ELEC 498 Final Report

Team 19: Local Speaker Identification and Speech Transcription

Hannah Berthiaume, 20114482

Aiden Horan, 20130908

Parker Rowe, 20122320

Adam Strom, 20111006

April 10, 2023

Presented To:

Dr. Alireza Bakhshai

Dr. Mike Korenberg

Dr. Alex Tait

Dr. Sean Whitehall

Dr. Saeed Gazor

# Executive Summary

Creating an Auto-transcription service for live, in-person conversation is a problem that has been worked on for more than a decade with different niche solutions implemented in the tech industry. Unfortunately, these solutions are not yet generalizable for general use purposes such as note taking during in-person meetings. In its current state, note-taking can be erroneous and distracting for the notetaker. Having a live speaker-labeled transcription service accessible after meetings for further reference can help take pressure off a meeting note-taker, improve the accuracy of information from the meeting, and make that information accessible to more employees. Creating an accurate speech-to-text service has already been achieved and many products already exist that can accomplish this. The difficulty in creating a live transcription service has been correctly identifying each speaker without the use of a keyword at the beginning of each sentence. The goal of this capstone project was to create a complete live transcription service that could translate a person’s speech to text and output the text auto-labeled with the speaker's name. The tool was to take advantage of pre-made open-source speech-to-text models and integrate them with a new speaker detection service created by the capstone group. The tool was expected to be able to predict who is speaking in real-time with a minimum accuracy of 70%, and with less than one second of latency. In addition, the service would be coupled with an intuitive UI and be trained on 4 distinct voices while also detecting when a new person is speaking. To create the speaker detection service a convolutional neural network was utilized and trained in 5 categories. 4 of the categories were the designated speakers on which the model would be trained, and the remaining category was a collection of speakers that would be used for the model to predict when a new voice is being heard. The model was trained using Mel Frequency Cepstral Coefficients extracted from 1-second-long clips. This was implemented in combination with OpenAI’s open-source speech-to-text transcription model nicknamed “Whisper” to create the transcription aspect of our service. This was integrated into a web application with a clean intuitive frontend and backend service that controlled the audio clips coming in, preprocessed them, and fed them through the speech-to-text and speaker recognition model mentioned above before sending them back to the frontend to be displayed to the user. Our final solution has an 86% accuracy rate in identifying the speaker on unseen testing data, less than 1-second latency between the first voice being detected and a transcription being displayed as well as the prediction taking less time per audio clip than the length of an audio clip avoiding any growing lag. With time spent to further develop and scale this service, it could transform many workplaces in how meetings are run by increasing efficiency and accessibility.

Table of Contents

[Executive Summary i](#_Toc132056483)

[List of Figures iv](#_Toc132056484)

[List of Tables iv](#_Toc132056485)

[1.0 Motivation and Background 1](#_Toc132056486)

[1.1 Requirements 1](#_Toc132056487)

[Functional 1](#_Toc132056488)

[Non-Functional 1](#_Toc132056489)

[Constraints 2](#_Toc132056490)

[1.2 Problem Scope 2](#_Toc132056491)

[2.0 Design 2](#_Toc132056492)

[2.1 Speaker Identification Subsystem 2](#_Toc132056493)

[2.2 Speech to Text Subsystem 3](#_Toc132056494)

[2.3 User Interface Subsystem 4](#_Toc132056495)

[2.4 Runtime Environment Separation of Concern 4](#_Toc132056496)

[3.0 Implementation 5](#_Toc132056497)

[3.1 Overview 5](#_Toc132056498)

[3.2 Communication 6](#_Toc132056499)

[3.3 User Interface 7](#_Toc132056500)

[3.3.1 UI Framework 7](#_Toc132056501)

[3.3.2 Audio Recording & Processing 8](#_Toc132056502)

[3.4 Backend Server 9](#_Toc132056503)

[3.4.1 Software Overview 9](#_Toc132056504)

[3.4.2 Phrase Handling 9](#_Toc132056505)

[3.4.3 Voice Activity Detection (VAD) 10](#_Toc132056506)

[3.4.4 Multithreading 11](#_Toc132056507)

[3.4.5 Speaker Identification 12](#_Toc132056508)

[3.4.6 Speech Transcription 13](#_Toc132056509)

[3.5 Project Timeline 15](#_Toc132056510)

[3.6 Bill of Materials 16](#_Toc132056511)

[3.7 Training Data 17](#_Toc132056512)

[4.0 Testing and Evaluation 17](#_Toc132056513)

[4.1 Accuracy 17](#_Toc132056514)

[4.2 Latency 19](#_Toc132056515)

[4.3 Prediction Speed 19](#_Toc132056516)

[4.4 Processing Speed 20](#_Toc132056517)

[4.5 User Interface 21](#_Toc132056518)

[5.0 Considerations 24](#_Toc132056519)

[5.1 Social Considerations 24](#_Toc132056520)

[5.2 Safety Considerations 25](#_Toc132056521)

[5.3 Environmental Considerations 25](#_Toc132056522)

[5.4 Economic Considerations 26](#_Toc132056523)

[5.5 Regulatory Compliance 26](#_Toc132056524)

[5.6 Professional Considerations 27](#_Toc132056525)

[6.0 Compliance with Specifications 27](#_Toc132056526)

[7.0 Conclusions and Recommendations 28](#_Toc132056527)

[7.1 Future work 28](#_Toc132056528)

[7.1.1 Improve Scalability 28](#_Toc132056529)

[7.1.2 Improve Versatility 29](#_Toc132056530)

[7.1.3 Introduce a Continuous Learning Component 30](#_Toc132056531)

[7.2 Additional Learning 30](#_Toc132056532)

[7.3 Societal Impact 31](#_Toc132056533)

[7.4 Conclusion 32](#_Toc132056534)

[References 33](#_Toc132056535)

[Appendices 35](#_Toc132056536)

[Appendix A: Team Contributions 35](#_Toc132056537)

[Appendix B: Major Code Blocks 35](#_Toc132056538)

[User Interface 35](#_Toc132056539)

[Data Preprocessing 36](#_Toc132056540)

[Backend Server Code 36](#_Toc132056541)

[Appendix C: Project Planning 42](#_Toc132056542)

# List of Figures

[Figure 1 - System Diagram 6](#_Toc132054831)

[Figure 2 - Example of state variables written in JS. These variables determine how Svelte updates the visual state of the UI 7](#_Toc132054832)

[Figure 3 - Example of Reactive Binding of UI to state 8](#_Toc132054833)

[Figure 4 - Example value and output derived from phrase\_class\_counts of a 17 second phrase. 10](#_Toc132054834)

[Figure 5 - Code block on the backend server which executes the computationally expensive transcribe() and identify\_speaker() functions in parallel and awaits their result. 11](#_Toc132054835)

[Figure 6 - Code block of the backend's transcribe() function which takes the path of a .wav file and outputs what was said in it (if anything) 14](#_Toc132054836)

[Figure 7 - Table of prediction speed delays as noticed by users 20](#_Toc132054837)

[Figure 8 - Processing speed of 300, 1sec audio samples 21](#_Toc132054838)

[Figure 9 - User Interface while it is connecting to the model 23](#_Toc132054839)

[Figure 10 - User Interface when it is unable to connect to the model 23](#_Toc132054840)

[Figure 11 - Functioning User Interface 24](#_Toc132054841)

[Figure 12 - Manual .wav header insertion function. This header is prepended to each audio chunk before being sent to the backend](#_Toc132054842)

[Figure 13 - Script for extracting MFCCs from an audio file and saving them to a Numpy data file](#_Toc132054843)

[Figure 14 - Backend server code, main file: backend/app.py](#_Toc132054844)

# List of Tables

[Table 1 - Comparison of time it takes to perform both transcription and speaker identification for a 1000ms clip of audio with different threading techniques. 11](#_Toc132054845)

[Table 2 - Bill of materials for the project 16](#_Toc132054846)

[Table 3 - User Interface Messaging 22](#_Toc132054847)

[Table 4 - Project Specifications and their Completion Status](#_Toc132054848)

[Table 5 - Table of team contributions throughout the project](#_Toc132054849)

[Table 6 - Project timeline as defined in the blueprint report](#_Toc132054850)

# 1.0 Motivation and Background

Transcriptions can be important tools used during meetings or presentations to ensure all information is properly communicated. Keeping a transcribed version of spoken dialogue can be useful for record-keeping [1]. It also ensures content is accessible to all individuals, including those who are hard of hearing [2]. Speech-to-text models are not new, development has been progressing on these models for many years [3]. Some of these incorporate speaker identification through services such as Microsoft Azure’s “Speaker recognition” service or Google Cloud’s “Speaker ID”. However, neither of those services are offered for free. Neither of them can be run locally either, requiring that users have an internet connection to access the service. This project is unique in that it makes this technology more accessible to users that may not have stable internet connections or be willing to compromise their privacy but sending data off to other services.

