A Real-Time Speech Processing System for Medical Conversations

by

Jing Bo Yang

Supervisor: Michael Brudno
Collaborator: Jixuan Wang

April 2018

This page is intentionally left blank.

A Real-Time Speech Processing System for Medical Conversations

B.A.Sc. Thesis

by

Jing Bo Yang

Division of Engineering Science
April 2018

Supervisor
Michael Brudno
Professor
Department of Computer Science

Collaborator
Jixuan Wang
PhD Candidate
Department of Computer Science

Abstract

Tablet computers equipped with note-taking applications have started to shift people away from taking notes on paper, but the medical community has yet to benefit from this growing trend. Doctors today are still struggling to take notes during interviews, primarily due to stress, time constraint, and the amount of information that has to be recorded. Current note-taking applications only focus on providing convenient user interface; and smart meeting systems often aim at recording the meeting for remote attendance. Neither of the two are suitable for interview style conversations. To address doctors' demand, we have developed an application that acts as a note-taking platform and it is capable of processing audio live during the conversation. In real time, audio stream is diarized and transcribed, then displayed to the user in a chat-like interface. We believe this application, especially the underlying infrastructure for collecting structured data, is an important step toward reducing doctors' workload and for providing real-time inference based assistance. More work is already underway to address issues associated with speech diarization and to improve efficiency of the audio processing system.

Acknowledgment

First of all, I would like to thank my supervisor, Professor Michael Brudno from Hospital for Sick Children, for providing me with this excellent opportunity to work on audio processing. Professor Brudno had always been accommodating and willing to offer assistance. I also enjoyed conversing with him on a variety of topics after our formal meetings.

Of course, this work could not have been completed without the unrelenting support I received from PhD. candidate Jixuan Wang. It was a great pleasure to work with Jixuan. His excellent knowledge in software engineering and perseverance in academic research had been tremendous during the past year.

I would like to appreciate the emotional support I received from other members of Professor Brudno's research group. They are Aryan Arbabi, Sana Kawar, Menghan Chen, Mohammad Firouzi, and Nicole Sultanum.

I would also like to extend my thanks to to Andrew Maksymowsky and Krista Pace for their technical and administrative assistance. It would not have been possible to gain access to various facilities from SickKids without their help.

Last but not least, I would like to thank my parents for encouraging me during many difficult times throughout the year and my life in general.

Table of Contents

List of Figures

List of Tables

1 Introduction

Note taking is an essential component for doctor-patient encounters. These notes contain a variety of information ranging from personal information to diagnosis and prescriptions. Most of these notes are created during conversations or immediately after. The difficulty associated with note-taking is not only related to the amount of information that needs to be processed, but also the fact that doctors are under stress and strict time constraint to complete these tasks. Therefore, we should not rely on doctors to record every detail of the meeting, nor expect theses note to contain all critical information.

It goes without saying that medical record is one of the most important source of information in doctors' decision making process. Hence there is certainly the need, and strong demand, as demonstrated by interests shown during our presentation sessions, for an intelligent note-taking application that could reduce doctors' workload and ensure notes are as comprehensive as possible. In addition, infrastructures laid down for an intelligent note-taking system will also be useful for providing live feedback to doctors, thus acting as an extra source of assistance during interviews. Without doubt, live audio processing of doctor-patient encounter would be a great non-intrusive way to collect data for the said live assistance. In fact, Mize et al.[1] have found that interviews in which doctors focuses on computer monitors are rated as less-personal and the doctors being less experienced. This finding further stresses the importance of having a non-intrusive method of data capturing during medical conversations.

Numerous studies have been done on automated multi-media information processing. For example, Banerjee and Rudnicky[2] proposed SmartNotes to process meeting data. However, their work focuses more on note sharing, not improving the experience of note creation nor collecting audio data.

Prior to our work, the research team had started developing a Windows-based intelligent note-taking application, PhenoPad. The team had been actively promoting it to medical specialists for its potential to replace traditional note-taking methods. PhenoPad provides an integrated interface that collects hand-written notes, audio and images. Text data can already be relayed to an inference engine built to generate rel-

evant medical information. The ultimate goal of PhenoPad reaches far beyond that of traditional note-taking applications. We believe its data collection infrastructure is only the starting point of a computer program that will fundamentally change doctors' experience.

For this project, we have built a complete audio processing system for PhenoPad to collect and process audio data during doctor-patient encounters. During the process, PhenoPad's audio-related user interface had been re-vamped for more informative display. Furthermore, a Ubuntu-based audio processing server with the capability to process audio streams from multiple clients had been set up. We have also started exploring the possibility of integrating a microphone array and a camera into our system for more convenient device connection and more accurate results.

This thesis will first introduce necessary background related to speech processing. Then, we will explain how our systems are implemented and integrated together. We will talk about empirical results and propose new approaches that could improve our application. Since this project has a strong focus on implementation rather than attempting to improve on the current state-of-art algorithms, we will place more emphasize on implementation details in the following sections.

Before we begin, it is important to understand that speech recognition and speech diarization are two similar yet unique fields of study. Speech recognition aims to understand what is spoken, which needs to draw upon an existing vocabulary of known words and how they should be pronounced. In contrast, speech diarization answers the question of who is speaking. Trying to answer this question without knowing audio characteristics of attending speakers makes the problem much harder. The upcoming literature review and previous works section (Section 2) explains each of these two problems in detail.

2 Literature Review and Previous Work

In this section, we explore methodology of previous academic works on speech diarization and existing implementation of diarization and recognition algorithms. Since we have oriented this project toward implementing a live audio processing system for real-world applications, we primarily focused on experimenting with existing implementations and re-producing results of past academic works.

For the rest of this report, we will explicitly state whether the topic is related to speech diarization or speech recognition. As mentioned in Introduction (Section 1), speech diarization is process used to answer the question of "who spoke when", where as speech recognition provides answer for "what is being said". We also use ASR (automatic speech recognition) interchangeably with speech recognition. We refer to a combination of these two capabilities as "speech processing".

2.1 Basic Audio Processing

We start with basic audio processing functions because they are common to many audio processing related tasks.

Voice Activity Detection Voice activity detection is the attempt to identify audio segments that contain speech. An naive method is to compute RMS (root-mean-square) power1 of an utterance. However, this method is susceptible to level of ambient noise and quality of audio input device.

$$is\_speech = \sqrt{\frac{1}{N}\sum_{n=1}^{N} x_n^2} > threshold \tag{1}$$

Aside from this naive approach, there had been other approaches that utilize phonetic features, such as those done by Barabanov et al.[3] There had also been attempt to use DNN (deep neural networks) to reduce computational cost of voice activity, as demonstrated by Ko et al.[4] Ko's method runs 3.7× faster and has 9.54% (relative) im-

provement in accuracy against WebRTC VAD.

These developments have not challenged the position of WebRTC VAD [5], which had been one of the most popular toolkit for voice activity detection. From reading source codes of WebRTC VAD [6], it is obvious that Google's implementation uses a pre-trained Gaussian Mixture to determine whether a given audio frame contains speech.

Mel Frequency Cepstrum Coefficient MFCC is a feature on which almost all audio processing algorithms are based. It is a further processing of FFT (Fast Fourier Transform) power coefficients, treating these power terms as a time series signal. Mel-scale can be expressed as shown in Eqn 2. Log10 of these terms are often taken to retrieve decibel-level values.

$$m = 2595 log_{10}(1 + \frac{f}{700}) \tag{2}$$

Overall, steps required to compute MFCC can be summarized as the following. Note that these steps are taken for each sliding window. Size of the sliding window and offset between windows depend on application.

1. Compute FFT of a small window (typically 10-30 ms)
2. Compute Mel-scale power of FFT results
3. Convert Mel-scale power terms into decibel-powers by taking logarithms
4. Compute DCT (Discrete Cosine Transform) on the result
5. Re-scale MFCC results if necessary

2.2 Speech Recognition

Speech recognition algorithms have evolved substantially. Researchers have moved on from computing Dynamic Time Wrap distances to identify individual words, toward

utilizing n-gram models and Hidden Markov Models to perform syllable-level matching. In addition to MFCC features, researchers have used i-Vectors for acoustic modeling, with varying degrees of success[7][8]. Today, neural network-based models have also been developed to take advantage of recent advances in machine learning.

Kaldi - An Open Source Implementation Kaldi is a well-known state-of-art audio processing toolkit developed Povey et al[9]. Kalid has become popular because it provides numerous audio processing functions, including those described in Section 2.1, as well as wrappers for scientific computing kits LAPACK[10] and BLAS[11]. These scientific computing libraries and OpenFST[12] form the backbone of Kaldi. Users typically choose GMM for syllable level classification then use HMM (or in other words, Finite State Transducer) for forming complete words. In addition, Kaldi also supports SRILM[13], a language modeling toolkit, for processing n-gram language models.

These features make Kaldi a flexible, albeit having a high learning curve, platform to build customized language models and to experiment with different speech recognition algorithms. Recently, Kaldi integrated neural network based recognition system, created and tested by Vesely [14]. This new system, using DNN-HMM modeling, is 7% more accurate than the traditional GMM-HMM models. Vesley's paper on DNN based acoustic modeling found no difference between cross-entropy trained models and sequence-discriminative trained models, both of which can reach word error rate (WER) of as low as 18.4%.

Numerous language models already exist as part of Kaldi's default distribution package. Differences among these pre-trained models primarily lie in training datasets and acoustic structures. For example, the ASpIRE challenge [15] provides a model trained using UPenn's Fisher English Training Speech [16]. This audio corpus is extracted from English conversational telephone speech (CTS). Although Fisher's corpus only contains common English words, its spoken nature is well-suited for the purpose of understanding conversations. This model has a word error rate (WER) of 15.6%, higher than 8.57% of an SRE16 Xvector Model [17], which is developed based on an i-Vector based receipt [18]. The Xvector model is a further improvement over iVector model.

2.2.1 Commercial Speech Recognition Platforms

We have never attempted to implement a speech recognition system in-house because ASR platforms are widely available from numerous different sources. Unlike Kaldi, which is the most popular speech recognition platform for research, there exists many commercial speech recognition APIs. Unfortunately, we can only use these platforms for testing purposes, due to internal regulations on data privacy imposed by Hospital for Sick Children.

Google From a glimpse of Google's own description of its speech API [19], the best feature that it offers is it's ability to work in a noisy condition. As expected, Google is using its machine learning platform for the execution of its speech recognition API. From the experience of using Google's API, we discovered that it is capable of recognizing a wide range of terms, including those used for medical purposes. This is confirmed by Google's claim that it can tailor the speech models according to context of speech. It also recently added word-alignment, allowing us to match speech diarization result with ASR text. This capability is crucial because we wish to create conversation-style interface in the application.

