**Assignment: Design an End-to-End AI Voice Assistance Pipeline**

**Objective:**

Design a pipeline that takes a voice query command, converts it into text, uses a Large Language Model (LLM) to generate a response, and then converts the output text back into speech. The system should have low latency, Voice Activity Detection (VAD), restrict the output to 2 sentences, and allow for tunable parameters such as pitch, male/female voice, and speed.

Technologies Used:

Transformer (Pipeline)

Torch

Numpy

speech\_recognition

VAD

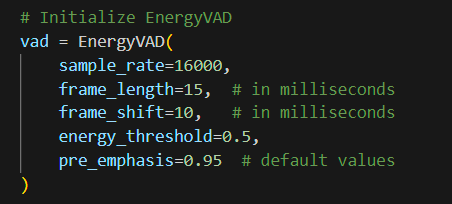
Whisper

google.generativeai (for Gemini)

edge-tts

**Step 1:**

Voice-to-Text Conversion:  
As the first part of this pipeline I had to implement a Speech recognition model that would let me transcribe the text. For speech and audio detection we are using a python library called VAD (Voice Activity Detection) that enables the program to calibrate and identify speech efficiently. The following parameters used for tuning the program to identify speech effectively:

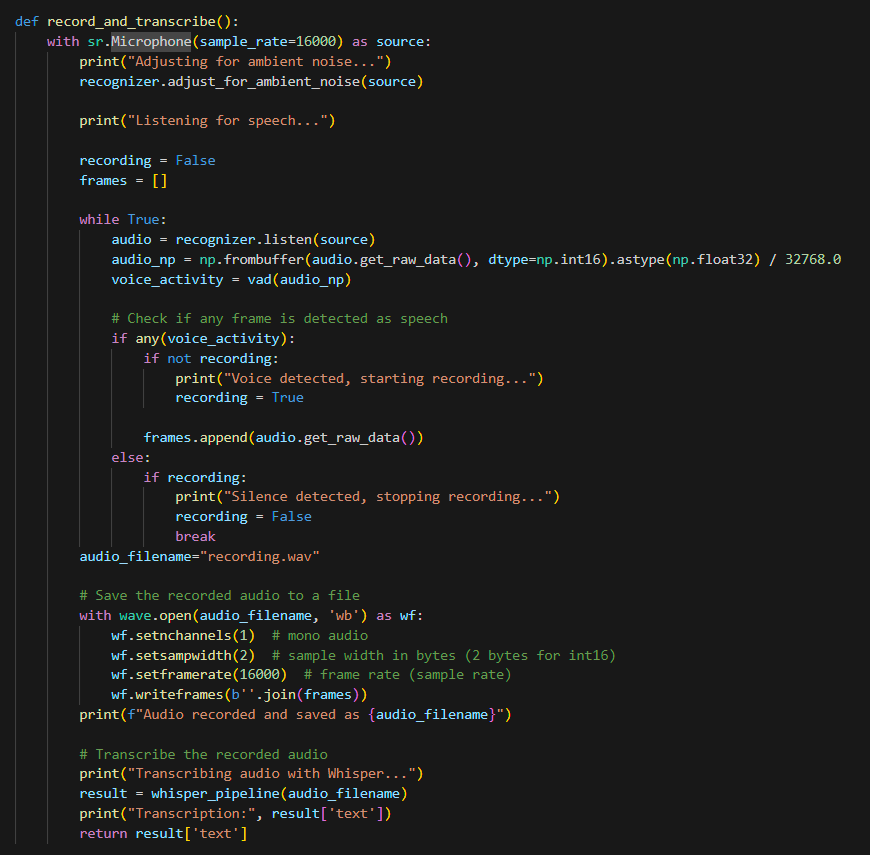


The value for energy\_threshold and sample\_rate were already given in the assignment document.

For recording the audio speech\_recognition library was used. For detecting audio we first record the ambient noise to calibrate the microphone to accommodate any background noise. We then use the parameters from EnergyVAD class to detect any speech. The EnergyVAD class returns a boolean numpy array that determines whether speech was detected at that particular frame. If speech was detected that frame is appended to the Frames array containg all other audio frames which are then stitched together and exported as a wav file.

The audio file is then passed onto the Whisper model as a file path. Whisper is an Automatic Speech Recognition model by OpenAI that is used to transcribe audio files. The audio files are processed by extracting features like Mel Spectrograms. The model has a transformer based encoder decoder architecture with attention masks. The encoder processes these features into hidden representations and the decoder takes these hidden representations and predicts the corresponding text. The predicted text is then tokenized and using a pre-trained language model to ensure that the predicted text is accurate and contextually right.

