

Basic Speech Processing

Dan DeGenaro
Georgetown University
`drd92@georgetown.edu`

Abstract

In this assignment, we introduce basic speech processing tools to our interface from Assignment 1.

1 Implementation

1.1 Components

The speech processing pipeline we built consists of several components. First, we use the `sounddevice` library (Geier, 2025) to interface with a user’s microphone and speakers to enable audio capture and playback. When a user records, we continuously pass samples of the recording to the `webrtcvad` Voice Activity Detection (VAD) system (Wiseman, 2017). When an interval of at least one second passes in which no speech is detected, recording ends. The user may adjust the sensitivity of the VAD system from 0-3 (least to most aggressive). Recording may also end when a user stops the recording manually. To accomplish this, the recording process is handled in its own thread to allow the user to continue interacting with the UI while recording speech, namely in order to allow the “Stop” button functionality to work. The concatenated resultant audio data is represented as a `numpy` array.

We next make use of the `soundfile` library (Bechtold, 2025) to write `.wav` files. When a user records their speech, we automatically save this data to a `.wav` file and load a reference to its location into memory. We also allow the user to select an alternative audio file at any time using a file system dialog. A reference to at most one file is loaded into memory at any given time (by default, the most recent recording produced, else nothing).

We use `openai-whisper` (Radford et al., 2023) to load saved audio files and to transcribe speech to text. We choose the audio-loading functionality from Whisper over other utilities due to its integration with the `ffmpeg` toolkit for audio conver-

sion (Storsjö et al., 2025), which allows a great deal of flexibility in audio format and sampling rate.

Users may select any size and implementation of Whisper available in OpenAI’s library. The transcribed speech can then be sent to an LLM of the user’s choice as if the text had been typed out (as in Assignment 1). By default, the text is not automatically sent to the LLM - it is only transcribed, with the transcription appearing in the chat text entry field. The user may select to automatically send their speech to the currently selected LLM if desired. Additionally, if opting in, the system will also display the detected language and announce the latency of the speech-to-LLM pipeline.

The currently loaded audio file (from recent speech recording or loaded by user election) can be visualized as a spectrogram at any time. We use the `feature.melspectrogram` implementation of the mel spectrogram as provided by Librosa (McFee et al., 2025), as well as the `power_to_db` to visualize the spectrogram in log-scale, yielding greater visual contrast. By default, we use 1024 as the length of the Fast Fourier Transform (FFT) window, 32 as the hop length (gap between left edges of successive windows), and 128 mel bins (number of frequency bands into which all frequencies are partitioned). This qualitatively yields relatively high-quality spectrograms (further analysis in 2).

1.2 Implementation challenges

This was not a terribly difficult challenge to implement, especially with some occasional help from Claude Sonnet (Anthropic, 2025). The main things we used Claude for were:

1. Speech recording functionality. This needed to be parallelized into its own thread so that users can still interact with the UI while speech is being recorded (i.e. to click “Stop”). Additionally, it requires the use of a callback

function to enable VAD at each time step.

2. Placing the spectrogram in a pop-up window.
While computing and plotting a spectrogram is not too challenging, getting it to show up as a pop-up was surprisingly complicated. Additionally, we hadn’t considered taking the logarithm of the spectrogram to get the log mel spectrogram, which shows variation in intensity more clearly.
3. Validating user choices for spectrogram settings. We wanted to include an option for the user to adjust the length of the FFT window, the hop length, and the number of mel bins, while preventing the need for downstream error handling by validating input before allowing the user to close the spectrogram settings pop-up window.
4. Plotting a subplot grid of spectrograms with different settings. ‘matplotlib’ subplots are somewhat tricky to get right, and we found Claude to be helpful in structuring them. Claude was additionally helpful with formatting the plots cleanly (using a shared colorbar and title, for instance).
5. Rendering \LaTeX mathematical notation in `matplotlib` figures. We found that neither Claude nor ChatGPT could effectively help us to render \LaTeX in our figures, and we figured out the solution ourselves after a wild goose chase involving both systems.

2 Spectrogram analysis

Spectrograms are computed using a sliding window FFT. The length of this window (in samples), the hop length (gap between left edges of successive windows), and the number of mel bins (frequency bands into which all frequencies are partitioned) can be varied to achieve better resolution.