IBM IBM offers both speech recognition and speech diarization capabilities on its BlueMix platform [20]. Although it offers similar styled JSON results with word-alignment output, IBM's API only works well in quiet environments. Empirical testing using audio snippet of an Youtube interview [21] shows that its speech recognition engine is less accurate than that offered by Google and its diarization accuracy is also questionable.

Microsoft Microsoft also offers a speech recognition engine via its Azure cloud [22]. It does not offer a diarizaiton engine like IBM, but it does provide capability to verify speaker identity using existing user profile [23]. Given the sample application provided by Microsoft, we can summarize its workflow in Fig 1. This interface is able to enroll a speaker with a short audio, then matches subsequent audio segments with existing user profiles. A confidence level is reported back to application developers once the API has finished processing. More users can be enrolled at runtime, with maximum enrollment

audio length of 1 minute. Notice that Microsoft's speaker identification interface is different from Microsoft's speaker verification API, which requires more enrollment audio segments.

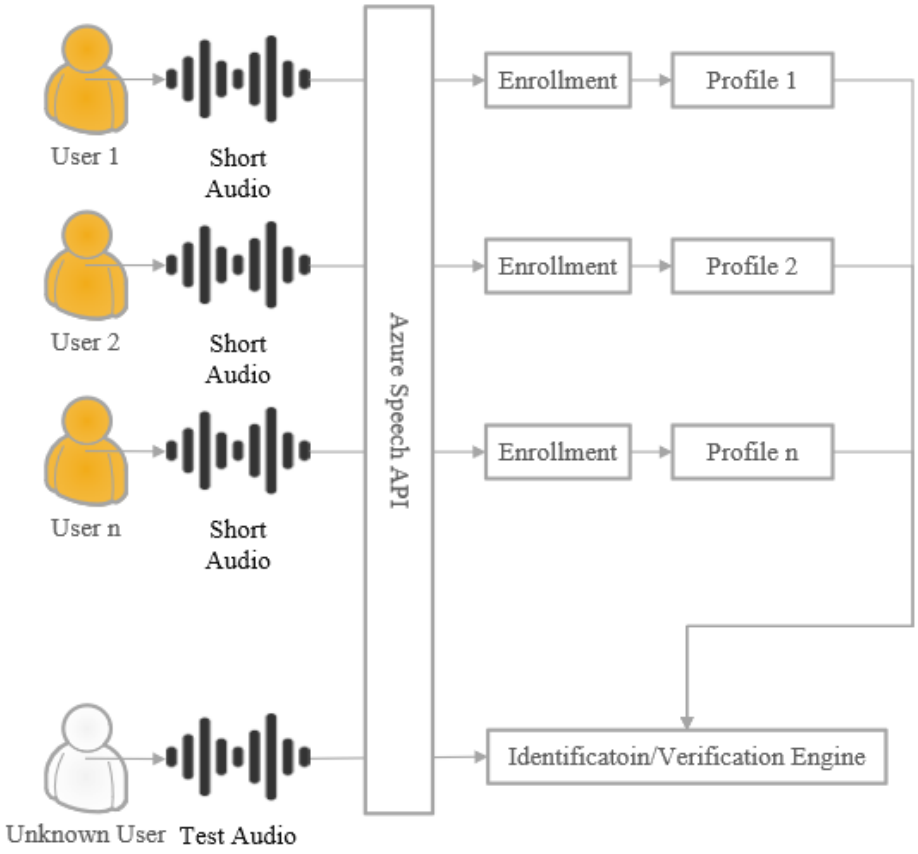


Figure 1: Workflow of Microsoft's speaker verification interface

2.3 Speech Diarization

We started off the project with an investigation of existing diarization algorithms. We will first explain some necessary mathematical concepts common to these diarization algorithms before delving into academic works.

2.3.1 Necessary Knowledge

As mentioned before, speaker diarization depends on the same features used by speech recognition algorithms. Even methods involving neural networks use MFCC (with delta and delta-delta) features as their input. A brief description of MFCC has been introduced in Section 2.1.

Prior to popularization of neural networks, most diarization frameworks use Gaussian Mixture Model to represent speaker profiles. A standard GMM can be expressed in the form of Eqn 3, where an individual Gaussian can be expressed as Eqn 4.

$$p(x) = \Sigma_{k=1}^{K} \pi_k \mathcal{N}(x|\mu_k, \Sigma_k) \tag{3}$$

$$\mathcal{N}(x|\mu, \Sigma) = \frac{1}{(2\pi)^{D/2}} \frac{1}{|\Sigma|^{1/2}} exp\{-\frac{1}{2}(x-\mu)^T \Sigma^{-1}(x-\mu)\} \tag{4}$$

Typical expectation maximization algorithm is available from $sklearn$. This algorithm can be summarized as an iterative computation of Eqn 5. Expression $L(\theta; X, Z)$ represents the likelihood of observing Z, latent data, given a set of training data X, using a model represented by θ. In this case, θ is a collection of variables that describe a set of Gaussian Mixtures outlined in Eqn 3.

$$\begin{aligned} \theta^{t+1} &= argmax(Q(\theta|\theta^{(t)})) \\ where\ Q(\theta|\theta^{(t)}) &= E_{Z|X,\theta^{(t)}}(log(L(\theta; X, Z))) \end{aligned} \tag{5}$$

We often use BIC to assess "similarity" between GMMs. We typically use a threshold BIC score to determine whether two GMMs need to be merged. The Bayesian Information Criterion can be expressed as Eqn 2.3.1. Then $\Delta BIC = BIC(M_i, M_j) - BIC(M)$.

$$\begin{aligned} BIC(M) &= log(L) - \frac{\lambda}{2} Plog(N) \\ BIC(M_i, M_j) &= logL(X_i|M_i) + logL(X_j|M_j) - \lambda Plog(N) \end{aligned} \tag{6}$$

2.3.2 Literatures on Speech Diarization

Workflow of speech diarization algorithms can be summarized in Fig 2. Major differences between techniques used for diarization center around clustering method and clustering criterion. Audio feature selection has also been explored in various works.

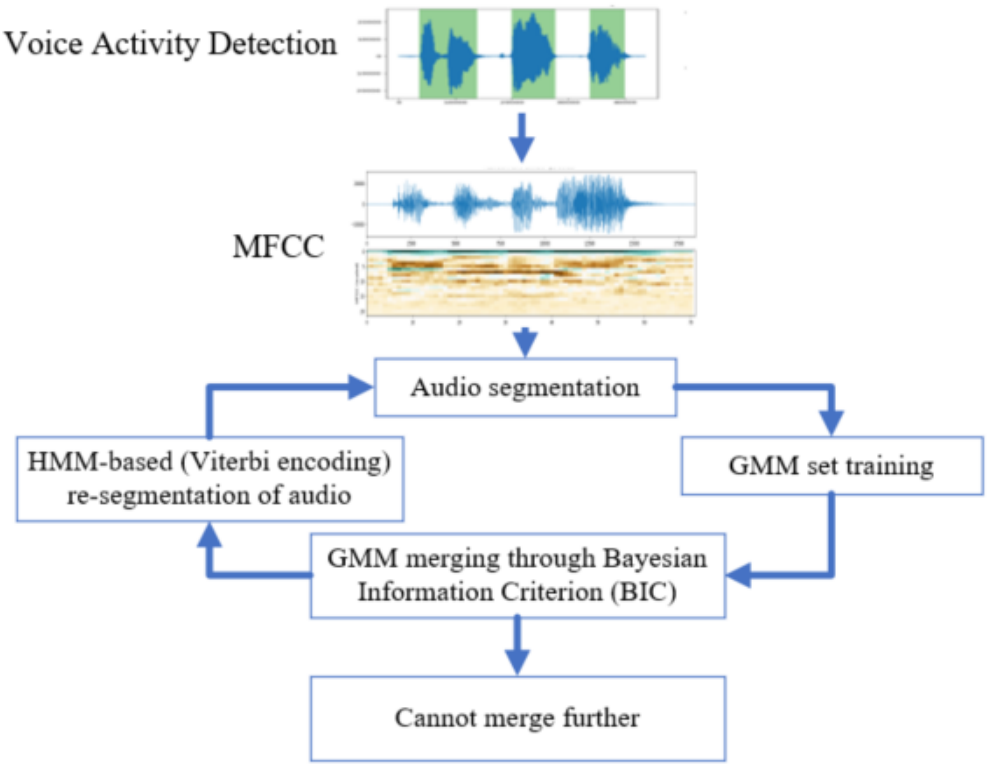


Figure 2: Common workflow of speech diarization algorithms

Clustering and Segmentation Evans et al. [24] outlined the difference between bottom-up and top-down clusterin mechanism. A top-down approach starts with 1 GMM and then iteratively breaks up GMM if the likelihood of a set of audio segments being uttered by the given model is below a threshold. In contrast, the bottom-up approach starts with a number of GMMs (larger than maximum number of speakers possible in the situation), and then attempts to merge data points associated with each GMM according to a given threshold. This work reveals that while bottom-up and top-down

methods have mixed accuracies as-is, the top-down method has lower diarization error with purified data. This certainly raises the importance of data purification, as both methods experience improved accuracy with purified data.

At the mean time, Liu et al.[25] experimented using GMM distance and BIC score as clustering criterion. Bayesian Information Criterion, described in Eqn 2.3.1, along with Akaike Information Criterion, are two popular criteria used for model selection. In contrast, Liu's measurement of GMM distance disregards the statistical implications of Gaussian Mixtures, but rather focuses on geometric properties of the model. Kotti et al.[26] also introduced KL-Divergence (Kullback-Leibler divergence) as another method of GMM merging.

Works done by Friedland et al. [27] is a great example on effect of feature selection. It is well understood that MFCCs alone are not sufficient for speech diarization. Delta and delta-delta MFCCs are used by most academic works reviewed for this project. [27] proposes to use long-term acoustic features in addition to MFCC and delta-MFCC for GMM training. [28] is another article that put effort in explaining the effectiveness of long-term features. These works found that pitch has high speaker discrimination power, which is not in contrary to common sense. Diarization error rate of Berkeley's ICSI system, of which Friedland is a main contributor, varies wildly from dataset to dataset. Nevertheless, there is a average of 4% decrease in error rate with top-10 most relevant long-term features.

Additional Segmentation Methods While the approaches mentioned above attempts to assign audio segments to speakers, works has also been done to identify locations of speaker change. Kotti et al.[26] mentioned methods to analyze difference between neighboring audio segments to locate potential speaker change location.

Recent developments in neural network has spurred interests in applying LSTM (long-short term memory) DNN (deep neural network) and RNN (recurrent neural network) instead of relying on standard GMMs. A unique approach is take by [29], which uses DNN to generate a speaker "embedding" instead of traditional MFCC and FFT based features for GMM training. Instead of attempting to assign audio segments to speakers, it finds speaker transitions by maximizing cosine distance between embed-

dings from different speakers during training. According to the report, it has around 10% improvement over accuracy achieved by LIUM, a state-of-art diarization package, reaching as low as 15% for CQT-gram features.