## 1.1 Requirements

The design of the system relied entirely on the needs of the end user. In this case, the functional requirements outlined below specify what the system *must* do to fulfill these needs, while nonfunctional requirements describe *how* the system would fulfill those needs.

### Functional

1. Allow the user(s) to record their conversation live.
2. Free and open-source
3. Runs locally on hardware – no internet needed.
4. The system will provide a live transcript of what is being said by the user(s) in a user interface.
5. The system will identify who is speaking for a given segment of transcribed audio.

### Non-Functional

1. The system’s user interface is intuitive, aesthetically pleasing, and responsive.
2. The transcription and identification output in the user interface has no perceivable delay to the end user(s), (i.e., it “feels live”) .
3. The system can run on average consumer hardware.

### Constraints

1. **Hardware:** our most capable device is running an Intel i7-8550u processor with a NVIDIA MX150 GPU. This device must be able to run the system to the specifications outlined above.
2. **Time:** the entire preprocessing-transcription-identification pipeline must run in under 1 second since each new audio chunk is sent in 1 second intervals. Any slower and it was found the “real-time” perception of the system was lost.

## 1.2 Problem Scope

The project goal was to create a service that could create a complete transcript of a conversation between 2 or more individuals and provide this to the user. This larger problem has been broken into 3 sub-components that combined would provide the complete service. These sub-components are the speaker identification service that detects who is speaking, a speech-to-text service that would transcribe the audio of what is being said, and a web interface that displays the full transcription to the user. Each of these sub-components has their own subjective goals as defined by the team.

# 2.0 Design

## 2.1 Speaker Identification Subsystem

The speaker identification subsystem was decided to be created from scratch for this project. This was decided as few speaker identification services are available for free and fewer fulfill the use cases of the project. In addition, providing a high-quality speaker identification service improves the novelty of the final product. When deciding on the scope of this subsystem the team first looked at how this feature was implemented in other transcribing services. Looking at services such as Zoom or home assistants revealed that common methods used for speaker identification in these products would not properly suit our use case. Transcription with services like Zoom use multiple microphones and the name of the speaker logging into the call to depict who is speaking at a given time. This method would not work as a solution to our problem as it would require all individuals to wear individual mics pre-labeled in our software, which would both be costly to the users and require too much overhead. Home assistants use a keyword or phrase trained on an individual’s voice to detect the speaker. This method, although closer to our ideal solution, still fails as it would heavily interrupt the flow of a conversation and lead to poor user experience. The solution landed on was closer to that of Microsoft Azure which is a neural network trained on a larger sample set of individuals speaking and uses that to predict the speaker. Discussion with the advisor, Dr. Sanaz Seyedin, suggested that this would require at least 20 minutes of recorded audio from each speaker for the model to achieve a suitable performance of at least 70% accuracy. These factors lead to the decision to limit the number of voices this model was trained on to just 4 individuals in the capstone group. This limited the work and allowed a more dynamic approach in the data collection stage as adjustments could be made when the group found barriers while training the neural network or found new methods that showed improved efficacy. Stretch goals, of adding additional classes to identify when a new voice is speaking, increasing the number of voices in the model, or adding a continuous learning component to the model, were created as well but were identified as not necessary for the scope of this project. What was not in the scope of the project was for the voice identification to work through significant noise or to correctly predict multiple speakers at the same time.

## 2.2 Speech to Text Subsystem

With the expected workload of the speaker identification component of the project being high, it was suggested by the capstone group’s advisor Dr. Sanaz Seyedin to limit the scope of this component of the project. To fulfill this recommendation while still developing a high-quality transcription service it was decided to use a pre-made Open-Source speech-to-text model that would be integrated into the final solution. It was decided that this solution would have to have the capability to run in parallel with the speaker recognition subsystem. In addition, the speech-to-text service would still have to run with considerable accuracy, being able to pick up and accurately output the words being said in real-time. In addition, it would have to be cheaper than other full transcription services that included dictation which tends to cost over $100.00 per year [4].

## 2.3 User Interface Subsystem

The user interface component of this project was planned to be aesthetically pleasing and intuitive. It would include a simple interface that would allow the user to begin and end recording a conversation, it would also contain an area to display the live transcription with an indicator of who is speaking and what they said. The user interface would display the speaker identification model’s best guess as to who is speaking for a given segment of audio. To emphasize the feeling of real-time, it was also decided that the user interface would be able to both update the predicted speaker and the words detected as the user spoke into it, allowing the system to correct any words said previously as more context arrives. In the final design, the user interface was decided to run in the browser as the web platform offers unparalleled accessibility and the ability to run on almost any platform.

## 2.4 Runtime Environment Separation of Concern

While the browser is a versatile platform for user interfaces, in terms of compute and high-performance multithreaded machine learning tasks, it does not offer the same level of flexibility and ease of development as a Python application can. This fact left the speech-to-text transcription and speaker identification subsystems needing to be run in a Python context running on the same computer, where the front-end user interface and backend can communicate with each other. Deciding to decouple the user interface and the more computationally expensive subsystems allowed for more modularity and ease of development during the project and in the future. The backend’s scope is to meet the requirements indicated above relating to real time output and minimizing latency, the results of which are discussed in 4.0 Testing and Evaluation. Meanwhile, the front-end interface’s scope is to provide the user an intuitive interface.

# 3.0 Implementation

## 3.1 Overview

The final implementation involves two main components which contain the three major required subsystems discussed in section 2.0 Design. These components run simultaneously on the computer and can be described as follows:

1. **Frontend User Interface**
   * Browser-based
   * Takes microphone input from the user
   * Streams audio chunks to backend
2. **Backend Server**
   * Python-based
   * Detects if speech occurs in a given audio chunk
   * Transcribes what is said in each chunk (Speech-to-Text subsystem)
   * Identifies who is speaking (Speaker Identification subsystem)

The following diagram depicts the client-server architectural style of the system and the flow of an audio chunk from input to output.

Text

Description automatically generated with medium confidence

Figure 1 - System Diagram

## 3.2 Communication

To allow the user interface and backend server to communicate live, the team opted to use WebSockets, a bidirectional and low-latency communication protocol. Using an open-source WebSocket library, Socket.io, which offers standard SDKs for JS (frontend) and Python (backend), allowed the client and server components to communicate with one another using a publisher-subscriber model. This model of communication allows both the client and server to subscribe to events they care about and emit events to any subscribers who care to listen. The two events that comprise the majority of the system’s functionality are:

1. **audio\_chunk:**

Emitted by the client and handled by the server. 1000ms of byte-encoded wav audio is sent as the payload.

1. **transcript\_update:**

Emitted by the server and handled by the client. The payload is a JSON object containing a phrase id, transcribed text, and a speaker which is either appended or updated to the running Transcriptions state variable mentioned in 3.3.1 UI Framework.

## 3.3 User Interface

### 3.3.1 UI Framework

The front-end user interface is based on the browser stack of languages: JavaScript (JS), Cascading Style Sheets (CSS) and Hypertext Markup Language (HTML). Over the last twenty years, multiple software frameworks have been developed to help in the creation of fast, reactive, and easy to develop user interfaces. To increase the team’s productivity, the UI framework Svelte was chosen for its ease-of-use, familiarity within the team, and its ability update the UI based on internal state of variables. As such, the frontend’s visual state relies entirely on the internal variables present in the accompanying JavaScript code.

Graphical user interface, application

Description automatically generated

Figure 2 - Example of state variables written in JS. These variables determine how Svelte updates the visual state of the UI

Among the variables seen in Figure 2, the one with the greatest visual impact on the user interface is the running Transcriptions variable. This is a JSON object which contains key-value pairs of what the team has called “phrases,” which the UI visually splits up by having a unique timestamp, text, and speaker assigned to it. This variable is updated/appended to through event listeners triggered by the backend, where the backend passes a JSON object which specifies a phrase id, transcription text, and speaker.

