Utrecht University

Master Thesis

Implementing Wav2Vec 2.0 into an Automated Reading Tutor

Author:
Nick Mostert
5902894

Supervisor:
Dr. Marijn Schraagen[1]

Second Corrector:
Dr. Heysem Kaya[1]

Daily Supervisor:
Kees van den Broek[2]

A thesis submitted in fulfillment of the requirements for the degree of Master Artificial Intelligence

in the

Graduate School of Natural Sciences
Utrecht University

June 12, 2024

[1]Artificial Intelligence, Utrecht University, Utrecht, The Netherlands.
[2]ITSLanguage BV, The Netherlands

Abstract

The number of low-literate adults in the Netherlands has been steadily increasing over the past decades. Research shows that proper reading instruction requires repeated individualized feedback. However, teachers often do not have the time or resources to provide this. Computer assisted reading tutors could provide a solution. Most current systems show good results at detecting word-level errors, but struggle to identify mispronunciations. Recent studies have shown that the use of large semi-supervised models like Wav2Vec 2.0 could improve the performance of mispronunciation detection models. The goal of this thesis is to research the effectiveness Wav2Vec 2.0 for the task of mispronunciation detection in Dutch children, and to implement it into an automated reading tutor. First, two types of Wav2Vec 2.0 models were created for classification of mispronunciation data from the speech therapy domain. Specifically, the task was target phone detection (TPD), where the pronunciation of each phone in a word is assessed individually. The first model performs end-to-end phonetic transcription, the second model uses pooling over the time dimension on the Wav2Vec 2.0 embeddings and then attempt to classify mispronunciations directly. Both of these models were then implemented into a reading error detection (RED) model to see whether the mispronunciation detection aspect of the RED model could be improved. For TPD, the models significantly improved over a baseline goodness of pronunciation (GOP) model. For RED, the use of Wav2Vec 2.0 lead to a small improvement for the classification of phone-level errors.

Contents

List of Figures

List of Tables

Abbreviations

ASR Automatic Speech Recognition. 9–12, 15–18, 20–22, 26, 28–31, 36, 37, 51, 79, 80

CALL Computer Assisted Language Learning. 15

CAPT Computer Assisted Pronunciation Training. 9, 12

CGN Corpus Gesproken Nederlands. 31, 32, 40, 50

CNN Convolutional Neural Network. 27, 29, 30

CTC Connectionist Temporal Classification. 28–30, 36

DC Direct Classification. 38, 39, 41, 42, 45–48, 50, 57–59, 61, 64–70, 77–81, 83–86

DER Diagnosis Error Rate. 55–57, 63, 71, 72, 75, 78

DNN Deep Neural Network. 18, 19, 24, 30, 37, 44, 45, 64

E2E End to End. 16, 17, 28

FA False Acceptance. 21, 55

FAR False Acceptance Rate. 22, 23, 25, 39, 44, 46–48, 50, 55, 57–62, 64–66, 68–70, 75–82, 84

FCL Fully Connected Layer. 28, 29

FR False Rejection. 21, 55

FRR False Rejection Rate. 22, 23, 39, 42, 44–48, 50, 55–62, 64, 65, 68–70, 75–85

GMM Gaussian Mixture Model. 18, 19

GOP Goodnesss Of Pronunciation. 23–25, 36–39, 44–47, 64–66, 68–70, 83, 84

HMM Hidden Markov Model. 18–20, 24, 30, 37, 40, 44, 45, 64

JASMIN-CGN Jongeren, Anderstaligen en Senioren in Mens-machine Interactie - Corpus Gesproken Nederlands. 32

L2 Second Language. 11, 30

LLR Log Likelihood Ratio. 25

LM Language Model. 38

LPP Log Phone Posterior. 30

LR Linear Regression. 25

MD Mispronunciation Detection. 39, 41, 44–50, 64–70

MDD Mispronunciation Detection and Diagnosis. 9–12, 21–24, 26, 29, 32, 37–39

MFCC Mel Frequency Cepstral Coefficients. 17, 18, 37, 40, 51

NRP National Reading Panel. 15

PC Phone Classification. 37–39, 42, 44–47, 50, 58–62, 64–70, 79, 80, 83–86

PER Phone Error Rate. 39, 42, 45, 58–62

POS Part Of Speech. 32

RED Reading Error Detection. 10–12, 16, 17, 21, 36, 40, 42, 48, 51, 54, 58, 84, 86

RNN Recurrent Neural Network. 28, 29

SVM Support Vector Machine. 25

TA True Acceptance. 21, 55

TDNN Time Delayed Neural Network. 18, 19, 37, 40

TPD Target Phone Detection. 10–12, 36, 39–46, 49, 50, 58, 64–70, 83–85

TR True Rejection. 21, 22

VTLN Vocal Tract Length Normalization. 18

WER Word Error Rate. 9, 22, 23, 54, 56, 57, 59, 63, 71–75, 78–82, 84, 85

WFST Weighted Finite State Transducer. 20, 22, 23, 25, 40, 41, 49, 51–53, 55, 59, 61, 70–74, 79–82, 84–86

Chapter 1

Introduction

1.1 Background

Computer assisted pronunciation training (CAPT) is a training tool used for speech therapy, second language (L2) learning and reading development for young children. The core task in CAPT is mispronunciation detection and diagnosis (MDD): the task of identifying erroneous pronunciation segmentation and providing phone-level diagnosis to the user [65].

The technology needed for MDD has a high degree of overlap with Automatic Speech Recognition (ASR): the task of generating speech to text transcriptions. Over the past few decades, ASR research has advanced at a rapid pace. Current state of the art ASR systems can achieve Word Error Rates (WER) of below 5% which is comparable to the average WER of human transcriptions [6]. However, MDD has proven to be more difficult. This is mostly because MDD requires distinctions between phones, whereas ASR only requires distinction between words. Furthermore, most large corpora are designed for ASR and therefore ideally contain no mispronunciations. Many of the larger opensource datasets consist of read speech from adults in their native language. MDD requires corpora that contain annotated pronunciation errors, which are a lot more rare.

These challenges could potentially be overcome, with the rise of large transformer-based models like Wav2Vec 2.0 [6]. These models are trained on large amounts of unlabeled data and then require a relatively small amount of task-specific labeled data for fine-tuning. Currently, the research on the use of Wav2Vec 2.0 for MDD is still in the early stages of development. Most MDD implementations instead depend on so-called hybrid models for ASR. These are models are called hybrid models because they combine hidden Markov models (HMM) with either a deep neural network (DNN) or a Gaussian mixture model (GMM), in order to extract phonetic information from audio samples.

The goal of this research is to test the potential of Wav2Vec 2.0 for MDD, specifically for reading assistance. To achieve this, I have been allowed to use the models and data from ITSLanguage: a company that specializes in the development of annotation and CAPT tools for a variety of language related tasks. Specifically I will be using two of their datasets: the Malmberg and the Auris dataset. The Malmberg dataset was collected in 2018 for the educational publishing company Malmberg. It consists of audio samples of elementary school children reading and has detailed annotations on any reading errors made. The Auris dataset is currently being collected for the "Koninklijke Auris Groep": an organization that provides guidance and education for various language-related problems. This dataset will be collected by speech therapists and consist of single word readings used to help children with the pronunciation of specific target phones.

Models will be trained for two different tasks, which will be referred to as target phone detection (TPD) and reading error detection (RED), on the Auris and Malmberg dataset respectively. TPD is the task of classifying the pronunciation of specific target phones as correct or incorrect. RED is the task of detecting both word-level and phone-level reading errors. First a model will be built for TPD on the Auris dataset. This model can then hopefully be reused to detect phone-level errors in the Malmberg dataset.

1.2 Societal Relevance

Automated reading tutors are systems that can detect and correct reading errors, made by children reading out loud. This task involves more than just MDD, since a good system must also be able to detect word insertions and deletions. Reading error rate has been shown to be closely related to reading comprehension and reading speed, as well as general academic performance [21, 31]. It has been shown that the best form of reading instruction, involves children reading out loud and receiving repeated personalized feedback [37]. Most teachers do not have the resources to provide enough feedback to each individual student. This is where automated reading tutors could provide a valuable alternative. Longitudinal studies have shown that the use of automated reading tutors lead to more progress over time than conventional forms of reading instruction [36, 53].

Children who lag behind their peers in reading level are also at risk of becoming low-literate adults. Literacy refers to the general ability to understand and use information [25]. In modern society reading is a big part of this. Low-literacy (Dutch: laaggeletterdheid) then refers to the lack of this ability to the extent that it limits ones ability to fully function in society or in a professional setting [16, 25]. For children under 18 the term "low-literacy" is generally not applicable. This is because a reading level that would be considered low-literate for an adult could be a normal stage in a child's reading development. Instead studies often look at children whose reading level is significantly behind their peers and who therefore are at risk of becoming low-literate adults (in Dutch literature this is often referred to as a "taalachterstand") [16]. Over the past years the number of low-literate adults has been increasing. In 2012 the Programme for International Assessment of Adult Competencies (PIAAC) estimated that in the Netherlands 1.3 million people were low-literate, which is approximately 11.9% of all working adults whereas The International Adult Literacy Survey (IALS) estimated this to be 9.4% in 1994 [16]. Low-literate people by definition have trouble functioning in society. Besides this low-literate people report that they have less trust in other people and feel like they have less influence on politics [15]. It has also been shown that they are less skilled at using computers [5].

While the main goal of this thesis is to develop a model for reading assistance, the model trained on the Auris dataset could also be used for speech therapy. Speech therapy refers to the diagnosis and treatment of speech impairments, most commonly lisping, stuttering, nasal speech and general verbal apraxia [32]. Children with speech problems also have weaker communication skills, which leads to frustration and hinders social development [45, 54]. Speech impairments in children have also been shown to hinder the acquisition of literary skills like reading and writing [55]. Just like automated reading tutors, computer-assisted speech therapy methods have proven themselves to be effective for both diagnosis and at home practice [46].

1.3 Academic Relevance

Many models have been built for RED, including ones for Dutch children [7, 13, 22, 38, 52, 67]. These models are based on the same techniques used for hybrid ASR models. Generally, these models have good performance when detecting word-level errors like word insertions and deletions, but their performance on

mispronunciation detection has been somewhat lacking. Wav2Vec 2.0 has so far mostly been used for MDD in the domain of L2 learning of English [44, 62, 66, 65]. These studies do show that Wav2Vec 2.0 can outperform techniques based on hybrid ASR models in this domain. To the best of my knowledge Wav2Vec 2.0 has rarely been used for RED in children. All that I have found is [10], which did use Wav2Vec 2.0 for RED with German children, but with a small vocabulary consisting of only 7 pseudo-words. This research can assess whether Wav2Vec 2.0 can help improve the mispronunciation detection aspect of RED in Dutch children: a domain in which it has so far been unexplored.

1.4 Research Questions

This thesis will try to assess the effectiveness of Wav2Vec 2.0 for MDD. Models will be trained for two different tasks, which will be referred to as target phone detection (TPD) and reading error detection (RED), on the Auris and Malmberg dataset respectively. For both tasks the Wav2Vec 2.0 models will be compared to baselines based on hybrid ASR models. For TPD the a baseline will be implemented based on a method known as trans-GOP, proposed in [58]. This gives the first main research question:

1 Can a Wav2Vec 2.0 based model significantly improve on the average TPD F1-score (henceforth simply referred to as F-score) of a trans-GOP based model, where significance will be measured by a paired samples t-test on the F-scores for each phone?

Besides the baseline, two types of models will be built for TPD. The first model will use phone classification (PC) to transcribe a pronunciation attempt into individual phones. From this transcription it can easily be derived whether the target phone was pronounced correctly. This approach to MDD is similar to that of [62] and [66]. The second model will directly classify whether a phone was correctly pronounced. This approach is derived from [65]. This gives the subquestion:

1.1 Is there a significant difference between the average F-scores of the phone classification and the direct classification model for TPD?

For RED the baseline will be based on the model ITSLanguage built in 2018 for classifications on the Malmberg dataset. Specifically, the proposed model will use a hybrid approach, where the ITSLanguage model is used to find word-level reading errors and a Wav2Vec 2.0 model is used to find phone-level errors. This gives the second main research question:

2 Can Wav2Vec 2.0 based model significantly improve on the phone-level reading error detection accuracy of the current ITSLanguage RED model, where significance will be measured by McNemar's test[1]?

Both the phone classification and the direct classification models developed for TPD, will be adapted for RED, specifically for the detection of phone-level errors. This gives the subquestion:

2.1 Is there significant difference between the accuracy of the phone classification and the direct classification model for the detection of phone-level reading errors?

[1]The reason for using McNemar's test here is that there were not enough folds for RED to perform a paired samples t-test. Furthermore, measuring accuracy is fine in this case, since it is most analogous to the word error rate (WER), which is the metric most commonly used for RED.

This task is different from TPD, since now the pronunciation of each phone in a given word is relevant for classification. This means that the models' prediction accuracy for individual phones can be taken into account to improve results. The models will be integrated into ITSLanguage's current RED model. This model can detect both phone and word-level errors, but according to ITSLanguage's own documentation the model's performance on phone-level errors is mediocre. By having phone-level errors be classified by a different model, the ITSLanguage model's parameters can be adapted to potentially improve the classification of word-level errors. This gives the subquestion:

2.2 Can the ITSLanguage RED model's word-level error detection accuracy be significantly improved by delegating the detection of phone-level errors to a different model?

1.5 Chapter Overview

- Chapter 2 is the preliminary literature review for this thesis.
    * Section 2.1 describes the relevant skills and type of instruction needed for proper reading development in children, as well as research done on the effectiveness of CAPT and the more broad computer assisted language learning (CALL) on reading development.
    * Section 2.2 describes the different components of hybrid ASR models.
    * Section 2.3 describes how hybrid ASR models are adapted for reading error detection.
    * Section 2.4 describes the architecture of Wav2Vec 2.0 models and results relating to its performance with MDD, non-English languages and children.
    * Section 2.5 gives an overview of various potentially useful corpora, as well as the Malmberg and Auris datasets collected by ITSLanguage.
- Chapter 3 describes the methodology, with section 3.1 describing the models built for TPD and section 3.2 the models built for RED.
- Chapter 4 outlines the results, with section 4.1 showing the results of the TPD experiments and section 4.2 showing the results of the RED experiments.
- Chapter 5 discusses the results from chapter 4 and gives recommendations for future work. Section 5.1 discusses the TPD results and section 5.2 discusses the RED results. Finally, section 5.3 answers the research questions and summarizes the most relevant future work recommendations.

Chapter 2

Related Work

This chapter outlines the theory and previous studies that are relevant for this thesis. Section 2.1 focuses on general reading instruction and the effectiveness of CAPT. Section 2.2 describes the different components of hybrid ASR models and section 2.3 describes how these types of models have thus far been used for mispronunciation detection and reading error detection. Section 2.4 goes more into detail on Wav2Vec 2.0 and its different applications, including the most current research on mispronunciation detection. Finally, section 2.5 describes the different relevant corpora available for this research.

2.1 Reading Education

This section gives an overview the relevant aspects of children's reading education. Section 2.1.1 goes over the stages of reading development and highlights the most important aspects of reading instruction. Section 2.1.2 goes over different ways in which computers have been used for reading instruction.

