Assignment 2: Audio I/O

Emma Rafkin
Georgetown University
epr41@georgetown.edu

Abstract

Assignment two required building audio input/output (I/O) methods in Python including recording from the computer. Additionally, the code was augmented to generate spectrograms of the audio files as well as run the audio through an Automatic Speech Recognition (ASR) model and pass the text off to a Large Language Model (LLM). The following paper describes the implementation of these methods and a discussion of parameter selection and system performance. Code for this project is hosted on Github[1].

1 Introduction

Although electronic speech processing tools are commonplace nowadays, the act of digitally processing acoustic sound is still not simple. First, the computer has to attach to a microphone which takes the acoustic sound waves from the air and digitizes it. This digital signal must then be streamed in by the computer, which has to cache the object until some end signal where it can then be saved off to an intermediate file or passed along through some pipeline. Once the acoustic signal is captured, there are many methods for analysis available. Traditionally, linguists have used spectrograms to understand articulatory patterns in phonology. These spectrograms can be useful for machine learning across many fields in order to understand the digitized signal that will be fed as an input into a downstream model. Additionally, once the sound is captured it can be passed to a downstream model for further use in a pipeline. In order to demonstrate this capability, a speech-to-LLM pipeline was created for this assignment.

2 Implementation

Loading audio from a file, saving audio to a file, and audio playback are all relatively simple in Python. A .wav file can be easily loaded into an array of floating point numbers using the Librosa[2] library. Saving audio uses the Wave[3] library to take the floating point array and write it to a file. There are some parameters that can be set with the Wave library, but the basic method written for this assignment used a single channel as default and a sample width of 2. Finally, playback can be done by passing the floating point array to the SoundDevice[4] library which has a “play" function.

In contrast to loading and saving audio, recording audio from the computer is a bit more complicated. In order to write this method, Qwen-3-Coder-30B-A3B-Instruct (Hui et al., 2024; Team, 2025) helped to achieve the basic functionality, and then various pieces did not work or compile properly so they were fixed by hand. The goal of this method is to start recording and capture the signal until the user stops talking and then save the audio to a file. Voice Activity Detection (VAD) algorithms can be used to identify when speech starts and stops in an audio signal. The WebRTC-VAD[5] (Blum et al., 2021) library was used for basic VAD. This method opens up an input stream using SoundDevice. This requires an asynchronous callback function that takes the input from the stream and appends it to the output data object. In this callback function, the VAD algorithm is run on the audio data in order to detect if speaking stops. For speech to actually be considered "over", there should be multiple callback calls in a row where no speech is detected. The callback function is called every 0.02 seconds (this is a configurable variable) and it was set to require 10 consecutive non-speaking chunks in order for silence to be officially “detected". Once this threshold was met, the

[1] https://github.com/erafkin/ling4467_assignment_1

[2] https://librosa.org/

[3] https://docs.python.org/3/library/wave.html

[4] https://python-sounddevice.readthedocs.io/en/0.5.1/

[5] https://pypi.org/project/webrtcvad/

callback function raises a “stop" flag and the Input stream is closed. Then the resulting audio is saved to a file.

There are many libraries that generate spectrograms. MatplotLib, SciPy, and Torch were all tested, but it was found that Librosa was the easiest library to use. This module loads in the audio file and applies a Short-time Fourier transform (STFT) to it. This computes discrete Fourier transforms (DFT) over short overlapping windows in order to represent the signal in the time-frequency domain. Then it turns the amplitude of the data to decibels and displays it as a spectrogram. As discussed further in Section 3, there are many parameters that can be set to generate different spectrograms, but in this case a log scale was used for HZ and the length of the windowed signal after padding was set to 2048. The rest of the parameters were left as their default values.

ASR models take in audio as input and return a predicted transcription of that audio. Whisper (Radford et al., 2022) is an open source model from Open AI that has been trained on 680,000 hours. Whisper offers a variety of different sized models. As this code was running on a CPU, the model that is used by default is Whisper Tiny which is 39M parameters instead of Whisper Large's 1.5B parameters. This model was run via a Huggingface[6] ASR pipeline.