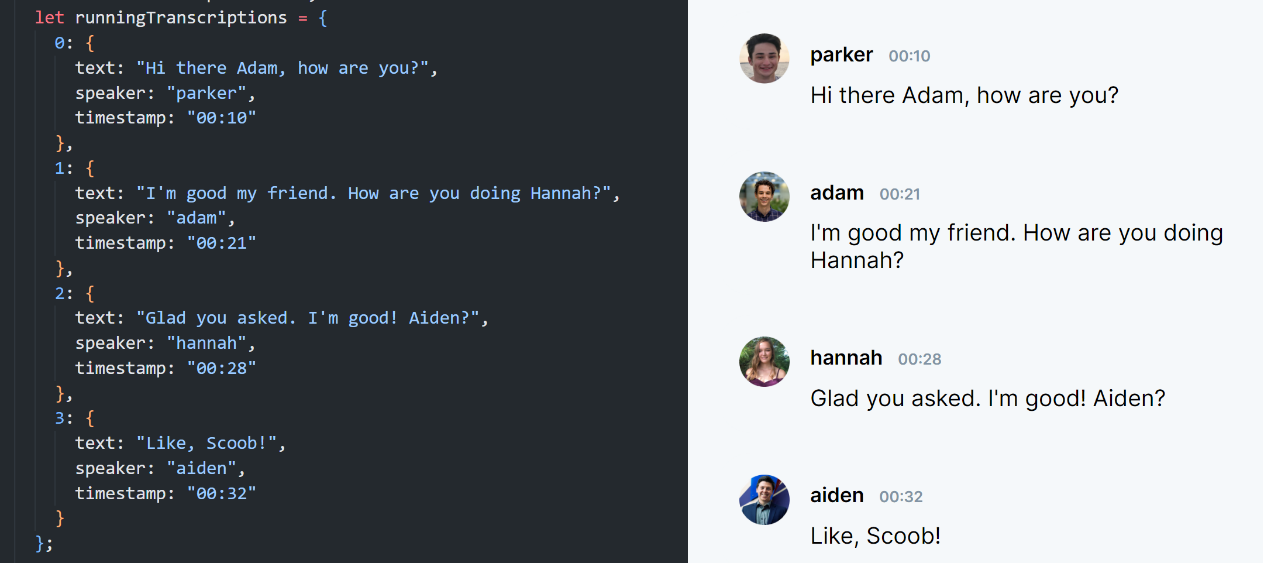


Figure 3 - Example of Reactive Binding of UI to state

### 3.3.2 Audio Recording & Processing

The front-end interface takes in microphone input using the standard MediaRecorder browser API. There were two main constraints this API enforced on the project that had to be overcome.

1. The input to both the speech-to-text model and speaker identification preprocessing step must be in a raw, uncompressed .wav format, while the MediaRecorder API (in its current implementation as of March, 2023) can only record audio in a .webm format across all modern browsers
2. The MediaRecorder API streams audio chunks where the initial chunk contains any audio header information (in our case .wav), while each proceeding chunk does not contain such header. To perform transcription and identification on a 1 second audio chunk, the models require the .wav header for each chunk.

To address constraint 1), the team found an open-source library called extendable-media-recorder which has the ability to encode the audio data from the MediaRecorder API into a .wav format required by the two audio models found on the backend [5]. While this new recorder API solved the first issue, it still would only append the .wav header to the first chunk. To address constraint 2), the team developed a function which can insert a valid .wav header at the beginning of each audio chunk before sending it off to the backend for transcription and identification, see Figure 12. The combination of solutions for 1) and 2) allowed the team to stream individual valid 1 second .wav audio chunks from the browser to a Python-based backend server for individual processing.

## 3.4 Backend Server

### 3.4.1 Software Overview

The backend server is a Python-based web server built using the Flask framework. The decision to use Python for the server was primarily due to the team’s use of the open-source library Tensorflow to develop our speaker identification model, in addition to the use of other open source models which are based on the PyTorch machine learning library. The server is constantly listening for events emitted by the client using the Socket.io library including: “connect”, “begin\_transcription”, “audio\_chunk”, and “end\_transcription”. The main audio processing loop occurs in the “audio\_chunk” event listener where the server receives 1 second of uncompressed audio in the .wav format, which is subsequently saves as a temporary file so it can be operated on further.

### 3.4.2 Phrase Handling

Phrases in context of this system are individual chunks of audio with transcribed text, an attributed speaker, and timestamp. For each phrase, the backend repeatedly appends each new second of audio to an audio buffer. It also maintains a running count of which speakers have been classified for each new second of audio added to the phrase. The speaker with the most votes is therefore the most likely speaker for the entire phrase, this counting method helps combat flickering of who is speaking if a one second chunk were to be misclassified.

Text

Description automatically generated

Figure 4 - Example value and output derived from phrase\_class\_counts of a 17 second phrase.

To display conversational text in a concise and readable manner, phrases must have a logical beginning and end. The team decided that a new phrase should be determined by the following two conditions:

1. The current speaker has taken a two second or greater pause between speeches.
2. If a different speaker is being classified for majority of the last five seconds than who was speaking for the entire rest of the phrase, we can assume someone new is speaking and we should create a new phrase to correctly show somebody new is speaking.

To address 1), the team uses a voice activity detector for each segment of audio. If nobody is speaking for two or more seconds, the backend uses this as an indicator to begin a new phrase. Otherwise, if someone is talking, the backend knows to attempt speech-to-text and speaker identification for the given chunk. To address 2), the backend server maintains a first-in-last-out (FILO) queue of max length five of the last speakers identified for each audio chunk. If the majority of this queue is not the same as the majority count found in phrase\_class\_counts for entire phrase, the code interprets this as a new speaker starting to talk and should therefore start a new phrase to reflect this.

### 3.4.3 Voice Activity Detection (VAD)

The first step of the audio processing loop is to detect whether there is any speech detected in the 1 second chunk of audio. This has historically been a complex task, but thanks to the latest advancements in machine learning for voice activity detection, the team was able to use a drop-in solution called Silero VAD. Silero VAD is a pre-trained enterprise-grade Voice Activity Detector which can detect if someone is speaking in a 1000ms chunk in under 1ms of CPU processing time [6]. The team’s decision to use this was based on its flexibility of audio chunk length, speed, support of 8000Hz and 16000Hz bitrate, and ease of implementation through PyTorch. With only one line of code the team was able to check if someone was speaking within a given audio chunk

### 3.4.4 Multithreading

The two most computationally expensive operations of the system are performing speaker identification and transcription on an audio chunk. During testing the team found that running the software sequentially left the program waiting for transcription to finish before it could attempt to identify the speaker. This led to a processing delay of over 1 second, an effect which cascaded into the system being further behind with each new chunk it processes. This did not meet the non-functional requirements. To mitigate this, multithreading was used to allow the program to offload both computationally intensive tasks from the main thread into their own threads, where they can be operated on simultaneously and not waste CPU time waiting.

Table 1 - Comparison of time it takes to perform both transcription and speaker identification for a 1000ms clip of audio with different threading techniques.

|  |  |  |
| --- | --- | --- |
|  | Single Threaded | Multi-Threaded |
| Average Time to Complete (ms) | 1435ms | 725ms |

Using Python’s concurrent.futures library, the team was able to execute the two operations in parallel and await their output before proceeding to emitting the results to the user interface.

Text

Description automatically generated

Figure 5 - Code block on the backend server which executes the computationally expensive transcribe() and identify\_speaker() functions in parallel and awaits their result.

### 3.4.5 Speaker Identification

If it was found that a user was speaking for the given 1 second audio chunk, the backend server starts to perform speaker identification in its own thread. The first step of speaker identification is to perform preprocessing, then feed the extracted features into the team’s custom trained speaker identification model.

#### Data Preprocessing

Mel Frequency cepstral coefficients (MFCCs) were used as an input to the model. MFCCs are a widely used feature extraction technique in speech and audio processing. The process of extracting MFCCs involves taking a short-time Fourier transform of the audio signal and mapping the resulting power spectrum onto the Mel scale, The Mel scale is a perceptual scale of pitch that relates to the way humans perceive the pitch of sounds. It is based on the concept that the human ear's response to sound is non-linear, meaning that the perception of pitch changes with changes in frequency. The Mel scale is used in fields such as acoustics, speech processing, and music to measure the perceived pitch of sounds. The Mel scale is then divided into a number of equally spaced bins, and the logarithm of the energy in each bin is taken. Next, a discrete cosine transform is applied to these logarithmic energies to obtain a set of cepstral coefficients. The resulting MFCCs capture information about the spectral shape of the audio signal and are often used as features for tasks such as speech recognition, speaker identification, and music classification [7].