LIUM - An Open Source Implementation Like Kaldi in the field of speech recognition, LIUM [30] is popular among those working on speech diarizaiton. LIUM follows traditional speech processing workflow. It first computes MFCC (mel-frequency cepstral coefficients), generates i-Vectors, segments and clusters GMM (Gaussian Mixture Model) using BIC (Bayesian Information Criterion) and performs re-segmentation with Viterbi-decoder based HMM (Hidden Markov Model). This workflow is shown in Fig 3. This diagram should be viewed in comparison with Fig 2, as entry points for LIUM are directly related to steps in speech diarization. Moreover, LIUM can also be used for ASR purposes, as shown by Rousseau et al.[31], who utilized LIUM for English to French translation.

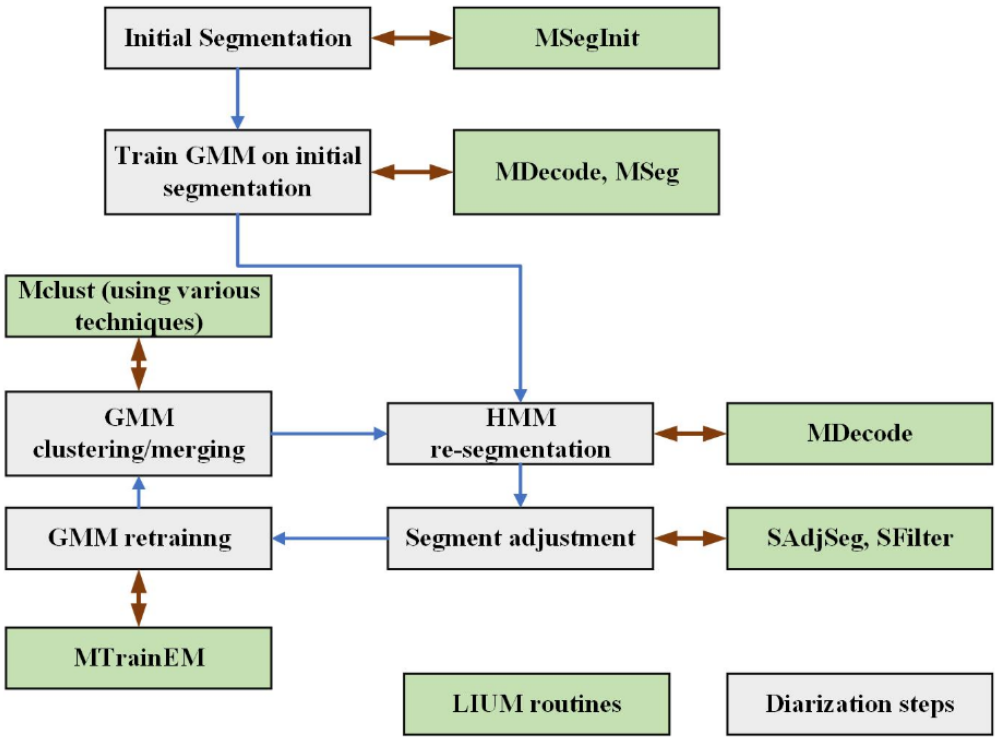


Figure 3: Workflow of a typical GMM based diarization algorithm and its corresponding LIUM toolkit entry points

As stated in these reports, LIUM focuses on text-independent speech diarization, which means no ASR is needed prior to distinguishing speakers. This is far more difficult than text-dependent diarization, but allows diarization and recognition to take place in parallel, as both requires basic audio features like MFCC.

Authors of LIUM claim that the engine is able to achieve WER between 8.35% to 24.49% [32] [33]. These results in line with those reported by Berkely-based researchers who worked on ICSI speaker diarization systems [27]. In addition, a previous Engineering Science thesis [34] has used LIUM on mobile phones, achieving better accuracy than those reported by LIUM's authors due to a slightly different metric used to account for "extra" speakers.

2.4 Live Diarization

All the works listed above focus on performing diarization on a recorded audio segment. They do acknowledge that speaker profile is typically unknown prior to diarization, but none mentioned the need for diarization in real time. Works done on diarization during meeting, as discussed in [35], requires existing speech model of attending speakers. These works are in the field of live speaker identification, not our goal of performing live speaker diarization.

Huang et al. [36] has worked on building a efficient system for real-time speech diarization, but this work has little description of its live retraining routine. The difficulty in a real-time speech diarization system is not the speed of segmenting, but when the models needs to be re-trained using more data. Its definition of real-time seems to be the ability to process a recorded audio using less than length of the audio, not running a diarization engine in parallel with the recording session.

In fact, there had been little progress toward executing the above algorithms during live meetings except commercial APIs. It is unclear how IBM (section 2.2.1) implemented its diarization system, but its poor performance can certainly be explained by the lack of research in the field. Amazon has already rolled out its own transcription system [37] and will include the feature to identify speakers. However, it does not specify if it requires enrollment of known speakers.

3 Implementations

We have effectively completed the implementation of all components necessary for a real-time speech processing system. The exact algorithms chosen to be implemented were selected from a variety of candidate algorithms. Tests involving speech recognition systems were less thorough, as Kaldi was known to be the state-of-art package used by many existing systems. In the case of speech diarization, we have attempted to build numerous systems toward the final goal of using such a system in the end product.

Since our project aims to complete the implementation of the said audio processing system, we will also described some implementation details in this section. For the same reason, experimental accuracies are not a critical measure. Instead, we evaluated various algorithms empirically. Furthermore, system hardware requirement was also taken into consideration for algorithm selection, as the system needs to support processing multiple audio streams and execute in parallel with incoming audio streams. For our purpose, it is more logical to combine the result and implementation sections.

3.1 Speech Recognition

There simply does not exist a pre-trained model that completely satisfy out requirement. We cannot pass audio streams to commercial APIs due to Sickkid's regulations on data privacy. Research language models, as mentioned before, are trained and tested on popular corpus like AMI [38] and Fisher [16]. We are primarily interested in their acoustic model, the model describing how syllables are pronounced and joined, rather than the language model, which describes how phrases are formed.

3.1.1 Comparison of Language Models

ASpIRE Language Model ASpIRE language model created by Harper and Mary [15] is an example included in the Kaldi speech processing package. ASpIRE is trained on Fisher [16], a dataset on conversational telephone speech.

LibriSpeech Language Model LibriSpeech language model is built by Panayotov et al. [39] using a corpus of audiobooks from LibriVox project [40].

Result As expected, ASpIRE language model performs better than LibriSpeech model. Quantitative results are not necessary as the difference is significant enough at test time. This difference in accuracy could be attributed to the way that people speak differently for conversation and for audio books. Audio book pronunciations are more formal and emotionless, while sentences spoken during conversations contain more variations and less monotonic.

3.2 Customized Language Model

Since the default ASpIRE language model only contains daily vocabulary, it is necessary to extend it by injecting medical terms. With a n-gram model for speech recognition, we have to not only expand the vocabulary of our model, but also put these words into context. During the process of creating our own language model, we had to be careful to not disturb the n-gram structures of the existing ASpIRE model. N-gram probabilities are best maintained, as observed in practice, when appearances of common words are minimized.

To address the need of obtaining medical terms with related context, we created a number of scripts to scrape abstracts from PubMed [41], a database for life sciences and biomedical topics. Basic steps involved in processing this data is listed in List 3.2.

For this task, we have scanned through approximately 2 million PubMed document IDs. From these abstracts, we were able to find more than 500K unique words. A close inspection of the unique words reveals that many of them are mis-spells, typos, incorrectly constructed words that should be broken up by hyphen, etc. Numbers made up a significant number of unique words, because we consider each alpha-numeric sequence as a word to accommodate potential specialist terms.

1. Download PubMed abstracts using $python$'s $Bio.Entrz$ toolkit in parallel
2. Pre-process downloaded text to exclude punctuations, separate sentences, and remove extraneous contents

3. Remove sentences that only contain "standard" vocabulary (we want our language model to be medical-focused)
4. Construct a unique word list
5. Remove words with less than 25 occurrences (25 is an empirically determined number)
6. Identify vocabulary that is not present in a pre-defined "standard" English vocabulary list
7. Truncate sentences to 3 words ahead and behind those identified vocabularies. This is our new corpus.
8. Apply CMU's Sphinx model to generate pronunciation (can be improved by scraping from medical dictionaries)
9. Generate a language model based on n-gram model and post processed texts
10. Merge the generated language model with default Aspire model
11. Re-generate HCLG graph for Kaldi

After cleaning up unwanted words, we have trimmed our vocabulary down to 82K unique words, plus all "standard" English vocabulary. Note that thoroughly cleaning the language corpus is critical for performance and accuracy. A bulky language model with excessive amount of unwanted n-grams can slow down probability computation. Erroneous spellings could also lead to incorrect transcription of audio.

Using this filtered corpus and ASpIRE language model, we are able to recognize a large number of specialized terms. From empirical testing, it is particularly good at chemical names, like because many chemical names are word tuples - word tuples have higher chance of being recognized than individual words. Recognition rate for diseases, symptoms and phenotypes are also satisfactory.

3.3 Speech Diarization

3.3.1 Previous Attempts

With Microsoft Speaker Identification API As described in section 2.2.1, Microsoft provides a speaker identification/verification system on Azure. Since the Microsoft API relies on known audio profiles, we have to adapt it to dynamically enroll new speakers as more audio stream is collected.

We attempted to build such a dynamic system to actively enroll new speakers and use the verification API as a diarizaiton agent. Basic workflow of this system is demonstrated in Fig 4. Unfortunately, Microsoft's API is fundamentally designed for speaker verification. It cannot handle continuous streaming of audio. This API would also be prohibitively expensive for long conversation (estimated cost of $15 for an hour of audio. Nevertheless, experience with the Microsoft API taught us important lessons on designing a live diarizaiton system, which is certainly useful for the current version of our work.

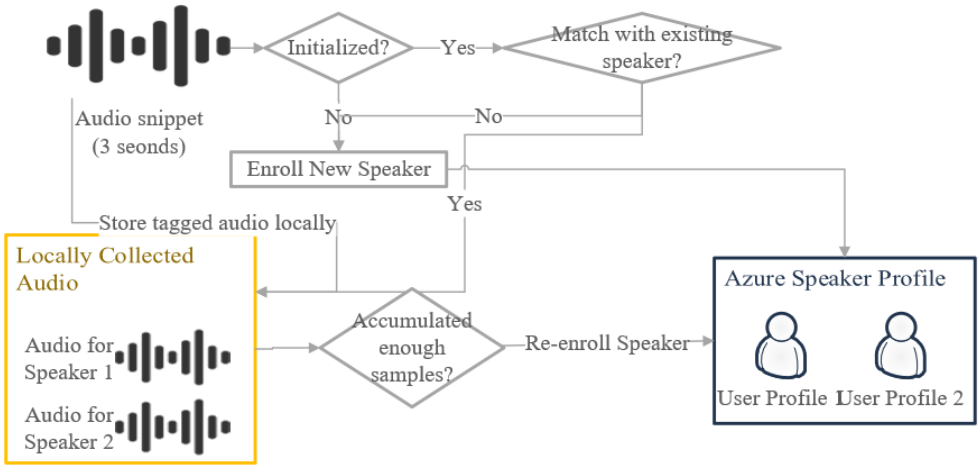


Figure 4: Workflow of a system that makes use of Microsoft's speaker identification API

GMM Based Model Most of the work prior to 2012 used GMM-based method to create clusters for speakers. This method uses a "parameterless" model because nothing but

the number of Gaussian per mixture is required at start up. It is very computationally expensive, yet its result is on the boarder line of satisfactory.

