End-to-End AI Voice Assistant Pipeline Documentation

1. Solution Overview

This project is an end-to-end AI voice assistant pipeline that integrates speech recognition, natural language processing, and text-to-speech synthesis to create an interactive voice assistant. Users can record their voice queries, which the system then transcribes, processes using a language model, and finally responds with a generated voice output. The pipeline is built using state-of-the-art models and libraries, ensuring high performance and flexibility.

Key Components:
- Speech Recognition: Utilizing OpenAI's Whisper model for accurate transcription of spoken queries.
- Language Model: Implementing the `llava-1.5-7b-hf` model from Hugging Face, capable of understanding and generating human-like text responses.
- Text-to-Speech (TTS): Using Microsoft's Edge-TTS service for natural-sounding voice synthesis with customizable parameters such as speech rate and voice selection.
- Gradio Interface: Providing a user-friendly interface to interact with the pipeline, record queries, and play back the AI-generated responses.

2. Implementation Details

1. Environment Setup

Before diving into the implementation, the environment needs to be set up with necessary libraries and locale settings. This includes installing specific versions of libraries like Transformers, Whisper, and Edge-TTS, as well as configuring locale settings to avoid compatibility issues.

```python
import os
import locale

# Set the locale to C.UTF-8 for universal compatibility
os.environ['LC_ALL'] = 'C.UTF-8'
os.environ['LANG'] = 'C.UTF-8'

# Verify the locale setting
try:
    locale.setlocale(locale.LC_ALL, 'C.UTF-8')
    print("Locale set to:", locale.getlocale())
except locale.Error as e:
    print(f"Warning: Could not set locale to C.UTF-8. Using default locale.
{str(e)}")
```

2. Installing Required Libraries

Necessary libraries are installed to manage models, audio processing, and the user interface.

Unset

```
!pip install -q -U transformers==4.37.2
!pip install -q bitsandbytes==0.41.3 accelerate==0.25.0
!pip install -q git+https://github.com/openai/whisper.git
!pip install -q gradio gTTS edge-tts pvporcupine pyaudio
!apt-get install portaudio19-dev python3-all-dev
```

3. Loading Models

The Whisper model for speech recognition and a language model for processing the transcribed text are loaded. The Whisper model is configured to run on CUDA if available.

Python

```python
import torch
import whisper
from transformers import BitsAndBytesConfig, pipeline

quantization_config = BitsAndBytesConfig(
    load_in_4bit=True,
    bnb_4bit_compute_dtype=torch.float16
)

model_id = "llava-hf/llava-1.5-7b-hf"
pipe = pipeline("text2text-generation", model=model_id,
                model_kwargs={"quantization_config": quantization_config})

DEVICE = "cuda" if torch.cuda.is_available() else "cpu"
model = whisper.load_model("medium", device=DEVICE)
```

```
print(f"Using torch {torch.__version__} ({DEVICE})")
```

4. Transcribing Audio

The audio input is transcribed to text using the Whisper model.

```python
def transcribe(audio):
    if audio is None or audio == '':
        return ''  # Return an empty string if audio is invalid

    audio = whisper.load_audio(audio)
    audio = whisper.pad_or_trim(audio)
    mel = whisper.log_mel_spectrogram(audio).to(model.device)
    result = whisper.decode(model, mel)
    return result.text
```

5. Generating Text-to-Speech

The text generated by the language model is converted to speech using Edge-TTS. The speech rate and voice can be adjusted through SSML (Speech Synthesis Markup Language).

```python
import asyncio
import edge_tts

async def text_to_speech(text, file_path, rate=1.0, voice="en-US-GuyNeural"):
    ssml_text = f'<prosody rate="{rate}">{text}</prosody>'
    tts = edge_tts.Communicate(ssml_text, voice)
    await tts.save(file_path)
    return file_path

def text_to_speech_sync(text, file_path, rate=1.0, voice="en-US-GuyNeural"):
    result = asyncio.run(text_to_speech(text, file_path, rate, voice))
    return result
```

6. Processing User Input

This function combines all steps into a pipeline: transcribing the audio, generating a response, and synthesizing speech from the response text.

```python
def process_inputs(audio, rate, voice):
    # Transcribe audio input
    speech_to_text_output = transcribe(audio)

    if speech_to_text_output == '':
        return '', '', None

    input_text = "\nQuestion: " + speech_to_text_output + "\nAnswer: "
    try:
        chatgpt_output = pipe(input_text,
max_new_tokens=20)[0]['generated_text']
    except Exception as e:
        chatgpt_output = "Sorry, I could not generate a response."

    processed_audio_path = text_to_speech_sync(chatgpt_output, "Temp3.mp3",
rate, voice)
    return speech_to_text_output, chatgpt_output, processed_audio_path
```