The process of collecting MFCCs from the training data involved dividing a set of uncompressed, labelled audio files into 1 second frames of audio, which were normalized to account for differences in volume between audio files then individually passed into an MFCC extraction function. When running the model in real time, 1 second of audio is taken from the microphone buffer at a time and fed into the MFCC extraction function. This process needs to be optimized, as computing the Fourier transform takes O(n log n) time [8]. The other processes that need to occur within the same timeframe are prediction of the speaker and transcription of the contents of speech. In order for the live microphone buffer to not overflow, preprocessing and prediction must occur in a shorter amount of time than the length of the audio frame. This was adjusted by modifying the length of a frame fed into the MFCC extraction algorithm. Otherwise, the system will lag by an increasing amount of time. The frame length also affected the accuracy of the model. It was found that 1 second was suitable for this value. The script to perform this extraction can be found in Appendix B: Major Code Blocks: Data Preprocessing.

#### Speaker Identification

The speaker identification step was comprised of a feed forward neural network that took twelve levels of MFCCs as inputs, had one output for each speaker plus one output for an unidentifiable speaker, and three dense hidden layers: one with 256 nodes, one with 128 nodes, and one with 64 nodes. Each node used a rectified linear unit (ReLU) as an activation function, with the exception of the output layer, which used SoftMax for classification.

To train the model, a batch size of 32 was used with a maximum of 100 epochs. Two techniques were implemented to combat overfitting. Dropout, which randomly removes a certain number of connections between nodes each epoch, was implemented with a coefficient of 0.1 on the input layer, 0.25 on the next two hidden layers, and 0.5 on the final hidden layer. Early stopping would occur if the model did not improve significantly over 20 epochs. In addition to these, learning rate reduction on plateaus was used to help achieve the highest accuracy possible. The learning rate of the model would be reduced by a factor of 0.75 every 4 epochs where the model did not improve.

The model is utilized in real time by accepting a set of MFCCs extracted from one second of audio data. The model would classify each second with one speaker that it believed was talking in that clip and this classification is added to the running count of classifications for who is speaking for a given phrase.

### 3.4.6 Speech Transcription

Speech transcription is performed using an Open-Source speech-to-text model released by OpenAI in September of 2022, nicknamed Whisper [9]. Whisper is a transformer-based sequence-to-sequence general-purpose speech recognition model. It is trained on a large dataset of diverse audio and is also a multitasking model that can perform multilingual speech recognition, speech translation, and language identification. Given this, the capability of the system is not restricted to just the English language, as all other subsystems for speaker identification and VAD are language agnostic. Whisper is accessible through a Python library which the team has wrapped in a function, transcribe(). Transcribe takes a .wav file path as its input, and the Whisper model reads this file, performs transcription, and among multiple outputs, responds with the plaintext of what was said in the audio clip. The transcribe() function returns this plaintext to its invoker.

Text

Description automatically generated

Figure 6 - Code block of the backend's transcribe() function which takes the path of a .wav file and outputs what was said in it (if anything)

There are multiple active discussions on how once could manipulate Whisper into becoming a real-time transcription model rather than relying on prerecorded audio files [10]. One method the team found to make Whisper feel “real-time” is through the concept of phrases. As mentioned in 3.4.2 Phrase Handling, the backend keeps a running audio buffer of concatenated 1000ms .wav audio chunks which comprises a phrase. By feeding in this audio buffer to the Whisper model as each new audio chunk arrives and is appended to the buffer, the Whisper model is constantly re-transcribing the audio buffer with increasing context as the phrase runs longer. The initial 1000ms of audio may have incomplete transcriptions of words, but subsequent seconds provide context for the Whisper model to finish cut-off words and disambiguate homophone words (e.g. "toe" and "tow"). By constantly providing updates to a given phrase through a phrase ID shown in Figure 3, a phrase is updated visually for the user and improved upon in accuracy as phrase grows in length.

## 3.5 Project Timeline

The project timeline defined in the Blueprint report was followed closely during the development of the system. It can be observed in Table 6, which can be found in Appendix C: Project Planning. An additional column was added to show the completion dates for each task. All timeline goals were achieved within the deadlines. On occasion the predetermined buffer periods were utilized. For example, the shipping of the ordered parts took longer than anticipated but remained within the allocated buffer period.

In addition, further work was completed that was not initially anticipated. Supplementary meetings with advisor Dr. Sanaz Seyedin were scheduled. These meetings each served a different purpose. For example, one meeting involved inquiries into the specifics of audio pre-processing and their parameters. These meetings have been added to the Project Planning Table (Table 6). Also, while it was still under development, a demonstration of the project was given at the First Year Discipline night. Attending the First Year Discipline night contributed to the development of our project because it offered an opportunity to test our project on various voices in a noisy environment. Using live audio from such a busy space provided useful insight into the limitations of our hardware.

## 3.6 Bill of Materials

Table 2 - Bill of materials for the project

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Item | Purchase Date | Qty | Cost | Supplier |
| Mic Shure SM58S (with switch) [11] | November 9th 2022 | 1 | $134.47 | Manufactured by: Shure Incorporated  Cosmo Music - The Musical Instrument Superstore |
| Protection plan for microphone | November 9th 2022 | 1 | $24.99 | Asurion Consumer Solutions of Canada Corp |
| Shure X2U XLR-to-USB Signal Adapter | November 9th 2022 | 1 | $111.87 | Manufactured by: Shure Incorporated  Sold by: Amazon.ca |
| Protection plan for signal adapter | November 9th 2022 | 1 | 19.99 | Asurion Consumer Solutions of Canada Corp |
| 5 Pack Foam Cover, Handheld Microphone Windshield | November 9th 2022 | 1 | $11.29 | Manufacturer: Z ZAFFIRO  Sold by: Trevored |
| **Total:** |  | 5 | $302.61 |  |

## 3.7 Training Data

Training data was collected iteratively over the course of the project. Initially, two minutes of audio was collected from each team member for the purpose of developing a minimum viable product, and more audio was added in five-to-ten-minute increments to improve the accuracy of the model.

Data was collected from the four team members who would be demonstrating the model. In addition, a smaller amount of data was collected from a set of people which would be used to train an “unknown” category. The final model was trained on approximately twenty minutes of active audio data for each category.

Data was recorded using a Shure SM58S dynamic microphone. This microphone was selected in order to capture training data with a high sample rate while reducing background noise. It also allowed for effective live demonstration in a noisy environment.

To ensure that the model would work in real time, it was necessary to create as similar conditions as possible to spontaneous speech while recording data. This was done by reading transcripts of ted talks and other live media in order to replicate that tone of speaking. Textbooks and newspaper articles were also used as material to supplement this, as well as freeform speech without reading from a script. The training data was fed through the entire frontend pipeline before performing feature extraction to ensure that any changes in sample rate, compression, or distortion that occurred in this process would not create a bias in the model.

# 4.0 Testing and Evaluation

## 4.1 Accuracy

Testing the accuracy of our model was a process done in multiple stages. The first stages of the accuracy test would occur while the model was being trained. The collected data was already split between training, validation, and testing data to determine and measure the model’s accuracy. While the model was being trained during each epoch its accuracy in predicting the data it was being trained on would be output to see the model’s progress. At the end of each epoch, the model’s accuracy would be measured on the validation data which it would not have been trained on within that epoch but may have been trained on in a previous epoch. Finally, when the model was finished training it would be measured up against the testing data split. This is data that the model has not seen at all in its entire training process but was still split from the same original data that the model was trained on. These measures of accuracy are standard in training a neural network to measure a model’s ability to complete a task(site). This initial testing gave insight into the general performance of different models and the effectiveness of different strategies used. Testing accuracy specifically is what was used to compare against the original goal for accuracy. At this step, the model was able to achieve an accuracy of 95%, 25% higher than the original goal for the project. Past this more testing was completed to see how the model performed under different conditions. This included doing a round of testing on new pre-recorded data that was not a subset of the original data set that the model was trained on. This was included as a round of testing in case the original set of testing data was overly similar to the training and validation data giving skewed results. This round was found to give very similar results to the previous round of testing and found to be redundant so it was eventually phased out of the accuracy testing process. Past this, two more live tests were used on models that were found to have promising results. This included testing directly on the model with nothing else running. This was a test to see how the model’s accuracy was affected by noisier, more chaotic audio inputs. The groups would take turns speaking into the recording microphone varying parameters like distance from the microphone volume of voice and amount of background noise. The output from this test would be the model’s predictions at the end of each frame as well as a final guess as to who was speaking. This gave the group insight into whether the training data needed to be updated and if so, what new elements should be introduced into the training data to make the model more stable. This was followed by a test on the actual web interface with the model integrated within it. This was used to test how the model was able to handle the loss in the data quality as it got sent from the front end to the backend as well as the processing that was performed in the backend. These last 2 tests were difficult and inefficient to measure directly due to the data going into the model being life and not pre-labeled. Due to this, it was decided by the group that they would be required to be given exact measurements in these stages and instead were used to inform future developments to the training data and model.