In-House Model As described in the literature review (Section 2), we attempted to use a custom-built model that follows the diarization workflow from previous literature. Our custom model uses basic MFC coefficients and allows various clustering criterion, such as cosine distance and BIC (Bayesian Information Criterion). The re-segmentation strategy is also a custom-built Viterbi-decoer algorithm that attempts to enforce minimum speech length before transition.

Although this in-house model is sufficient in performing "diarization" on artificially constructed data points, it is nowhere near the required accuracy for high-dimension MFCC from realistic audio samples. A comparison between its low-dimension and high-dimension performance is shown in Fig 5. Real world data has much higher spread and does not necessarily follow Gaussian distribution. The engine could not properly construct GMMs using these data points.

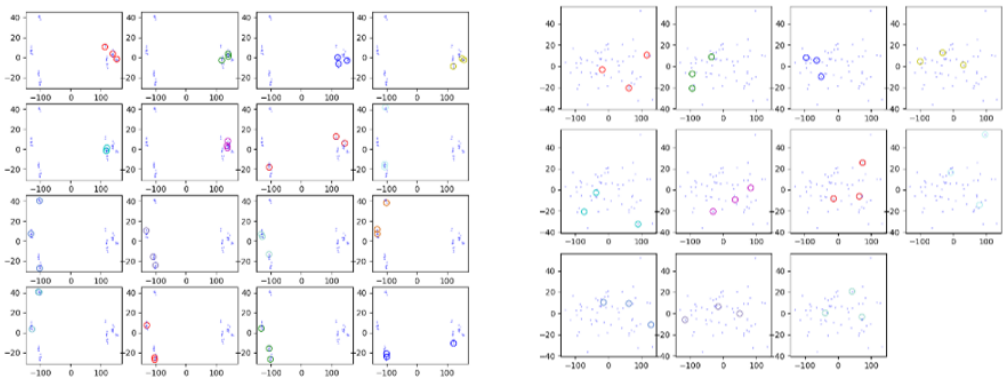


Figure 5: Clustering results for artificially constructed data and real world data

LIUM Diarization Package LIUM speaker diarization engine is a Java-based package. As described in the previous section (Section 2.3.2), LIUM is an popular model

among researchers. We built a Python wrapper for the LIUM engine to pre-process audio files and post-process segment files. Moreover, we have tunned LIUM parameters according to AMI corpus, which is better suited for conversation-style applications.

Since LIUM model is fundamentally based on GMM, we require at least 30 seconds of initial training with both speakers speaking for approximately the same amount of time for the model to function reasonably well. Short audio segments are then classified based on this partially trained model at regular intervals (5 seconds). After a longer interval (30 seconds), all previously recorded audio is re-segmented and the GMMs are re-clustered to improve model accuracy. Fig 6 is the live-diarizaiton result of LIUM for an example AMI corpus.

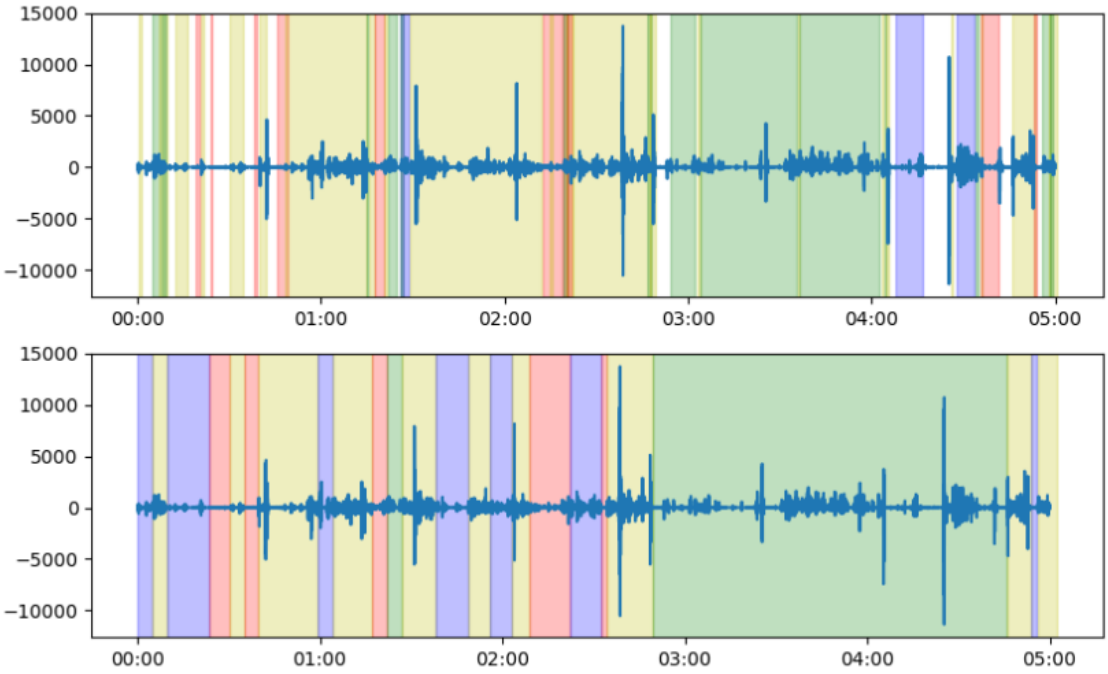


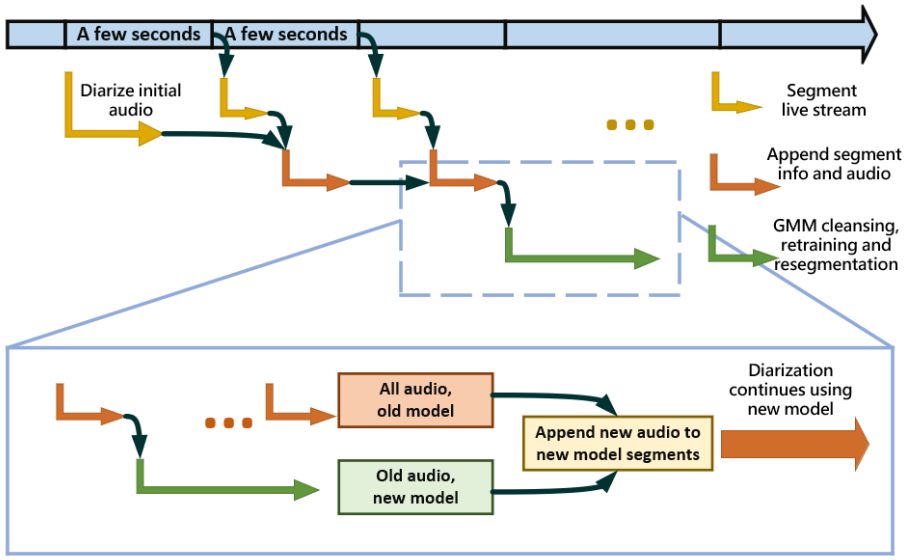
Figure 6: Diarization result for a 5 minute AMI snippet. Top is the ground truth. Bottom is LIUM result.

The model re-training strategy does provide satisfactory accuracy overtime, but it has a severe impact on efficiency of the algorithm. To be more specific, 5 seconds of au-

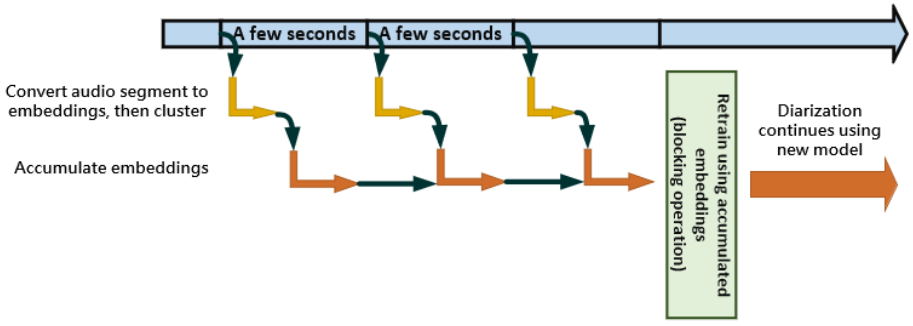
dio segment can be processed in less than 5 seconds, but re-training the entire model is an expensive operation. We not only have to deal with audio received while the current model is being re-trained, but also have to spare resources for the re-training effort. At least an Amazon c4.4xlarge instance (16 CPU cores and 30GB of memory) is necessary for multiple concurrent processing and re-training processes.

3.3.2 Neural Network Based Model

Most of our contribution to the neural network based model is to adapt it for live-diarizaiton. In the infrastructure implemented on our server, we used the same workflow as the one used for LIUM diarization engine. The periodic model re-training workflow is shown in Fig 7.



(a) With traditional GMM model. Re-training takes significantly longer, cannot be made a "blocking" operation.



(b) With neural network calculated embeddings. Re-training is fast.

Figure 7: Workflow of our live diarization engines

3.3.3 Microphone Array

We have purchased a ReSpeaker Microphone Array [42] to help with the task of speech diarization. We have setup the microphone array to record using a 7.1 channel setting, which requires a firmware upgrade to 0x32 version from its stock 0x30 firmware.

Although ReSpeaker comes with built in audio direction computation that can be accessed via an on-board register, it has limited resolution. We have opted to use ODAS [43], a generic audio processing library built to suppose the XMOS sound card on-board of ReSpeaker. ODAS is able to compute direction of sound for up to 4 possible sound sources. The number of sources tracked can be modified through ODAS source code.

3.4 System Integration

Speech diarization and speech recognition has been verified to work independently. We integrated these two functionalities into a server hosted on Amazon Web Service, then migrated to SickKids' High Performance Framework (HPF). Prior to setting up the environment on HPF, we conducted experiments on Amazon AWS. Amazon AWS is more flexible than HPF and provides better performance than HPF infrastructure. A brief report on server performance can be found in section A.1.