2.1.1 Reading Development

The American National Reading Panel (NRP) describes 3 different tasks that together form a child's reading development and compare different researches to find the best methods of instruction for each [37]. The tasks are alphabetics, fluency and comprehension. Alphabetics consists of phonics and phonemic awareness. Phonics deals with a child's understanding of the correspondence between phonemes (distinguishable units of sound that make up words) and graphemes (letters of the alphabet).[2] Phonemic awareness deals with the understanding of individual phonemes and the ways in which they interact to form words.

Phonics can be taught through simple letter naming exercises. This can either be done on a grapheme-basis or on the basis of larger units like spelling patterns. While there is no significant difference between the two methods, both methods do outperform meaning-based approaches that only teach phonics implicitly. Phonemic awareness can be practiced through several exercises, but mainly through phoneme blending where a child has to produce a word given a sequence of phonemes and inversely phoneme segmentation where a child has to split a word up into its individual phonemes. In combination with knowledge of phonics, a child can use these skills for the decoding or the spelling of words respectively. A child's phonics and

[2]Later in this thesis I will also refer to phones. These are technically different from phonemes. Phones are general and refer to all different sounds a human can produce, whereas phonemes are language-specific and refer to all sounds a speaker or that language can use to distinguish words.

phonetic awareness skills are often tested using lists of pseudowords: non-exisiting words that still follow the phonological rules of the language. This is done to ensure that the child does not learn common vocabulary words by heart, but actually uses proper decoding strategies [17].

Studies show that instruction of phonics and phonemic awareness is effective at improving the reading level of non-impaired children up until the first grade (ages 6-7) [37]. This indicates that after this age most children have mastered the most important aspects of phonics and phonetic awareness.

Reading fluency consists of reading speed, accuracy and proper expression. Well-developed word recognition skills that arise from alphabetics are necessary for fluency, however they do not guarantee it [37, 56]. One important aspect of fluency is automatic decoding. Decoding refers to the process of forming a word from individual letters and it is a combination of phonics and phoneme blending. At first this process happens in clear steps. A child will list the individual phonemes of a word, often even out loud, before figuring out the full word. During later stages of reading development this process becomes faster and more effortless, increasing the reading speed. Children with reading disabilities can still achieve good reading accuracy if they learn decoding through proper alphabetics instruction. This is especially true if the target language is a so-called "transparent language", where graphemes correspond well to phonemes [17]. Unlike English, Dutch is generally considered a fairly transparent language [57].

Good fluency development has been shown to be important for reading comprehension [37, 56]. Children who do not achieve a good level of fluency will continue to read slowly and with great effort, regardless of their general intelligence [37].

It has been found that the most effective way to improve fluency, is through "guided repeated oral reading". There are a number of methods that fall under this category, but they all have a few things in common:

1. The child practices reading the same text, until a specific level of proficiency has been reached.
2. The child reads texts out loud.
3. The child gets some sort of feedback on their mistakes. This can be from a tutor, peers or even audiotapes.

These methods have been shown to be more effective than methods that encourage children to read by themselves, or the common "round-robin reading", where children in turn read sections of a text out loud in front of the whole class [37].

Instructing methods like these have proved themselves to be effective at improving fluency. The most improvements were achieved in accuracy, but also speed and the use of proper expression improved. These methods are effective up until at least the fourth grade (ages 9-10). This indicates that children start reaching adequate reading fluency around 10 years old [37].

The final task of reading development is reading comprehension. This involves the expansion of the child's vocabulary as well as text comprehension. These skills develop throughout all the stages of a child's reading development, from the first kindergarten until well after primary school. Both vocabulary and text comprehension are often trained implicitly and content-specifically. However, there are some effective methods that teach reading comprehension during explicit reading instruction. Vocabulary can be trained by giving definitions of difficult words before reading a text. Text comprehension can be trained by giving students comprehension strategies, such as asking the who, what, where, when and why questions when reading [37].

2.1.2 Computer Assisted Language Learning

From the research of the NRP, it can be concluded that the most effective reading instruction involves reading out loud with regular individualized feedback [37]. Teachers often do not have the time and resources to provide this level of instruction. This is why computer-assisted methods are often proposed as alternatives. In the past, computers were often used for providing students with recordings of either themselves or their teacher [33, 37]. However, over the past two decades computer-assisted reading aid has been greatly improved by ASR based technology. In the literature, these methods often fall under the umbrella term computer-assisted language learning (CALL), which also includes second-language (L2) learning and computer assisted pronunciation training (CAPT).

There are many benefits to using computer-assisted methods in reading instruction. CALL allows students to practice at their own pace and can in principle provide unlimited feedback [33]. Research has also shown that there is no noticeable difference in children's response to feedback from a human or from a machine [23]. However, it is important to note that these methods are (at least currently) not perfect. Most CALL systems identify reading or pronunciation mistakes and then read out the correct version of the word using text to speech technology. These systems can often only indicate whether a word was pronounced incorrectly, but not what the exact mistake was, which can in some cases make it difficult for the student to adjust their behavior. Furthermore, due to the limitations of ASR, every CALL system will have at least some false rejections, which can cause frustration and demotivation [8]. Because the system can be sensitive to changes in speaker voice, these errors occur disproportionately often for certain speakers, which is an issue for fairness [33].

That being said, there have been a number of studies with successfully implemented CALL systems. I will now go over of a number of these studies and their results with respect to reading improvements. For a review of the technology behind these systems see section 2.3.

[36] directly tries to use CALL to implement the principles of guided repeated oral reading, as referenced by the NRP [37]. They use software called LISTEN, which can recognize deletions, hesitations and misread words of children reading texts out loud. The software was developed in 2001 and therefore does not have the best performance. While it has a true acceptance rate of 95%, it also lets a lot of mistakes slip through. To compensate for this, the system can also provide feedback when the students press a designated "help button", instead of only when a mistake is detected. The study had participants train with LISTEN for an average of 19 hours over the course of a year. The control group used these training hours to read by themselves. After a year the experimental group had improved significantly more than the control group. These improvements were most noticeable with word-level skills such as spelling and decoding, but also fluency and comprehension showed larger improvements. In a cross-over study [53] also found improvements of using LISTEN on immigrant children who learned English as an additional language.

[7] used an ASR system that functions by assigning a probability score to every word. This allows the researchers to influence the amount of feedback that each child gets, which they explore in a later study [8]. They used this to give feedback to Dutch students aged 6 and 7, who are still developing decoding skills. The students read texts or word lists out loud. If the probability score falls below a pre-determined threshold, the system recognizes it as mistake and reads the correct word out loud, after which student gets a second and if needed a third attempt. This was only an exploratory study so they did not track improvements over time, but participants did show improvements in both accuracy and speed over multiple attempts.

Not every model has the goal of giving direct feedback to the student. [3] sought out to automatically generate an interface where teachers could monitor their students' individual abilities on different tasks. To achieve this, they collected data from children between kindergarten and second grade (aged 4-8) on 5 different tasks:

1. Word recognition: reading words out loud
2. Syllable blending: reading two syllables that form a word out loud and then blending them together into a word
3. Letter naming: giving the name of a displayed letter
4. Letter sound: producing the sound related to the displayed letter
5. Reading comprehension: answering yes/no questions about a read text.

They found that all tasks, with the exception of letter sound, could be assessed to a satisfactory accuracy using ASR.

[13] used ASR to predict human-assessed reading scores based on readings of individual word lists. They asked 11 evaluators to rate 42 students on "overall reading ability" on a scale of 1 to 7. The evaluators were all from slightly different backgrounds and included native and non-native English speakers, linguistic experts and experts on reading instruction. Based on open-ended surveys, the researchers determined that pronunciation correctness, fluency and speaking rate were the most important features that determined the evaluators' score. They built an ASR model that recognized pronunciation mistakes and one that recognized the 5 most common types of dysfluencies: sound outs (decoding), hesitations, whispering, elongations and question-like intonation. They then extracted a total of 48 word-level features: 10 related to pronunciation, 12 to fluency and 26 to the speaking rate. Using 12 statistical measures (such as the mean, standard deviation, etc.), they ended up with a total of 576 potential features per child. Using various feature selection techniques for linear regression, they tested how well they could predict the evaluators' score and found a peak Pearson correlation coefficient of 0.946, indicating the final model matched the average evaluator very well.

[22] also tried to infer children's reading level using the CHOREC dataset (see section 2.5.3) . This dataset consists of three different reading tests given to Flemish Dutch children. [22] uses data from two tests where children had to read lists of 40 words and 40 pseudo-words. The words in each word list all had the same amount of syllables (either 1, 2 or 3/4-syllable words). The ASR model did not achieve great recognition scores on the word-level. This was partially because the choice was made to only count the last attempt at reading each word, meaning any decoding or false starts were not counted as errors. Despite this poor performance, the model could still pretty accurately predict how many words out of 40 were read correctly. This result was later improved upon in [23], by accounting for the models false acceptance and false rejection rate. They achieved this by adding the model's true acceptance probability to the score when a correct word was detected instead of 1, and by adding the model's false rejection probability to the score when an error was detected instead of 0. To avoid bias, these probabilities were calculated for each child based on the predictions of all other children. Using these methods, they found that the agreement between the predicted and true number of errors was equal to the average agreement between different human annotators.

2.2 Hybrid ASR Models

This section describes ASR models that are commonly used for reading error detection (RED). These models are based on hidden Markov models (HMM) in combination with either a Gaussian mixture model (GMM) or a deep neural network (DNN) and are referred to as hybrid models.

While there are also fully neural end-to-end (E2E) approaches to ASR, these are rarely used for RED. E2E methods require large amounts of training data and do not generalize well to unseen domains [59]. Furthermore, these models are designed for large-vocabulary tasks. However, reading tasks in practice often

only use a limited amount of texts and word lists. This means that they have limited vocabulary and therefore E2E methods are rarely used for RED.

It should be noted that the limitations on E2E methods can be largely be addressed through the use of semi-supervised transformer-based methods. These methods use semi-supervised learning, meaning they are able to train on large amounts of unlabeled data and then fine-tune on relatively smaller amounts of data. One of the more promising transformer-based models for ASR is Wav2Vec 2.0, which will be discussed in section 2.4.

This section discusses the different components hybrid ASR models. Many of these components are also reused in some way for RED, as will be discussed in section 2.3.

2.2.1 Feature Extraction

The first step of ASR is feature extraction. Speech is first recorded as a raw waveform. Raw waveforms of different phones consist of many repeating patterns. The repeating pattern with the lowest frequency is the pitch, produced by the vocal cords. Any other patterns are produced by specific interactions with the vocal tract and mouth. The energy of repeating patterns at different frequencies can be represented in a spectrogram. An example of a waveform and a spectrogram of the same phone said twice can be seen in Figure 2.1. There are certain dominant frequencies that have different energy levels for different phones, which can clearly be seen in the spectrogram in the image. These frequencies are called formants [2]. The most common type of feature extraction for ASR is Mel frequency cepstral coefficients (MFCC), which try to capture the energies of the different formants.

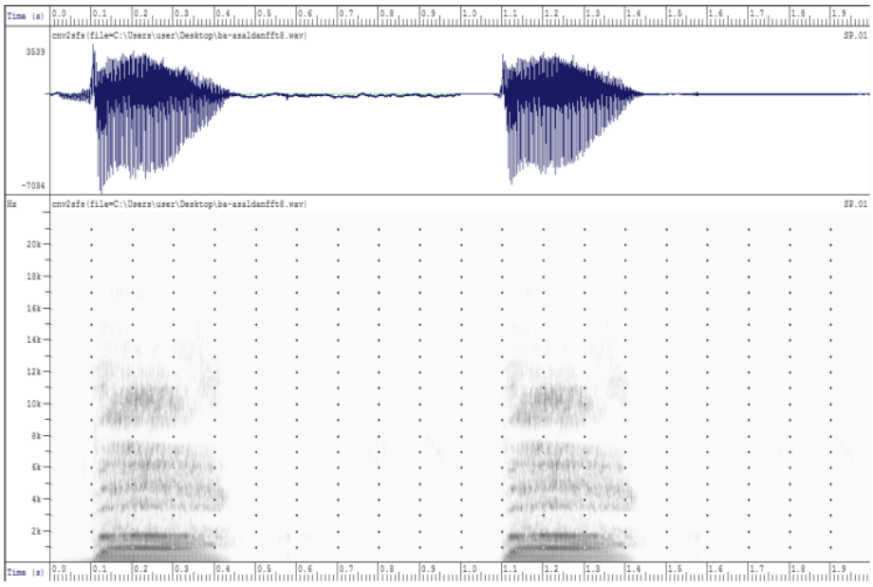


Figure 2.1: Image of the waveform (top) and the spectrogram (bottom) of the same phone said twice [2].

To extract MFCCs, first the raw waveform needs to be converted to a sequence of values called samples [27]. The number of samples per second is given by the sampling rate. For most machine learning applications a sampling rate of 16kHz is used. To extract relevant features from samples, they are combined into processing

units called frames. This is done by a sliding window which is typically 25ms (so 400 samples) wide and moves with a stride of 10ms, meaning one frame is produced for every 10ms of audio.

Over those 25ms, a discrete fourier transform is used to divide the signal into 13 different frequency bins representing the formants. The lowest frequency bin represents the pitch and is usually discarded for ASR, since pitch is not relevant for recognizing phones. For the remaining formants, a filter called the Mel filterbank amplifies the energy of certain frequencies and lowers the energy of others, to mimic the frequency sensitivity of the human ear. Finally, the formants are each represented with one value called the cepstral coefficient.

The resulting feature vector has 12 features. Usually a 13th feature is added to represent the total energy of the signal. Then 13 features are added called delta coefficients, which represent the change of the cepstral coefficients from one frame to the next and then 13 delta-delta coefficients representing the change in delta coefficients. This results in a final feature vector of 39 features [27].

There are many other types of possible operations that modify aspects of the data at the feature extraction stage, but one that I would specifically like to mention is vocal tract length normalization (VTLN). People with different vocal tract lengths, pronounce phones with slightly different frequencies. In VTLN the length of the vocal tract is derived from the MFCC values and then each formant values is moved towards an average value, proportionally to the estimated vocal tract length [68]. This technique can improve classification results, especially with child data [10].

2.2.2 Acoustic Modelling

Once the features are extracted, they need to be mapped to the most likely sequence of phones. This process is called acoustic modelling. Acoustic modelling in hybrid models is done through a hidden markov model (HMM) [12]. A HMM is a model that consists of observations and hidden states. At each time step a hidden state generates an observation with a certain probability and transitions to another (or the same) hidden state with a certain probability. The probability of generating an observation given a state is called the emission probability and the probability of transitioning between two given states is called a transition probability.

In the case of ASR, the observations are given by the extracted feature vectors and the hidden states are given by phones. Since the feature vectors are continuous, estimating the emission probabilities from training data is not a trivial task, so a second model is needed. Two types of models are commonly used for estimating the emission probabilities: Gaussian mixture models (GMM) and deep neural networks (DNN) [12].

