# Voice Cloning
# Project Report : CS 7643

Andrew Perry Chu
achu47@gatech.edu

Richie Youm
syoum3@gatech.edu

Nolan Piland
njpiland@gatech.edu

James Liu
jliu348@gatech.edu

## Abstract

*Text-to-Speech (TTS) systems have become commonplace in today's society. TTS systems are found in day-to-day applications like virtual assistants, but they are also applied in critical assistive technologies for people with visual or vocal disabilities. We aim to explore state of the art techniques which employ deep learning to build models that perform TTS translation. We also aim to explore methods for extending these models to use short voice samples as an additional input, generating speech which effectively clones that speaker's voice. We followed a transfer learning approach, training a first model on a speaker verification task [4], in order to generate a speaker representation. Then, we modified a known TTS model, Tacotron2 [13], to use the speaker representation as an additional input to reproduce the nuances of the speaker's vocal characteristics in the output speech. In this paper, we lay out the motivations, approach, challenges, and results we encountered in our exploration of these techniques.*

## 1. Introduction

### 1.1. Objective

Text-to-Speech (TTS) systems are programs which aim to convert text to audio reflecting human speech. A number of neural architectures have been created which succeed in this task, however a current area of research is how to modify these systems to control the vocal characteristics (accent, tone, pacing, etc.) of the generated speech. In this paper, we explore a method for few-shot ”voice cloning” - in other words, creating a TTS system which given input text and a small voice sample from an arbitrary speaker, produces audio reflecting the input text with that speaker's vocal characteristics.

### 1.2. Relevance

Many people in today's society interact with TTS systems - a common example is virtual assistants or GPS navigation where voice feedback is provided to the user by generating text then ”reading” it back to the user. In addition, TTS systems are critical tools for individuals with visual or vocal impairments in the form of screen readers and speech-generation devices respectively. Typically, TTS systems offer limited customization outside of selecting regional dialects for a small set of fixed voices, and require large quantities of recordings of an individual speaker's voice to develop natural sounding models.

TTS approaches which enable ”voice cloning” would allow users to fully customize speech generation. Applied successfully, this would introduce a highly personalized experience to TTS interaction, allowing each user to listen to voices which fit their comfort or create speech which best represents them. These models also have the potential to drastically cut the amount of data needed to develop a customized TTS experience if the models are able to generalize to various speakers well.

### 1.3. Current Practice

Today, basic neural TTS is typically performed in two stages as two separate sequence-to-sequence tasks. Between the two tasks, a mel spectorgram is used as an intermediate frequency representation. The mel spectrogram is generated by applying short-time fourier transform to the raw audio to create a frequency spectrogram, then applying mel scaling which mimics human perception of pitch.

The first TTS stage is a text to mel spectrogram encoder-decoder which takes in a text sequence, creates an encoding based on character or phoneme embeddings, then decodes the sequence to the mel spectorgram frequency representation. The second TTS stage is a mel spectrogram to waveform vocoder converts the mel spectrogram frequency representation into a raw audio waveform.

These models are typically trained on a large number of samples of one individual's voice. This presents limitations in requiring significant length of high-quality recordings from a single speaker. In addition, the generated speech mimics the vocal characteristics of that individual.

More recent research in the area of TTS and ”voice cloning” introduces the concept of a speaker encoder [4]. The speaker encoder produces a speaker embedding given audio samples of a speaker's voice. The speaker embedding is then introduced to the mel spectrogram encoder-decoder in addition to the text input, so that the encoder-decoder can learn to produce speech output conditioned on the speaker's voice.

### 1.4. Data

We used two datasets commonly used for TTS model training, LibriSpeech[7] and LJSpeech[3]. Both datasets contain pairings of text snippets from public domain writing combined with audio clips of English speakers reading those texts. We modified the base datasets by normalizing the audio to range between -1.0 to 1.0, and generated mel spectrograms from the normalized audio for training purposes.