3.4.1 Building A Speech Processing Server

Underlying Infrastructure Audio streaming and initial processing server can be obtained from open-source projects. However, we have to infuse these existing works with our customized speech diarization engine. We made use of code from GStreamer[44] as the basis of our server. It provides code example for audio processing worker, worker management, and a simple client. This server implements `python tornado` package and communicates via WebSocket| protocol.

To make the server more robust, we modified the management unit to handle unexpected disconnection from clients. More audio processing workers can be launched when requested, allowing multiple clients to connect to the same server. Fig 8 presents the server's workflow from launch to sending results back to connected clients. We experimented with multiple AWS settings and eventually settled on a `c5.2xlarge` instance (8 CPU cores and 16GB of memory) for testing purposes. The implementation on SickKids infrastructure uses the same amount of resources, except with weaker CPU single-core performance.

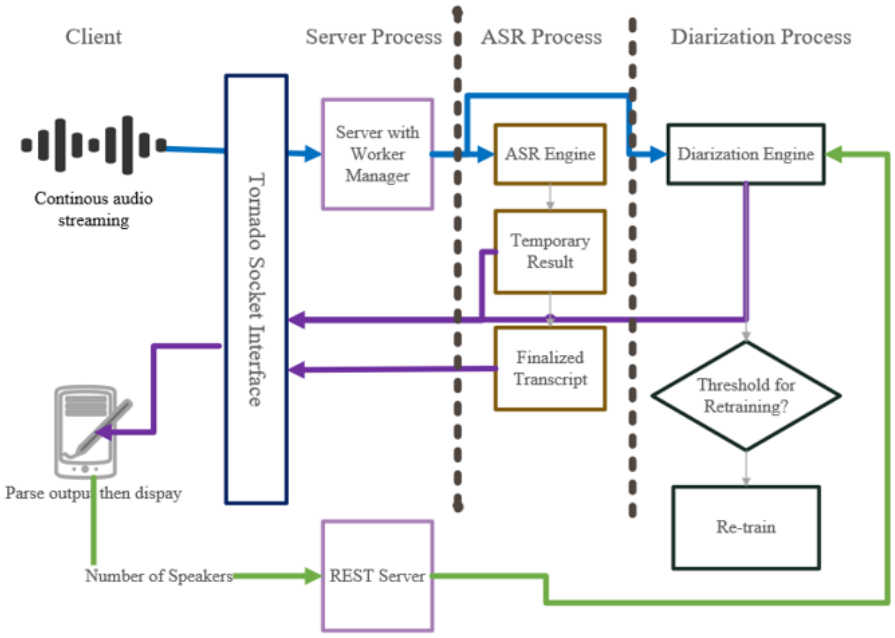


Figure 8: Asynchronous processing of speech recognition and speech diarization

Memory Management Through experiments, we discovered that Python garbage collection does not correctly dispose objects stored in cross-process communication channels even after related processes has exited. This behavior prevents memory from being recycled after a client disconnects. To address this issue, we implemented more verifications and manual garbage cleaning routines. Each worker, upon disconnection, explicitly deletes all memory references to ensure clean hand-over of work to the server manager. The server manager itself also verifies the deletion of objects occupied by the exiting process. Processes are also explicitly killed, along with all potential child-processed, after client disconnects.

Availability Our current system automatically spawns new workers after workers disconnect. Although GStreamer can recycle workers, our addition of another diarization process on top of its recognition worker interferes with the process. After manually killing processes associated with disconnected/failed clients, we implemented a bash script to spawn more workers. We added a semaphore to keep track of total number of

spawned workers to make sure that server hardware resources can actually support the spawned number of workers. Additional client connection requests are not responded to.

A diagram summarizing how memory management 3.4.1 and availability 3.4.1 can be achieve is presented as Fig 9. Not shown in the diagram is that worker failure are treated the same way as client disconnection. These are automatically discovered, and then new workers will spawn.

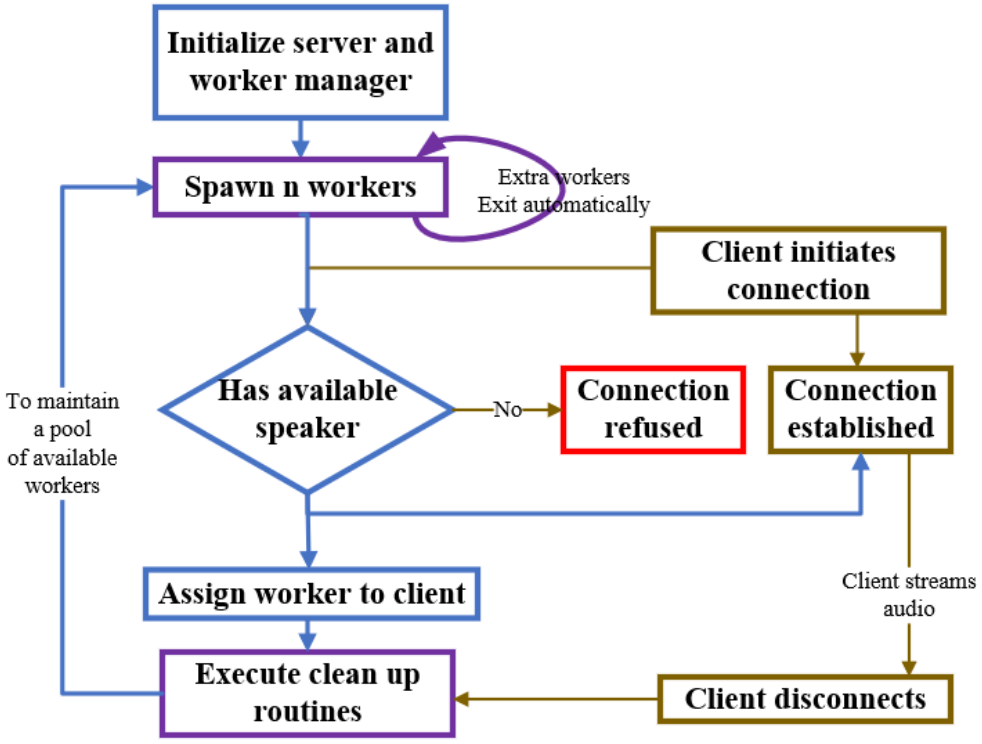


Figure 9: Process of assigning worker to client and basic worker management behavior

ASR Plugin Gstreamer [44] includes a *Kaldi* plugin to interface with the ASR engine, but the provided plugin can only work with a GMM based model, which is not accurate for our purpose. Instead of using the provided plugin, we used a plugin that uses neural network [45]. Parameters for this plugin has been changed to access the customized medical-focused language model described above.

Diarization Since the above open-source plugins do not handle speech diarization, nor can they correctly utilize multiple CPU cores on the server, we added a separate process alongside the worker thread to enable speech diarization. The JSON-based communication protocol was modified to accommodate results from diarization. To be more specific, diarization results are appended to temporary recognition results. Diarization and recognition essentially run asynchronously, and their results are synthesized at the front end.

We artificially delay diarization because speech diarization results are more accurate when the audio segment is long. Moreover, it would consume large amount of resources to compute for temporary results.

REST API Although Gstreamer is capable of handling various message types, it would be significant amount of work to alter the existing audio streaming channel to support other message types. Since we have added support for multiple speakers, along with user interfaces for this option, it became necessary to include another service to facilitate communication between server and client. REST based Flask package is a popular Python package that can complete this task. As shown in Fig 8, this interface communicates directly with the diarizaiton engine. At the current stage, it is only capable of notifying change in total number of speakers. Other messages can be easily added on REST infrastructure.

Resource Consumption It is clear from Fig 8 that each client connection to the server requires at least 2 CPU cores, assuming that both recognition and diarization are utilizing 1 CPU core each. Empirical testing shows that RAM utilization for each client connection increases slowly, likely related to the need for diarization engine to persist all previously processed audio segments. In total, at least 9 GB of memory is needed for each client connection. CPU utilization rate is typically at 100% or above (due to how Ubuntu evaluates CPU utilization rate). Over 100% utilization rate suggests the core is behind schedule on the amount of work assigned to it. In contrast, the diarization process only sporadically reach above 90% utilization, suggesting that the diarization algorithm we selected is capable of handling real-time task. It only falls behind briefly

when re-training is required.

More results and observations More results and observations are included in Section A.2. There, observations related to mis-alignment near speaker transitions and properties associated with the language model are briefly discussed. The ability of our speech recognition engine to capture specialist terms is demonstrated in Fig 15.

3.5 PhenoPad Front-End

Basic Interface Prior to our our work, PhenoPad had only been able to interface with either Google's or Microsoft's speech recognition engine. We designed an additional panel to display transcribed audio in the form of a chat conversation. Since there is a delay between speech recognition result and partitioning result from the diarization engine, we added an additional panel to display temporary results collected from our ASR engine.

Notice that the audio recorder can be stopped at any time to address potential privacy requirements. This requires us to save multiple tracks and to be able to preserve previous conversations for the same session.

To keep users informed of progress made by our ASR engine, we constantly update the latest sentence with the most-likely words. Each sentence will eventually become finalized after prediction result has stabilized. Long audio chunks are automatically separated into small audio chunks by our speech recognition engine to avoid clustering up display.

Fig 10 shows speech processing in progress. A combo-box is available for the user to identify the doctor, so that the conversation display can correctly display "incoming" and "outgoing" messages. Another figure with more detailed labels can be found at Fig 12.

Keyword Capturing As ASR result sentences become finalized, we send the entire sentence to an in-house text mining API to capture genetics-related terms. This result is displayed at the top right corner of the panel, as shown in Fig 10.

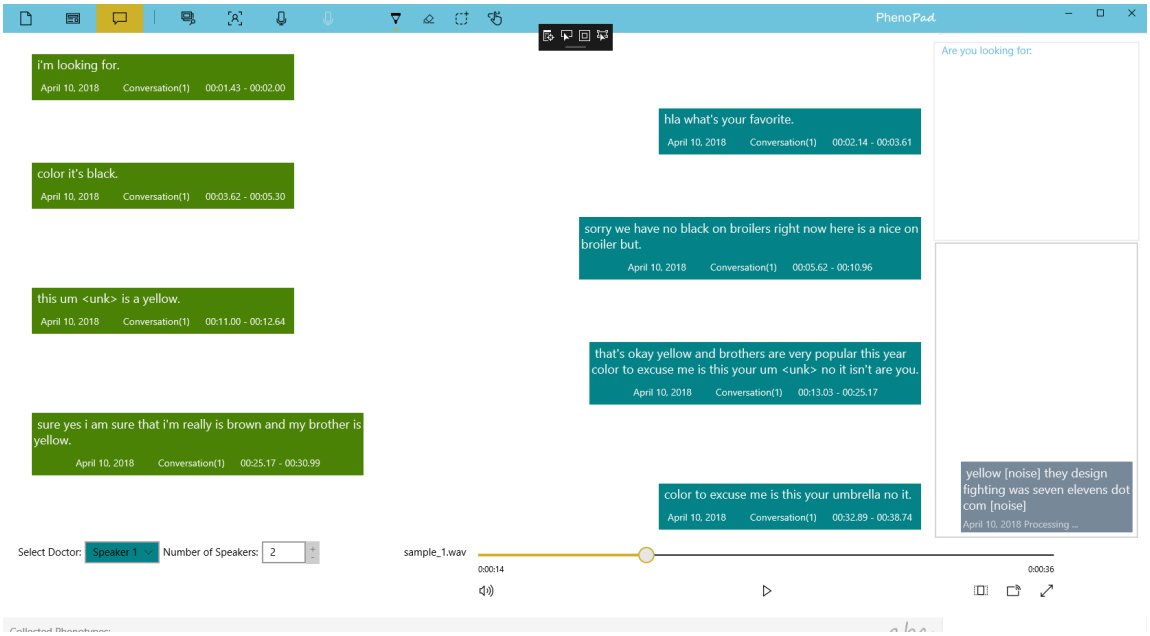


Figure 10: User interface for displaying speech recognition and diarization results. Right hand panel displays temporary results. The bottom is a replay bar. The center of the screen displays transcribed text in conversation-style.