The final module built was a Speech-to-LLM pipeline which connected the audio recording module to the Whisper ASR module and then passed off the transcribed text to the chatbot from the previous assignment which then returns the LLM response to the recording. The model used for the chatbot was GPT-OSS-120b (OpenAI, 2025) hosted on the Cerebras API.[7]

3 Spectrogram Analysis

Spectrograms were created for a recording of the phrase “Hello, my name is Emma". It should be noted that there was a lot of background noise in this recording from an air conditioning unit, so there is a lot of noise in the spectrogram. However, the speech signal is clearly captured by the Librosa spectrogram visualization. The human perception of sound is nonlinear but rather is logarithmic. Therefore, in order to properly analyze a spectrogram, the frequency should be displayed logarithmically rather than linearly. Figure 1 demonstrates how much clearer it is to visualize the formants in logarithmic frequency rather than linear frequency.

Another parameter that changed the resolution of the spectrogram was the NFFT parameter in the STFT function. This set the length of the windowed signal. The documentation suggested 2048 because it was “well adapted for music signals". However, as this was a speech signal I was curious if it might be fine if not better to lower or raise this value. 5 different values were tested, 3 of which can be seen in Figure 2. It appears that the default value of 2048 was still the best for this signal.

These small experiments demonstrated that the default values were the best for displaying spectrograms, therefore while there were many other parameters than could be adjusted, they were left alone as they are likely already optimized for speech visualization.

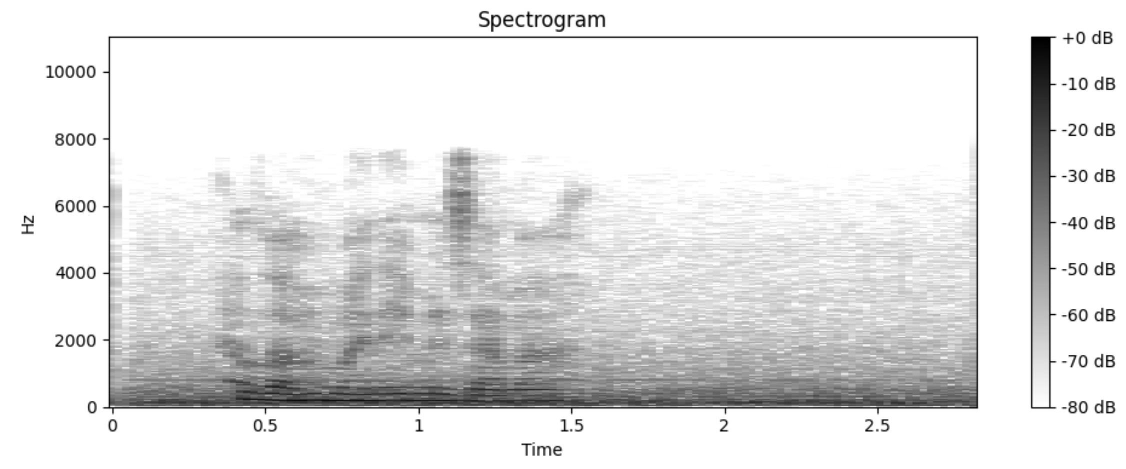
4 System Performance

The full speech-to-LLM pipeline takes around 3-4 seconds after the user is done speaking. The main latency issue comes from running the ASR model on a local machine. Even though this is the smallest Whisper Model, running it only on a CPU can be resource intensive and take time. Additionally, even though the LLM runs quickly because it is hosted by a much larger system, it takes time to await the HTTP response. The longer the answer the longer the response takes to process. For example the question “What is a good question to ask you?" elicited a 919 token response and thus it took over 4 seconds to run the pipeline. Generally, with “easy" prompts such as “Who is the governor of Illinois?" or “I would like to buy a hamburger", the system performed well. However, there are a few points of failure in the pipeline that can lead to incoherent output: audio capture, ASR, and LLM response.