Librispeech is a large dataset of audiobook recordings matched with text that contains multiple partitions of varying size and quality. We used the train-clean-100 subset, which is annotated as being clean recordings (limited background noise). This set contains roughly 100 hours of 16 kHz recordings from 251 unique English speakers reading public domain books from Project Gutenberg. The speakers include 126 male and 125 female speakers, and each speaker has around 25 minutes of recorded speech.

LJSpeech is a smaller dataset containing roughly 24 hours of 22.5 kHz recordings of a single female speaker reading public domain books and essays. This set was the base voice used in the original training of Tacotron2 [13].

For both LibriSpeech and LJSpeech, we generated mel spectrograms from normalized audio for training purposes.

## 2. Approach

### 2.1. Research Goals

Our project goal was to explore the use of speaker embeddings in the scope of TTS applications. In doing so, we aim to solidify our understanding and of data embedding as a tool for representing complex concepts within deep learning, as well as to better understand different applications of transfer learning.

This was done in three main parts. First we develop and train our own speaker encoder model according to current research in the field [4, 12], with the goal of generating speaker embeddings which the distinct capture the vocal characteristics of a speaker. Second, we modify and train existing neural TTS architectures [13, 8] to incorporate a speaker embedding as an input, with the goal of enabling voice cloning. Third, we construct a zero-shot environment which simulates an individual customizing a TTS experience, and evaluate quality of the generated speech.

$$L_{TI-SV}(\mathbf{e}_{ji}) = -\mathbf{S}_{ji,j} + \log \sum_{k=1}^{N} \exp(\mathbf{S}_{ji,k})$$

Figure 1: Text-Independent Speaker Verification Loss [12].

### 2.2. Speaker Encoder

#### 2.2.1 Speaker Verification

The most common approach to developing speaker encoders for TTS in academic literature is transfer learning from a speaker verification task [4], where the model attempts to predict whether two voice samples are from the same speaker.

Our speaker encoder model was adapted the architecture described by Jia *et al.* [4] with minor modifications. Our speaker encoder model is a three layer LSTM which takes in arbitrary length 80-channel mel spectrogram as input and has a 256-dimension hidden state. We apply a fully-connected layer with ReLU activation on the final LSTM hidden state to create the output speaker embedding.

#### 2.2.2 Training

To train our speaker encoder we followed the methodology described by Wan *et al.* [12], which describes a scalable loss metric for training generic speaker verification models. In particular, training is conducted by generating embeddings for a batch of N speakers with M utterances per speaker. The embedding of each utterance is compared against the average embedding of each speaker using cosine similarity. The target similarity is 1 when the utterance is compared to its own speaker's average embedding, and it is zero when the utterance is compared against other speakers. The target similarity is put into a matrix form for loss calculation.

#### 2.2.3 Softmax Loss

Wan *et al.* outline two different loss metrics for speaker verification which are suited for two different speaker verification environments: Text-Independent and Text-Dependent [12]. Given that our training dataset, LibriSpeech, contains speech samples of text which varies from sample to sample, we opted to use Text-Independent Speaker Verification Loss [12] in training our speaker verification model. Wan *et al.* define this Text-Independent Speaker Verification loss as shown in figure 1 where **S** represents values from a similarity matrix.

#### 2.2.4 Challenges

When training the speaker encoder model, we initially used smaller batch sizes of four speakers and four utterances per speaker. Validation accuracy and loss peaked very quickly and We later realized that this format limited the amount the model was able to learn since it was only asked to differentiate between a small amount of speakers in each instance. Experimentation with higher speakers and utterances per batch yielded better results.

When evaluating our speaker encoder model, we found that the generated speaker embeddings were sparse, having mostly zero values. Although empirically through clustering and use in Tacotron2 we found that the embeddings were useful for discriminating between speakers, we believed a more dense representation may provide more information about speaker characteristics to Tacotron2.