Conversation Replay Since results from ASR and diarization engine are not always correct, it is necessary to allow users to replay the conversation. This feature is particularly useful for the process of transferring interview content, such as notes, diagrams, and transcripts, to EHR. Each diarized audio is tagged with its relative location within its corresponding conversation. When clicked, as shown in Fig 10, the audio track seeks to 5 seconds prior to the start of the transcript.

Number of Speakers Total number of speakers can be changed via the dial-box toward the bottom of the display. Changes made to total number of speakers immediately initiates re-training on the server. Default speaker count is 2.

4 Future Work

After realizing that post-processing meeting information would not yield meaningful results without a robust infrastructure for data collection, we shifted our original focus toward implementation of such a real-time audio processing system. In the mean time, Jixuan, collaborator and main developer of PhenoPad, had refined other related user interfaces.

Even building an in-house audio processing system has proven to be difficult, primarily because we have under-estimated the amount of work involved to smooth out details not discussed by many research papers. Integrating previous academic work, such as putting LIUM and Kaldi together, took longer than expected as well. While we do appreciate the flexibility provided by these systems, the lack of documentation and instability had hindered progress.

Improvements can still be made to our existing system for additional stability and performance. Some problems outlined during interim report has not been addressed due to time constraint. The following section is a description of what could be done to improve our current system. Attempts will be made to alleviate the problems before work is handed back to Jixuan.

4.1 Speech Recognition

4.1.1 Performance

The original ASpIRE speech model works well on our platforms. A light weight, PocketSphinx, used on ReSpeaker, even runs on Raspberry Pi with acceptable accuracy. However, our customized language model lags behind on Sickids' HPF infrastructure. There is also noticeable, but sporadic delays on AWS instances. Knowing that real-time speech recognition have been achieved years ago with low performance commodity hardware, we believe the problem lies in complexity of our language model. Since we are confident that 82K words should not be a huge concern, given the vocabulary of commercial speech recognition implementations, it must be the complexity of our n-gram language model that is slowing down the computation. We will further simplify

our language model by repeating the filtering process described in List 3.2 to remove extraneous or low probability n-grams. Note that separating audio tracks will decrease worker load substantially, but we would like to have an efficient model that runs by itself.

4.2 Accuracy

We could train a language model using other datasets, that is, a language model trained using other publicly available dataset in addition to ASpIRE (Fisher corpus) and LibriSpeech (LibriVOX audio books). Since we are only interested in empirical results, we can mix a variety of datasets. As long as we are able to transform the given dataset to a unified transcription format, we can concatenate these datasets. If we were training with customized dataset, it would be desirable to train directly with Kaldi's nnet-3 implementation for best results.

4.3 Speech Diarization

4.4 Performance

We have already said that speech diarization runs fast enough for real-time processing, however, it has to stall during re-training. We have implemented an effective "branch-then-merge" algorithm to accommodate a model that is being re-trained, while keeping another old model for live use when we were developing a system based on LIUM. Merging between these two model is very fast relative to stall time taken by re-training. We could adapt this model for our current diarization algorithm as well.

4.5 Accuracy

We will continue to work on verifying accuracy for 3 or more speakers. The use of microphone array with locationing capability could also help with diarization accuracy.

4.6 Additional Hardware and Future Infrastructur

Work is already undertaken to offload audio and video acquisition to a Raspberry Pi Model 3. With this implementation, we aim to build a platform that can be installed or placed in meeting rooms. This hardware setup is illustrated in Fig 11. This setup diagram also includes potential communication channels required in the future. All connections will be bi-directional. The said device, Surface tablet and server will be able to communicate with each other. This setup will significantly reduce workload on Surface Book, extending its battery life in tablet mode.

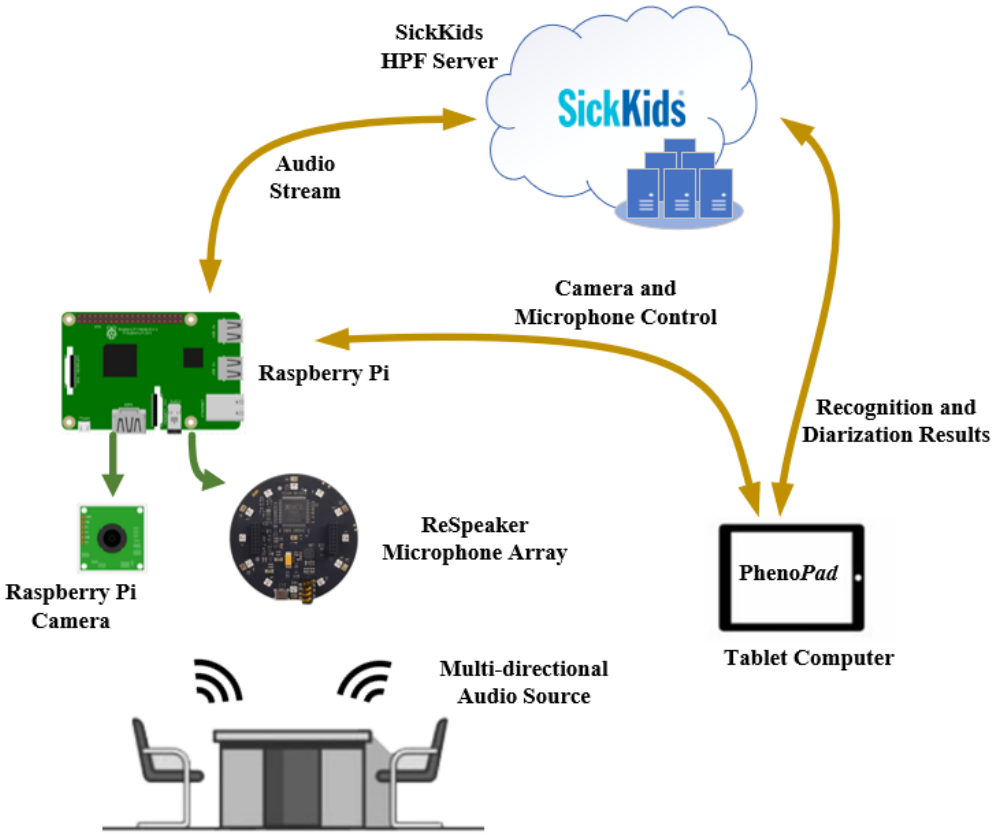


Figure 11: Potential hardware setup with Raspberry Pi and microphone array [46][47][48][49]

5 Conclusion

Over the course of this year, we have fully implemented a real-time audio processing system for medical conversations. This system is capable of recognizing medical-related keywords and identify lines spoken by individual participating speakers. The system now resides in SickKids' HPF cluster, ready to service multiple client connections. Most importantly, the user interface is clean and easy to use. With these goals achieved, we have reached most of the original design targets for this real-time audio processing system. While the system is still experiencing issues, upgrades will only be incremental.

Simply put, we have built a novel intelligent note-taking application that is well-suited for medical purpose and can be tailored toward other specialists from other professions. Although various note-taking and meeting systems have been made public, a synthesized product had never been implemented, let alone a functioning real-time audio processing system. We will start to formally introduce PhenoPad to people around SickKids as more stability and performance issues are routed out.

We did not improve accuracy of either speech recognition or speech diarization, because those were not primary goals of this project. We only intended to extract from the state-of-art research products and apply these algorithms to real world applications. As said, we will continue to improve on stability and efficiency of these systems and modify our user interface to accommodate demand from users.

Finally, we hope PhenoPad-like applications will become a staple for all medical specialists in the industry. We are confident that the infrastructure which we have built will significantly reduce doctors' workload for generating medical reports and provide vital information for real-time inference assistance. With introduction of more intelligent note-taking applications like PhenoPad, we will certainly see improved health care quality and happier doctor in the future.

References

[1] G. E. Mize, P. Hannaford-Agor, and N. L. Waters, *The state-of-the-states survey of jury improvement efforts: A compendium report*. National Center for State Courts Williamsburg, VA, 2007.

[2] S. Banerjee and A. I. Rudnicky, "Smartnotes: Implicit labeling of meeting data through user note-taking and browsing," in *Proceedings of the 2006 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology: companion volume: demonstrations*. Association for Computational Linguistics, 2006, pp. 261–264.

[3] A. Barabanov, D. Kocharov, S. Salishev, P. Skrelin, and M. Moiseev, "Voice activity detector (vad) based on long-term phonetic features," *ExLing 2016*, p. 33, 2016.

[4] J. H. Ko, J. Fromm, M. Philipose, I. Tashev, and S. Zarar, "Precision scaling of neural networks for efficient audio processing," *arXiv preprint arXiv:1712.01340*, 2017.

[5] "Webrtc vad," https://https://webrtc.org/, accessed: 2018-04-08.

[6] "Webrtc vad source code," https://github.com/ReadyTalk/webrtc/blob/master/webrtc/common_audio/vad/vad_core.c, accessed: 2018-04-08.

[7] M. Karafiát, L. Burget, P. Matějka, O. Glembek, and J. Černockỳ, "ivector-based discriminative adaptation for automatic speech recognition," in *Automatic Speech Recognition and Understanding (ASRU), 2011 IEEE Workshop on*. IEEE, 2011, pp. 152–157.

[8] S. Garimella, A. Mandal, N. Strom, B. Hoffmeister, S. Matsoukas, and S. H. K. Parthasarathi, "Robust i-vector based adaptation of dnn acoustic model for speech recognition," in *Sixteenth Annual Conference of the International Speech Communication Association*, 2015.