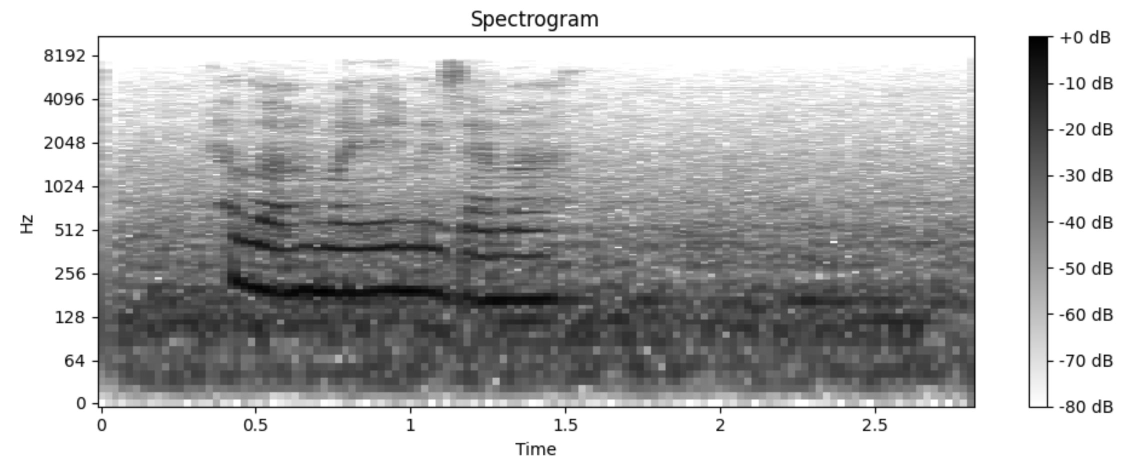
It is difficult to fine the right threshold of silence for the VAD module. In a noisy room (loud AC) the silence measured had to be short because it can be sensitive to background noise and therefore never stop. In a quiet room, the threshold could be larger to account for natural pauses in speech. In the future, it would be optimal to allow the threshold to be determined by the base level of noise in the environment to account for this and prevent hard

[6] https://huggingface.co/docs/transformers/en/main_classes/pipelines

[7] https://github.com/Cerebras/cerebras-cloud-sdk-python



(a) Linear scale



(b) Logarithmic scale

Figure 1: Linear vs Logarithmic frequency in spectrograms

coding. An example of this happening can be seen in Figure 3a. No speech was captured because the threshold was too low (because it was initially set for a very noisy space where it was harder to reach the silence threshold), which lead to a Whisper hallucination of “Thank you very much!", which was then passed off to the LLM.

Although Whisper Tiny is a powerful model relative to many others, its small size can lead to transcription mistakes or hallucinations. Even though the model being called happens to be multi-modal, the method by which the model is called only passes through the text. Therefore, if the transcription is wrong, the LLM response will likely be incoherent. An example of this can be seen in Figure 3b, where the input recording was “How many native speakers are there of Nahuatl" and it was transcribed as “How many native speakers are there in the hotel?" which elicited a confusing LLM response. This likely occurred because Nahuatl is word that does not appear in English very often and the relatively few parameters of Whisper Tiny might not allow it to retain this low-resource word if it encountered it in training. It is also possible that I did not pronounce Nahuatl correctly.

Finally, errors can occur from the LLM. Figure 3c demonstrates a seemingly successful pipeline where the LLM hallucinated. The speech of “Who is the governor of Illinois?" was correctly transcribed and the LLM returned information about the current governor of Illinois, J.B. Pritzker. However, it lists J.B.’s given name as “Joseph Barack" instead of “Jay Robert". This is an example of an LLM hallucination that likely came from training data of Barack Obama being from Illinois.

Despite these failure points, each module can be improved if needed because the ASR model and the LLM can be swapped out and the threshold can be set depending on the environment.