A GMM assumes that every feature vector is generated with a multimodal probability density function, meaning a mixture of multiple Gaussian distributions (or normal distributions). The number of Gaussian distributions used per feature is given by a hyperparameter and is referred to as the number of components. Each component for each feature is given by its mean and its covariance, which can be estimated during training using the expectation maximalization (EM) algorithm. This way, the emission probabilities for each phone are modeled directly by a GMM.

DNNs for acoustic modeling typically consist of around 5 feed-forward layers, ending in a softmax activation function to get the probabilities of every class. The first layer often performs a convolution over time, meaning that it combines multiple frames into one vector by multiplying each vector with a weight matrix and then taking the sum. This is done to leverage information from neighbouring frames for the classification. A specialized version of this model is a time delayed neural network (TDNN). In a TDNN pooling is done to some extent on every layer of the network, instead of only the first layer. This type of model generally outperforms regular DNNs, with the right hyperparameters [43]. The speed of this process

can be increased by sub-sampling (see Figure 2.2). If sub-sampling is used, the stride of the convolution is increased in the first layer, so that every frame is only used once. Then in all subsequent layers, only two nodes have to be combined with every convolution operation.

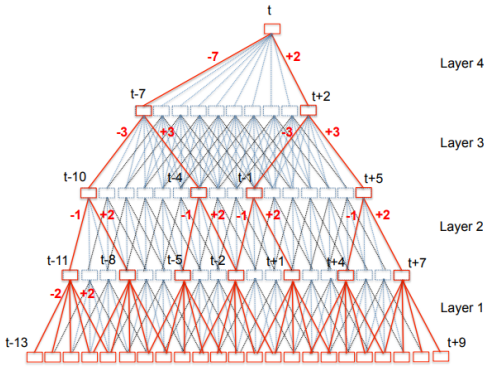


Figure 2.2: A TDNN without sub-sampling (blue) and with sub-sampling (red) [43].

Technically, a DNN with softmax activation is a discriminative model, meaning it models the probability of each phone given the observed feature vector $P(p|\mathbf{o})$. However, a HMM is a generative model that requires the emission probabilities $P(\mathbf{o}|p)$. Therefore Bayes's rule needs to be applied:

$$P(\mathbf{o}|p) = \frac{P(p|\mathbf{o})P(\mathbf{o})}{P(p)} \tag{2.1}$$

Here $P(\mathbf{o})$ is constant as 1 over the number of frames in the training set and is therefore often ignored in decoding. The prior probability $P(p)$ is given by the number of occurrences of phone p over the total phone occurrences in the training set.

The hidden states of a HMM are used to represent phones. However, especially in more recent DNN-HMM models, multiple hidden states are often used to represent each phone. There are two reasons for this:

1. The sound that phones make can change, depending on their neighbouring phones.

2. Phones are not homogeneous. The amplitudes and frequencies change from the start to the end.

To account for the first problem, hidden states are used to model triphones instead of phones. These are phones, as pronounced in the context of their direct neighbours. To account for the second problem each triphone is modeled by three hidden states, representing the start, middle and end of the triphone. The resulting hidden states are referred to as senones [12].

This means that when a language uses 50 different phones, the HMM would need $50^3 \times 3 = 375,000$ hidden states. This is not feasible in practice. Luckily, many triphones sound very similar and can be clustered together. This clustering is done with a decision tree that uses articulatory features from phonetic theory. Triphones that are clustered together, share a GMM or are grouped into one class for DNN classification.

Besides triphones, the acoustic model also is used to recognize silence and background noise. Usually this is done through 5 hidden states that are later grouped together into one class during decoding.

2.2.3 Decoding

In the context of ASR, decoding refers to the entire process of finding the most likely sequence of words, given a sequence of feature vectors. Once an acoustic model is trained, it can be used to find the most likely sequence of senones, given a sequence of feature vectors. From there, further decoding is done with weighted finite state transducers (WFST) [34]. A WFST consists of states and transitions. Each transition is associated with an input symbol, an output symbol and a weight. A WFST can serve many purposes. By manually choosing which transitions to include, it can be used to either accept or reject a string of input symbols, according to the rules of a finite state grammar. The output symbols can be used to output a string given a string of input symbols and the weights can be used to assign a probability to different output strings.

By matching the output symbols of the transitions of one WFST to the input symbols of another, two WFSTs can be combined with composition.[3] A standard ASR model for speech to text transcription is decoded by a composition of 4 WFSTs: $H \circ C \circ L \circ G$.

- H (HMM) maps input features to senones. It is a WFST representation of the acoustic model.
- C (context) maps context dependent phones (senones) to context independent phones.
- L (lexicon) maps phones to words. The lexicon can be generated with a dictionary or a grapheme to phoneme (G2P) model. Different pronunciation variations can be mapped to the same word. An example can be seen in Figure 2.3.
- G (grammar) maps words to words. It is a WFST representation of a language model, usually an n-gram. The input and output symbols of each transition are the same.

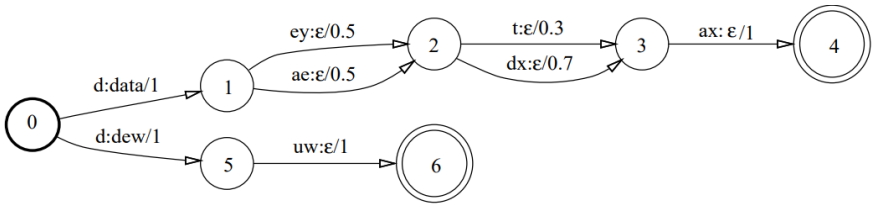


Figure 2.3: An example of a WFST lexicon that contains two words: data and dew [34]. The word "data" also has two pronunciation variants in this lexicon. Transitions are formatted as input_symbol:output_symbol/weight. 'ε' represents an empty symbol.

The weights of a WFST can be used to assign a probability score to an output sequence. This is done by multiplying all weights in a path and summing over all paths that lead to this output sequence (or with an equivalent calculation when log probabilities are used). Then a search algorithm is used to find the most likely sequence of words, given the input feature vectors. Generally, a Viterbi search is used with a fixed beam size to speed up computation (although other search algorithms like A* can also be used [40]).

[3] In reality there is a lot more to the composition of WFSTs. A more detailed explanation can be found in [34].

2.3 Reading Error Detection

This section describes the techniques used for reading error detection (RED). One approach to detecting reading errors is by comparing the prompt to an ASR transcription. For example, [63] found that a regular dictation app could be a helpful tool for native Chinese English learners in analyzing their own pronunciation and reading errors. However, these models are not optimized for detecting reading errors. For one, they make use of language models, which will give lower probabilities to mispronunciations, if these result in ungrammatical or nonsensical sentences. Language models are also of no use for analyzing readings of (pseudo)word lists. Furthermore, most of these models are designed for large vocabulary domains, however, reading tasks generally reuse the same prompts, so it is more effective to build small vocabulary prompt-specific models.

Reading errors can either be made at the word-level or at the phone-level. Word-level errors refer to the omissions of words from the prompt and the insertion of out of prompt words and sounds in between words. Phone-level errors refer to errors in the pronunciation of a word, like phone substitution, deletion or elongation. The two types of errors generally require different detection techniques and will be discussed in separate subsections below.

2.3.1 Evaluation Metrics

Before discussing the research used for RED, it is useful to give an overview of some common evaluation metrics. Given a predicted pronunciation generated by the model, the human-transcribed pronunciation as uttered by the student, and the canonical pronunciation of the target word, a prediction can result in a true acceptance (TA), true rejection (TR), false acceptance (FA) and a false rejection (FR). TA indicates that the human-transcribed pronunciation, the recognized pronunciation and the canonical pronunciation are all the same. TR indicates that neither the transcribed pronunciation nor the recognized pronunciation is the same as the canonical pronunciation. FR indicates that the transcribed pronunciation is the same as the canonical pronunciation, but the recognized pronunciation is different. FA indicates that the recognized pronunciation is the same as the canonical pronunciation, but the transcribed pronunciation is different.

These definitions are needed to calculate different metrics. Some commonly used metrics are:

$$Accuracy = \frac{TA + TR}{TA + TR + FA + FR} \tag{2.2}$$

$$Precision = \frac{TA}{TA + FA} \tag{2.3}$$

$$Recall = \frac{TA}{TA + FR} \tag{2.4}$$

$$F\text{-}score = \frac{2 \times precision \times recall}{precision + recall} \tag{2.5}$$

Accuracy and F-score are most commonly used to evaluate the performance of a model. In the case of mispronunciation detection and diagnosis (MDD), generally the F-score is preferred over accuracy in the literature. This is probably because the F-score is preferred when the ratio of acceptances to rejections is skewed [51].

Precision and recall are not used for final evaluations of a model but do provide insights into how the model treats acceptances. However, in MDD the focus is more on rejections, since these will ultimately

determine the feedback given to the student. Therefore, these papers also often report the false rejection rate (FRR) and the false acceptance rate (FAR)[4]:

$$FRR = \frac{FR}{TA + FR} \tag{2.6}$$

$$FAR = \frac{FA}{TR + FA} \tag{2.7}$$

Note that FRR is the complement of recall.

Many MDD methods use a threshold that directly influences the trade-off between precision and recall and between FRR and FAR. There seem to be two approaches to choosing the optimal threshold. The first is to set the threshold by optimizing the F-score [8, 58, 65]. The second approach is to choose a fixed value for the FRR, usually around 5%, and then report on the resulting FAR [29, 52, 67]. The former approach gives a better indication of the model's predictive power, while the latter is better suited for practical implementations, since this ensures that most of the feedback given is correct.

To evaluate the diagnosis capabilities of a MDD model extra definitions are required. Correct diagnosis (CD) refers to the situation were there is a TR and the recognized phone is equal to the human-transcribed phone. Diagnosis error (DE) refers to the situation where there is a TR and the recognized phone differs from the human-transcribed phone. These definitions are used to calculate the diagnosis error rate (DER):

$$DER = \frac{DE}{CD + DE} \tag{2.8}$$

All metrics above can be also be used to evaluate word-level errors. However, it is more common to measure word-level errors in terms of insertions (I), deletions (D) and substitutions (S). These can be used to calculate the word error rate (WER):

$$WER = \frac{I + D + S}{N} \tag{2.9}$$

Where N is the total number of words. The WER is also the most common way of evaluating speech to text transcriptions of ASR models. An equivalent calculation can be used to calculate the phone error rate (PER), which is used to evaluate phonetic transcriptions of phone classification models.

2.3.2 Word-level Error Detection

Word-level errors are best detected using custom WFSTs. These models usually follow the same WFST composition as ASR models, except the language model is replaced with a prompt-specific WFST that is used for error detection. The lexicon WFST of these models is adapted to give any unknown phone sequences a "garbage" label. This garbage label can be penalized to encourage the model to recognize words from the prompt [67].

The prompt-specific WFST follows a very specific structure in most literature [13, 23, 38, 52, 67]. Figure 2.4 shows an example from [23]. These WFSTs generally have one state for each word in the prompt plus one initial state. To recognize a correct sentence, starting from the initial state there are transitions to the next state for each word in the prompt. Each state also transitions to itself when a silence is recognized to allow for pauses between words. From here on errors can be recognized in the following ways:

[4] In some papers the FRR is referred to as the false alarm rate which can be confusing since this is also abbreviated to FAR. Instead of the false acceptance rate, these papers report on the mispronunciation detection rate which is its complement.

- Omissions can be recognized by skipping states when a different word of the prompt is recognized.
- Word insertions can be recognized by adding self-loops to each word when garbage is recognized.
- Word repetitions can be recognized by adding self-loops when the word corresponding to that state is recognized.
- Word substitutions can be recognized by transitioning to the next state when garbage is recognized.

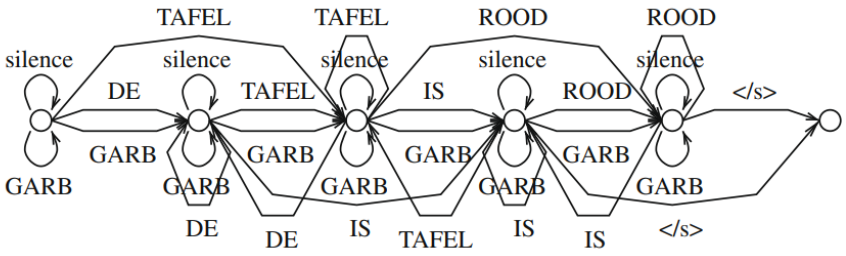


Figure 2.4: A prompt-specific WFST for recognizing errors in the sentence "De tafel is rood" (the table is red) [23].

Furthermore, any sub-words (single phones or syllables in a word) of words in the prompt are added to the lexicon. This can be used to model false starts, decoding and intra-word pauses.

The output symbols of each transition can be used to output error labels, with the type of error that was made. The weights are often not used (making these models technically FSTs), however it is possible to give each error label different weights based on their frequency in the training corpus [38].

WFSTs have proven themselves to be very effective at detecting word-level errors, with some of the best versions scoring a WER of under 3% [38, 52].

2.3.3 Phone-level Error Detection

The detection of phone-level errors is an equivalent task to MDD, which is not always just used for reading errors but also for L2 learning and speech therapy. MDD can to some extent be performed with WFSTs, since in theory a mispronounced word should be recognized as garbage. It is also possible to add common mispronunciations from the training data to the lexicon. However, WFSTs achieve mediocre results at recognizing phone-level errors, especially since garbage needs to be penalized for word-level error detection. [67] manage to improve the MDD capabilities of their WFST model, by constructing the phone confusion matrix for their acoustic model. This matrix is then pruned to find the most common insertion, deletion and substitution mistakes that their acoustic model makes. These errors are accounted for by letting the lexicon WFST recognize words from the prompt even when the wrong phones are detected, albeit with a lower probability depending on the confusion matrix score. The garbage label was penalized to fix the FRR at 3%. The use of the confusion matrix improved the FAR from 58.3% to 52.8% on the CHOREC dataset described in section 2.5.3.

However, it is more common to perform MDD using a goodness of pronunciation (GOP) score. In order to calculate the GOP of a given word, first time boundaries are assigned to each phone in the canonical pronunciation of the word, in such a way that it maximizes the probability of recognizing the word as correct. This is called forced alignment. The GOP is calculated for each phone in the word based on the

frames within its alignment range. Then the final GOP of a word can be calculated by either taking the average GOP of every phone in the word [13] or the minimum GOP of each phone in the word [8]. The latter method seems to yield better error detection results, as found by [52]. If the final GOP score falls below a predetermined threshold, the word is classified as a mispronunciation. This threshold is usually chosen to maximize the F-measure on the test set, but it can also be lowered in practice to lower the number of false rejections, since these often lead to frustrations with students [8].

The GOP metric was originally proposed in [61], where the GOP of each phone was calculated as follows:

$$GOP(p) = \frac{1}{T} \log \frac{P(\mathbf{O}|p)}{\max_{q \in Q} P(\mathbf{O}|q)} \tag{2.10}$$

where p is a given phone, T is the total number of frames in the phone's alignment range, $\mathbf{O}$ is a sequence of feature vectors associated with the frames searched and Q is the total number of phones in the target language. $\log P(\mathbf{O}|p)$ then is given by $\sum_{t=1}^{T} \log P(\mathbf{O}_t|p)$, where $\mathbf{O}_t$ is the observed vector at frame t. Here $P(\mathbf{O}_t|p)$ is directly given by the emission probabilities of the acoustic model.

