Designing and Evaluating a Forced Alignment Pipeline

James William Stonebridge
Herbert Alexander De Bruyn
Tyler Dierckman
Purdue College of Electrical and Computer Engineering
Under the supervision of Dr. Joseph G. Makin

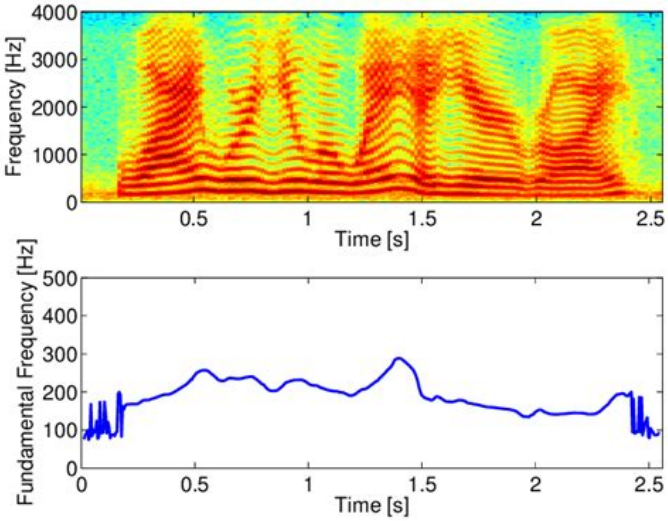

Fig. 1: Top: A speech file in the time domain, Bottom: A spectrogram of a speech file (frequency Domain). Notice how the Phonemes are more clearly separated in the spectrogram

***Abstract*—The field of speech decoding from neural activity faces a considerable challenge with the amount of labels needed to train a model to perform such a task. This burden can be lessened by reducing the number of possible labels.**

In this paper, we discuss the implementation and development of a forced alignment python package. How we Approached the pipeline design, what challenges we faced, and the limitations associated with the final design. The study encompasses everything from the initial setup and testing with non-human data to the actual experiments with relevant data from Makin Lab.

It includes the analysis of the extensive testing on the speech-to-text transcription Wav2Vec ASR model, as well as a discussion on the inconclusiveness of the word error rate results. Such outcome led to the decision of considering in the pipeline multiple transcription methods to be used with Montreal Forced Aligner - the chosen phoneme alignment model - which can be specified or determined with a threshold (Critical Error).

I. Introduction

Phonemes, which can be defined as the smallest units of sound in a particular language contributing to differences in meaning [1], are commonly relied upon in speech processing. They allow for the tokenization of linguistic elements beyond words, yielding a higher frequency of linguistic information in the time domain for a given segment of speech.

This has the potential to be useful when attempting to generate a map between a secondary input and speech by providing more distinct data points. For this purpose, phonemes for a speech segment are often generated through the use of a process known as forced alignment, by which orthographic transcriptions are automatically aligned to audio, generating a time-aligned list of phonemes [2]. Although this automatic process cannot yet rival the accuracy of manual phoneme alignment by trained linguists, it is often the only feasible way to generate phoneme alignments for large speech datasets, and is thus an important tool in speech processing.

This paper covers the exploration of existing forced alignment tools as well as the construction of a forced alignment pipeline for the use of Makin Lab. In part, Makin Lab focuses on decoding speech from implanted neural electrodes in humans. Previously, it has mapped electrode data using words as targets [3], but it sees value in the possession of a forced alignment system which could provide accurate phoneme alignments should they ever be needed. The goal of this research was therefore to create a simple-to-use pipeline capable of perform-

ing accurate forced-alignment given patient speech and original transcriptions.

II. Prior Work

A. Attempted Setups

Prior to completing The Collaborative Institutional Training Initiative's Human Subjects Research Basic Course and being granted access to the human data, the team decided to use Gilbreth. Gilbreth is a community Cluster optimized for running GPU intensive applications such as machine learning - in order to run the models used for phoneme alignment and speech-to-text transcriptions, allowing more flexibility if any model required further training. However, in doing so, the group encountered some issues when initially attempting to setup the repository.
To begin, it was necessary to check and often change all imported module file paths so they could be encountered within the directory.

Secondly, the group had some difficulties installing the necessary libraries due to kernel issues within Gilbreth, which were fixed by installing a package in conda with a python kernel to be used with a Jupiter Notebook, which allowed the packages to be installed via *pip install* and the code to run normally. The following code was used to fix the latest issue:

Algorithm 1 Environment Setup

Bash
$ conda activate cenv
(cenv)$ conda install ipykernel
(cenv)$ ipython kernel install –user –name any_name_for_kernel
(cenv)$ conda deactivate

B. Charsiu

The first of two forced aligners we looked at was Charsiu [4]. Charsiu is presented as a specialized successor of Wav2Vec2 [5]. The model reuses (most of) it's predecessor's architecture and trains on the TIMIT and Librispeech corpuses.

Charsiu fine-tunes the pretrained transformer wav2vec2-base but introduces an altered convolutional encoder. This new encoder allows the network to double its frame rate by lowering the stride of it's final layer.

Furthermore, The system is trained on the sum of two loss functions: Contrastive Loss L_m, from the original wav2vec paper [5], and Forward Sum Loss, which is introduced by Zhu et al. The formula for forward-sum loss is:

$$L_{FS} = -\sum_{X,Y \in S} LogP(Y|X) \tag{1}$$

- X: Speech Signal Encoding
- Y: Phoneme Encoding
- S: The optimal alignment of the two

L_{FS} exists to constrain all output from the transformer's attention matrix to be monotonic and diagonal. Its success can be easily proven by observing the ablation study provided by Zhu et al. (Figure 2). There is a clear monotonic, diagonal trend in the alignments using L_{FS}.

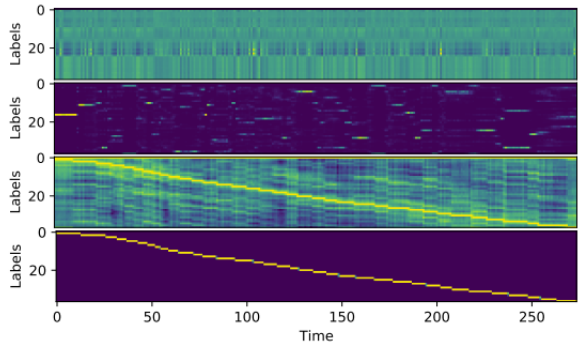

Fig. 2: Attention Matrices under different training conditions. Top to Bottom: 1) No training 2) only L_m 3) only L_{FS} 4) Both

When provided sentence transcriptions, Wav2Vec2 publishes phoneme error rates of 0.23. Charsiu slightly improves this with a phoneme error rate of 0.20 (table I).