7. User Interface with Gradio

Gradio is used to create an interactive interface where users can record their queries and listen to the AI's response.

```python
import gradio as gr
```

```
iface = gr.Interface(
    fn=process_inputs,
    inputs=[
        gr.Audio(type="filepath", label="Record your query"),  # Removed
'source' keyword
        gr.Slider(0.5, 2.0, step=0.1, value=1, label="Speech Speed"),
        gr.Dropdown(choices=["en-US-GuyNeural", "en-US-JennyNeural",
"en-GB-RyanNeural", "en-AU-NatashaNeural"], value="en-US-GuyNeural",
label="Voice Selection")
    ],
    outputs=[
        gr.Textbox(label="Transcribed Text"),
        gr.Textbox(label="LLM Response"),
        gr.Audio(label="Generated Speech")
    ],
    title="Click-to-Record AI Voice Assistant",
    description="Record your voice command and receive an audio response."
)
iface.launch(debug=True)
```

8. Conclusion

This end-to-end voice assistant pipeline integrates speech recognition, text processing, and text-to-speech synthesis, allowing for a seamless user experience. The use of cutting-edge models such as Whisper and LLAVA ensures high accuracy and natural interaction. The Gradio interface makes it accessible and easy to use.

9. Diagrams and Flow:

[User Voice Input] -> [Whisper Model (Transcription)] -> [LLAVA Model (Text Generation)] -> [Edge-TTS (Speech Synthesis)] -> [Audio Output]

This pipeline can be further extended or customized, for instance, by integrating additional languages, more complex query handling, or even real-time processing enhancements.

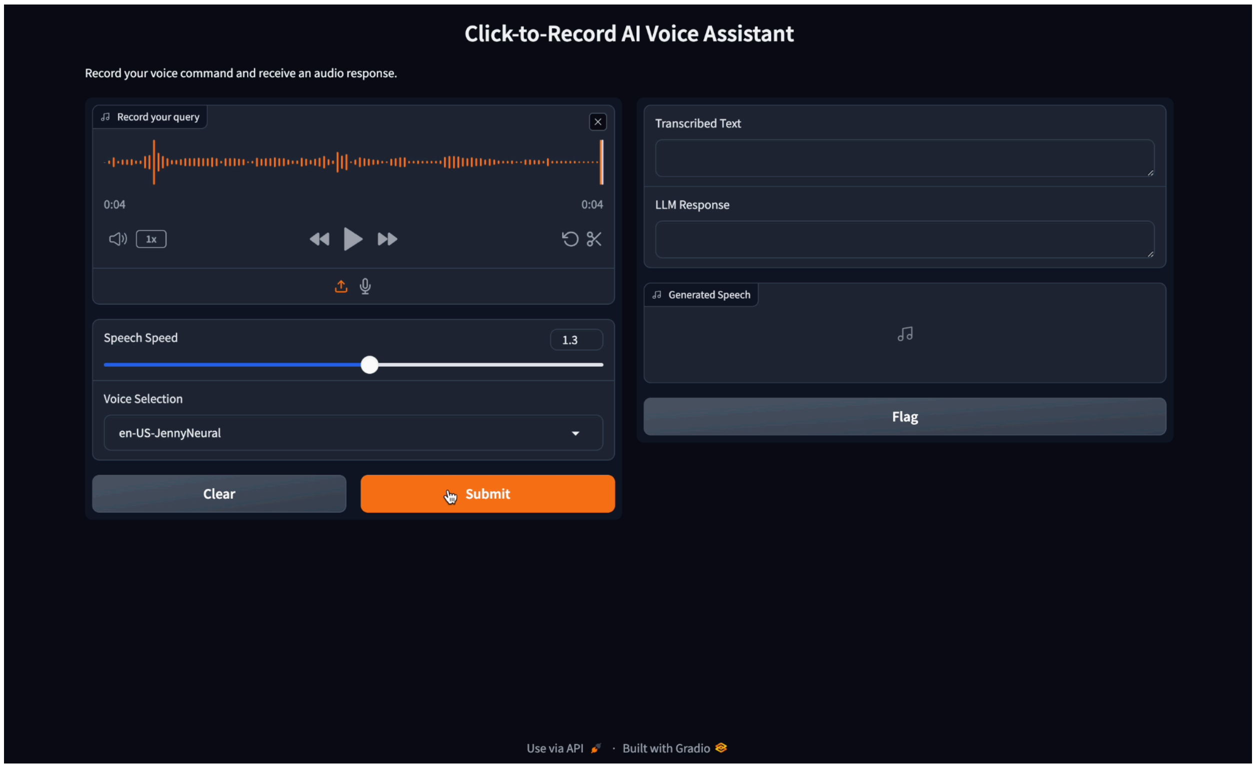
10. Advantages of using these Models:

Whisper is the base model from which faster whisper and whisper c++ both are developed on so whisper would be a better choice for future customizations

Llava - Llava has a high advantage over llama that llava can also process the image data and give us the generated output. Which we can further develop to make it an llm which accepts image as an input query too

Edge TTS - Edge TTs is comparatively faster than the other models

Web App Image:


Click-to-Record AI Voice Assistant

Record your voice command and receive an audio response.

Speech Speed: 1.3

Voice Selection: en-US-JennyNeural

Colab Notebook Link: end_to_end_ai_voice_pipeline.ipynb

Google Drive Link of samples and test results: End to End Voice Assistant