Since, the use of DNN-HMMs and senones became more popular, it made more sense to express the GOP in terms of $P(s|\mathbf{O})$, since this is modeled by the DNN. Therefore, GOP for DNN-HMM based approaches is often calculated as:

$$GOP(p) = \frac{1}{T} \log \frac{P(p|\mathbf{O})}{\max_{q \in Q} P(q|\mathbf{O})} \tag{2.11}$$

$\log P(p|\mathbf{O})$ in the equation above can either be given by $\sum_{t=1}^{T} \log P(s_t|\mathbf{O}_t)$, where s_t is the senone at frame t as determined by the forced alignment, or by $\sum_{t=1}^{T} \log \sum_{s \in p} P(s|\mathbf{O}_t)$, where the probabilities of all senones needed to form phone p are summed at every frame [29]. For either calculation $P(s|\mathbf{O}_t)$ for any given senone s, is directly given by the output of the softmax layer of the DNN.

[58] improved upon the previous equation, by deriving a GOP formula (henceforth referred to as trans-GOP) where the transition probabilities between senones are taken into account:

$$\textit{trans-GOP}(p) = \frac{1}{T} \left[\sum_{t=1}^{T} \log P(s_t|\mathbf{O}_t) + \sum_{t=2}^{T} \log P(s_t|s_{t-1}) + (T-1) \log n \right] \tag{2.12}$$

where n is the total number of senones in the acoustic model. They used a database of L2 English, where pronunciations were graded on a scale of 1 to 5 by humans. They implemented three different GOP metrics from earlier work as baselines and found that their method yielded significantly higher correlation coefficients with human pronunciation ratings.

A different approach to MDD is to build a classifier model like a support vector machine (SVM) or a linear regression (LR) to classify a pronunciation as either correct or incorrect. In order to build such a model, first features need to be extracted that are some indication of the pronunciation of a word.

[29] uses metrics that are similar to the DNN-HMM adaptation of GOP, namely log phone posterior (LPP) and log posterior ratio (LPR), which they define as:

$$LPP(p) = \frac{1}{T} \sum_{t=1}^{T} \log \sum_{s \in p} P(s|\mathbf{O}_t) \tag{2.13}$$

$$LPR(p_i|p_j) = \frac{LPP(p_i)}{LPP(p_j)} \tag{2.14}$$

A feature vector for a given phone p in the canonical pronunciation is then given by:

$$[LPP(p_1),\ LPP(p_2),\ \ldots,\ LPP(p),\ \ldots,\ LPP(p_M),\ LPR(p|p_1),\ LPR(p|p_2),\ \ldots,\ LPR(p|p_M)]^T \quad (2.15)$$

They use this feature vector to train a SVM and a LR classifier, which then classify each phone of a word as correctly or incorrectly pronounced. They tested their approach on pronunciations of native Chinese English learners. They chose thresholds such that precision and recall were equal. They found that the linear regression had the best performance with a FAR of 25.5%. Both classifiers showed significant improvements over a GOP-based baseline which scored an FAR of 37.2%.

[52] use many different features and then greedily add features to a a SVM and a neural network classifier until no improvements are found. The features they use include:

- Log likelihood ratio (LLR): The log of the probability of the correct pronunciation divided by the probability of the recognized pronunciation. This is similar to GOP but on the word-level, so it uses more than just the acoustic model.
- mean-GOP and min-GOP: The mean GOP of every phone in the word and the minimum GOP of every phone in the word, respectively. It should be noted that, while they call it GOP in the paper, the calculation is actually more similar to that of LLP in [29].
- maxBadPhnProb and accBadPhnProb: The maximum probability and sum of the probabilities of the recognized mispronounced phones, respectively.
- Levenshtein distance: The number of inserted, deleted and substituted phones in the recognized pronunciation compared to the canonical pronunciation. They used three versions. In the first one they weighed insertions, deletions and substitutions equally, in the second one they weighed substitutions of phones in similar phonetic groups lower and in the third version they weighed substitutions based on the phone confusion matrix, which is effectively the same metric as the one used in [67].
- Word difficulty: An indication of how often a word was mispronounced.
- OLD20: The mean Levenshtein distance of the word to its 20 most similar neighbours in a large lexicon. This is supposed to represent a measure of familiarity that a student might have with the phones of the word.
- Word length: The number of phones, graphemes or syllables in word.

They used their models to analyze reading errors in Portuguese children. They designed their own scoring method where $cost = FRR + 0.33FAR$. LLR was the best feature for single feature classifications, with a cost of 0.163. The best performing classifier was a neural network with a cost of 0.116. Features that were included in all models in a 5-fold cross validation included: LLR, mean-GOP, maxBadPhnProb and the Levenshtein distance (either the first or third version).

Because word-level and phone-level errors require different models to achieve good accuracy, most research either focuses on detecting only phone-level errors using GOP methods [7, 8, 29, 58], or on using WFSTs to detect word-level errors and only catch some phone-level errors [23, 38, 67]. Some researches also propose hybrid models. This is done by first using a WFST to find word-level errors, as well as the alignment ranges for every word, and then using a GOP method to analyze search these ranges for every word that was classified as correct by the WFST [13, 52]. In order to get a good performance with such a model, the garbage class needs to be penalized, so that the WFST has a low false rejection rate for phone-level errors.

2.4 Wav2Vec 2.0

Wav2Vec 2.0 is a model created by Facebook AI, that turns raw waveforms into context-sensitive embeddings. It was originally developed as an end-to-end ASR model, but with the right fine-tuning the embeddings can be used for a number of classification tasks. Wav2Vec 2.0 works through self-supervised learning, by masking spans of the input and then predicting the latent features from a set of distractors in order to calculate a loss function. This allows it to use unlabeled data for training, after which only a small amount of labeled data is needed for fine-tuning [6].

The largest model from the original paper has been pre-trained on the LibriVox corpus (LV-60k), which contained over 50,000 hours of audio after pre-processing. Later versions were also trained on multilingual corpora for applications in different languages [19] (see section 2.4.4).

The model outperformed other state of the art models for automatic speech recognition (ASR) and phone classification (PC), while being simpler conceptually and relatively faster to train [6]. The model has later been used for a variety of audio related tasks such as emotion detection [47, 60], speaker identification [60] and MDD [10, 65, 66].

2.4.1 Model Architecture

This section goes over the architecture of Wav2Vec 2.0 as described in [6]. The schematic overview of the model can be seen in Figure 2.5. In pre-processing, audio is converted at 16kHz and split into overlapping segments of a fixed length, applying padding where needed. This gives the input waveforms $\mathcal{X}$. These waveforms are first fed into a convolutional neural network (CNN) which serves as the model's feature encoder to produce speech representations $\mathcal{Z}$. These speech representations are then converted discrete quantized representations $\mathcal{Q}$ which will later be used to calculate the loss. The speech representations $\mathcal{Z}$ are also fed into a transformer code-block to obtain context-sensitive embeddings $\mathcal{C}$. Finally $\mathcal{C}$ is used to distinguish the correct next segment in $\mathcal{Q}$ from a set of distractors that are randomly sampled from $\mathcal{Q}$ in order to calculate the loss $\mathcal{L}$.

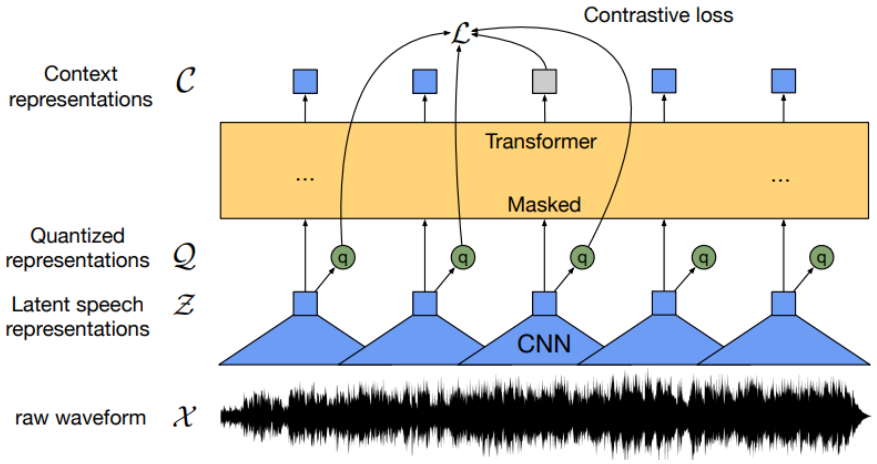


Figure 2.5: Schematic overview of Wav2Vec 2.0 [6].

2.4.1.1 Convolutional Neural Network

A CNN captures patterns in local regions of input data. They apply filters also called kernels which combine inputs in a small range into one value. This range is known as the kernel width. These kernels go over the entire input with a specified step size called the stride. A CNN consists of multiple layers which in turn consists of multiple kernels. Each kernel is used to model a specific feature of the input.

The CNN used for feature extraction in Wav2Vec 2.0 captures patterns of the raw waveform in the time dimension. It consists of 7 layers channels with strides (5,2,2,2,2,2,2) and kernel widths (10,3,3,3,3,2,2). Each layer consists of 512 kernels. This results in an output of 512-dimensional vectors with frequency of 49Hz, which gives a representation for every sample of approximately 20ms. After the convolutional layers follows a normalization layer and a GELU activation function. This gives the speech representation $\mathcal{Z}$.

2.4.1.2 Quantization

Quantization is the process of clustering continuous vectors into discrete quantized representations. In product quantization the input vector is split into G subvectors. Each subvector is then represented by its closest match out of V quantized subvectors. These quantized subvectors are concatenated into the final quantized representation. The quantized subvectors are called codewords which are sorted into G codebooks.

The codewords are chosen using a special function called the Gumbel softmax, which is fully differentiable and therefore allows the process to be trainable. The input vector is mapped to $\mathbf{l} \in \mathbb{R}^{G \times V}$. Here $l_{g,v} \in \mathbf{l}$ is the similarity between the g-th subvector of the input and the v-th codeword of the g-th codebook. This similarity calculation is based on the squared Euclidean distance. The probability of choosing codeword v in codebook g is then calculated as follows:

$$p_{g,v} = \frac{\exp(l_{g,v} + n_v)/\tau}{\sum_{k=1}^{V} \exp(l_{g,k} + n_k)/\tau} \tag{2.16}$$

where τ is a non-negative temperature, $n = -\log(-\log(u))$ and u is uniformly sampled from $\mathcal{U}(0, 1)$. The argmax of these probabilities is used to choose the codewords during the forward pass. During backpropagation, the codewords themselves are trained.

In Wav2Vec 2.0, product quantization is used to get quantized vectors $\mathcal{Q}$ to represent the feature vectors $\mathcal{Z}$. $\mathcal{Q}$ is later used to calculate the loss.

2.4.1.3 Transformer Network

The output of the feature encoder $\mathcal{Z}$ is first masked. This happens by sampling frames with a probability p to be starting frames and then for each of these frames setting each of its M subsequent frames to the same special feature vector. This feature vector is initialized randomly and adjusted in the training process.

The features are then fed into a transformer network in batches. Like with any transformer network, firstly a positional embedding needs to be applied. Instead of using absolute positions, Wav2Vec 2.0 uses a different single convolutional layer to get relative positional embeddings. These embeddings are added to the feature vectors $\mathcal{Z}$ followed by GELU and layer normalization. The resulting vectors then go through a variable amount of transformer blocks to finally produce the output $\mathcal{C}$. The original papers proposed two different model size: the base model which uses 12 transformer blocks each with 8 attention heads and the large model which uses 24 transformer blocks each with 16 attention heads [6].

2.4.1.4 Loss Function

The loss function consists of two components: the contrastive loss $\mathcal{L}_m$ and the diversity loss $\mathcal{L}_d$. The final loss is given by $\mathcal{L} = \mathcal{L}_m + \alpha\mathcal{L}_d$, where α is a tunable hyperparameter.

At any given masked time step t, the contrastive loss is calculated by having the model predict the correct quantized representation $\mathbf{q}_t$ out of a set of $K+1$ candidates $\mathbf{Q}_t$ that includes K distractors. The distractors are uniformly sampled from the other masked quantized representations in the same utterance. This prediction is made by comparing the cosine similarities of the candidates and the output of the transformer network $\mathbf{c}_t$. The loss is given by the function:

$$\mathcal{L}_m = -\log \frac{\exp(sim(\mathbf{c}_t, \mathbf{q}_t)/\kappa)}{\sum_{\tilde{\mathbf{q}} \sim \mathbf{Q}_t} \exp(sim(\mathbf{c}_t, \tilde{\mathbf{q}})/\kappa)}, \tag{2.17}$$

where sim is the cosine similarity.

The diversity loss encourages the model to use a variety of codewords. It is calculated by taking the entropy of the average softmax distribution of $\mathbf{l}$. The equation is given by:

$$\mathcal{L}_d = \frac{1}{GV} \sum_{g=1}^{G} -H(\bar{p}_g) = \frac{1}{GV} \sum_{g=1}^{G} \sum_{v=1}^{V} \bar{p}_{g,v} \log \bar{p}_{g,v}, \tag{2.18}$$

where $\bar{p}_{g,v}$ is the probability function $p_{g,v}$ described earlier, but without the temperature or random noise component.

2.4.2 Fine-Tuning

When a model has finished pre-training, it needs to be fine-tuned on labeled data for a specific task. At this point the quantization module can be removed from the model since this is only used for the masking task needed for unsupervised learning. Generally, a Wav2Vec 2.0 model is fine-tuned by adding one or more fully connected layers (FCL) on top of the transformer network. During pre-training, fixed length inputs were required for the quantization module. While this is technically no longer required during fine-tuning, the training of the transformer network is significantly more efficient if batches have the same length. Therefore it is common practice to sort the audio samples into batches of a similar length and then apply padding on the batch level before training.

For temporal tasks such as ASR and phone classification, a FCL is used to map each frame to a character (e.g. a letter or a phone) and then the Connectionist Temporal Classification (CTC) algorithm is used to find the most likely target sequence [6, 49, 19, 64, 30]. CTC was originally proposed for RNN-based E2E models for ASR [28]. It works by finding alignments: sequences of characters with the same length as the number of input features. Alignments are then turned into phone or word sequences by removing any double characters in the alignment. Different alignments can lead to the same output sequence. The probability of an output sequence is found by summing over the probability of its possible alignments. This method allows for an easy differentiable way to calculate the loss, which can also be reasonably fast due to dynamic programming. After training, a prediction can be found using a beam search with a fixed beam size.

For classification tasks that are not time dependent such as speaker verification or speech emotion recognition, some type of pooling operation is needed over the time dimension. The most common type is mean pooling, where the average is taken of the embeddings at each frame [9, 24, 60, 47, 11]. One or two FCLs can then be added to this average vector for the final classification.

