# Lizmotors Assignment

**A detailed description of your solution, including the choice of models, libraries, and parameters used.**

Overview of Solution: The solution uses a Speech-to-Speech pipeline, taking in an audio input and producing synthesized speech output through several models and libraries. Preprocessing will involve the pipeline for voice activity detection, transcribing speech into text, generating a response as text with the use of a language model, and converting text back to speech.

**Key Components and Choices:**

- Voice Activity Detection (VAD):
  - Model Used: silero-vad
  - Purpose: Speech segments detection model in an audio file. Isolates them from silence or noise.
  - Libraries: torch.hub, snakers4/silero-vad repo
  - Parameters: ONNX model for faster inference and 16 kHz sampling rate for audio processing.
- Speech-to-Text (Whisper):
  - Model Used : openai/whisper-small Processor and openai/whisper-tiny Model
  - Purpose: Transcribes detected speech segments into text.
  - Libraries: WhisperProcessor, WhisperForConditionalGeneration by Hugging Face
  - Parameters:
    - device_map="auto": Automatically assigns the model to the available GPU.
    - load_in_8bit=True: Load the model in 8-bit precision to use less ram.
- Response Generation:
  - Purpose: Generates a concise and precise text response based on the transcribed text input.
  - Libraries: AutoTokenizer, AutoModelForCausalLM from Hugging Face
  - Parameters:
    - load_in_4bit=True: Loads the model in 4-bit precision to optimize GPU memory.
    - device_map="auto": For GPU assignment.
    - max_memory={0: "15GB"}: Allocates 15GB of GPU memory.
    - max_new_tokens=64: Limits the response generation to 64 tokens.
    - temperature=0.1: Low temperature for more deterministic outputs.
    - top_p=0.9: Controls the diversity of the generated responses.
- Text-to-Speech (Parler TTS):
  - Model Used: parler-tts/parler-tts-mini-expresso
  - Purpose: Converts the generated text response back into speech.
  - Libraries: ParlerTTSForConditionalGeneration, AutoTokenizer from Hugging Face
  - Parameters:
    - Custom description for voice modulation.
    - CUDA-based processing for faster inference.
    - soundfile library to save the synthesized speech.

**Pipeline Execution:**

- Audio Processing: A VAD process is used to extract speech segments from the input audio file.
- Speech-to-Text: The audio segments are transcribed to text using the Whisper model.
- Response Generation: The transcribed text is then fed to the LLaMA model, which generates a concise response.
- TTS: This text response was synthesized to speech using the Parler TTS model, and then saved into an output file.

**Overview:**

This solution utilizes state-of-the-art models in every stage of this pipeline and optimizes the performance by quantization techniques such as 8-bit and 4-bit, and memory management on a GPU. It offers an end-to-end streamlined process that should enable real-time or near-real-time speech-to-speech conversion for applications such as virtual assistants or voice-interactive systems.

**Current Status:** I have made significant progress in implementing a Speech2Speech agent with the goal of creating a real-time demo using a Streamlit app. However, the integration with ngrok encountered issues, preventing full deployment. WebRTC could be a viable alternative for streaming audio from a browser microphone to a server, but due to time constraints from mid-term examinations, I was unable to implement it. I plan to continue developing this project to achieve near real-time performance.

**Performance Enhancement Suggestions:** The current solution has a latency of 21 seconds. This can be reduced by streaming the audio output using one of the following methods:

- **XTTS V2:** Tokenizes input text and feeds it to a speech synthesis model, producing audio chunks.
- **Orca Streaming Text-to-Speech:** Synthesizes audio while the Large Language Model (LLM) is still generating a response.

Additionally, utilizing a more recent GPU architecture that supports FP4 or faster INT4 kernels could significantly reduce output times, enhancing the real-time capabilities of the system.

**LLM Routing:** Incorporating LLM routing can optimize performance by assessing the complexity of the query and directing it to an appropriately sized LLM. This can be implemented using RouteLLM, which would improve both cost efficiency and response time.

**References:**


- LiveKit Voice Assistant
- HuggingFace Speech-to-Speech