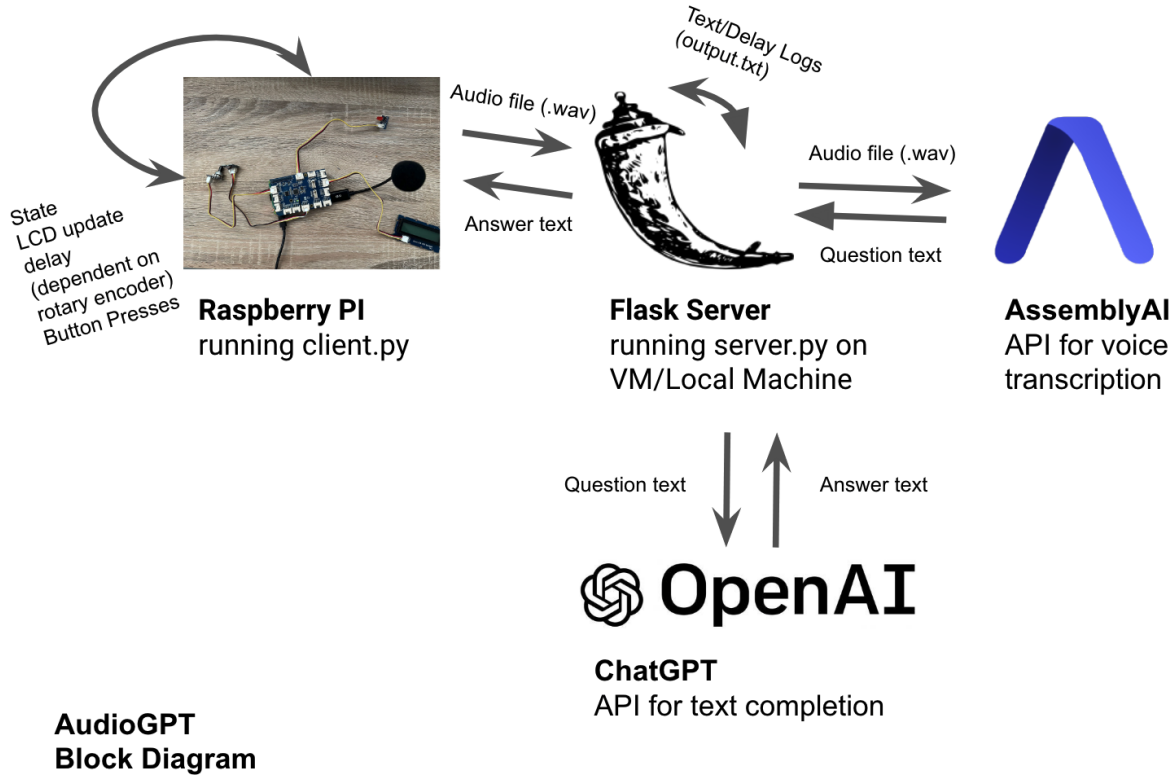
Fahim Kamal
EE 250

Final Project: AudioGPT

AudioGPT (a ChatGPT audio interface) is a four node IOT visualizer that takes audio data from the RaspberryPi (RPi) and returns a response from ChatGPT corresponding to the question recorded. The four nodes are the RPi running a client application, the Flask server running on a local machine, and the two API endpoints of AssemblyAI and OpenAI.



AudioGPT
Block Diagram

A user clicks a button attached to the GrovePi shield to trigger a recording event. Recording is done through the soundfile and sounddevice python libraries locally on the RPi. Using a HTTP post request, an application layer protocol, with the binary audio data (.wav) attached, the RPi sends the file to the flask server. The flask server which listens on a publicly accessible port, using the TCP protocol, downloads the file. Then, the flask server uses the AssemblyAI CLI to transcribe the recording.

AssemblyAI uses a neural network, specifically a transformer model, to predict the words of voice recording and returns a JSON response back to the flask server. The flask server takes the text corresponding to the audio, and sends a post request to the OpenAI chat completion endpoint. The post request also specified the model (gpt-3.5-turbo) and the role (user) in the

headers of the request (as per ChatGPT API reference). The delay in receiving each response is then recorded in output.txt.

Lastly, the response from ChatGPT is returned to the RPi. The RPi processes the text to emulate a scrolling effect by dividing the text into 16 character lines and showing successive lines twice (once on the bottom line, and once on the top line). Then, the RPi comes back to the idle state where it awaits another voice recording. The user also has the option to change the speed of LCD updates by using the rotary encoder in this stage. The value of the rotary encoder is mapped to 0.5 seconds for an LCD update to 5 seconds for an LCD update.

To visualize the delay at each of the endpoints, AudioGPT comes with a separate output.py script that outputs a matplotlib bar graph showing the delay for transcription (AssemblyAI) and the delay for chat completion (ChatGPT). Overwhelmingly, the delay arises from AssemblyAI which showcases one of the weaknesses of the IOT system. The system is bottlenecked by the time needed to transcribe the audio. One alternative would be to run the model locally and transcribe on the fly. However, the tradeoff would be requiring stronger computation on the edge. After looking at the AssemblyAI's documentation, they have a minimum threshold of 15-20 seconds to transcode and process the request before the model is run. This would explain why transcription takes on the order of 15-25 seconds although the question recordings are 3-8 seconds.

Another limitation of the system is not being able to update the speed of the LCD while a LCD update cycle (a response is being shown) is occurring. The solution to this would be to utilize interrupts to trigger an interrupt service routine that would handle the speed of the LCD updates. However, the GrovePi documentation does not readily offer an API that facilitates interrupts without significant revision. One possible solution would be to pass a callback function to the display function that reads the state of the rotary encoder and determines the appropriate delay. One problem in this approach is that time is wasted checking the state of the rotary encoder when the state has not changed.

Overall, AudioGPT was a good learning experience on working with different APIs, servers, and file I/O (particularly, when it came to recording audio, downloading audio, and writing the text/delay logs for analysis).