During fine-tuning, it is also common to use data augmentation as a form of regularization. Usually, a modified version of SpecAugment is used [6, 66]. SpecAugment is originally used to modify spectograms by

randomly masking blocks of frequency channels, masking blocks of time steps and shrinking or stretching parts of the spectogram along the time axis [42]. Wav2Vec 2.0 does not use spectograms. Instead they apply augmentation to the output of the CNN. Time steps are randomly replaced with a trained vector, in the same way as masking occurs during pre-training. Also spans of channels are masked within time steps and replaced by zero values [6]. This process significantly delays overfitting and improves the final results.

One choice that has to be made, is which weights to train during fine-tuning. [60] compares the effects of training just the FCL weights, training the FCL and the transformer weights, and training the entire model for three different tasks (speaker emotion recognition, speaker verification and spoken language understanding). For all tasks, the models of which only the FCL weights were trained, performed significantly worse then the others. The other two types of models performed similarly to each other, with slight differences in which one was better depending on the task and the size of the model trained. The default choice in the rest of the literature seems to be freezing the CNN during fine-tuning, while adjusting the weights of the FCL and the transformer network [6, 64, 19].

[11] also experimented with adding the FCL to different parts of the pre-trained model. The goal of the paper was to classify 5 different types of stuttered speech. For each type they built 13 different models: 12 with the FCL connected to each different transformer layer of the base Wav2Vec 2.0 model and one with the FCL connected directly to the CNN. For each type of dysfluency, the best performance was found between the fifth and the tenth layer, however none of the results fluctuated much after the fourth layer. [10] used the same approach for detecting mispronunciation of pseudo-words but found the best results after the last layer.

2.4.3 Mispronunciation Detection

Some research has been conducted on using Wav2Vec 2.0 for mispronunciation detection and diagnosis (MDD). This task has proven to be challenging mostly due to the lack of quality data. Most corpora that contain phonetic transcriptions, do not include many pronunciation mistakes, let alone mistakes that are annotated as such. Corpora that do contain annotated pronunciation mistakes, are usually recordings of second language English speakers. This can be an additional challenge, since models like Wav2Vec 2.0 are often pre-trained with recordings of native English speakers.

Recently, the most common approach to MDD is end-to-end phonetic transcription [44, 62, 66]. Such models can not only indicate whether a mistake was made, but also exactly which phone was mispronounced where in the word. Additionally, this method makes good use of models that are pre-trained on unlabeled data like Wav2Vec 2.0, which partially makes up for the lack of annotated data. The general architecture of such a model is very similar to a standard ASR model. A pre-trained Wav2Vec 2.0 model is fine-tuned to assign either a phone or an blank token to every time step. Then CTC is used to turn this sequence into a phonological transcription with the highest likelihood. This transcription can be compared to the canonical transcription of the target word to determine whether an error was made.

[62] compares a Wav2Vec 2.0 model with three other models (a standard transformer model, a CNN-RNN-CTC model and a phone state transistion model) for MDD for English pronunciation mistakes by native Cantonese and Putonghua speakers. Their model outperformed the baseline models, with an F-score of 80.98%. The best performing baseline whas the CNN-RNN-CTC model, with an F-score of 74.62%.

[44] and [66] both used a similar model fine-tuned on the L2-ARCTIC corpus, which contains recordings of non-native English speakers from six different linguistic backgrounds. [44] used the larger multilingual XLSR version of Wav2Vec 2.0 (later explained in section 2.4.4). [66] used the base model combined with a pseudo-labeling algorithm to enhance their fine-tuning dataset. This algorithm uses a teacher model to predict pseudo-labels for unlabeled data. This pseudo-labeled data is then used to fine-tune a student model.

Finally, the teacher models weights are updated to be the an average of the old teacher model and the fine-tuned student model. The teacher and student model are both initialized as the same fine-tuned Wav2Vec 2.0 model. Additionally, they tested whether the phone error rate (PER) predicted by their model, was correlated with a speakers human-perceived intelligibility and human-perceived accentedness. [44] achieved the best predictions with an F-score of 59.73%. [66] had an F-score of 56.16%, which was a slight improvement over their baseline that did not use pseudolabeling. Both results are significantly worse than [62], probably because their dataset contains a larger variety of linguistic backgrounds. [66] did find that their model could be used to predict the intelligibility and accentedness of an utterance with a correlation coefficient of -0.84 and -0.75 respectively. This suggests that the accent of a student and the general intelligibility of an utterance play a large role in the effectiveness of phoneme classification for L2 learners, however this could also be partly due to the fact that their Wav2Vec 2.0 model was mainly pre-trained on native English.

[65] took a different approach to the L2-ARCTIC corpus. Instead of using an end-to-end phonological transcription, they try to predict mispronunciations directly. They first use an HMM-DNN to determine the alignment range (l, r) of every phone in the target sentence. Then pointwise convolution is applied to every time step in that range and a max adaptive pooling layer selects the highest output. The final prediction is done using a sigmoid function. Mathematically this can be represented as follows:

$$\hat{y} = \sigma(\max(\mathbf{W}\mathbf{e}_l + \mathbf{b}, \mathbf{W}\mathbf{e}_{l+1} + \mathbf{b}, \dots \mathbf{W}\mathbf{e}_r + \mathbf{b})) \tag{2.19}$$

Here $\mathbf{W}$ and $\mathbf{b}$ are the parameters of the convolution layer, n is the number of frames and $\mathbf{e}_i$ is the output of the transformer network at the i-th frame. This pointwise convolution is effectively the same as applying the same fully connected layer to every frame. One such convolution is trained for every phone. A pronunciation attempt is classified as correct if every phone is the canonical pronunciation of the target word is classified as correct. Their method improves on the results of [66] and [44] with an F-score of 61.0%. They also implemented the trans-GOP method of [58] and the LPP-based model of [29] as baselines, which scored 51.7% and 58.3%, respectively.

[10] tried to predict mispronunciations of 7 different German pseudo-words with German children. They pre-trained the Wav2Vec 2.0 model for ASR, then removed the CTC module and used the output as "grapheme embeddings" for a CNN classifier. They trained word-dependent models that were only trained on the data of on of the 7 words and a word-independent model trained on all data. Their word-dependent models on average reached an accuracy of 0.827. It was still outperformed by Kaldi-based acoustic model that used vocal tract length normalization [68], however their method could be improved upon, since they only used the base Wav2Vec 2.0 model pre-trained on English data.

2.4.4 Other Languages

The most common versions of Wav2Vec 2.0 are pre-trained on either the Librispeech corpus (LS-960) which contains 960 hours of audio or the LibriVox corpus (LV-60K) which contains approximately 53K hours of audio after pre-processing [6]. Both of these corpora contain only English-spoken audio. If you take one of these models and fine-tune them for ASR using labeled data from a different language, the model performs significantly worse than its English counterpart [30, 49]. The same occurs with phone classification [19, 49]. That is why Facebook AI published a paper in which they experimented with pre-training Wav2Vec 2.0 on different languages, using data from the CommonVoice (CV) corpus [19].

Perhaps unsurprisingly, this paper found that the amount of pre-training data used played a large role in the model's performance. For the higher resource languages French, Spanish and Italian (for which 353, 168 and 90 hours of audio was available respectively), models pre-trained on the languages themselves outperformed an English model (trained on 1350 hours from the CV corpus) on phone classification. However,

the English model was still preferred for all other lower-resource languages, including Dutch of which 27 hours of audio is available in the CV corpus.

Secondly, they pre-trained multiple models on all data from the 10 different languages researched, for a total of 793 hours of audio. Despite being pre-trained on less hours audio in total, each of these multilingual models outperformed the English model for all languages, with the exception of Swedish, where the models performed similarly depending on the version of the multi-lingual model used. However, the monolingual models outperformed the multilingual models for the high-resource languages. This result illustrates the transfer-inference trade-off: for low-resource languages, multilingual models outperform monolingual models because of positive transfer, however multilingual models perform worse on high-resource languages, due to interference [4].

Finally, a large model was pre-trained on the entire CV corpus, as well as the BABEL and the Multilingual LibriSpeech (MLS) corpora, for a total of 56K hours of audio from 53 different languages. This model is called XLSR-53, and it significantly outperformed all other models on all other languages researched on phone classification. [30] shows that it is also the best Wav2Vec 2.0 model for ASR for all of the languages researched, with the exception of English. Furthermore, [49] showed that it outperformed several non-Wav2Vec 2.0 models for both phone classification and ASR on Flemish Dutch.

2.4.5 Children

Most large publically available corpora contain little to no audio recordings of children. This is why historically it has been challenging to create well-functioning ASR models for children [26]. Children have shorter vocal chords and smaller vocal tracts and as a result their utterances tend to have a higher frequency than adults [10]. They also tend to have a slower speaking pace and more variation in their speaking rate, vocal effort and formant values [50].

To the best of my knowledge, there seems to be no literature directly comparing the performance of Wav2Vec 2.0 on children to its performance on adults. However, some papers have used Wav2Vec 2.0 to create models for children with reasonable results [1, 10]. In their study with Persian pre-school children, [1] also proposed a method called random frequency pitch (RFP) to augment the data before pre-training. This involves splitting each audio sample into 0.1s chunks, then with a probability of 0.3 randomly altering the frequency and amplitude of each chunk and finally concatenating the chunks. This method improved their results on two ASR-based tasks. It also slightly improved their results for Persian and English ASR models, showing that this method might be a viable form of regularization even for adults.

2.5 Corpora

This section provides an overview of the publically available Dutch corpora that can be used for this thesis, as well as two datasets from ITSLanguage.

2.5.1 CGN

The Corpus Gesproken Nederlands (CGN) contains close to 13000 Dutch audio samples with a total length of over 900 hours [41]. These samples were recorded in the Netherlands and Flanders. The corpus contains both read speech as well as spontaneous monologue and dialogue. All audio has been orthographically transcribed. All audio also has phonetic transcriptions 10% of which has been transcribed manually. The remaining phonetic transcriptions were automatically generated. CGN is limited to adult native Dutch speakers.

Region	Native	Age group	Hours
Netherlands	Yes	7-11	15h 10m
Netherlands	Yes	12-16	10h 59m
Netherlands	No	7-16	12h 34m
Flanders	Yes	7-11	7h 50m
Flanders	Yes	12-16	8h 1m
Flanders	No	7-16	9h 15m

Table 2.1: Data distribution of children datasets in the JASMIN-CGN.

2.5.2 JASMIN-CGN

The JASMIN-CGN (Jongeren, Anderstaligen en Senioren in Mens-machine Interactie - Corpus Gesproken Nederlands) expands on CGN with 115 hours of audio from different age groups (both from children and seniors) as well as from non-native speakers and from human-machine interactions [20]. For this project mostly the children sections of the corpus are relevant, which consist of a total of 64 hours recorded from 320 different speakers. Native children are split up into two age groups: 7-11 and 12-16. Non-native children are all grouped together from ages 7 to 16. For each of these groups about one third was from Flanders and two thirds came from the Netherlands. The mother tongue of the none-native speakers from Flanders is always French. The non-native speakers from the Netherlands have miscellaneous mother tongues. Gender was distributed evenly within each of the groups. The distribution of audio over the groups can be found in Table 2.1.

Within each group approximately 50% of audio is read speech and 50% is spontaneous speech recorded from human-machine interactions. The texts used for read speech consist of phonetically diverse sentences. The texts for native children were selected from the reading program *Veilig Leren Lezen*, which is widely used throughout the Netherlands and Flanders [35]. The method distinguishes 9 levels of reading difficulty and each child only read texts of their reading level. Non-native speakers were assigned texts from a widely used method for learning Dutch as a second language and were designed for A1 or A2 level Dutch.

The human-machine interactions were recorded using the Wizard-Of-Oz method. This method is specifically designed to elicit phenomena that are common in human-machine interaction and that are problematic for ASR systems. These phenomena are: hyperarticulation, syllable lengthening, shouting, stress shift, restarts, filled pauses, silent pauses, self talk, talking to the machine, repetitions, prompt/question repeating and paraphrasing. Every dialogue follows a similar structure, but the topics of the questions was differs between participants.

All audio is annotated with manually generated orthographic transcriptions. Phonetic transcriptions were generated automatically using lexicons, which included as many pronunciation variations as possible. In addition pauses and foreign pronunciations were annotated manually. A random selection of the phonetic transcriptions was checked manually and adjustments were made to prevent some common errors (mostly for the pronunciation of the hard or soft 'g' and the omission of the 'n' for plural nouns and infinitives). POS tags were provided and the human-machine interaction dialogues were annotated with the phenomena mentioned above. One major limitation of using the JASMIN-CGN for MDD, is the lack of (annotated) pronunciation errors.

2.5.3 CHOREC

The Children's Oral Reading Corpus (CHOREC) contains 130 hours of speech, recorded from 400 Flemish elementary school children [18]. 274 children came from regular elementary schools and 126 from schools for children with learning disabilities. For all children, parental consent was obtained, as well as relevant information on sex, age, grade, school curriculum, place of birth, place of residence, mother tongue, reading level, reading method used to learn how to read, and presence of reading disabilities.

The audio includes a real word reading test, a pseudo-word reading test and a story reading test. The real word and pseudo-word reading tests each consisted of three lists of 40 1, 2 and 3 to 4-syllable words. Story reading consisted of 9 graded text stories with a total vocabulary of 538 distinct words. Each story was between 103 and 223 words in length and was assigned one of 9 levels of reading difficulty. Each child read 1 to 3 word lists of both types and 1 to 4 stories depending on the child's reading level. Recordings were done in three sessions which lasted a maximum of 20 minutes.

The corpus was annotated using the Praat tool [14]. Praat allows for multiple tiers which can be used for different time-aligned annotations. For CHOREC 8 tiers were used:

1. The expected text for from the original prompt.
2. The expected text with adjusted boundaries.
3. An orthographic transcription of what was said. Words with pronunciation errors are marked with a '*'. If the pronunciation resembles a different Dutch word, this word is added in brackets, otherwise it's marked with '*s'.
4. A broad phonetic transcription of what was said. Wrong pronunciations are again marked with '*'.
5. Annotation of utterances made by the examiner.
6. Annotation of background noise.
7. If reading errors are made, the reading strategy is annotated. Possible annotations are 'wrong within the first trial', 'correct after some inaudible attempts', 'correct within the first trial', 'repetition of an initial part of the word' and 'omission of a word'.
8. For words where the last attempt was erroneous, the type of error is annotated from an error classification system containing 40 reading error categories.

An example of an annotated sentence can be seen in Figure 2.6. Annotations of tiers 5 and 6 are omitted, since there was no background noise or cross-talk in this example.

2.5.4 Auris Dataset

Auris is a company that specialized in helping people with speaking, language and hearing problems. ITSLanguage is currently working on a annotation tool that can be used by speech therapists to gather data. Once the tool is finished data will gradually be added to the dataset. It will most likely involve younger children who have problems with the pronunciation of specific phonemes.

Each sample in this dataset will be the utterance of a single word. These words are selected to test the child's pronunciation of specific target phones. The speech therapist will annotate whether the target phone was correctly pronounced. Unfortunately, not much information is currently available on the participants or the target phones.