A paper comparing LTSM (long short-term memory) speaker diarization models from 2018 found some achieved an accuracy as high as 88% [12], however it was noted that many of these errors are caused by faulty VAD (voice activity detection), and not issues with actual speaker identification.

## 4.2 Latency

The goal of the project was to achieve a latency on the product of under 1 second from when the audio first began to be recorded and when an initial transcript prediction was displayed to the user. It was important to the product that it did not have a latency significantly greater than 1 second due to the goal of the product conducting itself in real-time. If the output was displayed significantly after the audio was recorded, even if the process was happening in real-time, it would become a useless feature for the user, and they would only end up using the final version of the transcript created. This was measured using the timestamps of packages sent to the backend and comparing them with the timestamps of the packages received from the backend in response. The final product was able to consistently beat the original goal for latency, but the exact measurements varied based on the machine running the program and other variable factors common in web development.

## 4.3 Prediction Speed

A functional requirement of this project is that it must receive an input of live audio frames, then be able to output the identified speaker in real time. For this requirement, the speed is in “real time” so long as there is no perceived delay to the user. When live audio input was implemented, users were surveyed to determine whether they noticed any delays between speaking and receiving the annotated transcript. This would determine whether or not the system ran in “real time”. Most of the surveyed users determined that the system has no delays and does run in real time, therefore this requirement is considered to have been achieved. The results of this brief survey can be observed in Figure 7.

Figure 7 - Table of prediction speed delays as noticed by users

## 4.4 Processing Speed

The project requirements outline certain metrics for performance that must be met for the project to be considered successful. Processing speed is being defined as the time required for the neural network to make a prediction on a given audio frame. This excludes other parts of the system, such as the Voice Activity Detection (VAD), the speech-to-text service and the front end. This requirement demands that the processing speed is less than the length of the audio frame itself. This ensures that the model must be able to process audio at least as fast as it receives it. This prevents backlogs and queues of audio that are waiting to be processed.

A performance test was devised to test this metric. Timers were added to the code to properly benchmark the performance of the model. They only encompass the function that performs predictions, ensuring only the neural network prediction is being counted. The project was run for 5 minutes, resulting in 300 samples of 1 second audio clips, as seen in Figure 8. During this time, users were speaking consistently at a fairly rapid pace, so the model is constantly making predictions. The mean time required for the model to make a prediction on each 1 sec audio sample was 94 milliseconds. This by far exceeds the performance goal of 1000 milliseconds for the input of 1 second audio clips.

Chart, bar chart

Description automatically generated

Figure 8 - Processing speed of 300, 1 sec audio samples

## 4.5 User Interface

The project requirements demand that the model is accompanied by a graphical user interface. This provides the user with a cohesive way to use the model. Its purpose is to display the generated transcript along with the correctly identified and annotated speakers. To keep the interface simple for new users, there is a single button at the top of the page with a distinguishable microphone icon. This recognizable symbol will be easy for users to understand [13], and indicates the button to start the program. When new users were introduced to the program, it was observed that they knew immediately how to start the program. When the user opens the program, it will initially display a message indicating it is connecting to the model, as seen in Figure 9. If this succeeds, this will be displayed as seen in Figure 11. Otherwise, an error message will be displayed as seen in Figure 10. The possible messages are detailed below in Table 3. Once the backend has connected and the user selects the microphone button, the program will start retrieving live audio from the device’s audio input and passing it into the model. The accompanying transcript is displayed along with the identified speaker. The font used on the interface was chosen as it is easier to read for people with dyslexia [14]. Functional testing was conducted to ensure this button works as expected. If it is pressed before the backend is connected, nothing will happen.

Table 3 - User Interface Messaging

|  |  |
| --- | --- |
|  | Messages displayed to user |
| The user interface is connecting to the backend | “Connecting to Backend…” |
| The backend is connected and running properly | “Connected to Backend” |
| The interface cannot connect to the backend | “Failed to connect to Backend” |

Graphical user interface, text, application

Description automatically generated

Figure 9 - User Interface while it is connecting to the model

Graphical user interface, text, application

Description automatically generated

Figure 10 - User Interface when it is unable to connect to the model

Text

Description automatically generated

Figure 11 - Functioning User Interface

Compatibility testing was conducted on both Windows and Linux platforms. The project has been able to run successfully on both, without unusual setup requirements. The project has not been tested on Mac due to lack of access, however there is no reason to assume it would not work similarly on that platform as well.

# 5.0 Considerations

## 5.1 Social Considerations

The accuracy and effectiveness of the model for underrepresented groups could be influenced by the diversity of voice samples used to train the app. If the app is trained on a biased dataset that is not representative of the population, it may not work well for certain groups of people. If the dataset only includes voice samples from people of a certain race, gender, or age group, the model may not be accurate for individuals outside of that group. To address this, as the training dataset is expanded, the diversity of voices in the training set should be carefully monitored, and the models should be tested on a diverse set of participants.

## 5.2 Safety Considerations

The safety considerations for this project are primarily related to data security and privacy of the audio data used. The audio files of user speaking to train the general model must be stored securely and anonymized to protect the identities of the people contributing to the base dataset from malicious actors using the dataset to impersonate others. This will be addressed by securely storing the raw audio data offline, only making the model weights publicly available.

Audio data from users of the model must also be protected as this can contain privileged information, such as doctor’s notes on a patient or sensitive meeting notes. This is addressed by keeping all added user data and transcribed notes on the machine running the model, so that no sensitive data leaves the user’s machine.

## 5.3 Environmental Considerations

Training the model efficiently, especially as it expands, will require the use of a graphics processing unit (GPU) in order to do so in a reasonable amount of time. GPUs require a significant amount of energy to operate. This energy consumption contributes to greenhouse gas emissions and other pollutants that are released into the atmosphere. The manufacturing process of GPUs involves the use of rare and valuable metals such as gold, silver, copper, and aluminum. The extraction and refining of these metals can cause additional environmental damage, including deforestation, soil erosion, and water pollution. The production of GPUs requires large amounts of energy and water, which can have a significant environmental impact. Finally, the disposal of GPUs can be problematic. When they reach the end of their lifespan they often end up in landfills, where the materials used in their construction can release toxic chemicals into the soil and groundwater. While it is difficult to address the issues surrounding the manufacturing of the GPU, effort will be made to select GPUs that are energy efficient and dispose of them properly at the end of their lifespan.

## 5.4 Economic Considerations

There are few economic considerations for this project as it is software based and free to use. Possible economic considerations for the future of the project would be primarily related to any costs associated with acquiring training data, as well as computational power and infrastructure.

## 5.5 Regulatory Compliance

In Ontario, the collection and storage of personal data are governed by the Personal Information Protection and Electronic Documents Act (PIPEDA) at the federal level and the Ontario Freedom of Information and Protection of Privacy Act (FIPPA) at the provincial level [15]. Under these laws, organizations and businesses are required to obtain an individual's consent before collecting, using or disclosing their personal information, and must only collect information that is necessary for the purpose of the collection. Organizations must also take appropriate measures to protect personal information from unauthorized access, use, or disclosure.

Individuals have the right to access their personal information held by an organization and request corrections on any inaccuracies. Organizations must respond to such requests within a reasonable time and must provide access to the information requested, subject to certain limited exceptions.

The Ontario Privacy Commissioner recommends that individuals obtain consent before recording any conversation to avoid any legal issues that may arise. Additionally, it is important to note that recording someone without their consent may be seen as a breach of their privacy and could result in legal action. These laws will be accounted for by gaining written, informed consent from anyone who contributes audio data to the dataset, and ensuring the data is stored in accordance with PIPEDA and FIPPA.

## 5.6 Professional Considerations

All team members take the above considerations seriously and act with professionalism towards each other and any external parties. No audio data will be used without the speaker’s explicit consent after explaining to them how the data will be used.

# 6.0 Compliance with Specifications

Table 4 outlines the functional, interface, and performance requirements for the design as specified in the project blueprint. All requirements were met by the final product.