References

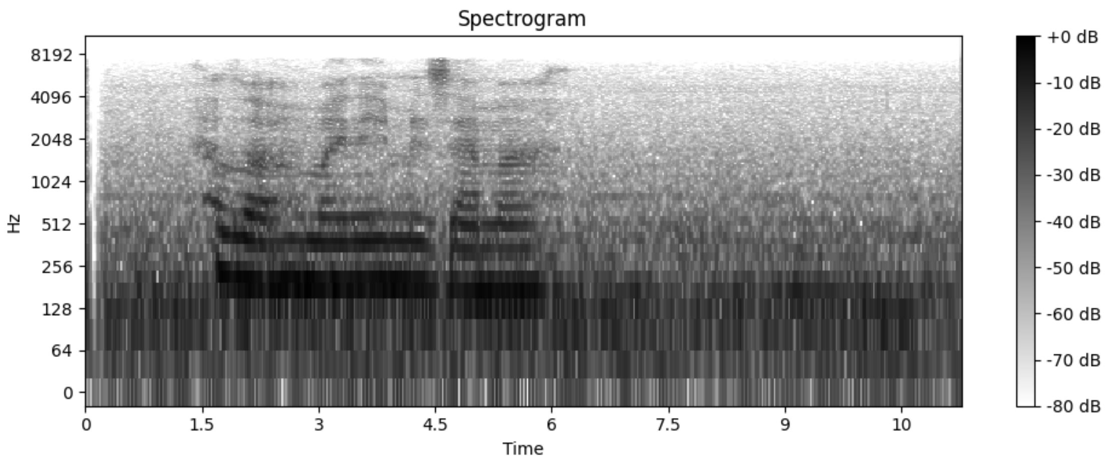
Niklas Blum, Serge Lachapelle, and Harald Alvestrand. 2021. Webrtc: real-time communication for the open web platform. *Commun. ACM*, 64(8):50–54.

Binyuan Hui, Jian Yang, Zeyu Cui, Jiaxi Yang, Dayiheng Liu, Lei Zhang, Tianyu Liu, Jiajun Zhang, Bowen Yu, Kai Dang, et al. 2024. Qwen2. 5-coder technical report. *arXiv preprint arXiv:2409.12186*.

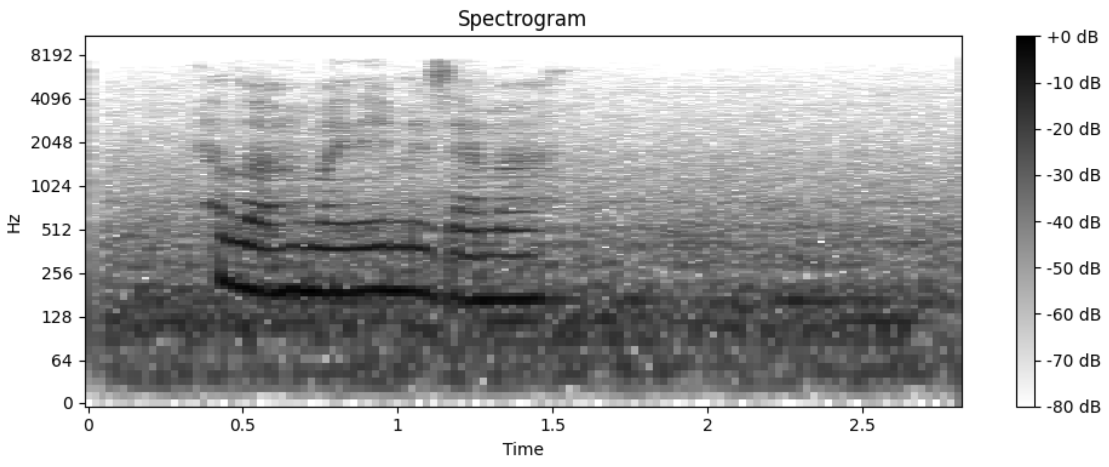
OpenAI. 2025. gpt-oss-120b gpt-oss-20b model card. *Preprint*, arXiv:2508.10925.