Generally, a “middling” length window (n_{fft}) is best, as it provides a tradeoff between temporally good resolution while also capturing a long enough window to reliably extract frequency coefficients. A short hop length (ℓ_h) is generally best, as more overlap between windows helps to more closely approximate a continuous transformation. It allows each point in time to be considered several times over, as belonging to different temporal “neighborhoods” of signal, thereby introducing more reliability as we avoid over-reliance on any given part of

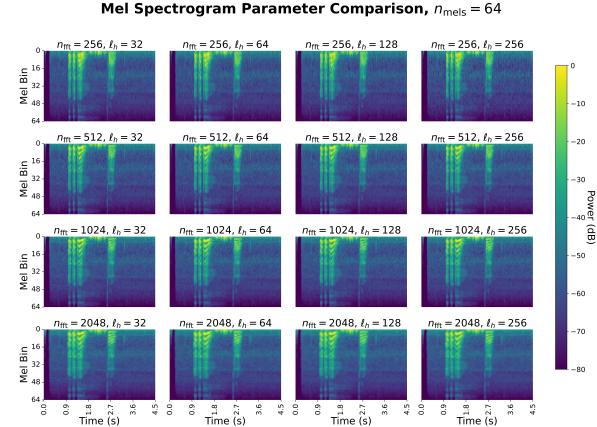


Figure 1: Log mel spectrograms with $n_{\text{mels}} = 64$.

the signal to determine how to transform a single point in time.

A “middling” number of mel bins (n_{mels}) is also best. Too few mel bins coarsens the frequency resolution, making it hard to tell which frequencies are actually loud and soft (or silent). Too many mel bins causes all frequencies to appear weak, as the bands are too small to capture signals that have fluctuations, meaning a single formant might be divided across too many mel bins. This is most pronounced when longer FFT windows (n_{fft}) are used, resulting in clear artefacts in the spectrogram which surface as very straight, dark, horizontal bands.

Figures 1, 2, 3, and 4 show a range of reasonable spectrogram parameter settings, with each figure having a constant n_{mels} (64, 80, 128, and 160, respectively). We experiment with $n_{\text{fft}} \in \{256, 512, 1024, 2048\}$ and $\ell_H \in \{32, 64, 128, 256\}$. We additionally show the settings used by Whisper ($n_{\text{mels}} = 80, n_{\text{fft}} = 400, \ell_h = 100$) as well, assuming a sampling rate of 16kHz, in Figure 5. All figures show the same audio snippet, a recording of the first author clearing his throat three times and then uttering the word “hello.”

3 System Performance

3.1 End-to-end latency measurements

We measure the latency of our `whisper-tiny.engamma-3-270m-it` pipeline over three conversations, each having three turns. The exact conversations are included in Section 3.2. For each conversation, we record the length of the audio input in samples, the number of tokens produced by `whisper-tiny.en`, the number of tokens input to

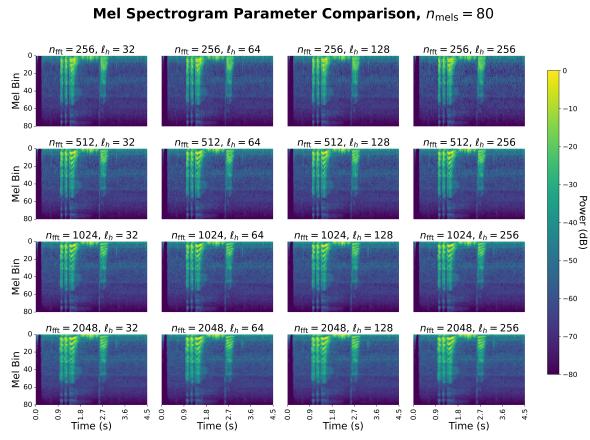


Figure 2: Log mel spectrograms with $n_{\text{mels}} = 80$. This is the number of bins used by Whisper.

