Part I: Overview

Virtual moderator aims to provide real-time feedback on conversations by determining *what* participants say and *how* they say it. By providing real-time metrics such as volume, pace, and sentiment, speakers can optimize the acoustics of their speech while also controlling for content by cross-referencing related topics produced by the virtual moderator.

Part II: System Flow

Analyzing pre recorded conversations is a relatively trivial task as information is always complete and computing power is virtually unlimited. However, complexity quickly arises on the frontier of real-time conversation analysis where the viability of a system is predicated upon quick, reliable results. To manage the many different tasks our virtual moderator is responsible for, we split our program into four threads— visualization, feedback, natural language processing, and audio analysis—with all threads sharing and updating a common data source.

The figure below depicts the flow for a single utterance, with each process occurring at a different time than the rest and the final chunk being stored as a speech instance object.

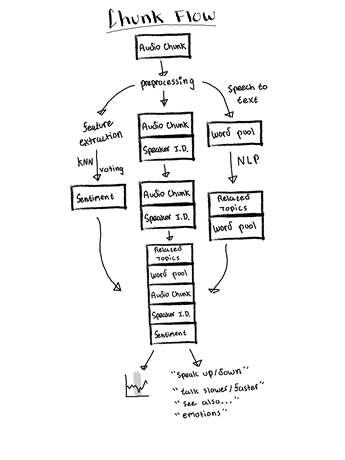


Fig 1. The flow for a single utterance

**Transcription:** While threading allows for the asynchronous completion of tasks, Google’s Speech-To-Text API provides transcription and timestamping services. Outsourcing transcription to Google allowed us to focus on the development of other metrics, although this approach constrains our system’s speed to that of Speech-To-Text, which is blocking.

Once the word-sized transcription is received from Google, Virtual Moderator collects the final chunk depicted above and places it in the conversation according to the Speaker ID and timestamps. By only processing transcription chunks, our team avoided parsing the audio stream for potential speech occurrences, which would significantly decrease our model’s speed/ reliability.

**Natural Language Processing:** Our original proposal presented fact verification as a stretch goal. After trying a variety of fact extraction techniques—including using transformers such as ROBERTA for named entity extraction—we determined that a successful run at fact verification would require time and computing resources beyond our disposal, so we pivoted to generating topics related to conversation content. Our initial attempt involved using ROBERTA[1] in conjunction with the CONLL NER dataset to develop sentence-based features which would then be used to label parts of speech, however, the order of magnitude class imbalances within the dataset and lengthy training times prevented this from being a viable approach.

Instead of shelving the BERT ecosystem as a whole— which has proven to be very effective in NLP tasks— we decided to leverage the BERT-based components of Google’s autocomplete feature through scraping. This was done after extracting noun phrases with SPACY[2]—a powerful python library founded upon pretrained neural networks—and then extracting the two most similar phrases from all possible combinations. Autocomplete suggestions were then scraped using these phrases and the top five suggestions—according to the sum of each suggestion’s similarity to each noun phrase— were returned. A simplified example of this process, and its outputs are shown below:

*Sentence*: Mayank Goel is a professor of machine learning and sensing at Carnegie Mellon.

*Most Related Phrases*: (a professor, machine learning), (machine learning, sensing)

*Top 5 Related Topics*: machine learning and sensing cmu, machine learning professor stanford, machine learning sensing, machine learning and sensing lab, adaptive sensing machine learning

As there is no standard for ranking suggestions, we were unable to filter suggestions based upon relevance. An improved analysis system should run a pre-trained transformer locally, in addition to organizing suggestions with respect to confidence. For the limited resources available in our real-time moderator, however, outsourcing computation proved to be effective, as our NLP thread induced little lag.

**Emotion detection:** The RAVDESS dataset[3] includes speech audio data for 60 trials in 24 actors, and it includes 8 different emotions. The emotion detection can be seen as a classification problem, where the input is a short audio segment, and the output is the predicted emotion class. Our pipeline first extracts MFCC features from the audio samples and then splits them into 25 ms windows with 10 ms hop between each window.

The classifier we chose is kNN classifier. We were able to achieve 92.5% test accuracy training kNN using the RAVDESS dataset with 10-fold cross validation.

**Speaker Identification:** In order for the system to work in a real conversational environment where multiple participants speak at the same time, the system must be able to differentiate between different speakers in order to give them individualized feedback. The task can be done in two methods.