One of the primary claims made by Charsiu Paper is the ability to effectively identify phonemes without the use of sentence transcriptions. To this end, it publishes precisions 0.55-0.60 when doing textless alignment. This is 10% worse than MFA

with text. Although novel, we found this innovation to be ill-suited to our purposes as the pipeline takes sentence transcriptions as input.

We elected not to use Charsiu for several reasons. Primarily, it is not well maintained. The last commit was made well over a year ago as of December 2023. Additionally, it cannot be installed without manually fixing multiple import bugs in their current repository. Beyond this, evaluation on our own datasets indicates that MFA outperforms Charsiu in all metrics (section IV-C).

Model	PER	Precision
Montreal Forced Aligner	No data	0.68
Wav2Vec2	0.23	No data
Charsiu	0.20	0.60

TABLE I: Published Phoneme Alignment Statistics, PER is unpublished for MFA and Precision is unpublished for Wav2Vec2

C. Montreal Forced Aligner

Shortly after Charsiu's precision and recall were evaluated and found disappointing, searching began for a more state-of-the-art forced alignment solution. Montreal Forced Aligner (MFA) was identified as the most promising choice based on data from recent research [6].

MFA is a command line utility which makes use of Kaldi ASR, a speech recognition toolkit, in order to perform many speech related processing tasks. In addition to generating forced alignments, MFA is also capable of generating pronunciations for out of vocabulary (OOV) words, generating a pronunciation dictionary from scratch, training new acoustic models, training a grapheme-to-phoneme (G2P) model, and adding word boundaries to orthographic transcriptions.

When installing MFA, the user is presented with several choices. It can be installed directly as either a Conda or a Pip package, or the source code can be cloned and run directly. Additionally, MFA simplifies the process of running in a virtualized environment via Docker, providing both complete images on the Docker Registry as well as a premade Dockerfile that can be modified and turned into a custom MFA-based image.

For the purposes of this project, running MFA via Docker presented several key advantages. Docker is a virtualization software which allows entire applications or even operating systems to run in their own isolated environments called containers. Containers contain everything necessary to run their contents independently of the host, such as application dependencies and their own filesystem. This allows containers to be deployed on any system running the Docker Daemon, Docker's background host process that manages container virtualization, regardless of what is installed on the host.

This is especially useful for applications like MFA which rely on Python, because a container can bundle together specific versions of Python and packages, eliminating common version issues. The use of Docker at its lowest level is relatively simple, with three important things to understand: dockerfiles, images, and containers. Dockerfiles are text files which contain instructions for building images. They can specify a base image, such as a version of Python or even an operating system, as well as commands to execute during the creation of the image, such as modifications to the image's file structure. After a dockerfile is complete, it can be built using Docker to form a docker image, which is like a sharable executable that contains the full container. Then, docker images can be actually run, which starts a virtualized environment in accordance with the dockerfile's specifications. Running images are known as containers, and they can be started, stopped, and deleted through the Docker Engine. Once a container is running, it can serve almost any purpose. Containers can interact with the host's file system through "volumes," can be interacted with directly using tools such as the Dev Container extension in VSCode, or simply run by themselves performing some task in the background.

After experimenting with Docker by creating several test containers from scratch and testing functionality, MFA's documented procedure was used to install and run MFA's pre-made Docker image. This required only two commands: "docker image pull mmcauliffe/montreal-forced-aligner:latest" to download the MFA image from Docker's image registry and "docker run -it -v /path/-to/data/directory:/data mmcauliffe/montreal-forced-aligner:latest" to run the image and create a usable container. One important thing to understand is the "/path/to/data/directory:/data" parameter in the run

command. This creates a volume, mapping a directory on the host system (left side of colon) to a directory on the container (right side of colon), allowing for speech and transcription data to be passed to the container. For example, the default parameter would map the directory "/path/to/data/directory" on the host system to "/data" on the container, sharing any data in these directories between the host and the container.

Once the container is running, tools like the Dev Container extension in VSCode can be used to manipulate the container's files and execute commands on it by opening a shell. Before any alignments could be conducted with MFA, the proper dictionary and acoustic model had to be installed. For this purpose, the commands "mfa model download acoustic english_us_arpa" and "mfa model download dictionary english_us_arpa" were run from a shell on the container, installing the ARPABET-based acoustic model and dictionary for US English. "mfa model inspect acoustic english_us_arpa" was run to validate this process, returning information about the acoustic model. Once the correct model and dictionary were installed, transcription and audio files were uploaded to the container through the use of the volume previously specified. This data can have any directory structure, but in order for MFA to process it properly, each transcription-audio pair must have the same base filename and be located in the same directory. For example, the data directory might contain a subdirectory for every distinct speaker, and each subdirectory might contain the files trial_1.txt through trial_n.txt as well as trial_1.wav through trial_n.wav. Each txt file should contain the transcription of its corresponding wav file as a string on one line. MFA does support other formats of corpus, but this is the simplest one and the one used in this implementation.

Once the data was visible to the container and in the correct format, the "mfa validate /data english_us_arpa english_us_arpa" command was run, "/data" being the data directory on the container. This verified the data provided in the context of MFA, ensuring that transcription-audio pairs were properly constructed and checking that the corpus did not contain too many out of vocabulary words. Ultimately, the "mfa align /data english_us_arpa english_us_arpa /mfa_data/aligned" command was run, performing forced alignment on the data in "/data." MFA will output .TextGrid files to the specified location (" /mfa_data/aligned") which contain, among other information, the phonemes present in a given .txt/.wav pair and their beginning and end times relative to the beginning of the audio file; these time aligned phonemes constitute the forced alignments.

After MFA was operational, testing began to validate it against the performance claimed in publications as well as against Charsiu. For this purpose, the TIMIT dataset was used, or more specifically, the test (as opposed to train) portion of a TIMIT set containing ARPABET phoneme alignments. This set was found publicly available in the documentation for Charsiu. Using a custom script, the dataset was refactored into a structure suitable for MFA. MFA was then run using the steps outlined above in order to generate .TextGrid files containing forced alignments for each transcription-audio pair in TIMIT. Another custom script was then used to extract phoneme alignments and evaluate MFA's accuracy relative to the included gold-standard ARPABET labels included in the dataset. For this purpose, the metrics of precision, recall, and phoneme error rate were used (averaged across all of TIMIT's test set), with precision and recall being evaluated with a 25ms absolute tolerance. 25ms is a widely used tolerance for the evaluation of forced-aligners [6].