Table 4 - Project Specifications and their Completion Status

|  |  |  |
| --- | --- | --- |
|  | **Specification** | **Specification Met?** |
| **1** | **Functional / Non-functional requirements** | |
| 1.1 | Given an audio frame of a determined length with one of the team members speaking for the length of the frame, classify the speaker with 70% accuracy. | Yes |
| 1.2 | Given a live input of audio frames, output which person is speaking in real time. | Yes |
| **2** | **Interface requirements** | |
| 2.1 | Develop a user interface that displays a live transcription of the inputted speech, tagged to indicated who is speaking | Yes |
| **3** | **Performance requirements** | |
| 3.1 | Model must make a prediction on a given frame faster than the length of the frame itself. (If the frames are partitioned into 40ms intervals, the model must make a prediction on one of these frames in less than 40ms.) | Yes |
| 3.2 | There must be less than 1 second of latency between the beginning of an audio input and the model making a prediction on the input. | Yes |

# 7.0 Conclusions and Recommendations

## 7.1 Future work

If given more time to develop the product there are a few elements that would be beneficial to the product's success and viability for general use. Primarily the additions to add to the project are improving the model's ability to scale, improving its versatility, and introducing a continuous learning component into the model.

### 7.1.1 Improve Scalability

In the final stage of this project, the model can distinguish between 4 voices. With its current state, the model decreases in accuracy for every additional voice added to the model. If this product was to be implemented in workplaces it would need to be able to distinguish between hundreds of voices which would bring down the accuracy far below the 70% minimum accuracy that this group hoped to achieve. For the product to maintain its high-level accuracy a different strategy would have to be implemented in its training. First, the number of voices within the training data would have to expand well past the number of individuals that are expected to use a single instance of the product. This would potentially include thousands of individuals. The model could then be trained to distinguish these voices. This will create a model that is more generalized in its ability to find what makes a voice unique from another given voice. From here the user would have to specify to the program how many users they wish the model to recognize. The data for these users would have to be recorded and fed into the model for retraining. The model architecture itself would have an additional final layer added to the end of it equal to the size of the number of users entered. The model would then be retrained with a much smaller learning rate to slightly adjust the parameters to learn the specific voices entered. This would create a more stable model as the number of voices approaches larger values. It may also have the effect of needing less data for each new speaker added to the model.

### 7.1.2 Improve Versatility

The model could also use improvements in its versatility of environments and situations in which it might be used. Currently, this model best works with one speaker at a time speaking into a single microphone within a fixed range from the microphone. To improve this model and its user experience it will need to be able to handle voices speaking from ranges of up to approximately 15m away from the microphone. This will allow users to have a single recording system at the center of a meeting room and be able to transcribe the conversation of all the individuals accurately. This would require adding training data for each speaker from varying distances. This would have an unwanted effect of the recordings also containing more noise and may require further preprocessing to ensure that just the voices are extracted from a recording before being fed into the model. The product could also benefit from greater consistency in its accuracy between different microphones. All the training data was recorded on a singular high-quality microphone to ensure the best results on the final accuracy of the model and to ensure it was not unintentionally learning the differences between microphones instead of voices. This left our model’s accuracy more sensitive to the microphone recording testing audio. This was seen when comparing audio from a laptop's embedded microphone versus the original recording microphone. Recording the training audio on a variety of microphones should create some increased resiliency to the model. There are potentially more preprocessing techniques that could assist in this process, but more research would be required to determine if it could correct this problem in its entirety. Related to the previous improvements mentioned in this section, the model displayed a lesser but still surprisingly good performance on accurately detecting voices through a video call. This gives promise to this product being used in hybrid meeting environments as well as in-person meeting environments. Using the techniques implemented in Zoom’s transcription service of labeling users based on the label of the microphone would only help if all devices joining remotely are individuals and not groups of people. If each remote device joining may contain multiple users, then the model must be able to handle the distortion to the voices that occur in between the recording of the voice on the remote device to rerecord the voice on the microphone feeding audio into the model. This will require further expanding the training data to account for this distortion when training the model.

### 7.1.3 Introduce a Continuous Learning Component

Introducing a continuous learning component into the model was a stretch goal for this project but was unable to be met due to time constraints. Currently to add a new voice to the model, the entire model must be completely retrained. This would become increasingly difficult as the number of voices in the model increases causing training time for the model to grow respectively. This would create poor user experiences if a company of a hundred or more employees wished to add a new speaker to the model and had to wait days for a new user to be recognized by the model. By implementing a continuous learning component into the model the time to add a new speaker would become constant allowing businesses to add and remove voices in the model much more easily. To complete this addition the model would already ideally have improved its ability to scale discussed in 7.1.1 Improve Scalability. The product would then have to automate partial retraining of the model when a new voice is requested to be added to the system. This would require using dynamic architecture methods to grow the model as new voices are added to the system. This is seen as an ideal approach to continual learning for this product as the new voices being added are incremental, the model can be trained offline, and accuracy is the greatest priority on new and old training data [16]. For the architecture to not become overly large as many voices are added over time the model would incrementally have to be retrained on all of the voices into a more compact architecture. Finally, a system would have to be implemented in the background to monitor and adjust the model in case catastrophic forgetting occurs, which is a common problem with continuous learning models [16].

## 7.2 Additional Learning

Through designing and developing this system, the team was immersed in the current state of speaker identification and speech-to-text systems available as paid products and open-source implementations. Through this exposure, the team learned of the exciting direction this field is heading toward as state-of-the-art machine learning-based models for speech-to-text, diarization, and classification are being open sourced by researchers. This has given undergraduate students with no advanced knowledge of such topics the power to design and integrate each component into a useful and novel product that many individuals could stand to benefit from. The team also learned of various software tools for frontend development, client-server communication, web servers, multithreading, audio processing, machine learning, and more. Given the limited scope, time, budget, and prior knowledge, the team was not able to explore every possible aspect of the system. The team identified multiple areas for further research listed above in 7.1 Future work that would increase the novelty and usefulness of this system.

## 7.3 Societal Impact

Prior to this project there was not a free and open-source, state-of-the-art, locally running, intuitive speech transcription and speaker identification system available to run on consumer computer hardware. It is possible to bring this product to market and monetize its features for capital gain. However, one of the key requirements the team set out was to create a product that was free and open source with the goal of lowering the barrier to accessibility for those who can’t afford or understand how to use existing solutions.

The team believes the solution outlined in this report could have an impact on existing products, for example: Nuance Communications Cloud-based Dragon Speech Recognition and Dragon Medical Dictation software [17]. This software is sold to enterprise businesses, lawyers, and doctors to transcribe speech accurately and quickly. At a cost of $500 per license for professional use, the release of a similarly performing free and open-source solution could undermine Nuance’s product offering in addition to opening the usefulness of live transcription to millions of those who cannot afford it.

Discussion online finds that a limiting factor for many businesses adopting transcription services is confidentiality. As majority of the available product offerings for transcription and identification run in the cloud, many companies (especially legal, government, and medical) are hesitant to have internal confidential conversations or patient dictations sent to a 3rd party off-premises [18].

## 7.4 Conclusion

In conclusion, this project has created a free and open-source speech transcription and speaker identification system that is locally running, intuitive, and accessible. By prioritizing affordability and data privacy, this solution has the potential to disrupt existing products and benefit users who previously faced access barriers. Overall, this project has the potential to make a significant impact on the market for transcription services and empower users who cannot afford or understand existing solutions.