Alec Radford, Jong Wook Kim, Tao Xu, Greg Brockman, Christine McLeavey, and Ilya Sutskever. 2022. Robust speech recognition via large-scale weak supervision. *arXiv preprint*.

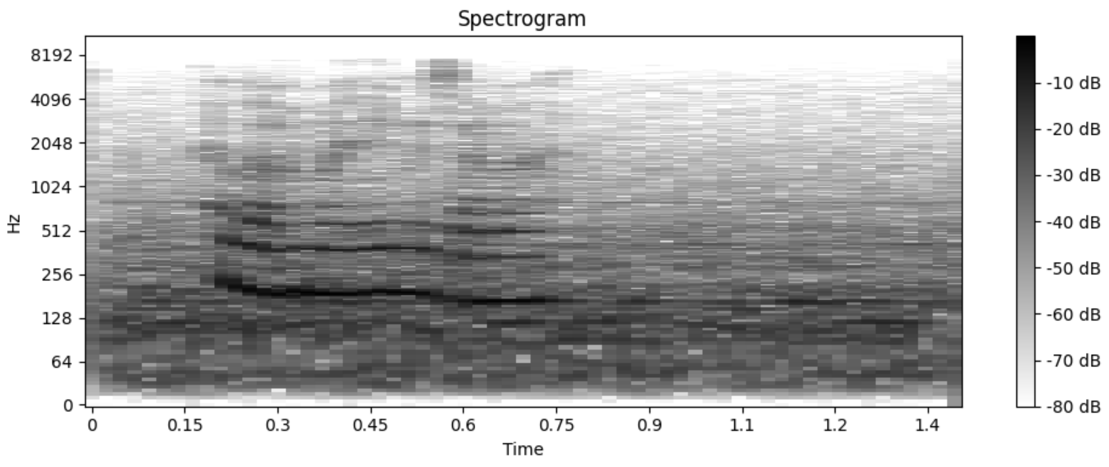
Qwen Team. 2025. Qwen3 technical report. *Preprint*, arXiv:2505.09388.



(a) NFFT 512

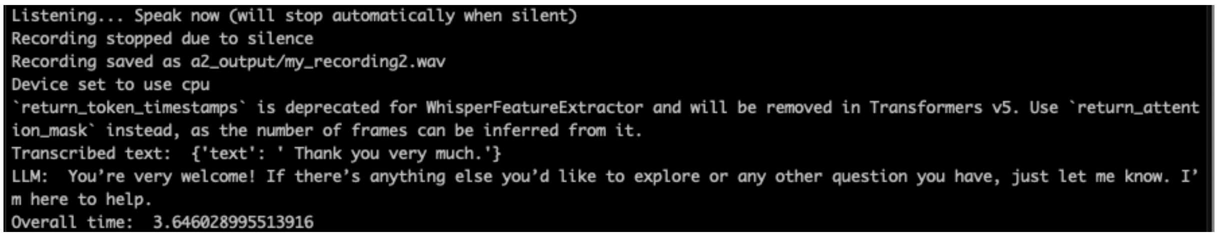


(b) NFFT 2048



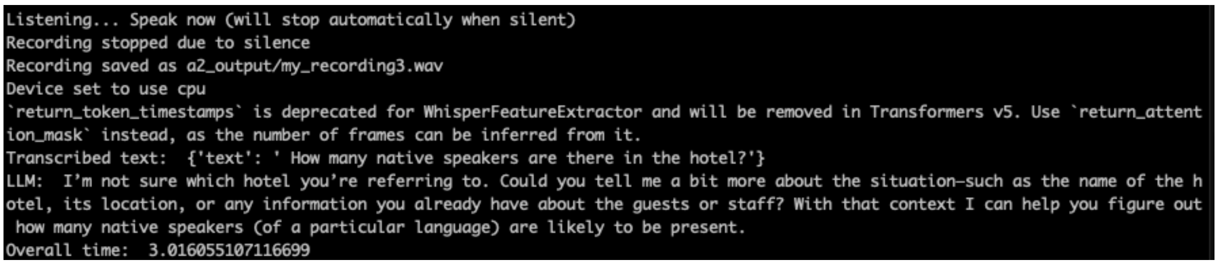
(c) NFFT 4096

Figure 2: Different values for NFFT, a parameter for the STFT function. NFFT sets the length of the windowed signal after padding with zeros.