TABLE II: MFA Test Results

Precision	**Recall**	**Phoneme Error Rate**
0.71250	0.72868	0.18319

In MFA's original publication [7], MFA achieved accuracies of 0.77 and 0.72 using 25ms absolute tolerances on Buckeye and Phonsay corpora respectively. In the paper, the exact methodology for calculating accuracy is not explained, only that the metric for a forced-alignment's accuracy is the absolute time difference between the force-aligned boundary and the reference boundary. The number of "accurate" phonemes by this metric could then be divided by the number of forced-alignments made (precision), the number of reference alignments (recall), or something else. Likely though, it represents

the number of accurate forced-alignments divided by the number of reference alignments, making it analogous to recall. It is notable that results of the local testing of MFA fall near or within the range of results obtained in MFA's publication. Additionally, the paper observed a 0.05 accuracy delta between corpora, justifying that testing on a completely different dataset (TIMIT) would be expected to produce slightly different results. From the testing done, MFA does adhere to the performance described in numerous publications, suggesting MFA was installed and is running properly. Additionally, MFA's precision and recall exceed Charsiu by 0.10-0.15, constituting a substantial improvement in forced-alignment performance. For this reason, the decision was made to move forward with constructing the alignment pipeline using MFA instead of Charsiu.

III. Pipeline Design

A. Installation and Use

We have configured our pipeline as a pip package. The package has been installed on the *Phonicon* machine of the Lab's server.

Algorithm 2 Installation and Use

Bash
pip install align-phonemes

Python
from align_phonemes import get_forced_alignment
get_forced_alignments(label, sound, output, transcription_method="Critical_Error", Critical_Error=0.30)

where:

label	= path to the block's label .txt file (see section III-E)
sound	= path to the block's label .wav file (see section III-E)
output	= A path for the output phonemes (formatting in III-E)
transcription_method	= Options are "Critical_Error", "Wav2Vec" or "Original"
Critical_Error	= default is 0.30 (See section III-B)

B. Sentence Transcription

Transcriptions play a large role in forced-alignment, and thus it is important for them to be as accurate as possible. For the purposes of this pipeline, that means a method is needed in order to obtain transcriptions using only a combination of recorded speech audio and the original sentences provided to patients (prescripts). There are two methods that were evaluated for performing this task: a Wav2Vec speech to text model and simply using the prescripts. These methods were compared in detail in Section IV - A, and no definitive conclusion was reached. In the majority of cases, where patients complied with instructions and read the prescripts properly, the prescripts more accurately matched the "true" transcriptions than Wav2Vec's transcriptions. However, when the patients do not or are unable to follow directions and read the prescript clearly, Wav2Vec has an advantage. This presented the interesting problem of how to choose which method to use when the optimal method could be different even across neighboring trials for the same speaker. In order to potentially circumvent this problem, a third optional experimental method was developed called "Critical_Error" (CE).

```
for wav_path, label_path in zip(
wav_paths, label_paths):
    original_transcript = ""
    with open(label_path, "r") as f:
        original_transcript =
        f.readline()

    samplerate, data = wavfile.read(
    wav_path)
    data = data.astype(np.float32)
    wav2vec_transcript =
        get_transcription(data,
        sr=samplerate).lower()

    if wer(wav2vec_transcript, original
        _transcript) > critical_error:
        start_time = os.path.basename(
        label_path)[:-4].replace(
        "-", ".")
        print(label_path)
        print(start_time)
        print(transcription_
```

```
        methods[start_time])
    transcription_
        methods[start_time] =
        "wav2vec"

    with open(label_path, ...
    "w") as f:
        f.write( ...
        wav2vec_transcript)
```

This function compares the word error rate of Wav2Vec's transcription and the prescript, and if this critical error exceeds a certain tolerance, rewrites the files so that forced-alignment is conducted using Wav2Vec's transcriptions instead of the prescripts. The logic behind this is that if Wav2Vec's transcription differs greatly from the prescript, then the prescript very likely differs substantially from the "true" transcription. This relies upon the fact that Wav2Vec is assumed to have a relatively constant error regardless of the "true" transcription. Take for example that the patient says something completely unrelated to the prescript. If the prescript were used, there would be a 100% word error rate, but if Wav2Vec were used, the error rate would be much lower, likely around 10%. Even if this 10% error from the "true" transcription deviated perfectly and ideally towards the prescript, there would still be a huge error between Wav2Vec's transcription and the prescript, which would then suggest that Wav2Vec is necessarily much closer to the "true" transcription. It should be noted that substantial quantitative testing on the impact of this method has not yet been conducted, but it seemingly has the potential to drastically reduce error in cases where patient speech deviates substantially from the prescript. In order to minimize the risk of inducing extra error, the critical error tolerance should be adjusted carefully through further testing.

In order to accommodate all transcription methods, the user must specify which method to use by passing "Original," "Wav2Vec," or "Critical_Error" and a numerical tolerance to the pipeline function as parameters. These signal the pipeline to use prescript, Wav2Vec, or CE transcriptions respectively, allowing the user to pick the method most suitable to them.

C. MFA

As previously discussed in Section II - C, MFA was selected as the forced alignment tool of choice over Charsiu and other alternatives due to its accuracy. Although MFA is very easy to run independently, especially in Docker, it is slightly more complicated to automate the use of a containerized process within a larger application. As the pipeline was being built in Python, it became necessary to find a way to manage a Docker container from within a Python script. The use of Python to execute shell commands was briefly explored, but then the Docker SDK for Python was discovered. This is a Python library which contains commands for directly interacting with the Docker Engine from within Python code, allowing containers to be started, interacted with, and then stopped automatically. This proved to be the perfect interface for interacting with containerized MFA, as it allowed MFA to be wrapped in a function and then called when necessary just like any other process. This circumvented many dependency issues that were encountered working with Charsiu, and allowed for more focus on adding other functionality to the pipeline.