# References

|  |  |
| --- | --- |
| [1] | K. Ungarsohn, "Verbatim transcripts for technical communication," *IEEE Circuits and Devices Magazine,* vol. 12, no. 5, pp. 51-53, 1996. |
| [2] | S. Kawas, G. Karalis, T. Wen and R. E. Ladner, "Improving Real-Time Captioning Experiences for Deaf and Hard of Hearing Students," *Proceedings of the 18th International ACM SIGACCESS Conference on Computers and Accessibility,* pp. 15-23, 2016. |
| [3] | R. Shadiev, W.-Y. Hwang, N.-S. Chen and Y.-M. Huang, "Review of Speech-to-Text Recognition Technology for Enhancing Learning," *Journal of Educational Technology & Society,* vol. 17, no. 4, pp. 65-84, 2014. |
| [4] | B. Turner, "Techradar.com," Techradar, 02 11 2022. [Online]. Available: https://www.techradar.com/news/best-speech-to-text-app. [Accessed 10 04 2023]. |
| [5] | C. Guttandin, "extendable-media-recorder GitHub: An extendable drop-in replacement for the native MediaRecorder.," 14 December 2019. [Online]. Available: https://github.com/chrisguttandin/extendable-media-recorder. [Accessed 7 January 2023]. |
| [6] | A. Veysov, "Silero VAD: pre-trained enterprise-grade Voice Activity Detector, Language Classifier and Spoken Number Detector," 23 November 2020. [Online]. Available: https://github.com/snakers4/silero-vad. [Accessed 9 January 2023]. |
| [7] | "MFCC Technique for Speech Recognition," Analytics Vidhya, 13 June 2021. [Online]. Available: https://www.analyticsvidhya.com/blog/2021/06/mfcc-technique-for-speech-recognition/. [Accessed 4 April 2023]. |
| [8] | C. Maklin, "Fast Fourier Transform," Towards Data Science, 29 December 2019. [Online]. Available: https://towardsdatascience.com/fast-fourier-transform-937926e591cb. [Accessed 4 April 2023]. |
| [9] | Open AI, "Whisper," Open AI, 1 September 2022. [Online]. Available: https://github.com/openai/whisper/. [Accessed 4 November 2022]. |
| [10] | Whisper Contributors, "Possible to use for real-time / streaming tasks?," Open AI, 21 September 2022. [Online]. Available: https://github.com/openai/whisper/discussions/2. [Accessed 3 January 2023]. |
| [11] | Amazon, "Mic Shure SM58S (with switch)," Amazon, 2022. [Online]. Available: https://www.amazon.ca/gp/product/B0000AQRSU/. [Accessed 22 November 2022]. |
| [12] | Q. Wang, C. Downey, L. Wan, P. A. Mansfield and I. L. Moreno, "Speaker Diarization with LSTM," *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP),* pp. 5239-5243, 2018. |
| [13] | S. Rosenbaum, J. O. Bugental and Tec-Ed, "Measuring the Success of Visual Communication in User Interfaces," *Society for Technical Communication’s,* vol. 45, no. 4, 1998. |
| [14] | L. Rello and R. Baeza-Yates, "Good fonts for dyslexia," *ASSETS '13: Proceedings of the 15th International ACM SIGACCESS Conference on Computers and Accessibility,* pp. 1-8, 2013. |
| [15] | Office of the Privacy Commissioner of Canada, "The Personal Information Protection and Electronic Documents Act (PIPEDA)," Office of the Privacy Commissioner of Canada, 2023. [Online]. Available: https://www.priv.gc.ca/en/privacy-topics/privacy-laws-in-canada/the-personal-information-protection-and-electronic-documents-act-pipeda/. [Accessed 6 April 2023]. |
| [16] | G. Parisi, R. Kemker, J. Part, C. Kanan and S. Wermter, "Continual Lifelong Learning with Neural Networks:," *Neural Networks,* vol. 113, no. 0893-6080, pp. 54-71, 2019. |
| [17] | Nuance Communications, "Dragon Speech Recognition Solutions," Nuance Communications, 2023. [Online]. Available: https://www.nuance.com/dragon.html. [Accessed 30 March 2023]. |
| [18] | K. Hashizume, D. Rosado, E. Fernández-Medina and e. al, "An analysis of security issues for cloud computing," *Journal of Internet Services and Applications,* vol. 4, 2013. |

# Appendices

## Appendix A: Team Contributions

Table 5 - Table of team contributions throughout the project

|  |  |
| --- | --- |
| Name | Overall Effort Expended |
| Hannah Berthiaume | 100 % |
| Aiden Horan | 100 % |
| Parker Rowe | 100 % |
| Adam Strom | 100 % |

## Appendix B: Major Code Blocks

### User Interface

Text

Description automatically generated

Figure 12 - Manual .wav header insertion function. This header is prepended to each audio chunk before being sent to the backend

### Data Preprocessing

A screenshot of a computer

Description automatically generated

Text

Description automatically generated

Figure 13 - Script for extracting MFCCs from an audio file and saving them to a Numpy data file

### Backend Server Code

from datetime import datetime, timedelta

import tempfile

import logging

import concurrent.futures

import numpy as np

import librosa

import soundfile as sf

import tensorflow as tf

import torch

import whisper

from flask import Flask

from flask\_socketio import SocketIO, send, emit

from utils.extract\_features import extract\_features

from utils.vad import is\_speaking

from utils.strip\_wav\_header import strip\_wav\_header

from utils.SpeakerHistory import SpeakerHistoryQueue

print("Num GPUs Available to TensorFlow: ", len(tf.config.list\_physical\_devices('GPU')))

transcription\_model = whisper.load\_model("tiny.en")

speaker\_model = tf.keras.models.load\_model("../saved\_models/4\_classes-04\_05\_2023\_22\_58\_25", compile=False)

speaker\_model.compile()

app = Flask(\_\_name\_\_)

socketio = SocketIO(app, cors\_allowed\_origins="\*", logger = False)

logging.getLogger("werkzeug").setLevel(logging.ERROR) # flask

logging.getLogger("socketio").setLevel(logging.ERROR) # socketio

logging.getLogger("engineio").setLevel(logging.ERROR)

FILE\_NAME = "audio"

audio\_bytes = b""

is\_first = True

torch\_use\_gpu = torch.cuda.is\_available()

speakers = ['hannah', 'aiden', 'parker', 'adam']

total\_transcript = []

# running phrase vars

last\_output = None

phrase\_audio\_buffer = b""

phrase\_timeout = 2 # seconds of silence required for a new phrase to start in live transcription

last\_phrase\_time = None

phrase\_id = 0

phrase\_class\_counts = {speaker: 0 for speaker in speakers} # creates a dictionary where the keys are the speaker names

                                                           # and the values are a counter of how many times the speaker

                                                           # detection thought it was them in a running phrase

running\_speaker\_history = SpeakerHistoryQueue(speakers)

"""

Write a blob of wav data as a temporary file where we can then load that

file and predict who is speaking in it using librosa and our trained

model.

"""

def save\_chunk\_to\_tempfile(blob):

    # create and save temp file

    fp = tempfile.NamedTemporaryFile()

    # print("\nTEMP FILE NAME" + fp.name)

    fp.write(blob)

    fp.seek(0)

    return fp

def save\_wav\_16(bytes: bytes, file\_name):

    # write audio bytes to wav file

    with open(file\_name + ".wav", "wb") as file:

      file.write(bytes)

    y, sr = librosa.load(file\_name + ".wav", sr=16000) # Downsample 44.1kHz to 16kHz for whisper to be able to read it

    sf.write(file\_name + "\_16.wav", y, sr, "PCM\_16")   # save the 16KHz file as 16 bit (also for whisper)

def transcribe(file\_name):

    result = transcription\_model.transcribe(file\_name, fp16=False)

    text = result['text'].strip()

    return text

"""

Identifies who is speaking in a given audio file and returns the result.

"""

def identify\_speaker(file\_name, average=False):

    duration = 1000 # ms - duration of the split audio clip in ms

    if average:

        sr = librosa.get\_samplerate(file\_name)

        print("SAMPLE RATE: ", str(sr))

        stream = librosa.stream(file\_name,

                            block\_length=64,

                            frame\_length=int(sr \* duration / 1000),

                            hop\_length=int(sr \* duration / 1000) \* 0.5)

        predictions = []

        # create 40ms clips of audio features to feed into the model and get the output

        for y in stream:

            normalized\_y = librosa.util.normalize(y)

            test\_frame\_features = extract\_features(normalized\_y, sr)

            pred = speaker\_model.predict(test\_frame\_features.reshape(1,len(test\_frame\_features)))

            idx = np.argmax(pred)

            predictions.append(speakers[idx])

        print("PREDICTIONS: " + str(predictions))

    else:

        y, sr = librosa.load(file\_name, offset=0)

        # print("SAMPLE RATE: ", str(sr))

        normalized\_y = librosa.util.normalize(y)

        test\_frame\_features = extract\_features(normalized\_y, sr)

        pred = speaker\_model.predict(test\_frame\_features.reshape(1,len(test\_frame\_features)))

        idx = np.argmax(pred)

        current\_speaker = speakers[idx]

        return current\_speaker

@socketio.on("connect")

def handle\_message(data):

    print("===============| Browser Client Connected! |===============\n")

@socketio.on("begin\_transcription")

def handle\_message(data):

    global is\_first, audio\_bytes, total\_transcript, last\_phrase\_time, phrase\_id, running\_speaker\_history

    audio\_bytes = b"" # clear the saved audio from the previous transcription

    is\_first = True

    total\_transcript = []

    last\_phrase\_time = None

    phrase\_id = 0

    running\_speaker\_history = SpeakerHistoryQueue(speakers) # clear the queue of speaker history

    print("===============| begin\_transcription |===============\n" + str(data))

    emit("ready\_to\_receive\_chunks")