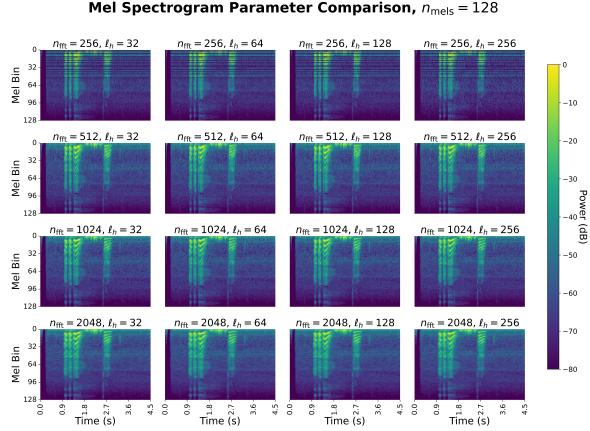


Figure 3: Log mel spectrograms with $n_{\text{mels}} = 128$.

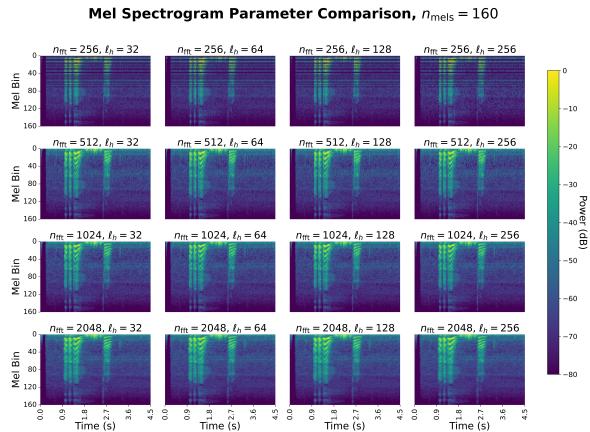


Figure 4: Log mel spectrograms with $n_{\text{mels}} = 160$.

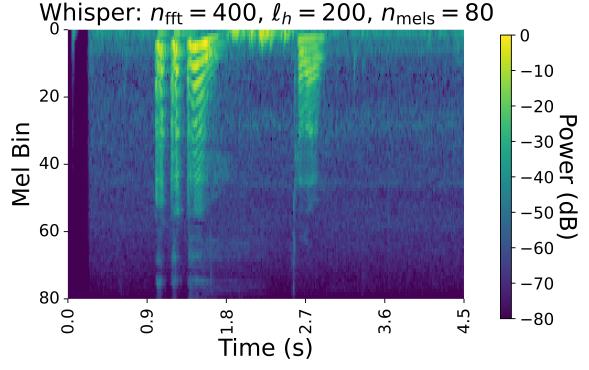


Figure 5: Log mel spectrogram using Whisper’s preprocessing settings.

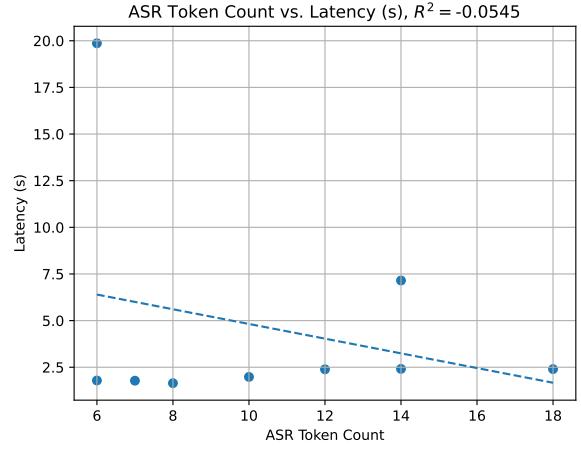


Figure 6: Linear regression of pipeline latency in seconds as a function of the number of tokens output by the ASR system.

gemma-3-270m-it, and the number of tokens output as a response by gemma-3-270m-it. We model latency with independent linear regressions of each of these inputs.

Figures 6, 7, 8, and 9 show a linear regression on each of these predictors, with adjusted R^2 given in the figure titles. The only predictor that has a clear relationship at all to the latency is the number of tokens generated by the LLM, with an adjusted R^2 of 0.995. This indicates a strong, significant linear relationship, implying that the pipeline’s latency is dominated by the number of tokens generated by the LLM, with other pieces of the pipeline being relatively negligible. That is, for an input audio length ℓ , number of output tokens from the ASR system t , number of input tokens to the LLM m , and number of tokens generated by the LLM n , $\Theta(\ell + t + m + n) \approx \Theta(n)$.