In order to create and run an MFA container in Python, the following code was used:

```
client = docker.from_env()

image_name = "mmcauliffe/montreal-
    forced-aligner"
image_tag = "latest"

client.images.pull(image_name, tag=
    image_tag)

container_data_directory = "/data"
container_aligned_data_directory =
    "/aligned"

volume_mapping = {
    local_data_directory: {'bind':
        container_data_directory,
        'mode':'rw'},
    local_aligned_data_directory:
        {'bind':container_aligned
        _data_directory,
```

```
        'mode': 'rw'}
}

container = client.containers.run(
    f"{image_name}:{image_tag}",
    command="/bin/bash",
    volumes=volume_mapping,
    stdin_open=True,
    tty=True,
    detach=True
)
```

This gets the MFA image from the Docker registry, assigns volumes to map directories between host and the container, and then runs the MFA container using some default parameters. These few lines of code spin up a whole dedicated virtual environment for MFA where alignment can take place. Once the container is running however, it will not actually perform any alignments without running the aforementioned commands in Section II - C. Luckily, the Docker Python package contains the "container.exec_run(arguments)" command, which provides a simple way to execute a list of string arguments as if they were run directly in a shell on the container. This makes the operation of MFA from within Python extremely simple, as all commands necessary to generate forced-alignments can be easily executed in sequence from the Python script. Additionally, due to the presence of volumes, any data generated during forced-alignment will be immediately accessible within the python script on the host system, allowing for further processing outside the container. After all commands have been run on the container, "container.stop()" and "container.remove()" should be present in the python script in order to delete the container and free up host resources. Essentially, this pipeline uses containerized MFA like it would a call to any other Python function, and all the overhead from Docker is cleaned up after every execution. This method ensures that MFA always has the exact dependencies it needs while minimizing the overhead caused by reliance on containerization.

D. Demarcation

One minor problem encountered during pipeline development was that transcriptions made by Wav2Vec had no sentence demarcation. Due to the way Makin Lab collected patient speech, each audio-transcription pair should contain two copies of the same sentence back to back. However, when Wav2Vec generates alignments, the two sentences could turn out slightly different, complicating the task of separating the transcription afterwards. In order to fix this problem, a novel solution was devised making use of the Python package Jiwer's word error rate function:

```
def find_split_index(input_list):
    lowest_wer = 1
    best_index = 0

    for index in range(len(input_list)):

        first_half = ' '.join(
            input_list[:index])
        second_half = ' '.join(
            input_list[index:])

        if first_half and second_half:
            test_wer = jiwer.wer(
                first_half, second_half)

            if test_wer < lowest_wer:
                lowest_wer = test_wer
                best_index = index

    return best_index, lowest_wer
```

This function works by sweeping a pointer across any input list and comparing the word error rates of the two strings consisting of space-separated arguments on each side of the pointer. It seeks to minimize the error between the strings by saving and returning the lowest error and the corresponding split index at which it occurred. Therefore, given a single list of words which theoretically contains two distinct but identical sentences (with possible errors), this function should return the index in the list corresponding with the start of the second sentence. This is based on the idea that the minimum error should always be in the correct spot as long as the actual sentences with errors are not very different. This technique is not foolproof and could likely be

improved in several ways, such as constricting the search near the middle or rejecting obviously wrong splits, but it still offers a very simple and relatively accurate means of accomplishing this task.

E. Input and Output

As aforementioned in Section III - A, the callable function which executes the whole pipeline, get_forced_alignments(), takes five input arguments.

Firstly, it takes the file path to a label .txt file which contains a list of speech trials and their associated prescript sentences. It should take the following format, where mike-on times are listed on the line after "mike-on times," each trial is listed on subsequent lines beginning with the time corresponding to the start of the trial followed by a comma and the prescript, and mike-off times are listed on the line after "mike-off times."

```
mike-on times
1.00, 1.00
2.00, this is an example sentence
3.00, this is an example sentence
4.00, this is an example sentence
mike-off times
5.00, 5.00
```

The second argument is the path to the .wav file which contains the audio for all trials in the label file. When the pipeline runs, this .wav file is split into a number of smaller .wav files, each corresponding with one trial in the .txt label file. This split is done automatically by chopping the input .wav file in the locations provided as the trial start times in the label .txt file.

The third argument is the path to the .json file which will store the forced-alignments as the output of the pipeline. It will have the following format, where each trial will exist as a new element in "alignments" and contain its start time, transcription method, sentence one transcription, sentence one aligned phoneme list, sentence two transcription, and sentence two aligned phoneme list:

```
{
"mike-on times": [
1.0,
1.0
],
"mike-off times": [
3.0,
3.0
],
"alignments": [
{
"2.0": {
"used sentence method": "Original",
"used-sentence 1": "example sentence",
"phoneme list 1": [
[
0.0,
0.55,
"[SIL]"
],
[
...
]
],
"used-sentence 2": "example sentence",
"phoneme list 2": [
[
0.0,
0.55,
"[SIL]"
],
[
...
]
]
}
}
]
}
```

The fourth argument, as discussed in III - B, allows the user to specify which method should be used to generate the transcriptions for forced-alignment. Specifying "Original" will use the prescript transcripts for all trials, specifying "Wav2Vec" will use Wav2Vec's transcriptions for all trials, and specifying "Critical_Error" will use the experimental critical error function discussed previously. In the case that this function is used, the fifth argument is used to specify the critical error tolerance.

F. Wav2Vec2.0

The speech-to-text model researched and implemented to perform the transcriptions within the pipeline was Wav2Vec 2.0. Wav2Vec is Meta's open-source framework for self-supervised learning of representations from raw audio data using transformers. The model inputs the raw audio in a multi-layer convolution neural network to retrieve latent speech representations of 25ms. These segments are fed to a quantizer - to choose a speech unit (shorter than phoneme) from a finite set of learned units for the latent speech representation - and a transformer - to add information on the audio sequence [8]. Finally, the model is trained with a contrastive task that distinguishes between true latencies and distractions. The implemented model was pretrained and fine-tuned with 960 hours of Libri-Light and Librispeech on 16 kHz sampled audio - any audio that differs from the desired rate must be resampled before being processed by the model - and can be found in Hugging Face [5].

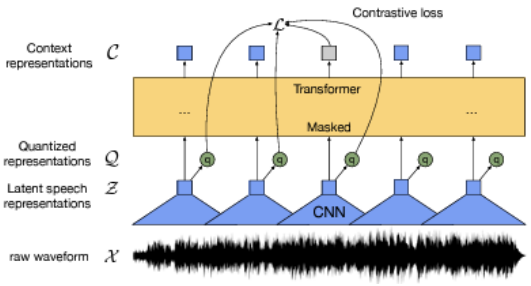

Fig. 3: Wav2Vec framework used to implement the model [5].

The implemented model was tested with clean and noisy audio from the CSTR VCTK Corpus data set prior to conducting the actual experiment to provide a baseline, which can be seen on Table III.

TABLE III: Wav2Vec Results on CSTR VCTK Corpus dataset

Metric	Dataset	Result
WER	CSTR VCTK Corpus (clean audio)	0.03521
WER	CSTR VCTK Corpus (slightly noisy audio)	0.08321

IV. Experiments

A. Experimental Metrics

We utilize 5 metrics in our methodologies.