@socketio.on("audio\_chunk")

def handle\_message(data):

    global is\_first, audio\_bytes, silence\_time, last\_phrase\_time, phrase\_audio\_buffer, phrase\_id, phrase\_class\_counts, running\_speaker\_history, last\_output

    # update the variables for the total running audio after each audio\_chunk is received

    audio\_bytes += data if is\_first else strip\_wav\_header(data) # if its not the first chunk, get rid of WAV header (first 44 bytes)

    is\_first = False

    # handle each audio chunk for live transcription

    # print("\nreceived audio chunk:\n", data)

    file\_pointer = save\_chunk\_to\_tempfile(data)

    temp\_file\_path = file\_pointer.name

    # initialize new\_phrase as False if we've spoken before, otherwise start it as true for the first run

    new\_phrase = False if last\_phrase\_time else True

    now = datetime.utcnow()

    # if its been more than 2 seconds since the last time a voice was detected AND

    # this isn't the very first run through (last\_phrase\_time would be None), then:

    #

    # the phrase is complete and we can clear the current phrase buffers

    if last\_phrase\_time and now - last\_phrase\_time > timedelta(seconds=phrase\_timeout):

        phrase\_audio\_buffer = b""

        new\_phrase = True

        phrase\_class\_counts = {speaker: 0 for speaker in speakers}

        running\_speaker\_history = SpeakerHistoryQueue(speakers) # silence, we don't care about detecting a speaker switch since we have a new phrase

        phrase\_id += 1 # incremement phrase ID if we're onto a new one

        print("Silence for at least 2 seconds... new phrase ")

    # if our current best guess of a speaker for the phrase has not been the majority of the last 5 seconds,

    # its time to start a new phrase since there's a good chance someone else started talking

    best\_guess\_for\_phrase = max(phrase\_class\_counts, key=phrase\_class\_counts.get)

    majority\_of\_last\_few\_guesses = max(running\_speaker\_history.counts, key=running\_speaker\_history.counts.get)

    if best\_guess\_for\_phrase is not majority\_of\_last\_few\_guesses:

        phrase\_audio\_buffer = b""

        new\_phrase = True

        phrase\_class\_counts = {speaker: 0 for speaker in speakers}

        phrase\_id += 1 # incremement phrase ID if we're onto a new one

        print("Someone else is speaking! New phrase ")

    # Voice Activity Detection - check if someone was speaking in this audio clip

    if is\_speaking(temp\_file\_path):

        last\_phrase\_time = now

        # append this audio to the buffer since the person is speaking.

        # if the phrase is a new one, start by keeping wav header, otherwise append chunks with no wav header

        phrase\_audio\_buffer += data if new\_phrase else strip\_wav\_header(data)

        # TODO better file name handling

        print("saving audio buffer, new\_phrase: " + str(new\_phrase))

        with open("phrase.wav", 'w+b') as f:

            f.write(phrase\_audio\_buffer)

        # create a threadpool to execute the two longest running functions (transcribe and identify\_speaker) in parallel, then wait for their executions to be complete

        try:

            with concurrent.futures.ThreadPoolExecutor(max\_workers=2) as executor:

                future1 = executor.submit(transcribe, "phrase.wav")            # live transcribe on the audio segment

                future2 = executor.submit(identify\_speaker, temp\_file\_path)    # make a prediction on who is speaking in the chunk

                running\_transcript = future1.result()

                current\_speaker\_guess = future2.result()

            phrase\_class\_counts[current\_speaker\_guess] += 1         # increment the count of how many times this speaker has been guessed for this phrase

            print("PHRASE CLASS COUNTS: " + str(phrase\_class\_counts))

            running\_speaker\_history.enqueue(current\_speaker\_guess)  # add the current speaker to the speaker history queue

            best\_speaker\_guess = max(phrase\_class\_counts, key=phrase\_class\_counts.get)  # the best guess of the current speaker is the one who has been guessed for majority of the phrase

            output = {

                "id": phrase\_id,

                "text": running\_transcript,

                "speaker": best\_speaker\_guess

            }

            last\_output = output # save this to use later if needed

            print("OUTPUT: ", str(output))

            emit("transcript\_update", output)

            # since we just got someone talking, prepare for more talking from them

            new\_phrase = False

        except:

            print(".\n") # hide the nasty whisper tensor size errors

    # END IF (is\_speaking)

    # finally, close the pointer to the temp file -- this should delete the tempfile too

    file\_pointer.close()

@socketio.on("end\_transcription")

def handle\_message():

    print("\===============| end\_transcription |===============\n")

    save\_wav\_16(audio\_bytes, FILE\_NAME)

    final\_transcript = transcribe(FILE\_NAME + ".wav")

    print("FINAL TRANSCRIPT:\n" + final\_transcript)

    emit("finished\_transcription", final\_transcript)

if \_\_name\_\_ == "\_\_main\_\_":

    socketio.run(app)

Figure 14 - Backend server code, main file: backend/app.py

## Appendix C: Project Planning

Table 6 - Project timeline as defined in the blueprint report

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| No. | Milestone | Due date | Date Completed | Responsible member(s) |
| 1 | Start researching methods for voice classification | Week 1 | Week 1 | All |
| 2 | Set up a private GitHub repository to store experimenting and final code | Week 2 | Week 1 | Parker |
| 3 | Research and project start buffer period | Weeks 2 -4 | Weeks 2 -4 | All |
| 4 | Create a rough plan for iterative approach for a complete voice classifier | Week 5 | Week 5 | All |
| 5 | Start regular code experimentation with librosa, Tensorflow, Keras, and other aspects to get familiar | Week 5 | Week 5 | All |
| 6 | Experimentation and planning buffer period | Weeks 5 - 7 |  | All |
| 7 | 1st Meet with Dr. Seyedin – Set Plan Based on Feedback on Proposal | Week 7 | Week 7 | All |
| 8 | Purchase Microphone | Week 7 | Week 9 | Aiden |
| 9 | Buffer period for getting microphone purchase approved and shipped | Weeks 7 - 9 |  |  |
|  | Meeting with Dr. Seyedin – to discuss audio pre-processing |  | Week 9 | All |
| 10 | Set up microphone and begin recording training data | Week 10 | Week 10 | All |
| 11 | Finish Recording Training Data | Week 11 | Week 11 | All |
| 12 | Pipeline for splitting training data & extracting audio features | Week 11 | Week 11 | Adam |
|  | Meeting with Dr. Seyedin – to discuss further pre-processing and audio parameters |  | Week 12 | All |
| 13 | Start implementing Text-to-speech (using external library) w/GUI | Week 12 | Week 12 | Parker |
| 14 | Start to Train & Test Binary Voice Classifier (is it this person? Yes or No) | Week 13 | Week 13 | Aiden + Hannah |
| 15 | Start working on a pipeline for using buffered audio input as an input for deep learning model | Week 13 | Week 13 | Adam |
| 16 | Buffer period for experimenting with training & testing a basic voice classifier | Weeks 13 - 15 | Weeks 13 - 15 |  |
| 17 | Finish Basic Binary Voice Classifier with audio input is a file | Week 15 | Week 14 | Aiden + Hannah |
|  | First-year Discipline Night |  | Week 15 | All |
| 18 | Finish Text-to-speech w/GUI | Week 16 | Week 16 | Parker |
| 19 | Finish Pipeline for buffered audio input for model (now the model can classify live audio) | Week 16 | Week 16 | Adam |
| 20 | Start collecting external datasets of voices for use in a generalized model for classifying new voices on-the-fly | Week 16 | Week 15 | Adam |
| 21 | Finish training & testing a voice classifier than can differentiate between 2 or more predetermined individuals. | Week 17 | Week 17 | Hannah + Aiden |
| 22 | Finish collecting external voice data for on-the-fly-recognition | Week 17 | Week 17 | Adam |
| 23 | Start training & testing a deep learning model for generalized voice classification of new voices | Week 17 | Week 16 | Hannah + Aiden |
|  | Meeting with Dr. Seyedin – to troubleshoot audio processing pipeline issue |  | Week 19 | All |
| 24 | Buffer period to work on final model | Week 17 - 20 | Week 19 |  |
|  | Meeting with Dr. Seyedin – how to increase accuracy |  | Week 20 | All |
| 25 | Finish generalized deep learning model | Week 20 | Week 20 | Hannah + Aiden |
| 26 | Buffer period to finalize details and prepare for demo | Week 20 - 22 | Week 21 | All |
| 27 | Open House | Week 22 | Week 22 | All |
| 28 | Final deliverable | Week 23 | Week 23 | All |
| 29 | Final project report | Week 24 | Week 24 | All |