[9] D. Povey, A. Ghoshal, G. Boulianne, L. Burget, O. Glembek, N. Goel, M. Hannemann, P. Motlicek, Y. Qian, P. Schwarz, J. Silovsky, G. Stemmer, and K. Vesely, "The

kaldi speech recognition toolkit," in *IEEE 2011 Workshop on Automatic Speech Recognition and Understanding*. IEEE Signal Processing Society, Dec. 2011, iEEE Catalog No.: CFP11SRW-USB.

[10] "Lapack," http://www.netlib.org/lapack/, accessed: 2018-04-08.

[11] "Blas," http://www.netlib.org/blas/, accessed: 2018-04-08.

[12] "Openfst," http://www.openfst.org/twiki/bin/view/FST/WebHome, accessed: 2018-04-08.

[13] "Srilm - the sri language modeling toolkit," http://www.speech.sri.com/projects/srilm/, accessed: 2018-04-08.

[14] K. Veselỳ, A. Ghoshal, L. Burget, and D. Povey, "Sequence-discriminative training of deep neural networks." in *Interspeech*, 2013, pp. 2345–2349.

[15] M. Harper, "The automatic speech recogition in reverberant environments (aspire) challenge," in *Automatic Speech Recognition and Understanding (ASRU), 2015 IEEE Workshop on*. IEEE, 2015, pp. 547–554.

[16] C. Cieri, D. Miller, and K. Walker, "Fisher english training speech parts 1 and 2," *Philadelphia: Linguistic Data Consortium*, 2004.

[17] "Sre16 xvector model," https://github.com/kaldi-asr/kaldi/tree/master/egs/sre16/v2, accessed: 2018-2-4.

[18] D. Snyder, D. Garcia-Romero, D. Povey, and S. Khudanpur, "Deep neural network embeddings for text-independent speaker verification," *Proc. Interspeech 2017*, pp. 999–1003, 2017.

[19] "Google speech api," https://cloud.google.com/speech/, accessed: 2017-10-10.

[20] "Watson api," https://www.ibm.com/blogs/bluemix/2017/05/whos-speaking-speaker-diarization-watson-speech-text-api/, accessed: 2017-10-10.

[21] "Peterson interview," https://www.youtube.com/watch?v=27dshxbbPlg&t=1s, accessed: 2017-10-1.

[22] "Azure api," https://azure.microsoft.com/en-us/services/cognitive-services/speech/, accessed: 2017-10-10.

[23] "Azure speaker api," https://azure.microsoft.com/en-us/services/cognitive-services/speaker-recognition/, accessed: 2017-10-10.

[24] N. Evans, S. Bozonnet, D. Wang, C. Fredouille, and R. Troncy, "A comparative study of bottom-up and top-down approaches to speaker diarization," *IEEE Transactions on Audio, speech, and language processing*, vol. 20, no. 2, pp. 382–392, 2012.

[25] T. Liu, X. Liu, and Y. Yan, "Speaker diarization system based on gmm and bic," in *International Symposium on Chinese Spoken Language Processing, Singapore*, 2006.

[26] M. Kotti, V. Moschou, and C. Kotropoulos, "Speaker segmentation and clustering," *Signal processing*, vol. 88, no. 5, pp. 1091–1124, 2008.

[27] G. Friedland, A. Janin, D. Imseng, X. Anguera, L. Gottlieb, M. Huijbregts, M. T. Knox, and O. Vinyals, "The icsi rt-09 speaker diarization system," *IEEE Transactions on Audio, Speech, and Language Processing*, vol. 20, no. 2, pp. 371–381, 2012.

[28] G. Friedland, O. Vinyals, Y. Huang, and C. Muller, "Prosodic and other long-term features for speaker diarization," *IEEE Transactions on Audio, Speech, and Language Processing*, vol. 17, no. 5, pp. 985–993, 2009.

[29] M. Rouvier and B. Favre, "Investigation of speaker embeddings for cross-show speaker diarization," in *Acoustics, Speech and Signal Processing (ICASSP), 2016 IEEE International Conference on*. IEEE, 2016, pp. 5585–5589.

[30] M. Rouvier, G. Dupuy, P. Gay, E. Khoury, T. Merlin, and S. Meignier, "An open-source state-of-the-art toolbox for broadcast news diarization," in *Interspeech*, 2013.

[31] A. Rousseau, L. Barrault, P. Deléglise, Y. Esteve, H. Schwenk, S. Bennacef, A. Muscariello, and S. Vanni, "The lium english-to-french spoken language translation system and the vecsys/lium automatic speech recognition system for italian language for iwslt 2014," in *Proceedings of International Workshop on Spoken Language Translation (IWSLT)*, 2014.

[32] S. Galliano, G. Gravier, and L. Chaubard, "The ester 2 evaluation campaign for the rich transcription of french radio broadcasts," in *Tenth Annual Conference of the International Speech Communication Association*, 2009.

[33] O. Galibert and J. Kahn, "The first official repere evaluation," in *First Workshop on Speech, Language and Audio in Multimedia*, 2013.

[34] D. D. Matteo, "Speech diarization on a mobile platform," 2011.

[35] G. Friedland and O. Vinyals, "Live speaker identification in conversations," in *Proceedings of the 16th ACM international conference on Multimedia*. ACM, 2008, pp. 1017–1018.

[36] Y. Huang, O. Vinyals, G. Friedland, C. Muller, N. Mirghafori, and C. Wooters, "A fast-match approach for robust, faster than real-time speaker diarization," in *Automatic Speech Recognition & Understanding, 2007. ASRU. IEEE Workshop on*. IEEE, 2007, pp. 693–698.

[37] "Amazon transcribe," https://aws.amazon.com/transcribe/, accessed: 2018-2-4.

[38] J. Carletta, S. Ashby, S. Bourban, M. Flynn, M. Guillemot, T. Hain, J. Kadlec, V. Karaiskos, W. Kraaij, M. Kronenthal *et al.*, "The ami meeting corpus: A pre-announcement," in *International Workshop on Machine Learning for Multimodal Interaction*. Springer, 2005, pp. 28–39.

[39] V. Panayotov, G. Chen, D. Povey, and S. Khudanpur, "Librispeech: an asr corpus based on public domain audio books," in *Acoustics, Speech and Signal Processing (ICASSP), 2015 IEEE International Conference on*. IEEE, 2015, pp. 5206–5210.

[40] “Librivox project,” https://librivox.org/, accessed: 2018-04-08.

[41] “Pubmed,” https://www.ncbi.nlm.nih.gov/pubmed/, accessed: 2018-04-08.

[42] “Respeaker mic array - far-field w/ 7 pdm microphones,” https://www.seeedstudio.com/ReSpeaker-Mic-Array-Far-field-w%2F-7-PDM-Microphones-p-2719.html, accessed: 2018-04-08.

[43] “Odas: Open embedded audition system,” https://github.com/introlab/odas, accessed: 2018-04-08.

[44] “Kaldi gstreamer,” https://github.com/alumae/kaldi-gstreamer-server, accessed: 2017-10-10.

[45] “Kaldi gstreamer plugin with neural net,” https://github.com/alumae/gst-kaldi-nnet2-online, accessed: 2017-10-10.

[46] “Meeting icon,” https://www.fotolia.com/id/111326919, accessed: 2018-04-09.

[47] “Sound icon,” https://thenounproject.com/term/sound-waves/209686/, accessed: 2018-04-09.

[48] “Sickkids,” http://www.sickkids.ca/, accessed: 2018-04-08.

[49] “Black device icons in flat style on white background.” https://www.dreamstime.com/, accessed: 2018-04-08.

A Appendix

A.1 Performance of Servers

Through the course of this project, performance of various platforms has become one of our primary interests. Since we have noticed substantial slowdowns on Sickkids's HPF framework, we run a few tests to evaluate the difference between HPF machines and those provided by Amazon AWS. It is also interesting to see how these server platforms fair against an Raspberry Pi, on which we have proposed to run the algorithms from.

For all algorithm runs, we use result from the 2nd run. This is to populate CPU cache with necessary content.

Table 1: Hardware Specifications

	Model Name	Frequency (GHz)	Cache (KB)
Sickkids	Intel Xeon E31xx (Sandy Bridge)	2.59	4096
M5.2xlarge	Intel Xeon 8175M	2.5	33792
C5.4xlarge	Intel Xeon Platinum 8124M	3.0	25344
Raspberry Pi Model 3	Broadcom BCM2837	1.2	512

Table 2: Test: sysbench –test=cpu –cpu-max-prime=20000

	Completion Time (s)	Speed Up against Sickkids (%)
Sickkids	26.523	–
M5.2xlarge	24.403	8.0
C5.4xlarge	22.639	14.6
Raspberry Pi Model 3	379.373	-1330.4%

Table 3: Test: stress-ng -c 1 –cpu-ops 5000

	Completion Time (s)	Speed Up against Sickkids (%)
Sickkids	22.76	–
M5.2xlarge	16.55	27.3
C5.4xlarge	15.66	31.2
Raspberry Pi Model 3	205.78	-804.1%

A.2 More Results and Observations

A.2.1 Labeled User Interface

Figure 12: Labeled user interface

This diagram should be clearer than the one shown as Fig 10. All sections of the user interface are colored.