Word-Error-Rate (WER) is a measure of the amount of Insertions I, Deletions D, and Substitutions S required to make one sentence match another. Its proper formula is:

$$\frac{S + D + I}{N} \tag{2}$$

Where N is the length of the sentence. Phoneme Error Rate (PER) is the same calculation but with Phonemes. Likewise, the Character Error Rate is a measure of how many characters need to change.

Precision is a method of measuring the accuracy of a Phoneme Alignment. The equation is:

$$\frac{correct\ phonemes}{prediction\ length}$$

Recall is very similar to precision but with one slight change

$$\frac{correct\ phonemes}{true\ sequence\ length}$$

B. Speech to Text

1) Dataset: The dataset used to conduct all tests aiming to evaluate the accuracy of the model consists of 23 blocks - a set of sentences spoken three times by the participant (two times aloud and one time silently) - with each triplet being a trial and a single sentence within the triplet a production. Each block contains roughly 40 trials, and all blocks total an amount of 919 trials distributed across two participants, a male and a female.

2) Methodology: In order to conduct the evaluation, a manual transcription of each block was necessary to record what the participant was saying exactly to accurately determine the accuracy of the model, as the participants can stray from the given directions and/or commit errors. The metric that was used and will be discussed in this examination is word error rate (WER) and character error rate (CER) in a per trial basis, which were calculated with manual transcription as the ground truth and the Wav2Vec transcription or the original trial transcript as the hypothesis. To begin with, to analyze the Wav2Vec model, the block audio needed to

be separated by trial, as the length of the audio decreases accuracy. From the original transcript ".txt" file, the actual start time of each trial are calculated by taking the offset of the file indicated time to the average of the "mike on times" (time when the microphone was turned on). Similarly, the end time of the trial is calculated using the start time of the following trial. With the trials period collected, the audio file (".wav") is sliced to obtain audio segments for all trials within the block, which are then resampled to 16 kHz and fed to the model to obtain the respective transcription for Wav2Vec. Once the trial transcription is obtained, the word error rate and character error rate are obtained with the use of the python library "jiwer".

3) Results: The purpose of this section is to convey the results of the experiments, as well as all the findings and inferred conclusions that could be supported by the descriptive statistics and plots. Due to the current support of Montreal Forced Aligner (MFA), which intakes word-based transcriptions, the word error rate results hold higher significance and value to our purposes and were the driving factor in decision making.

TABLE IV: Experiments Result Summary

Metric	Wav2Vec	Original
WER (avg)	0.10147	0.10532
WER (avg) w/o outlier ≥ 1.5	0.10147	0.09601
CER (avg)	0.03506	0.07996
CER (avg) w/o outlier ≥ 1.5	0.03506	0.06094

The first three graphs display the results for all trials per participant for word error rate for Wav2Vec (Figure 4) and the original transcription (Figure 5 and 6).

As it can be seen in the graphs (Figure 4 and 6), if outliers are not considered, the performance of the model for word error rate is slightly worse than the original transcription, which is supported by the average error in Table IV (10.147% to 9.601% respectively). However, there are few, but significant outliers within the results set, which is discussed in depth in the Limitations section, and currently presented by the spike in Participant's 2 results in Figure 5, which drive the word error rate

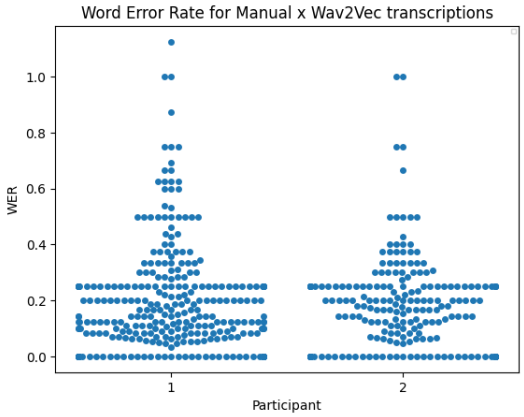

Fig. 4: Word error rate for Wav2Vec speech to text model transcription.

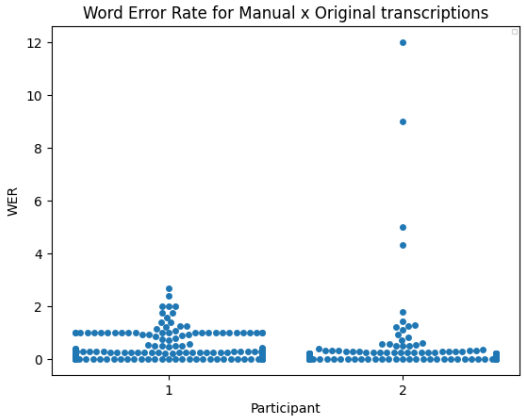

Fig. 5: Word error rate for original transcription.

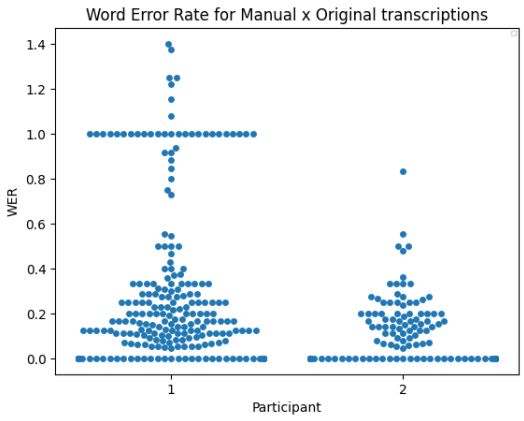

Fig. 6: Word error rate for original transcription without outliers greater than 1.5.

to be worse than its counterpart (10.532%) .
According to Figure 7, which represents a direct comparison of the Wav2Vec word error rate against

the original error rate, it can be determined that there is no clear and definite preference between the model's transcription and the original when using word error rate as the metric, even when removing all possible outliers greater than 1.5. This is supported by the apparent even distribution of results across the line x = y. In other words, there are as many points with higher error for Wav2Vec as there is for the original transcription.

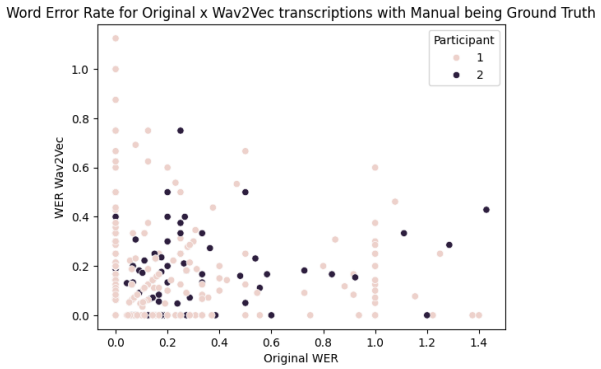

Fig. 7: Word error rate plotted for Wav2Vec against original transcription (no outlier greater than 1.5).