The first method is Spectral Clustering[4]. It first constructs an affinity matrix which is the cosine similarity between audio segments in a conversation. The affinity matrix is then refined using multiple techniques including Gaussian Blur, Thresholding, Symmetrization, Diffusion and Normalization. Then the matrix is eigen-decomposed into eigenvectors and eigenvalues. The number of clusters is determined by finding the largest gap in the eigenvalues. Each segment is embedded by the eigenvectors, and finally the K-Means algorithm is used to find out the clusters. The advantages of this method are that it does not involve deep learning and can thus be calculated without GPU quickly, and that it does not require prior knowledge of speakers (hence no enrollment), but the largest disadvantage is that it needs the whole conversation audio data in order to update the clusters, and this makes it unsuitable for realtime diarization.

Another approach to this challenge is to first have each speaker enroll in the system, and the system would train some “model” for this individual speaker, and later during the conversation the system would take a segment of the audio samples, put them into the models, and get similarity scores with different speakers. The speaker with the highest score is then the predicted speaker. Traditionally the model used would be Gaussian Mixture Model (GMM), where the voice of each individual is modeled as a combination of Gaussian distributions, and the parameters (the combination itself and the mean/variance of each distribution) are set during the enrollment process so that the training data gets a high similarity score in the GMM. In the conversation the likelihood scores of each audio segment are then calculated against all GMMs of the participants. In our experiments we used MFCC as our voice features and we’ve found out GMM didn’t perform well and is extremely prone to different kinds of biases, including volume differences, silences in training data and background noises. Before the presentation we didn’t have time to complete another approach which is based on neural networks, so we had to present this one with bad results.

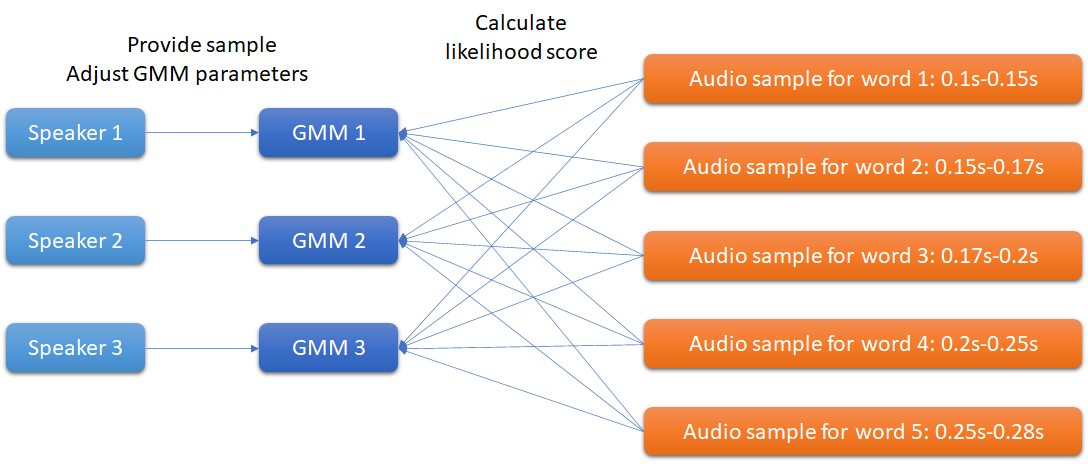


Fig 2.The structure of GMM speaker identification system

The neural networks based method uses LSTM to create an embedding representation vector[5] for each speaker during enrollment phase, and in the conversation the audio segments go through the same LSTM embedding network, and the resulting vector is then compared with the speaker representations using cosine similarity. The input to the LSTM is the MFCC sequence of the audio segment, and the last state of the LSTM is put into a linear layer, and the output of that layer is the final embedded vector. To train this LSTM embedding network we used the TIMIT dataset which includes about 600 different voice actors with 10 sentences each.

Additionally a generalized end-to-end loss function for speaker verification[6] is employed. After 950 epochs the equal error rate (the error rate when the false positive and false negative rate is the same with some acceptance threshold) reaches 0.0381. After this embedding network is trained, we feed the network with 1 minute enrollment speech data of each participant in order to create representation of each speaker. In the conversation, we first detect if the audio is silence by utilizing the WebRTC voice activity detector, and the actual speech data segments are fed into the network, and the embedded vector is compared with speaker representations using cosine similarity, and the one with the highest score is taken as the prediction. After the presentation we completed a prototype of this neural network based real time speaker identification mechanism.

