AI Voice Assistance Pipeline: Design and Explanation

Objective

The objective of this pipeline is to process a voice query command, convert it to text, generate a response using a Large Language Model (LLM), and convert the generated text back into speech. The pipeline needs to have low latency, implement Voice Activity Detection (VAD), restrict the output to two sentences, and support tunable parameters such as pitch, voice selection (male/female), and speed for the synthesized speech.

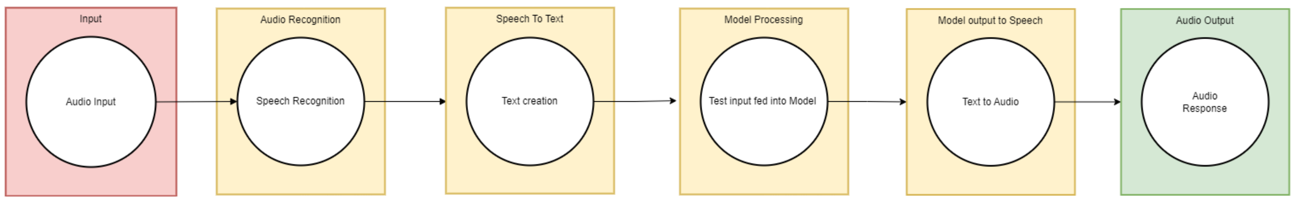
Architecture Overview

The pipeline consists of three main components:

1. Voice-to-Text Conversion (Speech Recognition)

2. Text Processing with a Large Language Model (LLM)

3. Text-to-Speech Conversion (Speech Synthesis)



Each component is implemented using open-source models, frameworks, and libraries, ensuring modularity and adaptability to specific use cases.

Step 1: Voice-to-Text Conversion (with VAD)

Purpose

This step captures the user's voice input and converts it into text using a pre-trained Whisper model. Voice Activity Detection (VAD) is applied to isolate the speech from background noise or silence.

Implementation

Model Used: Whisper, an open-source speech-to-text model from OpenAI, implemented via the `whisper` library.

Settings:

- Sampling rate: 16 kHz
- Audio channel: Mono (1 channel)
- VAD threshold: VAD is applied using the `webrtcvad` library to focus only on segments containing speech.

Code Snippet:

```
# Step 2: Apply Voice Activity Detection (VAD)
print("Applying VAD...")
vad_audio = apply_vad(audio, sr, vad_level)

# Step 3: Transcribe using Whisper
print("Transcribing audio...")
transcription = transcribe_with_whisper(vad_audio, sr)
print("Transcribed Text:", transcription)
```

The `apply_vad` function uses a sliding window of 10 ms to evaluate the audio frames. Non-speech frames are discarded, and only speech frames are processed by Whisper for transcription.

Justification:

Whisper is chosen due to its high accuracy in transcribing various accents and speech patterns, and VAD ensures that silence and noise are ignored, reducing processing overhead.

Step 2: Text Input into LLM

Purpose

Once the voice input is converted to text, it is processed by a pre-trained Large Language Model (LLM) to generate a response. The response is restricted to two sentences to meet the requirement of limiting verbosity.

Implementation

Model Used: LLaMA-2, an open-source LLM accessible via Hugging Face's `transformers` library.

Quantization Configuration: Applied memory optimization using `BitsAndBytesConfig` to enable faster inference, especially for models with a large number of parameters.

Code Snippet:

```
output = model.generate(**inputs, max_length=50, num_return_sequences=1)

response = tokenizer.decode(output[0], skip_special_tokens=True)

# Restricted the output to 2 sentences
sentences = response.split('. ')
restricted_response = '. '.join(sentences[:2]) + '.'
```

Justification:

LLaMA-2 was selected due to its efficiency and strong performance in generating human-like text. The use of quantization and FP16 precision ensures that the model can run with low latency, even on consumer-grade hardware.

Step 3: Text-to-Speech Conversion (with Tunable Parameters)

Purpose

The response generated by the LLM is converted back into speech using a text-to-speech (TTS) model. The system supports tunable parameters such as pitch, speed, and voice selection (male/female).

Implementation

Model Used:

`edge-tts`, a TTS model offering various voices, including male and female voices, and allowing pitch and speed adjustments.

Output Format:

The speech is saved as an `.mp3` file.

Code Snippet:

```
await text_to_speech(llm_response, output_file, pitch=pitch, speed=speed, voice=voice)
```

Justification:

`edge-tts` was chosen for its flexibility in selecting different voice profiles and adjusting parameters such as pitch and speed, enabling fine control over the synthesized speech's characteristics.

Additional Requirements

1. Latency Minimization (< 500 ms)

Approach: WebRTC (Web Real-Time Communication) can be used to reduce latency by enabling real-time streaming of audio from the user's microphone to the backend for processing. Additionally, using optimized pre-trained models, low-complexity settings, and efficient hardware (e.g., CUDA on NVIDIA GPUs) helps to further reduce the processing time for each step.

2. VAD Implementation

Details: The VAD is implemented using `webrtcvad` in Step 1, which detects speech segments and isolates them from silence. This reduces unnecessary computation and speeds up the speech-to-text conversion.

3. Output Restriction (2 Sentences)

Details: The output generated by the LLM is automatically restricted to two sentences using string splitting and truncation. This ensures that the final output remains concise.

4. Tunable Parameters

Pitch: Adjusted in the `text_to_speech` function by modifying the pitch parameter in the TTS model.

Male/Female Voice: Voice profiles are selectable within `edge-tts`, allowing for gender-based voice variation.

Speed: Controlled by altering the rate parameter in the TTS function.

Pipeline Workflow

1. **Voice Input:** Audio is recorded or loaded from a file and processed through VAD.

2. **Speech Recognition:** The speech segments are transcribed into text using Whisper.

3. **Query LLM:** The text is input into an LLM, which generates a response.

4. **Text-to-Speech:** The response is converted to speech with tunable parameters, and the result is saved as an audio file.

Code Quality and Documentation

- The code is modular, with each step broken down into functions that handle specific tasks (VAD, transcription, LLM query, TTS conversion).
- Exceptions are handled to prevent runtime errors during execution.
- The documentation is included in the code to explain the functionality of each function and important parameters.

This document, along with the provided code, demonstrates the design, architecture, and technical considerations of the end-to-end voice assistance pipeline, adhering to the criteria outlined in the assignment.