Moreover, the following figures (8 - 10), displays the results for all trials using the character error rate as metric.

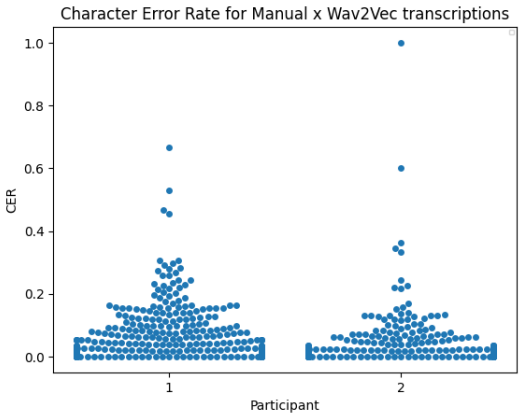

Fig. 8: Character error rate for Wav2Vec speech to text model transcription.

By comparing Figures 8 and 10, it is clear that the error is greater for the original transcription - 6.094% in contrast to 3.506% from Wav2Vec. This relation is only aggravated by the outliers seen in Figure 9 for participant 2, resulting in an average

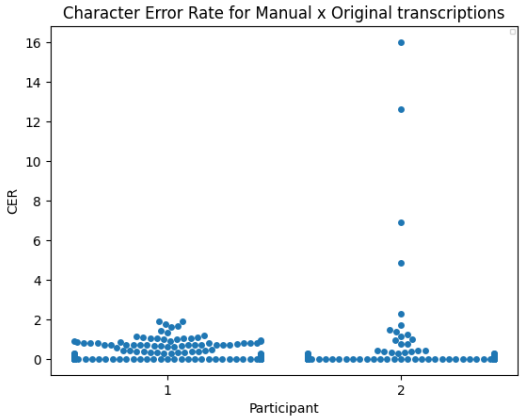

Fig. 9: Character error rate for original transcription.

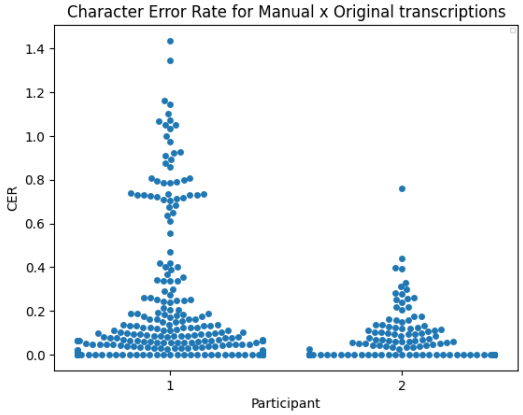

Fig. 10: Character error rate for original transcription without outliers greater than 1.5.

error of 7.996%.

Therefore, the same inconclusiveness derived from word error rate cannot be said about character error rate. According to Figure 11, which is a direct comparison between both alternatives being discussed, there is a clear preference towards the model's transcription, as there is much higher concentration of points in the low error rate for the model than there is for the original transcription.

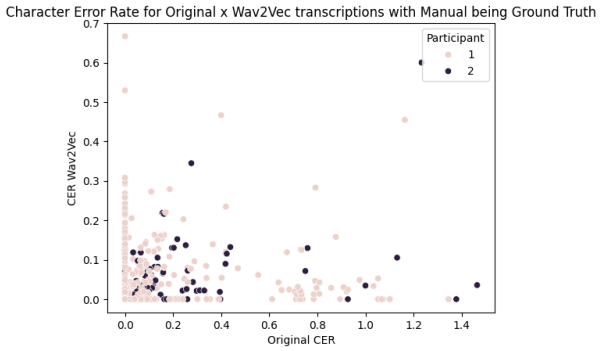

Fig. 11: Character error rate plotted for Wav2Vec against original transcription (no outlier greater than 1.5).

Both the descriptive statistic on Table IV and the swarm plots correctly indicates that word error rate (10.147% average) is significantly higher than character error rate (3.506%) - a 6.641% difference - which is the expected result considering the model's framework and training, which is based on a finite set of speech units that are shorter than phonemes, not words. Therefore, resultant words are more likely to be wrong than characters, and why there were rare occurrences of non common English vocabulary words.
However, since our most significant metric is word error rate, which was not conclusive about whereas the model has better accuracy than the original or not, a definite conclusion could not be made. Since the accuracy appeared to be conditioned to the circumstances of the trial, more specifically the ability of the participant to follow the instructions accordingly and fully read the production twice or not, a compromise was made in our pipeline to account for such instances - the critical error "path".

C. Phoneme Alignment

1) Motivaiton: We have two Alignment Models which claim similar performance. It is known that they perform well on the TIMIT dataset; however, we seek to understand how each will perform on our corpus of speech.

2) Datasets: For this experiment, two distinct datasets are utilized. The first, which shall be referred to as HIPPA, contains sentences spoken by various participants in a hospital setting. Time-aligned Phoneme labels had to be created for the HIPPA dataset (see section V-B). The second dataset, TIMIT, acted as a control. There were many questions regarding the quality of our team's phoneme transcriptions. Hence MFA on TIMIT, which contains professionally labeled phonemes [9], was used to compare.

3) Methodology: As mentioned in the prior experiment, HIPPA participants speak in trials, which contain two productions of the same sentence. Once again, the decision to evaluate by trial was made. This increases the probability that participants finish within evaluation time and decreases the probability of sounds being cut short. Within the experiment 3 comparisions take place. Charsiu's output is evaluated against the HIPPA transcriptions, MFA's output is evaluated against the HIPPA transcriptions and, finally, MFA is evaluated against the (professional) transcriptions in the TIMIT test set.

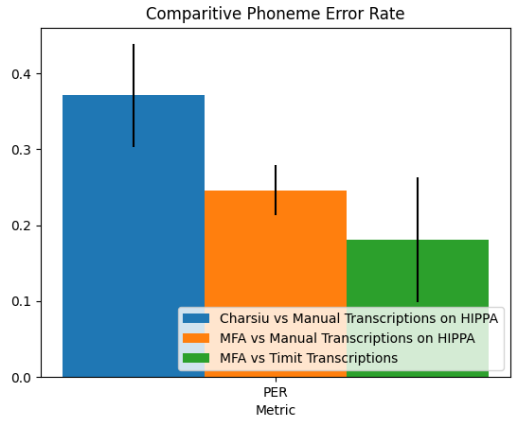

Fig. 12: Average PER performance of MFA and Charsiu. Lines represent standard error.

