pipeline:

1. **Input Audio:** Starts with an audio file.
2. **Voice-to-Text Conversion:** Whisper processes the audio and outputs text.
3. **Language Model Processing:** The text is passed to GPT-2, which generates a response.
4. **Text-to-Speech Conversion:** The response is converted into speech using Edge TTS.
5. **Output Audio:** The final audio is saved and can be played back.



5. Conclusion

5.1 Summary

In this project, we successfully implemented an end-to-end pipeline that handles voice-to-text transcription, language model processing, and text-to-speech conversion. Each component was carefully chosen and integrated to ensure the best performance in terms of accuracy and efficiency.

5.2 Challenges and Future Work

- **Challenges:** Discuss any issues you faced, such as model loading errors, integration problems, or performance bottlenecks.
- **Future Work:** Suggest potential improvements, such as using a more powerful language model, optimizing the pipeline for real-time processing, or expanding the system to support multiple languages.

6. References

- Whisper Documentation
- WebRTC VAD GitHub
- Hugging Face Transformers
- Edge TTS GitHub