We utilized the code at <https://github.com/HarryVolek/PyTorch_Speaker_Verification> as the basic LSTM model and training code, and built upon it to add real time identification function to it. The resulting code is available at <https://github.com/dodo0822/PyTorch_Speaker_Verification>. Also demo videos are available at <https://drive.google.com/file/d/1Guyx0Pkvv26UxvvzqkVGNW8MHu4tX9gt/view?usp=sharing> and <https://drive.google.com/file/d/1o9ZVI9xMM9T3xejv2rSrJZ3-osyymCyv/view?usp=sharing> . In our experiments it worked much better than GMM, but it is also susceptible to biases that occurred in the enrollment data such as volume difference and background noise. To mitigate this we use the pydub library to normalize the volume across different enrollment audio clips and librosa to detect voice activity within each audio clip.

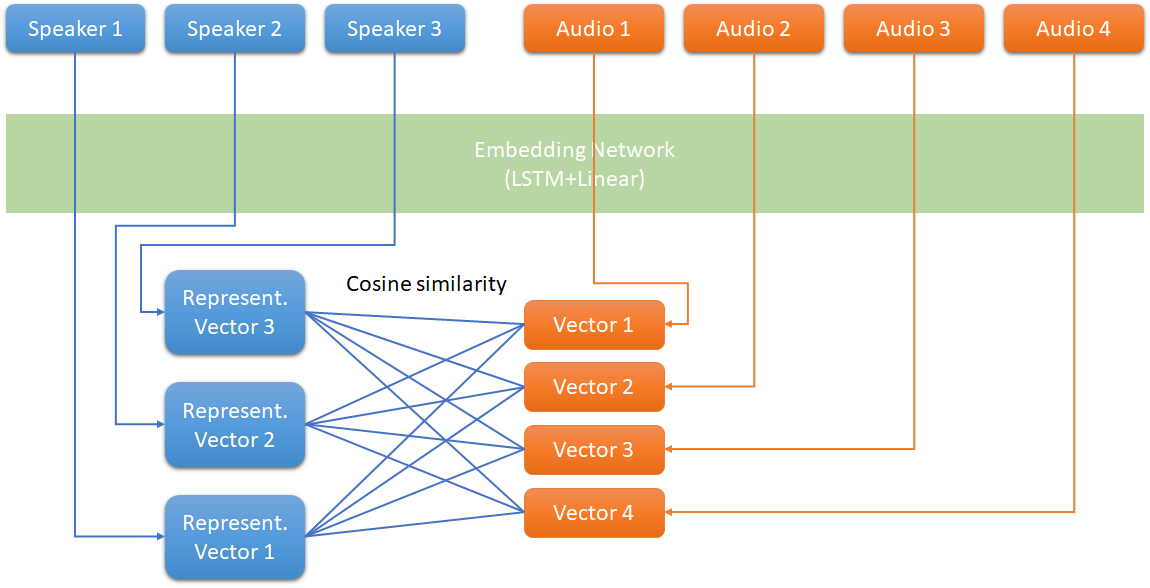


Fig 3. The structure of LSTM speaker identification system

However the challenge of volume difference still persists if the conversation takes place over a Zoom meeting where each participant has different volume in the conversation, because there is no way to normalize a partial audio segment without knowing the overall average volume of a whole utterance, so in the demo sessions I had to adjust Zoom’s volume so our volumes are balanced. We conclude that the preprocessing of the audio data is extremely important in order for the speaker identification system to work properly.

Part III :­ Display

Currently we display all the metrics on a simple text box in figure 2. Also a graph of the diarization result is displayed in figure 3. The metrics displayed include word per minute, percentage of participation, related topics and suggestions based on the metrics.