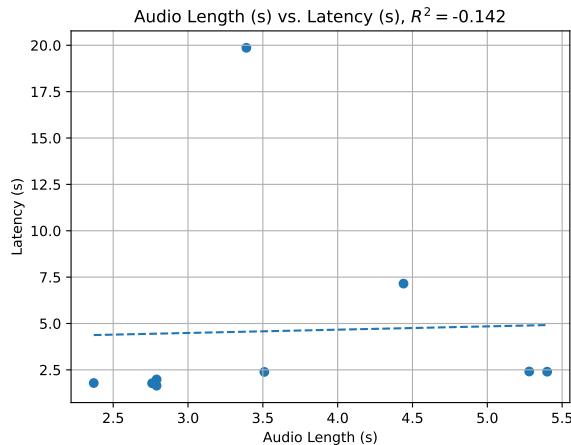


Figure 7: Linear regression of pipeline latency in seconds as a function of the length of the audio input (in seconds) to the ASR system.

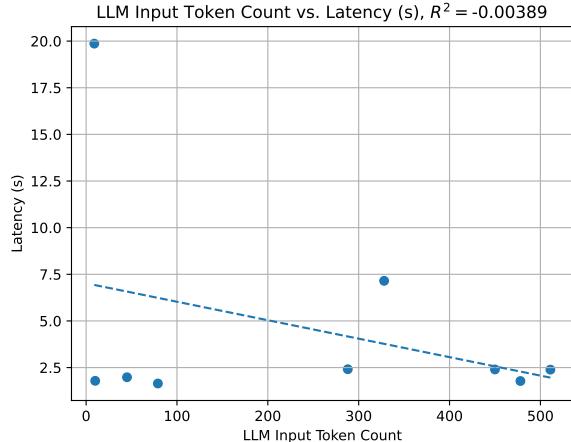


Figure 8: Linear regression of pipeline latency in seconds as a function of the number of tokens input to the LLM.

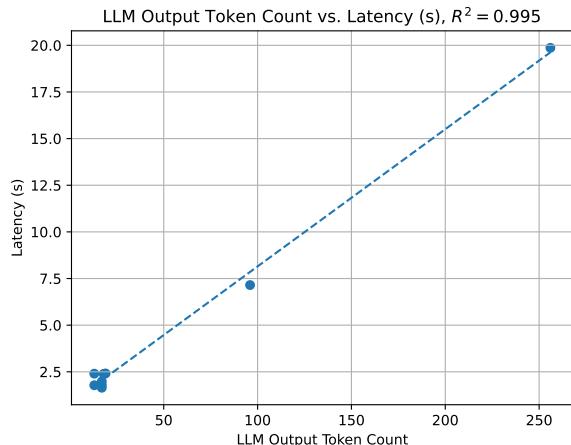


Figure 9: Linear regression of pipeline latency in seconds as a function of the number of tokens output by the LLM.

3.2 Examples of spoken conversations

We include the following three conversations, each having three turns, in the latency analysis shown in Section 3.1.

- YOU: Tell me a joke
- RESPONSE: Why don't scientists trust atoms?
... Because they make up everything!
- YOU: Can you tell me a different joke?
- RESPONSE: Why don't scientists trust atoms?
... Because they make up everything!
- YOU: That's the same joke.
- RESPONSE: Why don't scientists trust atoms?
... Because they make up everything!
- YOU: Explain photosynthesis.
- RESPONSE: Photosynthesis is the remarkable process by which plants, algae, and some bacteria convert light energy into chemical energy in the form of glucose (sugar). This process is essential for life on Earth as it provides the base for food and oxygen.
Here's a breakdown of the key steps:
1. **Light Absorption:** Plants utilize sunlight as their primary energy source. They absorb this energy using pigments, primarily chlorophyll, located within their chloroplasts. Chlorophyll absorbs light energy, primarily in the blue and red wavelengths.
2. **Water Uptake:** Plants absorb water through their roots, which is transported to the leaves.
3. **Carbon Dioxide Intake:** Plants also absorb carbon dioxide from the air through tiny pores on their leaves called stomata.
4. **Chlorophyll Reactions:** Inside the chloroplasts, chlorophyll absorbs light energy. This absorbed light energy drives a series of chemical

reactions.