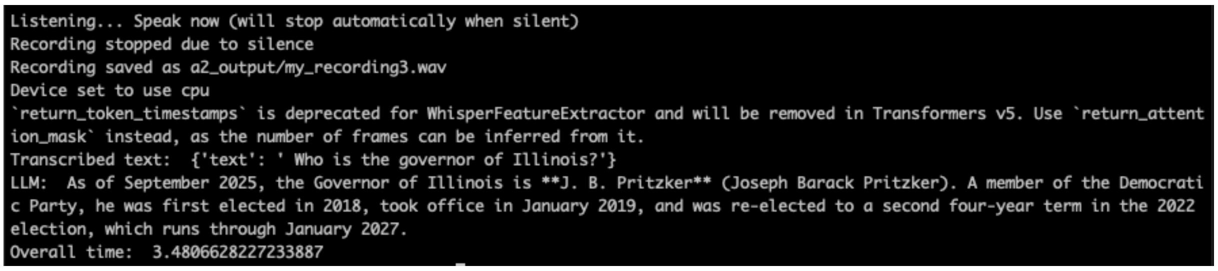
```
Listening... Speak now (will stop automatically when silent)
Recording stopped due to silence
Recording saved as a2_output/my_recording2.wav
Device set to use cpu
`return_token_timestamps` is deprecated for WhisperFeatureExtractor and will be removed in Transformers v5. Use `return_attent
ion_mask` instead, as the number of frames can be inferred from it.
Transcribed text:  {'text': ' Thank you very much.'}
LLM:  You're very welcome! If there's anything else you'd like to explore or any other question you have, just let me know. I'
m here to help.
Overall time:  3.646028995513916
```

(a) Silence threshold was set too low, setting off a hallucination pipeline.



```
Listening... Speak now (will stop automatically when silent)
Recording stopped due to silence
Recording saved as a2_output/my_recording3.wav
Device set to use cpu
`return_token_timestamps` is deprecated for WhisperFeatureExtractor and will be removed in Transformers v5. Use `return_attent
ion_mask` instead, as the number of frames can be inferred from it.
Transcribed text:  {'text': ' How many native speakers are there in the hotel?'}
LLM:  I'm not sure which hotel you're referring to. Could you tell me a bit more about the situation—such as the name of the h
otel, its location, or any information you already have about the guests or staff? With that context I can help you figure out
 how many native speakers (of a particular language) are likely to be present.
Overall time:  3.016055107116699
```

(b) Whisper transcription error leading to incoherent response from LLM



```
Listening... Speak now (will stop automatically when silent)
Recording stopped due to silence
Recording saved as a2_output/my_recording3.wav
Device set to use cpu
`return_token_timestamps` is deprecated for WhisperFeatureExtractor and will be removed in Transformers v5. Use `return_attent
ion_mask` instead, as the number of frames can be inferred from it.
Transcribed text:  {'text': ' Who is the governor of Illinois?'}
LLM:  As of September 2025, the Governor of Illinois is **J. B. Pritzker** (Joseph Barack Pritzker). A member of the Democrati
c Party, he was first elected in 2018, took office in January 2019, and was re-elected to a second four-year term in the 2022
election, which runs through January 2027.
Overall time:  3.4806628227233887
```

(c) LLM hallucination (Pritzker's name) yielding incorrect information

Figure 3: Examples of speech to LLM chats.