In both Figure 12 and 14 we see that MFA matches its published performance almost exactly when TIMIT is the baseline.

MFA achieves shockingly close, but slightly worse, results in all metrics when working on HIPPA. This result is incredibly promising especially given that HIPPA contains more noise than TIMIT. Nonetheless, it is worth being skeptical given the concerns discussed in section V-B.

Finally, it is notable that Charsiu lags in all metrics by a large margin. This is incredibly impressive for Montreal Forced Aligner, given that it is older than Charsiu by 4 years.

This data strongly indicates that MFA produces superior alignments to Charsiu. Additionally, it indi-

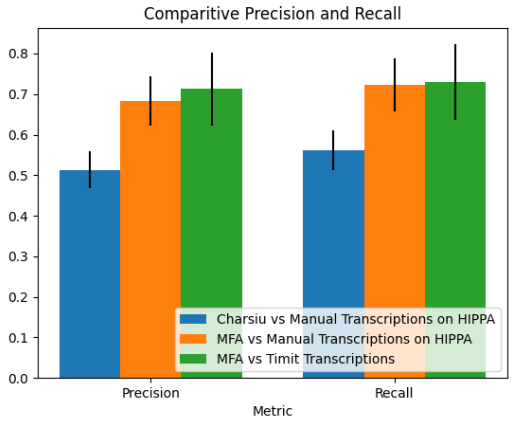

Fig. 13: Average Precision and Recall performance of MFA and Charsiu. Lines represent standard error.

cates that MFA's performance on HIPPA is promising.

D. Wav2Vec2.0 Noise Evaluation

1) Motivation: During the experiment outlined in section IV-B, it became clear that noise affected transcription quality. To this end, I empirically verify this and better understand the magnitude of the issue.

2) Dataset: TIMIT serves as the source of labeled sound files for this project. Noise was obtained from multiple corners of the internet, primarily YouTube and film sites. However, in all cases, noise resembled hospital ambiance.

3) Methodology: Wav2Vec is evaluated on three types of noises, random, background conversation and light continuous beeping. Random samples from the 5 minutes of hospital ambience are superimposed on a timit sound file. Precautions are taken to avoid aliasing when dealing with different sample rates. In order to keep all clips with superimposed noise remained intelligible to the human ear, noise was multiplied by a random factor of 0.2-0.7.

TABLE V: Wav2Vec2.0 Performance when Transcribing Noisy Audio

Noise Type	Average WER
None	0.101
Random	0.853
Background Conversation	0.923
Slow continuous beeping	0.724

4) Results: Observing table V, we confirm that noise has a significant detrimental effect on transcription. It appears that background conversation is the most detrimental form of noise and continuous beeping is less detrimental. This is unsurprising given the characteristics of conversation would presumably be less distinguishable from phonemes.

Nonetheless, WER of this magnitude is generally illegible. Hence, we conclude that noise is a significant factor to be considered and it is worse if it resembles speech.

V. Limitations

A. Dataset

To begin with, it is important to highlight that the data set was basically created by the group with the manual transcriptions, which is the basis - ground truth - in all tests. Therefore, since the manuscripts were based off human ability to listen and understand different accents, the likelihood of some level of bias caused by human errors and the fact that the original transcription was used as reference for doubt-clearing purposes is real.

As previously stated, the data set accounts for only two participants - male and female. Therefore, some inherent bias in the dataset is present, as it does not account for much variability with participants, overlooking important factors in the accuracy and functionality of the model, such as accent and pronunciation.

The data set contains slightly noisy audio files - hospital background noise - which were not filtered before running the model, which might have contributed to the current results.

B. Transcription

As previously established, the testing had outliers which impacted the results considerably, especially for the comparison between manuscript and original transcription. These outliers are directly correlated to a specific instance of errors which constitute in the participant speaking considerably less than it was supposed to. An example of such instance is for the production "the young people turn out", in which the participant only said "the" throughout the whole trial, which resulted in nine insertions - therefore a word error rate of 9 (Figure 3).

It is important to note that our phoneme transcriptions were equally imperfect. To understand the degree of imperfection in the phoneme labeling, we labeled a sample from TIMIT. Our label was then compared to the professional transcription. It was found that there was a WER of 0.33, precision of 0.66 and recall of 0.71. These metrics serve as a

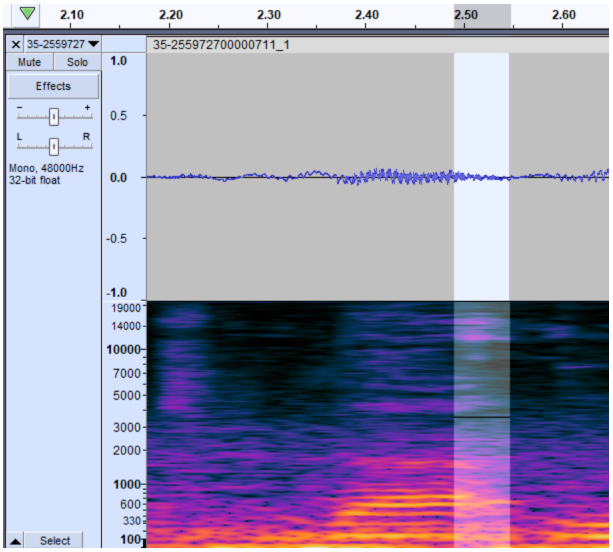

Fig. 14: The Audacity Interface that was utilized to label Phonemes. The highlighted section is a "T", which is roughly .23 milliseconds long. Barely visible, incredibly brief phonemes such as this one make it incredibly difficult to manually align phonemes

rough indicator of Error in our manual Phoneme transcriptions. Looking at precision alone, we can infer that 66% of the phonemes I labeled were of the correct type and within 20ms of the start time, or 2 out of 3 times. However, I would strongly advise this is viewed as an upper bound on error. Many of the mistakes made were trivial, for example, labeling 'TH' when the true phonemes is 'DH'.

The prevailing opinion among the lab is that speech decoders primarily classify the motor commands of participant's mouths. If this hypothesis is true, the difference between 'DH' and 'TH' (among other acoustically close phonemes) may be significant as they do require quite different motions.