When applying the speaker encoder model to Tacotron2 training, we found that the mel spectrograms generated in the Tacotron2 pipeline were different from the format our speaker encoder was implemented with which lead to diverging results. Performance improved after retraining the speaker encoder using the Tacotron mel format, normalizing the audio, and removing zero padding when running the speaker encoder.

## 2.3. TTS Architecture

To generate speech, we used an existing TTS implementation as a baseline and modified the text to mel spectorgram encoder-decoder to work with our speaker embedding model.

### 2.3.1 Tacotron2

Tacotron2 is a speech synthesis model first introduced in Natural TTS Synthesis by Conditioning WaveNet on Mel Spectrogram Predictions [13]. It uses a sequence-to-sequence architecture to convert character strings into mel spectrogram predictions which a vocoder network, like WaveNet, can be used to generate audio outputs.

On the input side Tacotron2 creates a per-character embedding, which is fed through a series of three convolution layers and a bidirectional LSTM layer. The encoder output is consumed by an location sensitive attention layer before being decoded in the same way as the encoder in reverse.

We started with a baseline Tacotron2 implementation provided by Nvidia [5]. The original architecture was modified to concatenate the speaker embedding to each individual character embedding after the initial convolution layers. This ensured that the speaker embedding was not spatially correlated with character inputs, and that the embedding was present at every time step.

During training, the data loader was to be modified to load a pre-trained checkpoint of the speaker embedding model and run inference on the computed mel spectrogram for every audio sample. Any alteration to the speaker embedding model requires retraining the Tacotron2 model to match. The loss used was the same as in the original Tacotron2 implementation, MSE loss on the output mel spectrogram values and binary cross entropy on the gate values used to indicate end of sequence. At inference time, a provided recording would used to generate the embedding.

### 2.3.2 WaveGlow

WaveGlow is a neural vocoder first described in Waveglow: A flow-based generative network for speech synthesis [8]. WaveGlow improved on prior vocoders by introducing a architecture without auto-regression, enabling much faster training and inference while maintaining quality similar to state of the art models.

We used a WaveGlow implementation provided by Nvidia [6] as the vocoder to generate audio from our modified Tacotron2 generated mel spectrograms. We treated the vocoder as a fixed model using the provided pre-trained weights and did not attempt to fine tune the output.

### 2.3.3 Challenges

The base Tacotron2 model was originally trained on the LJSpeech dataset, which only included one speaker. This led to the model ignoring the speaker embedding as an input, so additional alterations needed to be made to adapt the data loader to work with Librispeech. This potentially introduced bias as this is the same dataset that the speaker embedding model was trained on: any success training Tacotron2 using it might not generalize to the entire embedding space.

In our original proposal, we indicated that we planned on using a pre-trained instance of Tacotron2 to facilitate rapid prototyping of a speaker encoder. As we progressed, we discovered that we needed to make modifications to the base Tacotron2 implementation to be able to consume the speaker embedding output by the speaker encoder. This ultimately disallowed our initial plans to use an existing trained instance of Tacotron2. As a result, we wound up having to train Tacotron2 on our own which introduced an unforeseen time-delay in terms of how quickly we could prototype and iterate on our speaker encoder. This significantly impacted our ability to experiment with both the end-to-end TTS pipeline and speaker encoder.

## 2.4. Integration

After training the speaker encoder and TTS models, we created an integrated inference system to test the output. The system takes an arbitrary speech sample and text as input. The speech sample is converted to mel spectrogram and a speaker embedding is created based using the speaker encoder. Then the text and speaker embedding are fed into our modified Tacotron2 followed by WaveGlow to generate speech samples for evaluation.

## 3. Experiments and Results

### 3.1. Speaker Encoder

Due to resource-intensive training for Tacotron2, our initial experiments focused on optimizing the quality of the speaker encoder by itself. The primary variable we experimented with was the batch size, or the number of speakers and utterances per speaker in each training instance. We started with smaller batches of 4 speakers and 4 utterances, but through experimentation found that 32 speakers and 8 utterances per speaker generated the best results.

