Documentation on Real-Time Speech-to-Text and Llama3.1 Integration

Linux (Ubuntu 24.04) seem to have problem with the microphone the device, Windows operating system have been used for the development of this working prototype. The prototype utilizes the GPU and CPU capability to produce the seamless performance of the system.

The real-time transcribtion is made possible through the open-source GitHub package named RealTimeSTT. This package provided easy-to-use, low-latency and GPU enabled speech-to-text library for real time applications. It listens to the microphone and transcribes voice into text. This package is ideal for use-cases such as voice assistants and the applications requiring fast and precise speech-to-text conversion.

The latest version of the package was v0.3.92 which provides the AudioToTextRecorder class whose instances can be used to initialize and transcribe the spoken words into the written text.

The installation procedure is pretty much straight forward with the cuda-enabled PyTorch as the dependency to be installed separately. The anaconda environment with Python 3.12 was used for the installation of PyTorch for GPU acceleration and the RealTimeSTT for transcribtion.

Commands:

```
conda create -n stt_env python=3.12
conda activate stt_env

conda install pytorch torchvision torchaudio pytorch-cuda=12.1 -
pip install RealtimeSTT
```

Use of dedicated earphone was required for the seamless transcribtion. The speech-to-text package have the features such as voice activity detection, real-time transcription and wake word activation. WebRTCVAD used for initial voice activity detection and SileroVAD for more accurate verification. Faster_Whisper is used instant and GPU-accelerated transcription. Porcupine or OpenWakeWord s used for wake word detection.

Python Script:

```
from RealtimeSTT import AudioToTextRecorder
import torch

# File to store the transcribed text
TRANSCRIBED_TEXT_FILE = "transcribed_text.txt"

def process_text(text):
    print(f"Recognized Text: {text}")
    # Save the transcribed text to a file
    with open(TRANSCRIBED_TEXT_FILE, "w") as file:
        file.write(text)

if __name__ == '__main__':
    try:
        # Check available models
        print("Available models: base, small, medium, large, lar

        # Check if CUDA is available
        cuda_available = torch.cuda.is_available()
        if cuda_available:
            cuda_device = torch.cuda.get_device_name(0)
            print(f"CUDA is available. Using device: {cuda_devic
            print(f"Memory Usage: {torch.cuda.memory_allocated(0
        else:
            print("CUDA is not available. Falling back to CPU.")

        print("Initializing recorder...")
```

```
        # Initialize the recorder with GPU support if available,
        device = "cuda" if cuda_available else "cpu"
        recorder = AudioToTextRecorder(device=device, model="bas
        print(f"Recorder initialized on {device}. Waiting for in

        # Start real-time transcription
        print("Speak now...")
        while True:
            recorder.text(process_text)

    except KeyboardInterrupt:
        print("\nProgram interrupted. Exiting gracefully...")

    except Exception as e:
        print(f"An error occurred: {e}")

    finally:
        try:
            if 'recorder' in locals() and recorder:
                recorder.stop()
                print("Recorder stopped.")
        except Exception as cleanup_error:
            print(f"Error during cleanup: {cleanup_error}")
```

Ollama Integration was made possible through the following Python script. The Llama 3.1 model of 8b parameter is used which respond to the transcribed text as the conversation with a bit high latency which need to be fixed in next iteration of development.

Python Script

```
import os
import time
from ollama import chat, ChatResponse
```

```
# Path to the transcribed text file
TRANSCRIBED_TEXT_FILE = "transcribed_text.txt"

def process_with_llama(prompt):
    """
    Uses the Ollama Llama model to process the given prompt.
    """
    try:
        # Query the Llama model
        response: ChatResponse = chat(model="llama3.1", messages
            {'role': 'user', 'content': prompt}
        ])
        return response.message.content
    except Exception as e:
        return f"Error querying Llama: {str(e)}"

def process_transcribed_text():
    """
    Continuously monitors the transcribed text file and processe
    """
    last_processed_text = None

    while True:
        # Check if the transcribed text file exists and has cont
        if os.path.exists(TRANSCRIBED_TEXT_FILE) and os.path.get
            with open(TRANSCRIBED_TEXT_FILE, "r") as file:
                transcribed_text = file.read().strip()

            # Only process if the text is new
            if transcribed_text != last_processed_text:
                print(f"New transcribed text detected: {transcri
                last_processed_text = transcribed_text

                # Generate Llama response
                print("Processing with Llama...")
```

```
                llama_response = process_with_llama(transcribed_
                print("Llama Response:")
                print(llama_response)

        time.sleep(1)

if __name__ == "__main__":
    print("Waiting for transcribed text...")
    process_transcribed_text()
```