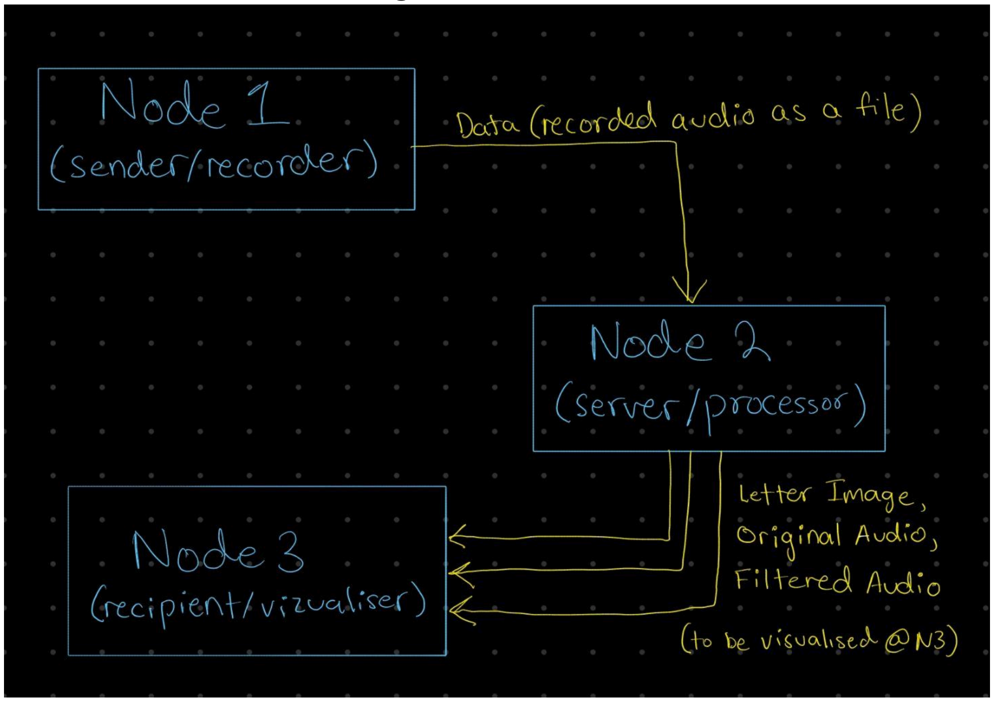
EE250 Spring 2025 LetterWriter Final Project
Deccan Maniam, Sahil Pandit

What is the LetterWriter?
Our final project, LetterWriter, is a system adapted from the movie *Her (2014)*. The system essentially allows a user to send a digitally "handwritten" letter to someone using their voice– i.e. they speak into their device and a digitally "handwritten" letter is produced and sent to the intended recipient.

Our three node system **communicates using HTTP requests** and does the following:

- The first node is the **recorder** and will record the user's voice, save it as an audio file, and send it out to the second node in the system, the server, for processing
- The second node is the **server/data-processor** and will receive the audio file, **perform audio filtering** to isolate/make more prominent the user's voice, **transcribe the audio**, **convert it into a digitally "handwritten" image**, and send it (along with the original and filtered audio files) to the recipient (N3)
    - Audio filtering is done using FFTs and inverse FFTs to separate unwanted frequencies from the audio and then recombine the remaining frequencies
    - Audio transcription and letter generation is done using off the shelf, pre-trained OpenAI models
- The third node is the **recipient/visualiser** and will receive and display the three files it receives from the server on a simple webpage – the "handwritten" letter, the original audio, and the filtered audio

What does it look like at a high level?


What does it use?
As mentioned earlier, the system is comprised of three nodes – in our case this was demonstrated using a RaspberryPi as Node 1, a Laptop running Windows as Node 2, and another Laptop running Ubuntu as Node 3 – and **uses HTTP** as the communication protocol.

The reason for HTTP is because our project is pretty much completely built in Python and the HTTP implementation using requests and app routes was of great convenience not only in implementation but also development (made it easier to modularise the code and split up tasks in development).

As also touched upon earlier, the main forms of processing present in this project are in the form of audio filtering, transcription, and handwriting generation.

Audio filtering was done by using the Fast Fourier Transform and Inverse Fourier Transform methods as part of the *numpy* library. Moreover, we used libraries like *librosa* and *sounddevice* to handle managing and creating audio files as they are versatile enough to deal with most audio file types (mp3, wav, m4a, etc.) as inputs and can construct .wav files given frequency samples respectively.

For transcription, we used OpenAI Whisper, an off-the-shelf transcription model. For handwriting generation, we used https://github.com/ankanbhunia/Handwriting-Transformers, which is a pre-trained transformer that uses pytorch under the hood.

What were some challenges/issues?
Some of the biggest challenges we faced mainly came in the form of an ambitious scope, unaccounted for hardware considerations, and miscellaneous technical issues (e.g. audio encoding and file management standardisation).

The ambitious scope proved challenging because we ultimately ended up with a project that is a little more off the shelf and unrefined than we intended. For example, we initially wanted to have the system such that the handwriting generation was more from scratch or personalised than the current implementation. In an ideal scenario we would have been able to give the user a way to train a model with their own handwriting so that the letter that is produced is in line with the sender's writing style. On top of this, minor things like audio recording and file processing proved an unexpected challenge because of hardware incompatibilities and differences across the nodes we were using – e.g. using an internal mic on a laptop was far easier than configuring a USB microphone on the RaspberryPi. Additionally, things like a more natural mode of interaction with the recording node was something we wanted to aim for – i.e. allow a user to continue speaking until they were finished (e.g. signified by a keyboard interrupt, a keyword such as "...END RECORDING", or after a defined period of silence). Finally, we intended to use the RPi as our server node initially, but we overlooked the heaviness of something like audio transcription and image generation and so changing what device does what led to unseen consequences – e.g. The need to use a USB microphone instead of an internal mic.

Ultimately, the main takeaway from this project comes down to planning and time management – having an ambitious scope is good as it incentivises work (because it's interesting and… just cool) BUT we came to realise the importance of having a more holistic high level view when it comes to planning – e.g. Having some plans that are too barebones, just right, and (intentionally) too ambitious as ways of balancing essential functionality with desired functionality; it also just makes it easier to course correct and adjust in situations where plans/visions get derailed because of unforeseen circumstances.