#### 3.1.1 Evaluation

To analyze speaker encoder performance, we looked at trends in a variety of metrics: the loss being optimized in training, the accuracy of the model being generated, and overall how well the embedding being generated from the speaker encoder clustered around utterances of the same speaker. The loss and accuracy measures are readily self-explained. In evaluating clustering performance, we analyzed silhouette scores [10]. Quantitative values (loss, accuracy, and silhouette scores) were computed and tracked over a reserved validation set and plotted over a per-epoch basis. Figure 2 shows trend lines of these values across training a speaker encoder for 300 training epochs.

Ideally, we're looking for minimized loss, maximized accuracy, and maximized silhouette scores. As shown in figure 2, we begin to see performance trail off in later epochs and any performance gains seen appears to be due to run-to-run variance in performance. That being said, the overall quality of the scores we're seeing appear to be lacking. In terms of silhouette scores per speaker, we would expect to see higher values which would indicate more healthy clustering, and thus, more healthy indication that the speaker encoder is correctly providing a representation of a speaker from an utterance independent of the text being read. Additionally, loss appears remain rather large (600) and accuracy remains relatively low (40%) across the entirety of the training process. We ultimately chose to proceed with the project leveraging the model trained after 273 epochs in hopes to better analyze what exactly was going on with the model.

Visually inspecting some of the embeddings being output by our selected speaker encoder, we found that our encoder was outputting very sparse representations of a speaker, with a majority of values in the embeddings being "0". This could potentially be due to a lack of varying data needed to build up a proper speaker encoder. This being said, we continued to explore the embeddings being produced to see how well it appeared to cluster in speaker-space.

### 3.2. Clustering

Intuitively, there are two clusterings of utterances we considered, one being by each speaker itself and another by gender. Clustering by speaker is our ultimate goal for the speaker encoder; clustering by gender is something we believed would mirror speaker performance since male and female voices typically have tones which fall between different bands. Given the number of clusters for gender, we believed we would see quick improvements in gender-based silhouette scores (figure 2d), which we did see in early training epochs. However, we saw that performance begin to fall off in later epochs, seeming to indicate that the encoder is attempting to learn higher-level representations beyond tone, or something other may be wrong with the implementation.

Despite seeing concerning behavior across some of our quantitative metrics, we performed additional qualitative analysis in the form on t-SNE over output embeddings over the reserved validation set. Figures 3 and 4 provide t-SNE visualizations for the model trained for 273 epochs across gender and 10 randomly selected speakers respectively.

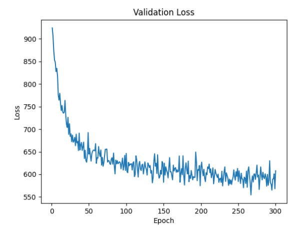
In looking at the t-SNE plot over gender, it appears that there is indeed some clustering around gender. This clustering wasn't as distinct as we originally believed it would be, but this shows some promise that our encoder is beginning to tease out nuances in speaker verification and may be building out a speaker space which can capture features which make up a speaker's voice.

To maintain clarity in plotting results on a per-speaker basis, we show the embeddings generated from all utterances from a random sample of 10 speakers in figure 4. Here, we begin to see some of our more promising results as we see there are indeed some distinct clusters being formed for some speakers. This being said, there are still some clusters not being correctly partitioned, specifically the speakers represented by the yellow and green points. However, this gave us hope that the model was performing some form of proper speaker encoding!

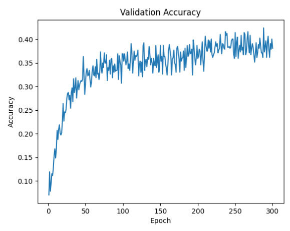
### 3.3. End-to-End TTS

Producing intelligible results by Tacotron2 with speaker embedding required several iterations. Training from scratch on LibriSpeech, which contains multiple speakers with around 30 minutes per speaker produced only unintelligible noise even after multiple days. After finding better success bootstrapping with training on a single speaker, we believe the variation in speaker was too great to allow the model to converge properly.