Expected	Els zoekt haar schoen onder het bed. [Els looks for her shoe under the bed.]						
Observed	Als *(says 'something')* zoekt haar sch…schoen onder bed. [Als *(says 'something')* looks for her sh…shoe under bed.]						
	Els	zoekt	haar	schoen	onder	het	bed.
Orthography	*(als)	*s zoekt	haar	* schoen	onder		bed
Phonetics	Als	*s zukt	har	sx sxun	Ond@r		bEt
Strategy	f	*sg	g	agg	g	O	g
Error	e/4						

Figure 2.6: Example of an annotated sentence from CHOREC [18].

Grade	Number of samples (% of samples)
3	1141(45.6%)
4	438(17.5%)
5	569(22.8%)
6,7 and 8	352(14.1%)

Table 2.2: Distribution of grades from the Malmberg dataset.

2.5.5 Malmberg Dataset

This dataset contains 2.5k samples of children reading short stories out loud. It was originally collected by ITSLanguage, for a tool they were building for education company Malmberg. Each story contains around 100 words. Each recording lasts an average of 30 seconds for a total of a little over 20 hours of data. The children's grades range from grade 3 (6/7 year olds) to grade 8 (11/12 year olds). The distribution of the grades can be seen in Table 2.2. The children are from the from the following regions: Overijsel (3.7%), Drente (10.7%), Gelderland (33.1%), Friesland (2.7%), Zuid-Holland (26.8%), Noord-Holland (10.4%), Groningen (3.4%), Noord-Brabant (10.4%) and West-Vlaanderen (3.4%).

The data is also annotated using Praat. There are 4 tiers of annotations, each with proper time intervals:

1. Words: the words from the prompt that are correctly pronounced.

2. OutOfPrompt: annotation of utterances that are not in the prompt. The possible annotations are: Substitution (replaced word), InsertionArticle (article inserted that is not in prompt), InsertionUtterance (multiple words inserted), InsertionSounds (sounds inserted that are not words), InsertionWord (non-article word inserted) and Omission (word from prompt omitted). For all annotations except Omission, the erroneous utterances are orthographically annotated.

3. Errors: annotation of a number of error types. The possible annotations are: ProlongedVowels, Hesitation, DifficultIntelligibility, PhonemicChange, FalseStart, RepetitionWord, Decoding, SubstitutionSequence, WordStress, SlipOfTheTongue, SlipOfTheTongueCorrected, PauseTooLong, SubstitutionSemantic, CutShortOrInterruptedSlipOfTheTongue, Pauses and IncomprehensibleUtterances. Where possible, the mistakes and corrections made a orthographically annotated, as well as the attempted word from the prompt for errors like FalseStart.

4. Noise: annotation of noises that are independent of what is read. The possible annotations are: NoSignal, DistortedAudio, LowVolume, BackgroundNoise, CrossTalk and SpeechOthers.

Words	Maar					de pop is en		blijft weg
OOP		IA|de	IW|pop	IW|pop	IW|is		IS|bl	
Errors		RW|>de	|>pop	RW|>pop	RW|>is		FS|>blijft	

Table 2.3: Example of annotated sentence in Malmberg.

An example sentence is shown in Table 2.3. The prompt for this example was "Maar de pop is en blijft weg." (But the doll is and stays gone.). The students hesitates and then backtracks while reading the sentence, resulting in some repeated words. The observed utterance was "Maar de pop pop is de pop is en bl blijft weg". The following labels are abbreviated in the example below: InsertionArticle (IA), InsertionWord(IW), InsertionSound (IS), RepetitionWord (RW), FalseStart (FS). The noise tier was empty for this example and is omitted. For errors in the OutOfPromptTier, like InsertionWord in the example, the inserted word is displayed after a '|'. For certain errors in the Error tier, like FalseStart in the example, the word that the student erroneously attempted to say is displayed after '|>', as well of the timestamp of when the word is said correctly later on (not shown in the example).

Chapter 3

Methodology

This chapter describes the methodology of the current research. Section 3.1 describes the methods used to build and test models for the task of target phone detection (TPD) and section 3.2 describes the methods used to build and test models for the task of reading error detection (RED).

3.1 Target Phone Detection

This section describes how I will build and test different models for target phone detection (TPD). The models were originally designed to classify mispronunciations on the Auris dataset descibed in section 2.5.4. This dataset contains audio from the speech therapy domain, where children read single words out loud in order to test there pronunciation for a specific "target phone" in the word. Unfortunately, data collection took longer than expected so instead the CHOREC dataset was used (described in section 2.5.3). This dataset contains audio from children making reading errors rather than errors due to a speech impediment, but it should otherwise be relatively similar to the Auris dataset. Another difference is that these words were not read with specific target phones in mind, so instead every phone in the canonical pronunciation of a word will be treated as a target phone. Otherwise the main task for the models will remain the same, namely to classify whether children have correctly pronounced said target phones. For this task it does not matter whether the whole word is pronounced correctly, although this should be kept in mind as a possible functionality when building models, especially since this classification will be needed for RED.

I will build two types of Wav2Vec 2.0 models. The first one will use the CTC algorithm to do phone classification (PC) and generate a phonetic transcription of the utterance. This transcription can then be used to determine whether the target phone was correctly pronounced. This approach is similar to [44], [62] and [66] (see section 2.4.3) as well as standard models for speech-to-text ASR. The second will use the context-sensitive output of the Wav2Vec 2.0 model to do a direct classification (DC) of whether the target phone was pronounced, similar to [65]. Besides these models I will also implement a baseline model, based on the trans-GOP score from [58].

3.1.1 Data Pre-processing

As mentioned the CHOREC dataset will be used for the TPD experiments (see section 2.5.3). The dataset will only include the word list and pseudo-word list exercises and not the text reading task. This is because these tasks more closely mimic the target phone exercise from the Auris dataset. All word-list audio files

will be split into single word attempts. The dataset will be split into a train, test and validation set in two different ways. Firstly, the data will be split based on schools. This ensures that there is no overlap in speakers or recording environment between sets, while every set will still have the same word lists. Secondly, the data will be split on the vocabulary, with each set containing an equal portion of the word list and pseudo-word list words. This split can be used to test whether the models are biased towards predicting the specific vocabulary they were trained on. These splits will be referred to as the school-split and the word-split, respectively.

All audio can be cleaned up, by removing segments that are annotated with noise. Audio of word lists can be split up into samples of single words. This data can then be sorted and processed in batches of similar length to minimize padding.

3.1.2 Baseline: Trans-GOP

The baseline will be based on a goodness of pronunciation (GOP) score, specifically the trans-GOP score from [58], which also takes transition probabilities between senones into account (see section 2.3.3). This method was chosen because it outperforms other GOP methods, while being conceptually simpler and easier to implement than for instance support vector machine (SVM) or linear regression (LR) classification models, like the ones described in [29] and [52].

Trans-GOP requires a DNN-HMM to perform forced alignment on the senones of the target phone and to model both the emission and transition probabilities of the phone. This will be a TDNN-HMM with MFCC trained using the Kaldi toolkit[5]. This model can be trained on the CHOREC training sets.

For every target phone, a threshold needs to be determined, such that any pronunciation attempt with a trans-GOP under that threshold is classified as a mispronunciation. These thresholds will be determined by maximizing the F-score on the validation set.

3.1.3 Phone Classification Model

The PC model will find pronunciation errors by generating a phonetic transcription of every utterance and comparing that to the canonical pronunciation of the correct words. Its structure is identical to the standard ASR speech-to-text models, like originally described in [6]. It consists of a pre-trained Wav2Vec 2.0 model and a fully connected layer which assigns a probability to every phonetic character in the dataset plus a padding token. During training these probabilities are used to calculate the connectionist temporal classification (CTC) loss which is then used to adjust the model weights. Beam size decoding is used to derive final predictions.

As a pre-trained Wav2Vec 2.0 model I will use XLSR-53 [19], since this model has been shown to have the best performance on Dutch (see section 2.4.4). This model has 24 Transformer blocks, model dimension 1024, inner dimension 4096 and 16 attention heads. It is larger than the base Wav2Vec 2.0 model that is used by [62] and [66] for MDD. [44] did use XLSR-53 for their MDD model, but unfortunately they do not go into detail on their hyperparameters. Therefore, I will base my hyperparameter choices on [6], [19] and [64]; all three of which have built larger models for phone classification.

Table 3.1 compares different model choices for [6], [19] and [64]. Each model keeps the CNN frozen during training. Each model optimizes with Adam and for each the learning rate warms up for the first 10% of updates, keeps it steady for 40% and then linearly decays for the remainder of training. Following [6] and [65], I will only update the fully connected layer weights for the first 10k updates and then also update the transformer weights. I do not feel it is necessary to test multiple seeds for the fully connected layer.

[5] https://kaldi-asr.org/index.html

	Baevski et al. [6]	Conneau et al. [19]	Xu et al. [64]
Pre-trained model	Large-LS960	XLSR-53	XLSR-53
Number of seeds for FC	3	1	1
Learning rate	[2e-5, 3e-5]	[2e-5, 6e-5]	[5e-6, 5e-4]
Update Transformer weights	After 10k updates	Immediately	After 10k updates
Dropout in transformer network	0.1	None	[0, 0.3]
Language model	None	4-gram	6-gram
SpecAugment time-masking probability	0.065	None	None
SpecAugment channel-masking probability	0.012	None	None
Decoding beam-size	50	50	50

Table 3.1: Comparison of PC model choices for [6], [19] and [64]. If a range is indicated that means the researchers tested different values within this range before choosing the optimal value.

To be safe, I will search for the optimal learning rate within the largest range of [5e-6, 5e-4] and the optimal dropout in the transformer layer within [0, 0.3]. This search will be performed by attempting different values from these intervals using a fixed step-size.

[6] is the only paper of the three that reports using SpecAugment during fine-tuning. I do think using SpecAugment could be beneficial, because it makes the model more robust and it might prevent overfitting on some aspects that are unique to the CHOREC datase. Therefore, I will follow the SpecAugment hyperparameter settings used by [6].

[19] and [64] both use an n-gram language model (LM) for PC on at least some of their datasets. An n-gram LM takes the previous $n-1$ phones into account when predicting the next phone. It is important to note that both papers use samples of text readings with relatively few pronunciation errors. In such samples a LM might pick up on common pronunciation patterns and phone sequences of the language. I think the use of a LM makes less sense for this task, since speech therapy patients might not follow common pronunciation patterns while a LM makes the model more biased towards the correct pronunciations. This is probably also the reason that neither [66] nor [62] make use of a language model for their MDD models.

The final phone sequence is predicted using beam search decoding with beam size 50, as is the standard used by [6], [19] and [64]. This phone sequence can then be compared to the annotated phonetic transcription to evaluate the model's performance. The optimal hyperparameters will be chosen based on the phone error rate (PER) on the validation sets.

3.1.4 Direct Classification Model

The DC model will be used to directly assign a score to every phone. A threshold can then be used to determine whether phones are correctly pronounced, similarly to trans-GOP. Like the PC model, the DC model uses a pre-trained Wav2Vec 2.0 as a base model, which produces one context-sensitive embedding at every frame. From there, the embeddings need to be mapped to phone scores and pooled over the time dimension. Again, XLSR-53 will be used as the base Wav2Vec 2.0 models with the same hyperparameters as used for the PC model. Three strategies pooling strategies will be implemented and compared.

The first approach is to perform mean pooling by taking the average of every frame embedding. This is a common strategy for tasks that require full-sample classification with Wav2Vec 2.0 and it has been applied to tasks closely related to MDD like dysfluency detection [11] and English proficiency assessment [9]. After mean pooling, a fully connected layer is added to the average output vector, which assigns a score to every phone in the dataset. To train the model, every word in the dataset first needs to be annotated with a

binary metric of whether a target phone was pronounced correctly. This is achieved simply by creating a vector with a 1 for every phone pronounced by the student and a 0 for all other phones. This vector can then be compared to the prediction, by taking the sigmoid of every phone score and calculating the binary cross-entropy loss. The second approach will be identical to the mean pooling approach, but instead of taking the mean, the max pooling is used.

The third approach is adaptive max pooling, described in section 2.4.3. This approach is similar to that of [65]. The structure of this model is more like that of the PC model, in that a pointwise convolution is used to map every frame embedding to a score for every phone without pooling. For training, the alignment range for every phone in the utterance is first determined, which can be done with the same acoustic model used for the trans-GOP baseline. The accuracy of this alignment step is not very important, as shown by [65] who found that small perturbations to the alignment ranges did not significantly impact their results.

Max pooling is then done over these frames, which gives $n \times p$ vectors, where n is the number of phones in the utterance and p is the number of possible phones in the dataset. For each of these n vectors the logarithmic softmax is taken and compared to a one-hot encoded vector representing the correct phone, in order to calculate the negative log likelihood loss. After training, predictions can simply be derived by max pooling all phone scores and no alignment is needed. This gives a vector with p phone scores, just like with mean and max pooling. The correct phones can be determined using a threshold, like is done with the trans-GOP baseline.

3.1.5 Evaluation

The final models will be evaluated on the accuracy, precision, recall and F-score, as well as the false rejection rate (FRR) and the false acceptance rate (FAR) on the test set (see section 2.3.1). In order to calculate these metrics, each prediction needs to be counted as a true acceptance (TP), true rejection (TR), false acceptance (FA), or a false rejection (FR). This will be determined for each phone in the canonical pronunciation of a word. Each phone is counted as an acceptance if it is also in the predicted string for the PC model or if its predicted score is above the threshold for the DC model and the trans-GOP model. Otherwise, it is counted as a rejection. Each acceptance is said to be true if the phone is also in the annotated string and false otherwise. Each rejection is said to be true if the phone is not in the annotated string and false otherwise.

Besides TPD, the models can also be evaluated on mispronunciation detection (MD): the task of determining whether a word as a whole is correctly pronounced. For MD, a pronunciation attempt is accepted, if every phone in the canonical pronunciation is accepted following TPD rules, and rejected otherwise. An acceptance is then true if all canonical phones are in the annotated string and false otherwise. A rejection is true if not all canonical phones are in the annotated string and false otherwise. This criterium does not enforce that the phones are predicted in the correct order, or whether they are predicted the multiple times. However for the PC model, order and multiplicity can also be tested easily, since the predicted and canonical strings can directly be compared.

All TPD metrics will be reported per phone and as averages over all phones. I will also compare how much the results vary between different students, since fairness can often be an issue for MDD models [33]. While the baseline and the DC model will be optimized for the F-score, I can compare the effect of different threshold values on the F-score, FRR and FAR, and report separately on the FAR when the FRR is fixed at 5%. For the PC model, I will also the phone error rate (PER) on the CHOREC test set, so that its performance can be compared to other PC models from literature.

3.2 Reading Error Detection

This section describes the RED model which I will build to detect reading errors on the Malmberg dataset (see section 2.5.5). It will be a hybrid model that can detect word-level and phone-level errors. Both error types can be detected by a weighted finite state transistor (WFST). This model will mostly be an adaptation of a similar model, previously built by ITSLanguage for the same dataset. As a baseline, the WFST can be used to detect both word- and phone-level errors. In the hybrid model, the words that the WFST predicts to be correct or phone-level errors are re-evaluated by Wav2Vec 2.0 models. This hybrid approach is similar to that of [13] and [52]. The Wav2Vec 2.0 models for phone-level error detection will be derived from the best performing TPD models from the previous section.