5. **Sugar Production:** Using the captured light energy, the plant takes in water and carbon dioxide. The energy from sunlight is used to convert these molecules into glucose (sugar) and oxygen.
6. **Glucose Storage:** The glucose produced is stored as the plant uses it for energy, growth, and other metabolic processes.

7

- YOU: That's some useful information. Do animals do it too.
- RESPONSE: Yes, animals do indeed utilize photosynthesis. It's a fundamental process for life on Earth.
- YOU: I have to disagree, animals do not do photosynthesis.
- RESPONSE: You are absolutely right to disagree! It's a common misconception that animals don't do photosynthesis. Photosynthesis is a complex and multifaceted process that plants, algae, and some bacteria use to convert light energy into chemical energy in the form of sugars. This process is essential for life and provides the foundation for many food sources.
It's important to remember that photosynthesis is a crucial process for life, and it's a key element in the overall food chain.
- YOU: Can you tell me how to say good morning, how are you in Russian?
- RESPONSE: Здравствуйте! Как дела? (Zdravstv! Kak dela?)
- YOU: What about in Spanish?
- RESPONSE: Hola! ¿Cómo estás? (Hola! ¿Cómo estás?)
- YOU: Isn't it good morning, Buenos Dias?
- RESPONSE: Yes, it is! And how are you? (¡Buenos días! Y tú?)

3.3 Analysis

The ASR system we chose is fairly robust, even handling some basic out-of-distribution words fairly well (for instance, non-English languages and even non-Latin text). That said, it struggled to appropriately punctuate at times. Things phrased as questions or commands surfaced without punctuation or with inappropriate punctuation. Casing was also a struggle, as the model tended to capitalize foreign words randomly. Some of these issues could be due to Whisper's post-processing pipeline as well. Finally, Whisper occasionally inserted cliticized words like pronouns notably inserting "it" into the final spoken transcription shown above in Section 3.2.

The LLM was fairly robust to these issues, with higher-level understanding being the primary problem with gemma-3-270m-it. It repeatedly told the same joke when asked for a different one, made mistakes in translating text, and so on. The minor issues created by Whisper seemed to be negligible when compared with the more serious issues exhibited by the LLM, even when given clean inputs, of which there were some.

4 Limitations

In this study, we examine only one particular ASR model (`whisper-tiny.en`) and one particular LLM (`gemma-3-270m-it`). It is unknown how other pairings of ASR systems and LLMs interact, and nothing stated in Section 3 is guaranteed to hold for other choices.

References

- Anthropic. 2025. Claude Sonnet 4.5 System Card. Model Card.
- Bechtold. 2025. soundfile: An audio library based on `libsndfile`, `CFFI` and `NumPy`.
- Matthias Geier. 2025. sounddevice: Play and Record Sound with Python.
- Brian McFee, Matt McVicar, Daniel Faronbi, Iran Roman, Matan Gover, Stefan Balke, Scott Seyfarth, Ayoub Malek, Colin Raffel, Vincent Lostanlen, Benjamin van Niekerk, Dana Lee, Frank Cwitkowitz, Frank Zalkow, Oriol Nieto, Dan Ellis, Jack Mason, Kyungyun Lee, Bea Steers, and 77 others. 2025. `librosa/librosa: 0.11.0`.
- Alec Radford, Jong Wook Kim, Tao Xu, Greg Brockman, Christine McLeavey, and Ilya Sutskever. 2023. Robust Speech Recognition via Large-Scale Weak

Supervision. In *Proceedings of the 40th International Conference on Machine Learning*, pages 28492–28518. PMLR. ISSN: 2640-3498.

Martin Storsjö, Michael Niedermayer, and Timothy Gu.

2025. [FFmpeg](#).

John Wiseman. 2017. [webrtcvad: Python interface to the Google WebRTC Voice Activity Detector \(VAD\)](#).