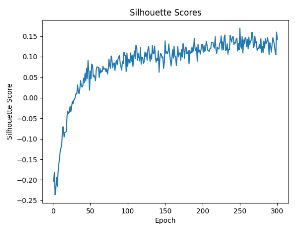
Our most successful strategy was to first train Tacotron2 with speaker embedding on the LJSpeech dataset, then further train it on LibriSpeech dataset. As mentioned previously, LJSpeech consists of only 1 speaker, which intuitively makes this dataset less ideal for the voice cloning task. However, we were interested in observing whether



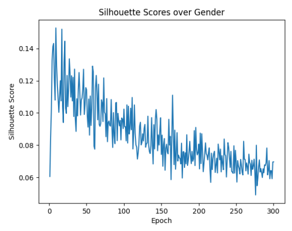
(a) Validation loss



(b) Validation accuracy



(c) Silhouette scores per speaker



(d) Silhouette scores per gender

Figure 2: Per epoch plots of quantitative metrics over 300 epochs of speaker encoder training.

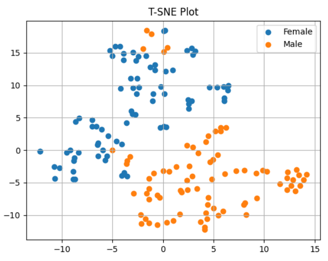


Figure 3: T-SNE plot by Speaker Gender

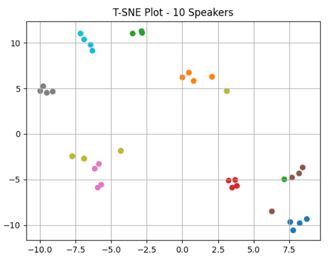


Figure 4: T-SNE plot of 10 random speakers after 273 epochs

the characteristics of the speaker such as intonations would be reflected in the generated audio, which is an important part of representation in speaker embedding. Furthermore, we hypothesized that a large single speaker data such as LJSpeech will be easier to learn the "speech" aspect, in which the "cloning" part could be learned from LibriSpeech data with multiple speakers and greater audio variance. Therefore, our goal of the experiments given the limitations was to observe a clear distinction between generated audio, even though they may not sound exactly like the original speakers.

To compare the audio generation performance, we used short audio clips from 4 well-known figures with the clips ranging 5 - 15 seconds: *Hermione* [11] from *Harry Potter*, pop singer *Ariana Grande* [2], CEO of Tesla and SpaceX *Elon Musk* [1], and former president of the United States, *Barack Obama* [9]. These speakers were chosen on the basis that they have relatively distinctive accents, depth, and intonations, therefore will be easier to recognize if the model learned the speaker characteristics. The clips were set to sampling rate of 16000 and converted from stereo to mono before being passed to speaker encoder model, due to the tacotron2 implementation we're leveraging.

We trained tacotron2 on LJSpeech dataset from scratch up to 27000 iterations with batch size of 32, which we found working slightly better compared to the smaller batch size of 16. While this comparison was made during early epochs when the generated audio still did not correlate well to the target text, we decided to focus our computing resource on the model producing better-sounding results at the time.

From the training results, we used a few checkpoints to generate audio for each speaker given the example text "*I need more time and resources.*", which we find it a befitting sentence given the nature of this project. In the earlier epochs, the generated output was mostly static noises, and at around 10000 iterations, parts of human words were present. At 27000, we were able to successfully produce fully understandable audio (Hermione, Ariana, Obama, Elon in sequence) which had identifiable varying emphasis and intonation in the generated speech between speakers.

Without even listening to the audio, we can also infer the pace of the words being pronounced, as well as the breaks between words such as 'time' and 'and' from the empty signals in mel-spectogram and horizontally spanning center part of the alignment in Figure 5.