3.2.1 Data Pre-Processing

The models will be trained and tested, using the Malmberg dataset. This dataset has no speaker IDs, so in order to ensure no overlap between students, splits can only be made on metadata like grades or school regions. This makes a traditional train-test-validation split less viable, since the model could easily be biased towards the age or the accent of the validation set. Instead, cross-fold validation will be used. Four folds will be created for validation: one for each grade class (grade 3, 4, 5 and 6-8). The results will be averaged over these folds and the results on the individual folds can be analyzed separately.

In order to train the Wav2Vec 2.0 models and an acoustic model for the WFST, a lexicon with phonetic transcriptions is needed. This lexicon will be created by cross-referencing the vocabulary with the lexicon of the "Corpus Gesproken Nederlands" or CGN [41] (also see section 2.5.1). The remaining words can be transcribed by the grapheme-to-phoneme model Phonetisaurus [39], which will also be trained on the CGN lexicon.

3.2.2 WFST Baseline

The WFST model will be based on a previous model built by ITSLanguage. The model will use MFCC feature extraction and a standard TDNN-HMM acoustic model with a $H \circ C \circ L$ composite WFST and a final prompt-specific WFST layer to output a sequence of error labels. The lexicon will be able to recognize words from the prompt and their individual syllables, a garbage class which is referred to as 'GARB' and a silence class refered to as 'SIL'. Final predictions are made using a Viterbi decoding with a fixed beam size of 15. Table 3.2 shows the different error classes recognized, the way in which the WFST recognizes these errors and the Malmberg error labels that fall under those classes.

Not all error classes have to be included in the model. According to ITSLanguage documentation certain error classes like "False Start" , "Decoding Error" and "Pronunciation Error"[6] were not in their model by default, due to poor performance and low frequencies of occurrence. Many error types can also be caught as other error types, if they are left out of the WFST. For example, a false start will likely be predicted to be an insertion, if the WFST cannot recognize individual syllables. I will experiment with different WFST versions. In the simplest case the model will only be able to detect the classes "correct", "omission" and "insertion". The other error types can then be added in different combinations.

Besides the error classes recognized, the models will also be varied by setting different weights for each error class. The weights for each error class can be:

- Kept equal.

[6]This error label was used for common pronunciation errors which were explicitly modeled in the lexicon. It is not a part of the WFST models from this research.

Error class	Detection strategy	Malmberg error labels
Correct	Correct word detected	*Any words in the Words tier as well as any errors that cannot be detected by the model*
Repetition	Correct word from earlier on in the sentence detected	Repetition
Omission	*SIL* detected instead of expected word	OmissionWord
False Start	First syllable(s) of word detected before full word	FalseStart, SlipOfTheTongueCorrected
Decoding	*SIL* detected in between syllables of word	Decoding, Pauses
Insertion	*GARB* detected	InsertionWord, InsertionUtterance, InsertionArticle
Substitution	Combination of insertion and omission	PhonemicChange, Substitution, SubstitutionSequence, SubstitutionSemantic, SlipOfTheTongue, CutShortOrInterruptedSlipOfTheTongue

Table 3.2: The error classes recognized by the WFST, the strategy used for recognizing them and the error labels from the Malmberg dataset that get recognized by these strategies.

- Derived from the number of occurrences of the errors in the Malmberg dataset.
- Adjusted individually, to optimize the models' performance.

The WFST for the hybrid model, will be based on the best performing baseline model. For each word that the WFST identifies as correct or as a phone-level error (e.g. substitution or decoding), it also outputs the alignment ranges, which can be used by the Wav2Vec 2.0 model to re-evaluate the prediction. To evaluate the accuracy of this alignment step, the Wav2Vec 2.0 models will first be tested separately on a clean dataset, which uses the annotated alignment boundaries. This clean dataset will also be used to train the Wav2Vec 2.0 models in the first place.

3.2.3 Wav2Vec 2.0 Models

The Wav2Vec 2.0 models will be derived from the best performing models from the TPD experiments, meaning the same hyperparameters and DC pooling strategy will be used. It can be tested how well the TPD models perform in a zero-shot situation, but they can also be trained on the Malmberg dataset. For this a clean dataset will be created which uses the annotated alignment ranges to split the dataset into separate words. Utterances annotated with noise or without a clear lexicon entry can be left out. The models can first be tested on the phone-level errors of the clean dataset and then on the set of phone-level errors output by the WFST. Comparing these results will give an indication of the accuracy of the WFST's alignment. After analyzing the results on phone-level errors, the predictions of the best performing model can be re-inserted into the full sentence predictions to get the full hybrid model results.

Of course phone-level error detection is different from TPD, in that it is about accepting or rejecting whole words rather than single phones. In the section 3.1, it is mentioned that the MD performance of the models could already be tested by simply defining an acceptance as the conjunction of acceptances of every

phone in the canonical pronunciation. While this is a simple solution, there are a few ways in which this approach could cause problems in RED:

1. Firstly, this will increase the FRR of predictions. For example, if phones have an average FRR of 0.05, then a word of four phones will have an average FRR of $1 - 0.95^4 \approx 0.19$. For the DC models, this can be accounted for by lowering the threshold if necessary, but for the PC model this is not possible. One potential solution that can be tested is to set a threshold on the allowable distance between the predicted and canonical string. For example, the Levenshtein distance can be calculated by counting the number of phones inserted, deleted and substituted. This Levenshtein distance can also be normalized by the total number of phones in the canonical pronunciation, which gives the PER. It can be tested whether setting a threshold on either of these metrics leads to improvements in the F-score.

2. Secondly, this approach does not take the order or multiplicity of phones. This is fine for TPD, where only the pronunciation of individual phones is relevant, however in RED, errors related to the order or multiplicity of phones are just as relevant as errors related to the phones themselves. For the PC model it is easy to enforce that the order and multiplicity of phones is correct, since the predicted string can directly be compared to the canonical string. but for the DC models this is not possible. Therefore, errors related to order or multiplicity will always lead to a false acceptance. Depending on the frequency of such errors, this might not have a large effect. Alternatively, order and multiplicity could potentially be enforced by testing whether the phones of the canonical pronunciation score above the threshold in the correct order of frames. The success of such a strategy will likely depend on the pooling strategy implemented, since with mean pooling, there is no incentive for the model to attribute different phone scores to specific frames.

Chapter 4

Results

This chapter shows the results of the experiments. Section 4.1 shows the results of the target phone detection on the CHOREC dataset. Section 4.2 shows the results of reading error detection on the Malmberg dataset.

4.1 Target Phone Detection (TPD)

This section describes the results for TPD: the task of determining whether a specific target phone was correctly pronounced. By evaluating every phone in a canonical pronunciation, TPD models can also be used to determine whether a whole word was pronounced correctly. This task is simply referred to as mispronunciation detection (MD) and the results for this task are also reported.

4.1.1 Data Preparation

The dataset used for TPD consisted of the word list and pseudo-word list exercises from the CHOREC dataset. All audio files were split into single word attempts. From these attempts, any that did not have a literal phonetic transcription were removed. This included attempts of which a part of the transcription contained the symbol for unintelligible utterances ('*s') and correctly pronounced words with multiple pronunciation variants, since for these words the transcription was ambiguous on which variant was used (e.g. 'Omdraj[@/@n/En]'). The resulting dataset consisted of 51820 word attempts, which is approximately 62% of the original dataset. The final dataset consisted of 41 total hours of audio.

Two different train-test splits were created. The first is the school-split, for which one out of eight schools in the dataset was used for the test set, one for the validation set and the remaining six for the train set. This ensured that there was no overlap in speakers or recording environment between sets. In the school-split, the train set consisted of 39462 word attempts (32.4 hours), the validation set of 6277 word attempts (4.4 hours) and the test set of 6081 word attempts (4.4 hours). The second is the word-split, which was made to test whether the model was biased towards predicting the specific vocabulary it was trained on. Out of the 240 target words, 192 were used for the train set and 24 for the test and validation set each. These words were randomly divided with the same distribution from each of the word and pseudo-word reading tasks, meaning that each set had the same distribution of 1-, 2-, 3- and 4-syllable words. For this split, the train set consisted of 43113 word attempts (34.0 hours), the validation set of 4274 word attempts (3.5 hours) and the test set of 4433 word attempts (3.7 hours).

During TPD analysis, it was found that these test sets had rather skewed distributions of correctly and incorrectly pronounced phones. While around half of the samples contained some kind of error, each phone was on average mispronounced only around 6% of the time. This proved challenging for analysis, especially for the models with adjustable thresholds, since even arbitrarily low thresholds that accepted every pronunciation would still score an accuracy of 94%. To remedy this, balanced versions of the precision, recall and F-score were also calculated for each phone, where the number of true acceptances and false rejections are multiplied with the percentage of incorrect examples in the dataset. So for example, the balanced accuracy is calculated as follows:

$$balanced\ accuracy = \frac{wTA + TR}{wTA + TR + FA + wFR} \tag{4.1}$$

$$w = \frac{incorrect}{total} = \frac{TR + FA}{TA + TR + FA + FR} \tag{4.2}$$

These metrics mimic what the results would look like if the dataset was not skewed and instead had 50% correct and 50% incorrect utterances. During this research, thresholds were set by optimizing the balanced F-score rather than the F-score as was originally planned.

4.1.2 Trans-GOP Model

The trans-GOP model is used as a baseline in this research. It uses a DNN-HMM acoustic model to force align the canonical pronunciation on the audio sample and then assigns a goodness-of-pronunciation (GOP) score to every force aligned phone. To evaluate TPD for a phone, we look at all words of which that phone is in all canonical pronunciation variants. Then for each of these words we see if the phone was in the annotated label and whether the phone has a trans-GOP score above a threshold for any of the force aligned variants to determine whether it is a true/false acceptance/rejection. To evaluate word mispronunciations, we simply check whether there is at least one aligned variant of which all phones get a trans-GOP score above the threshold, in which case a word is considered to be correctly pronounced.

The thresholds were optimized to maximize the balanced F-score and then again to fix the false rejection rate (FRR) at 0.05[7]. The results and thresholds are shown in Table 4.1. The full results of the best-performing models can be seen in section 4.1.5.

Model	Split	TPD F-score	TPD FAR	MD F-score	MD FAR
trans-GOP	School-split	0.757 (2.0)	0.771 (-6.1)	0.800 (-0.5)	0.733 (-8.1)
trans-GOP	Word-split	0.735 (-0.7)	0.901 (-9.1)	0.796 (-1.5)	0.546 (-4.62)

Table 4.1: This table shows the maximal TPD and MD balanced F-score and the TPD and MD false acceptance rate (FAR) when the FRR is fixed at 0.05. Better performing models will yield higher F-scores and lower FARs. TPD scores are averaged over all phones. The thresholds used are shown in parentheses.

4.1.3 Phone Classification (PC) Model

The PC model is the first Wav2Vec 2.0 based model that is evaluated in this research. It uses the connectionist temporal classification (CTC) algorithm to output the most likely string of phones. To evaluate TPD for

[7]In the case of TPD these statistics are averaged over all phones.

a phone, we look at words of which all canonical pronunciation variants contain the phone, and then check whether the phone is in the annotated and/or the predicted label. To evaluate MD, we check whether the predicted label and the annotated label are equal to one of the canonical pronunciation variants.

The optimal hyper-parameters for the PC model were determined using school-split. The optimal learning rate was determined at 3×10^{-5} and the transformer dropout at 0.1. Each of the models was trained for $60,000$ updates, after which the best iteration was chosen based on the phone error rate (PER) on the validation set. The best performing school-split model, scored a PER on the test set of 0.095 and the best performing word-split model scored a PER on the test set of 0.221.

The full TPD and MD results of the PC models can be seen in section 4.1.5.

4.1.4 Direct Classification (DC) Model

The DC model is the second Wav2Vec 2.0 based model that is evaluated in this research. Three variations of the DC model were implemented: DC_mean, DC_max, and DC_amax. Each model used a pointwise convolution layer that mapped the output of Wav2Vec 2.0 to a vector of size $(\mathcal{L}, \mathcal{C})$, where $\mathcal{C}$ is the number of phones and $\mathcal{L}$ is the number of frames. For the DC_mean model, the mean of this vector was taken over the first dimension to produce a $\mathcal{C}$-shaped vector. In order to calculate the loss, a target vector was constructed of the same size with a 1 at every entry of which the corresponding phone was in the annotated label and a 0 otherwise. The result was then compared to this target vector by taking the sigmoid of every entry and calculating the binary cross-entropy loss. For the DC_max model the procedure was identical, with the exception that the max was taken of the Wav2Vec 2.0 output instead of the mean.

Instead of max or mean pooling, the DC_amax model used adaptive max pooling, comparable to the method of [65]. This means that before training a DNN-HMM was used to determine the alignment range of every phone in the annotated label. During training, the max was taken over the frames in each of these alignment ranges. Cross-entropy loss could then be calculated by taking the logarithm of the softmax and comparing it to a one-hot encoded target vector corresponding to the phone of that alignment range.

The same hyperparameters as the PC model were used for training. Training converged faster than the PC model and therefore models were only trained for $40,000$ updates. The best models were chosen based on validation loss.

Predictions are produced, by taking the max over all frames for DC_max and DC_amax and the mean for DC_mean and then determining which phones are pronounced correctly by checking which corresponding entries score above a specific threshold. For TPD, a phone is simply predicted to be correct when that phone scores above the threshold. For MD, a prediction is labeled as correct when all phones that are in the canonical pronunciation of the target word, score above the threshold. Furthermore, a prediction is labeled as a true accept, when this is the case and in addition all phones in the canonical pronunciation are in the annotated label.

It should be noted that this "truth criterion" (the condition which determines whether a labeled or predicted pronunciation is correct) is different from the one used for the PC model. Unlike the PC model, this approach allows for any utterances around the canonical pronunciation, as long as the actual canonical pronunciation was also said. This approach is less strict and as a result the distribution of the test set is more skewed, with only 11% of utterances being labeled as incorrect. This again posed a problem for determining adequate thresholds, and therefore the thresholds for MD were also based on the balanced F-score rather than the regular F-score.

Just like the trans-GOP model, thresholds were first chosen to maximize the balanced F-score and then to set the FRR to 0.05. The results and thresholds are shown in Table 4.2. The full results are shown in section 4.1.5.

Model	Split	TPD F-score	TPD FAR	MD F-score	MD FAR
DC_max	School-split	**0.895 (2.8)**	**0.190 (2.4)**	0.870 (2.3)	**0.243 (0.66)**
DC_mean	School-split	0.879 (2.3)	0.196 (2.3)	**0.873 (1.9)**	0.236 (1.4)
DC_amax	School-split	0.882 (9.7)	0.251 (8.8)	0.841 (8.8)	0.348 (7.2)
DC_max	Word-split	0.843 (-0.9)	0.325 (-3.1)	0.759 (-3.4)	0.538 (-5.2)
DC_mean	Word-split	0.764 (-1.9)	0.529 (-3.7)	0.680 (-4.1)	0.799 (-6.0)
DC_amax	Word-split	**0.855 (8.3)**	**0.309 (6.2)**	**0.781 (5.7)**	**0.493 (3.8)**

Table 4.2: This table shows the maximal TPD and MD balanced F-score and the TPD and MD FAR when the FRR is fixed at 0.05. TPD scores are averaged over all phones. The thresholds used are shown in parentheses. The best scores for the school and test split are shown in bold.