VI. Contributions

A. Herbert Alexander de Bruyn

1) To Project: Prior to being allowed to access the actual data set, this author worked with plotting spectrograms of audio segments of 5 seconds and attempted to plot the phonemes on top with correct start and end time, but Audacity ended up being a better and more accurate tool for aiding in the phoneme alignment. In addition, he worked in the evaluation of Wav2Vec model without the human data relevant to this project (using CSTR VCTK Corpus data set) to provide a baseline and check the model's implementation - clean and noisy audio. Performed the manual transcription of 14 blocks - for one production, two productions, and entire trial (final version used to avoid cutting off valid audio). Calculated all the metrics (WER and CER per trial and the average across all 14 blocks for both), as well as creating all the plots describing the entire manually transcribed dataset. Developed all side functions related to this evaluation and plotting (block parser, audio slicing, Wav2Vec evaluation, original evaluation, and others).

2) To Paper: This author wrote the following sections: Abstract, Attempted Setups (within Prior Work), Wav2Vec2.0 (within Pipeline Design) Speech-to-text (within Experiments), Dataset (within Limitations), Transcription (within Limitations), and Herbert Alexander de Bruyn (within Contributions).

B. James William Stonebridge

1) To Project: This Author's primary contribution was the configuration and development of the Pipeline. As the only pipeline developer with access to the HIPPA dataset, this meant almost all pipeline integration, testing, and, eventually, pip configuration fell under his responsibility. He also designed and implemented the Critical Error transcriptions, and trial blocks and proposed the data structure/flow of the final pipeline.

His secondary contribution involved creating our verification dataset. As with Alex, 14 blocks of sentences were labeled by production and later 10 blocks were labeled by trial. Additionally, he produced phoneme transcriptions of 16 sentences.

He wrote and conducted multiple experiments that obtained the trial/production WER and CER from the manually labelled HIPPA data. He wrote and conducted the Phoneme accuracy experiments on the HIPPA data. He performed an analysis of the effects of noise on Wav2Vec2.0.

Before access to HIPPA data, this author focused primarily on Evaluating Charsiu on TIMIT. This included finding and reading the Charsiu Paper and doing comparative analysis with other literature. Additionally, the team struggled with finding TIMIT's labeled phonemes early in the semester. It was this author who ultimately found and transcribed the parquet file containing the TIMIT phonemes. Finally, as the only teammate with experience on Gilbreth, the first few weeks involved helping the other authors access and utilize the system.

2) To Paper: Within Prior Work, this author wrote the Charsiu section. Within Implementation, this Author wrote the Installation Section. Within Experiments, this author conducted and wrote the Phoneme Alignment Accuracy and Wav2Vec2.0 Noise evaluation experiments.

C. Tyler Dierckman

1) To Project: After becoming familiar with Gilbreth, this author focused on testing Charsiu's forced-alignment capabilities on several datasets including TIMIT and Librispeech. When the decision was made to transition from SLURM jobs to Jupyter notebooks, he figured out how to create Jupyter kernels from Conda environments. After a version of TIMIT was found containing ARPABET phoneme labels, he wrote functions to calculate precision, recall, and phoneme error rate, and then evaluated Charsiu on TIMIT. After lackluster results, he researched other forced-aligners, discovered MFA, and began experimenting with it. He learned Docker and figured out how to install and run MFA, and then taught the other authors how to do so. He wrote a script to refactor TIMIT to test it on MFA, as well as another script to evaluate MFA's precision, recall, and phoneme error rate on all of TIMIT. He then worked on running containerized MFA from within a Python script, and wrote a rudimentary pipeline to calculate forced-alignments that became the core of the finished pipeline. He assisted Will with manual phoneme alignments by aligning 3 trials. He also devised the error-rate-based demarcation as well as the critical error transcription method. Finally, he wrote all the functions in the final pipeline that pertain to MFA, process MFA output, preform sentence demarcation, and generate and fill the output .json file.

2) To Paper: This author wrote the following sections: Introduction, Montreal Forced Aligner (within Prior Work), Sentence Transcription (within Pipeline Design), MFA (within Pipeline Design), Demarcation (within Pipeline Design), and Input and Output (within Pipeline Design).

REFERENCES

[1] F. Gobet, "Vocabulary acquisition," in *International Encyclopedia of the Social Behavioral Sciences (Second Edition)*, second edition ed., J. D. Wright, Ed. Oxford: Elsevier, 2015, pp. 226–231. [Online]. Available: https://www.sciencedirect.com/science/article/pii/B9780080970868530281

[2] U. B. P. Lab, "Forced alignment," Available at https://linguistics.berkeley.edu/plab/guestwiki/index.php?title=Forced_alignme (12/19/2023).

[3] J. G. Makin, D. A. Moses, and E. F. Chang, "Machine translation of cortical activity to text with an encoder–decoder framework," *Nature Neuroscience*, vol. 23, no. 4, pp. 575–582, Apr 2020. [Online]. Available: https://doi.org/10.1038/s41593-020-0608-8

[4] J. Zhu, C. Zhang, and D. Jurgens, "Phone-to-Audio Alignment without Text: A Semi-Supervised Approach," *ICASSP 2022 - 2022 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 5 2022. [Online]. Available: https://doi.org/10.1109/icassp43922.2022.9746112

[5] A. Baevski, Y. Zhou, A. Mohamed, and M. Auli, "wav2vec 2.0: A Framework for Self-Supervised Learning of Speech Representations," *Neural Information Processing Systems*, vol. 33, pp. 12 449–12 460, 6 2020. [Online]. Available: https://proceedings.neurips.cc/paper/2020/file/92d1e1eb1cd6f9fba3227870bb6 Paper.pdf

[6] H. Wu, J. Yun, X. Li, H. Huang, and C. Liu, "Using a forced aligner for prosody research," *Humanities and Social Sciences Communications*, vol. 10, no. 1, p. 429, Jul 2023. [Online]. Available: https://doi.org/10.1057/s41599-023-01931-4

[7] M. McAuliffe, M. Socolof, S. Mihuc, M. Wagner, and M. Sonderegger, "Montreal Forced Aligner: Trainable Text-Speech Alignment Using Kaldi," in *Proc. Interspeech 2017*, 2017, pp. 498–502.

[8] A. Baevski, A. Conneau, and M. Auli, "Wav2vec 2.0: Learning the structure of speech from raw audio," 9 2020. [Online]. Available: https://ai.meta.com/blog/wav2vec-20-learning-the-structure-of-speech-from-raw-audio/

[9] G. E. Hinton, "Improving neural networks by preventing co-adaptation of feature detectors," 7 2012. [Online]. Available: https://arxiv.org/abs/1207.0580