However, it was still difficult to attribute each audio to a particular speaker, as the output audio had no significant difference in the speaker tone and depth, which was expected given LJSpeech's constraint. We then continued to train this model on LibriSpeech for about 17000 iterations and tested two checkpoints - iterations 14000 and 17000 for comparison.

On iteration 14000, the model was able to pick up on speaker-specific characteristics as we expected, most notably on the depth and frequency of the voice. The words

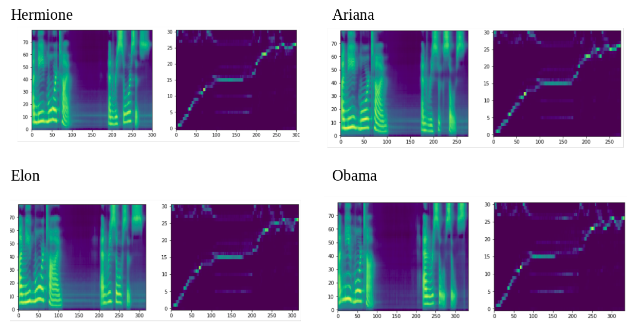


Figure 5: Generated mel-spectogram and encoder-decoder alignment

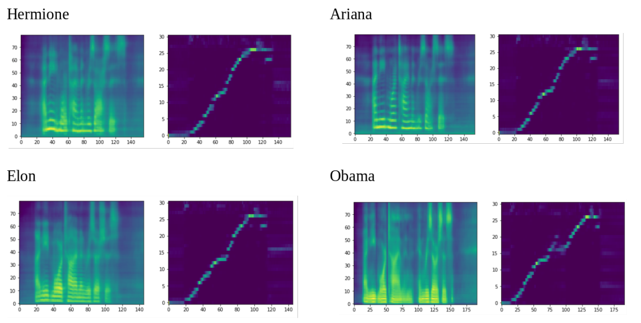


Figure 6: Generated mel-spectogram and encoder-decoder alignment for our best model

in the generated audio (in the same order) were a lot more coherent, and were spoken at a more natural pace. Both female speakers had a different voice from before – which in relative terms, are slightly closer to the original speakers, with *Ariana Grande*'s version having a slightly lighter voice compared to *Hermione*'s. Looking at the visualization shown in Figure 6, we also notice a smoother transition of mel-spectorgram and stronger encoder-decoder alignment compared to Figure 5.

More interestingly, male speakers had a more drastic difference. *Elon Musk*'s voice was much deeper than the female counterparts and the previously generated audio, and *Barack Obama*'s voice was even deeper than *Elon*'s – which is what we would expect. While they did not sound exactly like the original speakers, we were able to capture some parts of their speaker characteristics in the generated audio.

Interestingly, from iterations 17000 and onward, we noticed a sign of the model overfitting to the speaker embedding rather than producing intelligible sound such as this example from *Elon Musk*'s voice. While the depth and the tone representing the speaker was present similar to iteration 14000, the generated output experienced a severe degradation of text-to-speech quality as seen from the alignment in Figure 7.

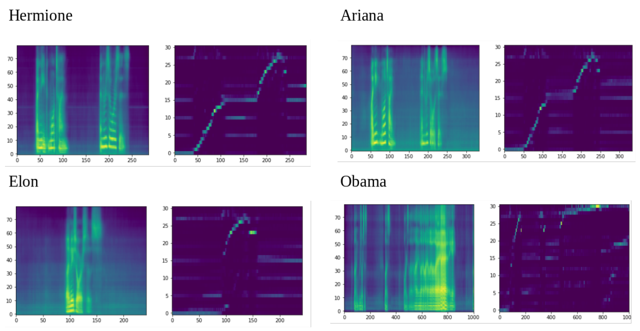


Figure 7: Generated mel-spectogram and encoder-decoder alignment for overfit model

From these results, we concluded that the model effectively learned different speaker features from speaker embedding concatenated with the convolution outputs in the lower layers, which means that our experiment was successful. For future iterations, more consideration should be put towards improving speaker encoder to better clone the original voice, as well as paying closer attention to the pre-processing required for each dataset.