A.2.2 Diarization Alignment

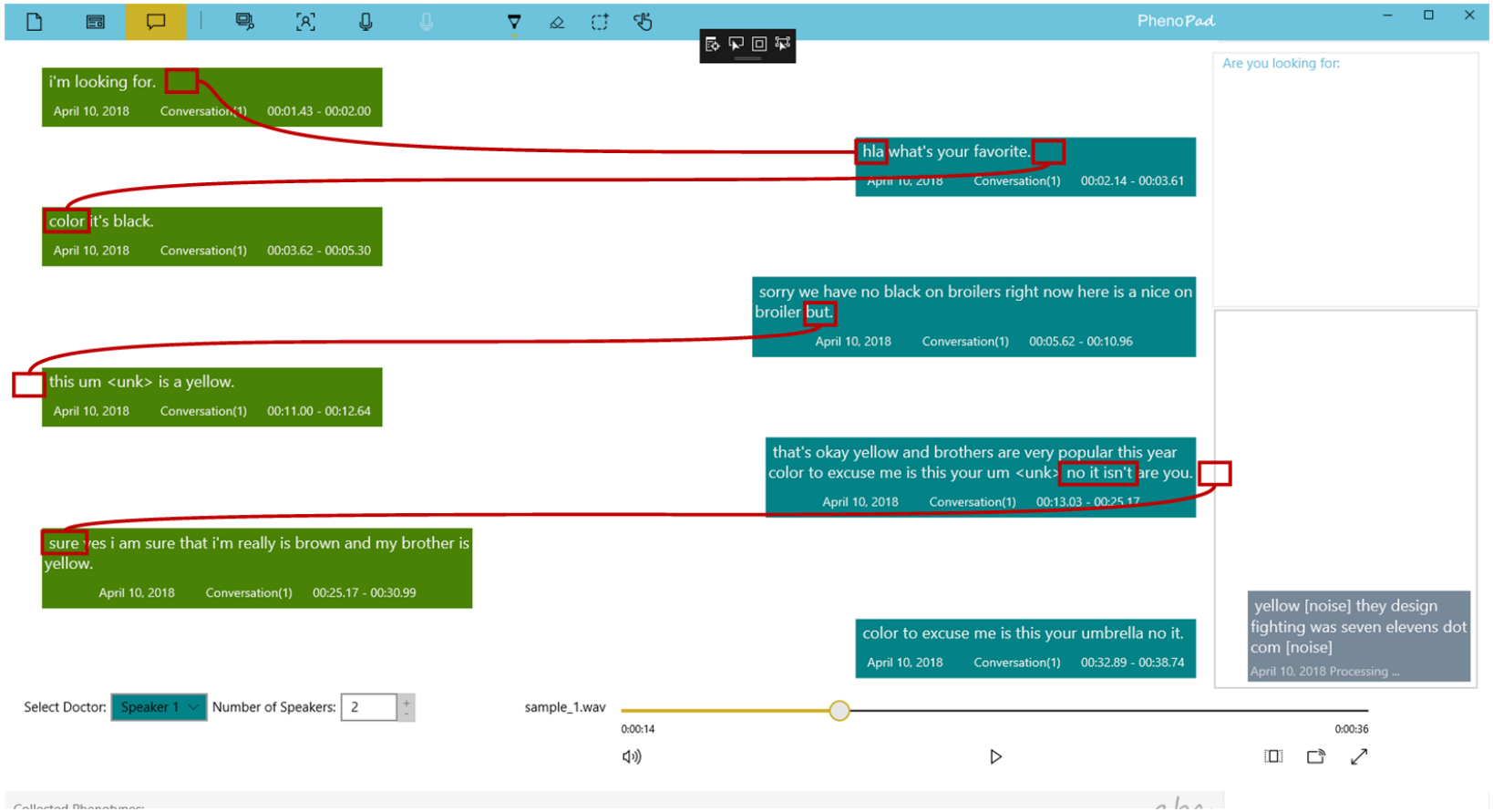


Figure 13: Shifted diarization results

This figure demonstrates mis-alignment in diarization result. Boundary words are difficult to separate because speaker change boundaries are often in the middle of these words. Our diarization engine only has resolution of around 15ms. It does not take a long time for us to say a simple word.

A.2.3 N-Gram Model - Umbrella is Difficult

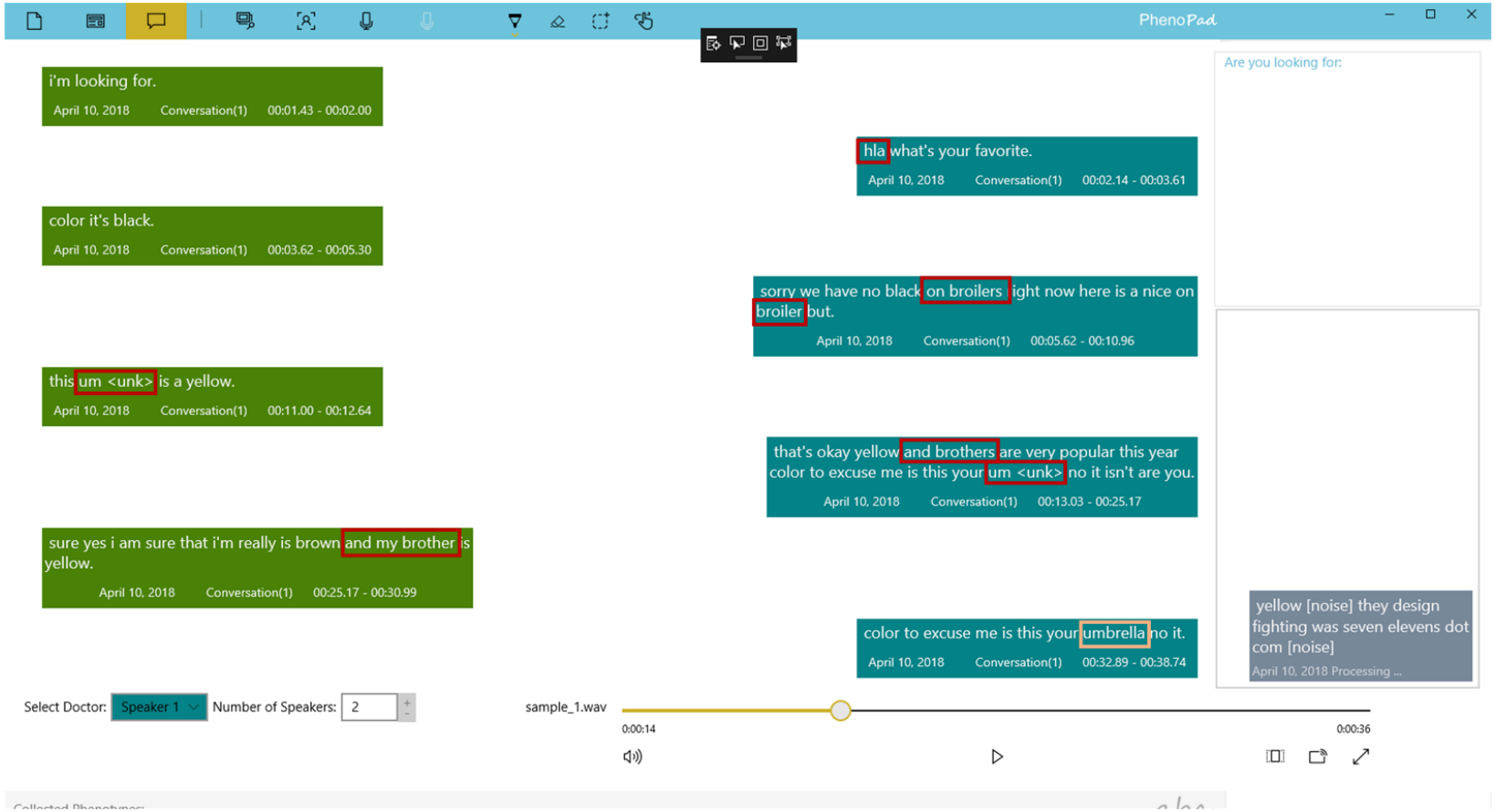


Figure 14: Speech recognition result for "umbrella"

Apparently it is very difficult to recognize "umbrella". It could be that Fisher corpus lacks references to umbrella. We were only able to recognize 2-gram "your umbrella". "your umbrella" should actually have higher probability than other mis-recognized combinations like "your favorite umbrella" or "this umbrella". People have little reason to speak those terms in telephone conversations. Our engine prefer terms related to "brother" over "umbrella" most of the time.

A.2.4 N-Gram Model - Special Terms

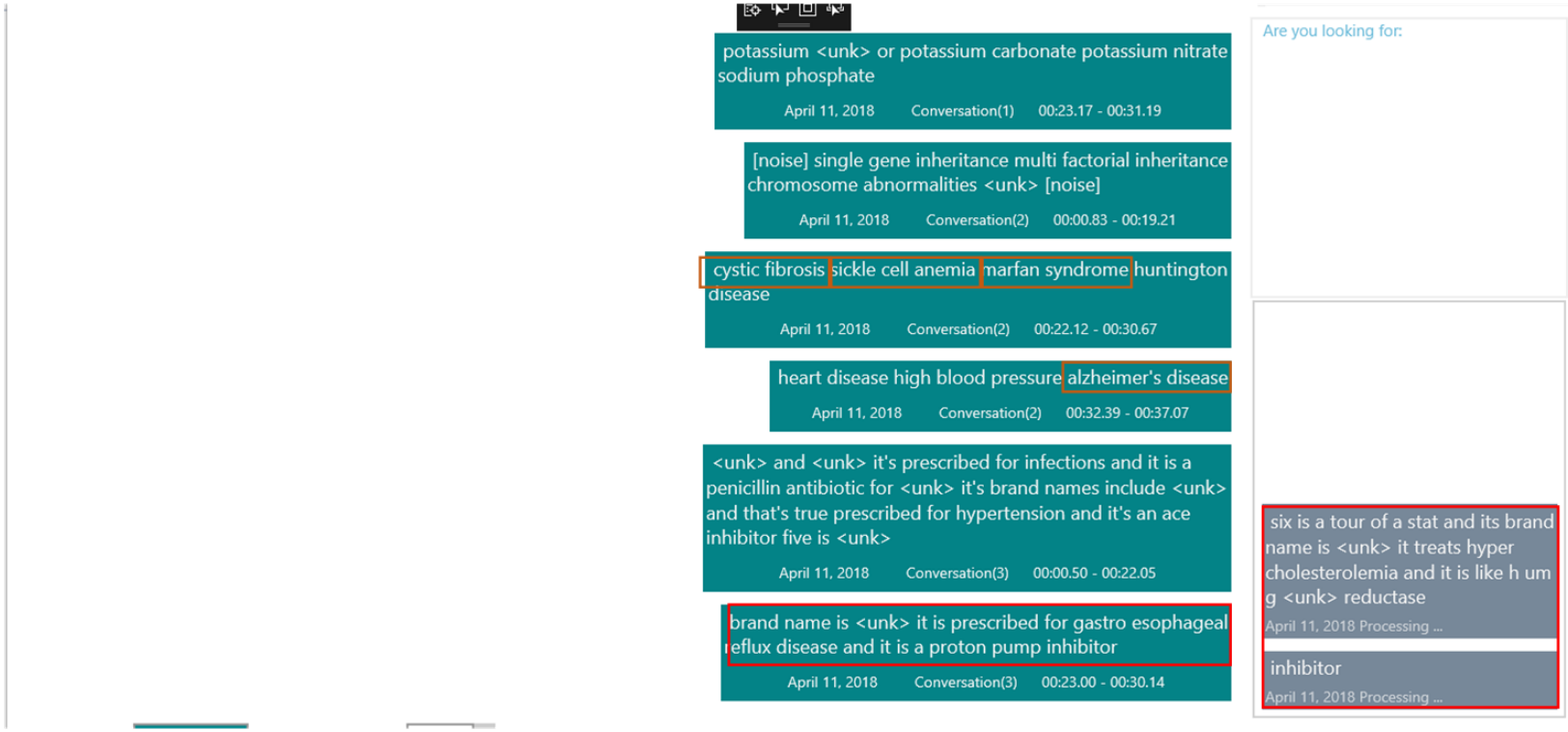


Figure 15: Speech recognition result for various chemical names, drug names and diseases

Our model is surprisingly good at terms that appear frequently on PubMed. Unfortunately, our model is not good with uni-grams. As circled in red, the engine is terrible at uni-grams, but it recognizes related diseases/symptoms just fine. This is because uni-grams naturally have smaller probability that bi-grams and tri-grams.

This page is intentionally left blank.