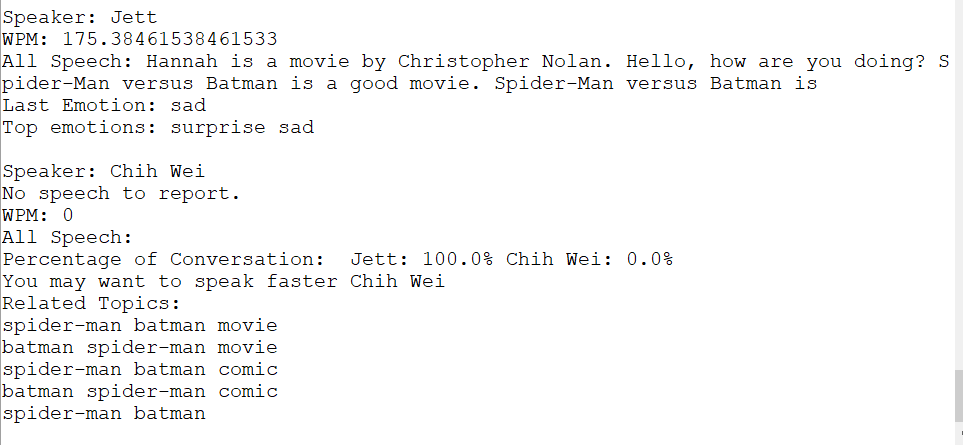


Fig 4. The display of the metrics in text

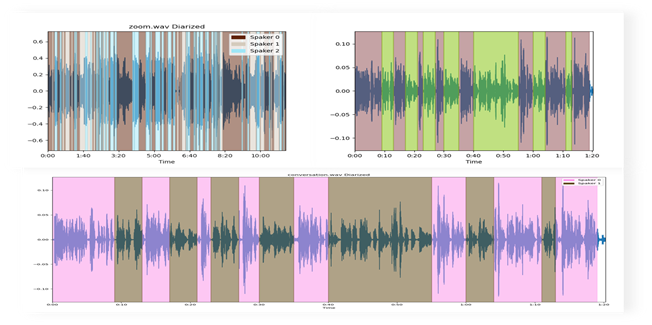


Fig 5. The diarization result

Part IV : Conclusion & Future development

In this project we developed a system which is capable of identifying individual speakers during a conversation and providing metrics and suggestions for the participants. The speech diarization and speaker identification is important and must be accomplished with high accuracy if the conversation occurs face-to-face. In our experiments we see that the neural network based method performs far better than traditional machine learning techniques such as GMM.

To make this component more robust we might need to come up with data normalization during real time conversation, and the continuous refinement of the enrollment data. For example we can update the speaker representation (and perhaps even the LSTM embedding network) at some time point during the conversation if the prediction confidence is high, and if the system yields low confidence it should ask the user to manually label the data so the system can train the model continuously in a semi-supervised learning.

The metrics should be displayed in a more unobtrusive way, such as a transparent screen overlay that the user can put onto a Zoom meeting, and we might want to give more suggestions based on the metrics instead of directly displaying the raw numbers.

The current workflow of the system allows only a small degree of data sharing between tasks. That is, in each task the features are calculated individually. If we add more metrics that depend on machine learning techniques, then the overall system performance might decrease quickly. We would need to introduce some degree of data sharing between tasks. For example, if more than one task needs the MFCC features of the audio segments, then we should do that only once and share the features between tasks. Also the tasks should ideally be independent of each other. For example, we want to perform the calculation of each metrics parallely, and after all the metrics are calculated for a given time period the system finally gives suggestions based on the metrics. Each individual program feature might be easy to implement on their own, but when they come together to form a large system, it is a much more difficult task to make them perform in high performance in a real time environment.

Part V: Reference

1. Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, & Veselin Stoyanov. (2019). RoBERTa: A Robustly Optimized BERT Pretraining Approach.
2. Honnibal, M., Montani, I., Van Landeghem, S., & Boyd, A.. (2020). spaCy: Industrial-strength Natural Language Processing in Python.
3. Livingstone SR, Russo FA (2018) The Ryerson Audio-Visual Database of Emotional Speech and Song (RAVDESS): A dynamic, multimodal set of facial and vocal expressions in North American English. PLoS ONE 13(5): e0196391. <https://doi.org/10.1371/journal.pone.0196391>.
4. Quan Wang, Carlton Downey, Li Wan, Philip Andrew Mansfield, & Ignacio Lopez Moreno. (2018). Speaker Diarization with LSTM.
5. Georg Heigold, Ignacio Moreno, Samy Bengio, & Noam Shazeer. (2015). End-to-End Text-Dependent Speaker Verification.
6. Li Wan, Quan Wang, Alan Papir, & Ignacio Lopez Moreno. (2020). Generalized End-to-End Loss for Speaker Verification.