## 4. Work Division

See Table 1.

## 5. Appendices

### 5.1. Code Implementation

Code implementation / modifications can be found on GitHub.

## References

[1] YouTube clip by *Elon Musk Sound Bites*. Elon musk on fossil fuel power for electric cars. https://www.youtube.com/watch?v=GT8xgUx5WGk. 5

[2] YouTube clip by *fr0zenintimeee*. *ariana grande speaking voice evolution*. https://www.youtube.com/watch?v=QxbbjJXebIQ. 5

[3] Keith Ito and Linda Johnson. The lj speech dataset. https://keithito.com/LJ-Speech-Dataset/, 2017. 2

[4] Ye Jia, Yu Zhang, Ron Weiss, Quan Wang, Jonathan Shen, Fei Ren, Patrick Nguyen, Ruoming Pang, Ignacio Lopez Moreno, Yonghui Wu, et al. Transfer learning from speaker verification to multispeaker text-to-speech synthesis. *Advances in neural information processing systems*, 31, 2018. 1, 2

[5] NVIDIA. Tacotron 2. https://github.com/NVIDIA/tacotron2, 2020. 3

[6] NVIDIA. Waveglow. https://github.com/NVIDIA/waveglow/tree/

| Student Name | Contributed Aspects | Details |
|---|---|---|
| Richie Youm | Implementation, Analysis | Implemented forward pass and model connection of Speaker encoder. Integrated generated speaker embedding output to Tacotron2 model by concatenating with convolution layer outputs before LSTMs. Generate and compare sample audios with checkpoints throughout model training. Resolve dependency compatibility issues. |
| Andrew Chu | Implementation, Training | Implemented similarity matrix for GE2E [12] loss, finalized speaker embedding training script, adapted Tacotron2 dataloader to enable speaker embedding generation, ran large-batch speaker encoder experiment, set up and ran TTS model training experiments on GCP |
| Nolan Piland | Implementation, Analysis | Contributed GE2E [12] TI-SV loss implementation, ran preliminary exploration of speaker encoder, and provided speaker embeddings visualization for analysis |
| James Liu | Implementation | Implemented initial training script for speaker embedding model and made alterations to the Tacotron2 data loader to support LibriSpeech. |

Table 1: Contributions of team members.

5bc2a53e20b3b533362f974cfa1ea0267ae1c2b1, 2020. 3

[7] Vassil Panayotov, Guoguo Chen, Daniel Povey, and Sanjeev Khudanpur. Librispeech: An asr corpus based on public domain audio books. In *2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 5206–5210, 2015. 2

[8] Ryan Prenger, Rafael Valle, and Bryan Catanzaro. Waveglow: A flow-based generative network for speech synthesis. In *ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 3617–3621. IEEE, 2019. 2, 3

[9] American rhetoric. Barack obama speeches. https://www.americanrhetoric.com/barackobamaspeeches.htm. 5

[10] Peter J Rousseeuw. Silhouettes: a graphical aid to the interpretation and validation of cluster analysis. *Journal of computational and applied mathematics*, 20:53–65, 1987. 4

[11] *Harry Potter and the Philosopher's Stone*. YouTube clip by *Wizarding World*. *It's Leviosa, Not Leviosaaa!* https://www.youtube.com/watch?v=Qgr4dcsY-60. 5

[12] Li Wan, Quan Wang, Alan Papir, and Ignacio Lopez Moreno. Generalized end-to-end loss for speaker verification. In *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 4879–4883. IEEE, 2018. 2, 7

[13] Yuxuan Wang, RJ Skerry-Ryan, Daisy Stanton, Yonghui Wu, Ron J Weiss, Navdeep Jaitly, Zongheng Yang, Ying Xiao, Zhifeng Chen, Samy Bengio, et al. Tacotron: Towards end-to-end speech synthesis. *arXiv preprint arXiv:1703.10135*, 2017. 1, 2, 3