Code Snippet:



**Step 2:**

Text Input into LLM:

For the purpose of an end to end pipeline we can use any of the following LLMs to generate a response to our query:

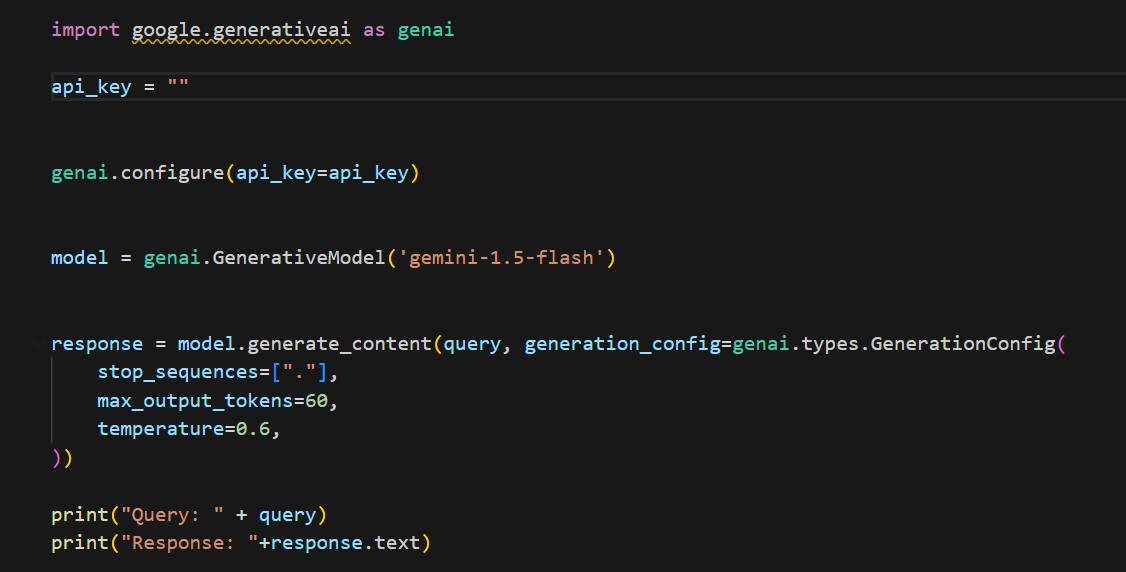
1. GPT-2 Large
2. Distil GPT
3. Falcon 7B
4. Vicuna
5. Lamma

For the purpose of this assignment I did try to implement these models directly into the pipeline.Falcun and Vicuna being very large models (upwards of 9GB) consumed quite a lot of memory. This led to resource exhaustion error in the pipeline. The lighter models like gpt2 and distilgpt didn’t consume as much memory as the ones mentioned above but the response generation capabilities of these models were subpar.

In the end I decided to implement Google’s Gemini API for the LLM part which made the pipeline fast and less resource intensive.

To implement the output restriction requirement of the assignment, max\_output\_tokens were set to 60 tokens and an end sequence was implemented. A temperature of 0.6 was also introduced to increase the model’s creativity. For this pipeline we are currently using the *gemini-1.5-flash* model which provides a fast and accurate response.

Code snippet:



**Step 3:**

Text-to-Speech Conversion:  
The generated text is then converted to an audio file using edge tts (Text to speech) which is a Neural TTS developed by Microsoft. Using edge tts we can convert the text along with various ajustible parameters like the gender of the voice (Male or Female), change the pitch of the voice (Lower for Deeper voice and Higher for a higher pitch voice) and change the rate (The speed ) of the dictation. The input text is sent to the Neural TTS model that analyzes the input to generate natural-sounding speech, mimicking human intonation, rhythm, and pronunciation. The system then produces an audio waveform, which can be customized by adjusting parameters like pitch, rate, and volume. Additionally, SSML tags can be included for finer control over speech delivery. Finally, the generated audio is outputted in formats like WAV or MP3, ready for use in applications, either through cloud services or on local edge devices for scenarios requiring low latency or offline capabilities.

Code Snippet:

A screen shot of a computer code

Description automatically generated

The resulting audio is then saved under the name “output.wav”.

Additional Requirements:

1. Latency: Suggest how to minimize the latency under (< 500 ms) using WRTC ?

To minimize latency under 500 ms using WRTC, optimize each stage of the AI voice assistant pipeline by using lightweight models and efficient code to reduce processing time. Implement parallel processing to handle tasks like voice detection and text processing simultaneously (Parallel processing).

1. VAD: Implement VAD to detect voice activity and ignore silence.  
   VAD was implemented to detect Voice activity and silence with the following parameters:

A computer code with white text

Description automatically generated

1. Output Restriction: Restrict the output response to a maximum of 2 sentences.

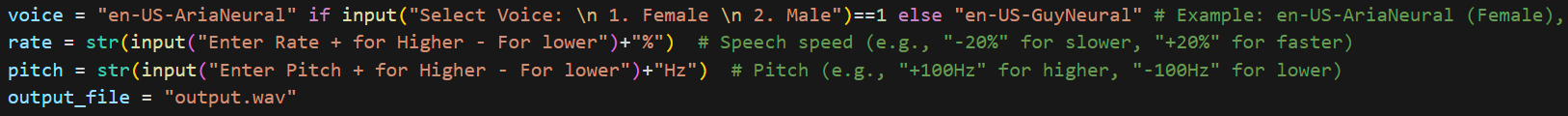
The following parameters were passed to the Gemini API to restrict the output:

A black screen with white text

Description automatically generated

1. Tunable Parameters: Allow for tunable parameters such as:
   * Pitch: adjust the pitch of the synthesized speech
   * Male/Female Voice: choose between different voices (e.g., Joanna or Samantha)
   * Speed: adjust the speed of the synthesized speech

All these were added while converting text-to-speech using edge-TTS. Following is the code snippet containing these tuneable parameters:



**Reference**

**https://huggingface.co/openai/whisper-large-v3**