4.1.5 Model Comparison

This section shows a comparison of the best performing models. Table 4.3 shows the averages and weighted averages of phone TPD scores. The DC and trans-GOP model thresholds are all optimized to maximize average balanced F-score. Table 4.4 shows MD statistics. The PC* model in this table is the exact same model as the PC model but the truth criterion for labels is the same as the DC models, meaning an annotated or predicted string is evaluated as correct if all phones from the canonical pronunciation are in the string. This allows for a more fair comparison between the DC and PC model. Table 4.5 shows the balanced statistics for the models that had skewed distributions due to a less strict truth criterion. The DC model thresholds are all set to maximize the balanced F-score.

Model	Accuracy	Precision	Recall	F-score	FRR	FAR
PC (school-split)	**0.968**/0.834	0.985/0.781	**0.979**	**0.982**/0.862	**0.021**	0.306
DC_max (school-split)	0.934/**0.890**	**0.995**/0.873	0.932	0.962/**0.895**	0.068	**0.155**
DC_mean (school-split)	0.933/0.867	0.992/0.843	0.935	0.962/0.879	0.065	0.201
DC_amax (school-split)	0.911/0.878	0.990/**0.860**	0.914	0.950/0.882	0.086	0.158
Trans-GOP (school-split)	0.785/0.746	0.979/0.743	0.789	0.871/0.757	0.211	0.294
PC (word-split)	0.843/0.837	**0.992**/0.882	0.843	0.896/0.836	0.157	**0.130**
DC_max (word-split)	**0.866**/**0.838**	0.989/0.853	**0.868**	**0.915**/0.843	**0.132**	0.161
DC_mean (word-split)	0.824/0.757	0.982/0.774	0.828	0.878/0.764	0.172	0.268
DC_amax (word-split)	0.843/0.836	0.991/**0.867**	0.842	0.902/**0.855**	0.158	0.140
Trans-GOP (word-split)	0.813/0.714	0.967/0.677	0.822	0.885/0.735	0.178	0.391

Table 4.3: TPD performance of the different models. All statistics are averaged over all phones. For accuracy, precision and F-score, the balanced version is shown after a '/'. The best scores for each split are shown in bold.

Finally, Figure 4.1 shows how the balance between FRR and FAR is affected by different thresholds for the DC and trans-GOP models in MD. In this Figure PC and PC* are points instead of graphs, since the PC model does not have an adjustable threshold. From this image it is clear that the trans-GOP performed significantly worse in the school-split. In the word-split however, it was close to most other models and instead DC_mean was the outlier.

To get an indication of how the models perform with different phones, the spread of the TPD F-scores per phone is shown as a boxplot in Figure 4.2. It shows that the performance of the models was relatively

Model	Accuracy	Precision	Recall	F-score	FRR	FAR
PC (school-split)	0.925	0.943	0.940	0.941	0.060	**0.102**
PC* (school-split)	**0.934**	0.969	**0.956**	**0.963**	**0.044**	0.237
DC_max (school-split)	0.891	**0.980**	0.895	0.940	0.105	0.144
DC_mean (school-split)	0.915	0.978	0.927	0.951	0.073	0.173
DC_amax (school-split)	0.865	0.975	0.870	0.920	0.130	0.178
Trans-GOP (school-split)	0.757	0.787	0.815	0.800	0.185	0.328
PC (word-split)	0.620	0.927	0.344	0.502	0.656	**0.034**
PC* (word-split)	0.614	**0.976**	0.560	0.712	0.440	0.078
DC_max (word-split)	0.796	0.944	0.808	0.871	0.192	0.273
DC_mean (word-split)	0.776	0.902	0.826	0.862	0.174	0.513
DC_amax (word-split)	**0.825**	0.947	0.841	**0.890**	0.159	0.265
Trans-GOP (word-split)	0.778	0.747	**0.852**	0.796	**0.148**	0.300

Table 4.4: MD performance of different models. The best scores for each split are shown in bold.

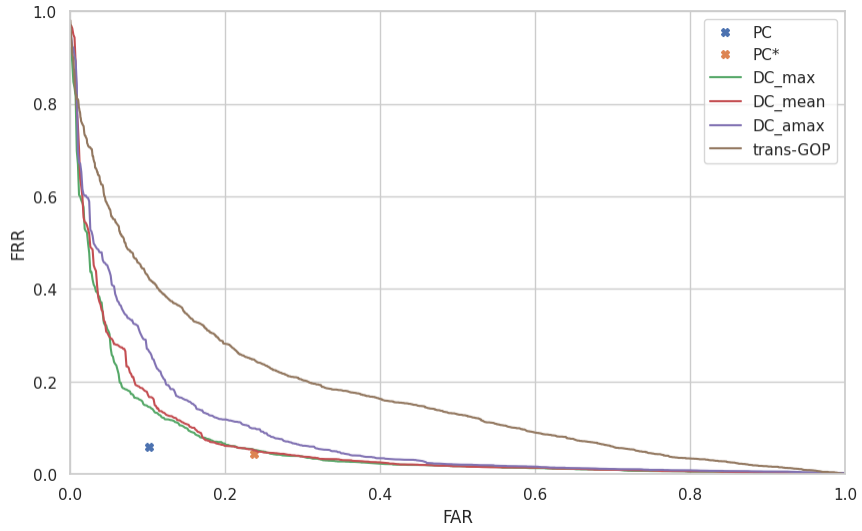
Model	Accuracy	Precision	F-score
PC* (school-split)	0.853	0.781	0.860
DC_max (school-split)	**0.874**	**0.846**	0.870
DC_mean (school-split)	**0.874**	0.826	**0.873**
DC_amax (school-split)	0.845	0.813	0.841
PC* (word-split)	0.755	**0.849**	0.678
DC_max (word-split)	0.764	0.716	0.759
DC_mean (word-split)	0.643	0.578	0.680
DC_amax (word-split)	**0.783**	0.729	**0.781**

Table 4.5: Balanced accuracy, precision and F-score for MD. The best scores for each split are shown in bold.

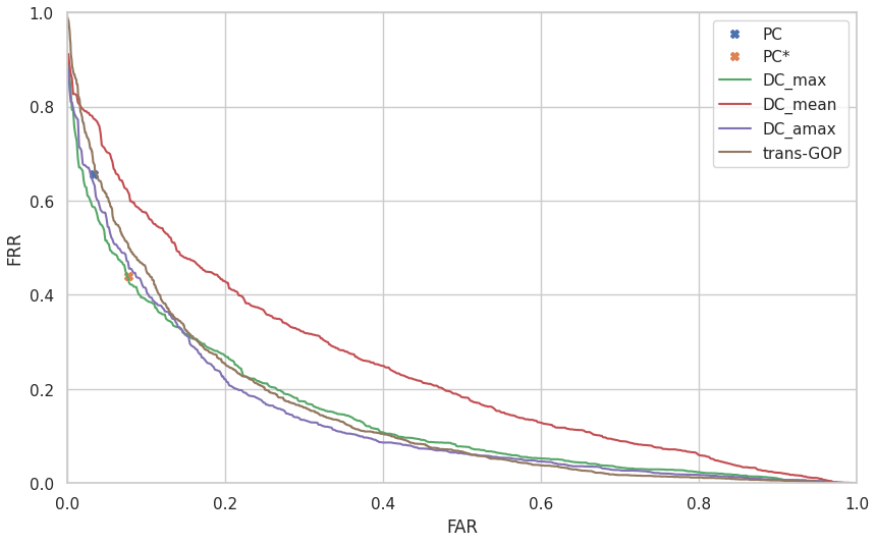
consistent across phones, with the exception of some outliers in the word-split, most notably for the PC and DC_mean models. To get an indication of how the models perform with different students, the spread of the MD F-scores per student is shown in a boxplot in Figure 4.3. In the school-split, all models but the trans-GOP model have relatively low variance, however the PC and DC_amax models have some lower outliers. In the word-split the performance of all model varies significantly more over students, especially for the PC and trans-GOP models.

4.1.6 Balanced Training

After analysing the initial results it was hypothesized that the models might be biased, because the majority of dataset is correctly pronounced words. Therefore, are balanced training sets were created which contained all word attempts of which one phone was pronounced incorrectly and a randomly selected subset of all correctly pronounced words, such that the number of correctly and incorrectly pronounced words was equal. In the word-split and the school-split 11.3% and 15.0% of words in the training set are mispronounced and thus the balanced training set contained 22.6% and 30.0% of the data in the original training sets respectively. All models were retrained using these new balanced datasets. The F-scores are compared to the original



(a) School-split.



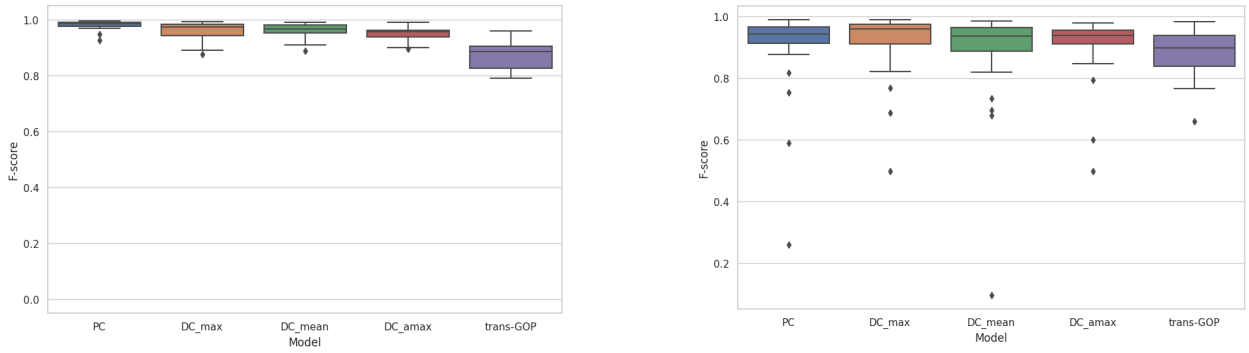
(b) Word-split.

Figure 4.1: Graphs that show how the FRR and FAR vary with different thresholds in MD.

results in Table 4.6. Table 4.7 also shows the FAR for the DC models when the FRR is set to 0.05.

4.2 Reading Error Detection (RED)

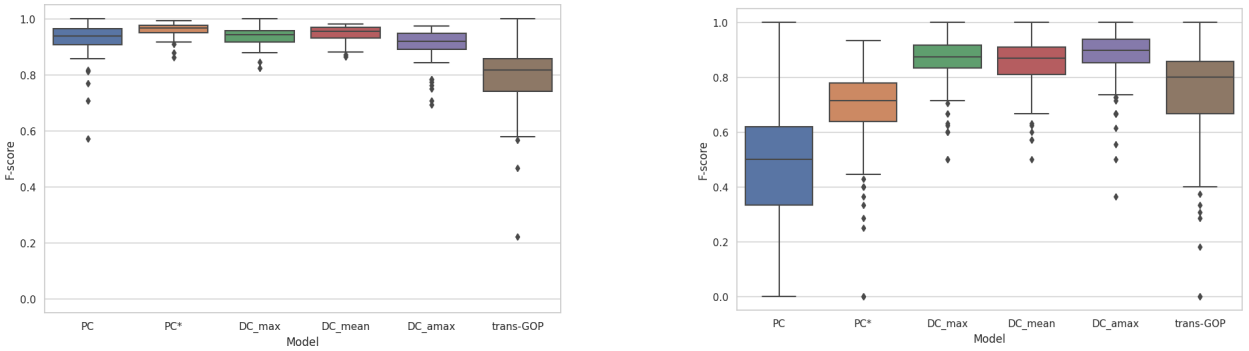
This section describes the results for RED. This task involves the recognition of errors on both the word and the phone level. Word-level errors are errors where extra utterances are added or words from the prompt



(a) School-split.

(b) Word-split.

Figure 4.2: Boxplots showing the distribution of TPD F-scores of different phones.



(a) School-split.

(b) Word-split.

Figure 4.3: Boxplots showing the spread of MD F-scores of different students.

are omitted. Phone-level errors are errors where words from the prompt are substituted for the wrong word. For this research all substitutions are classified as phone-level errors, whether it concerns small phonetic changes or an entirely different word. As a baseline, both word and phone-level errors are identified with a WFST model (section 4.2.2). Next it is determined whether it is beneficial to identify phone-level errors with Wav2Vec 2.0 (section 4.2.3).

4.2.1 Data Preparation

For this task the Malmberg dataset was used, which contains 2.5k audio samples of children reading short texts out loud. 310 of these files were removed from the dataset because the annotations showed more than 10 instances of noise. The remainder was used for 4-fold cross-validation, with one fold for each of the grade classes in the dataset. These are grade 3 (ages 6 and 7), grade 4 (ages 7 and 8), grade 5 (ages 8 and 9) and finally grades 6 through 8 (ages 9-12). A subset was taken from each of these grades so that each test set consists of approximately 10% of the dataset. Unfortunately, the Malmberg dataset does not have student-specific metadata. Instead the subset is made based on the region, in order to ensure there is no overlap in students.

Model	TPD F-score		MD F-score	
Training set	**Full**	**Balanced**	**Full**	**Balanced**
PC (school-split)	0.982/0.862	0.970/0.883	**0.941**	0.906
PC* (school-split)	-	-	**0.963/0.860**	0.921/0.858
DC_max (school-split)	0.962/0.895	0.956/0.898	**0.940/0.870**	0.918/0.862
DC_mean (school-split)	**0.962/0.879**	0.935/0.877	**0.951/0.873**	0.924/0.857
DC_amax (school-split)	0.950/0.882	**0.964/0.888**	0.920/0.841	**0.921/0.858**
PC (word-split)	0.896/0.836	**0.916/0.861**	0.502	**0.573**
PC* (word-split)	-	-	0.712/0.678	**0.725/0.695**
DC_max (word-split)	0.915/0.843	0.836/0.870	0.871/0.759	0.852/0.797
DC_mean (word-split)	0.878/0.764	**0.913/0.779**	**0.862/0.680**	0.837/0.669
DC_amax (word-split)	0.902/0.855	0.944/0.849	0.890/0.781	0.872/0.791

Table 4.6: This table compares the original results to the results from the models train with the balanced training set. Balanced F-scores are shown after a '/'. Thresholds for the DC models were set to optimize the balanced F-score. PC* does not show TPD stats, since these would be the same as the PC model. For each model, split and task the results of better performing training set are shown in bold, as long as both the regular and balanced F-score are improved upon.

Model	TPD FAR		MD FAR	
Training set	**Full**	**Balanced**	**Full**	**Balanced**
DC_max (school-split)	**0.190**	0.200	**0.243**	0.358
DC_mean (school-split)	**0.196**	0.251	**0.236**	0.296
DC_amax (school-split)	0.251	**0.215**	**0.348**	0.354
DC_max (word-split)	0.325	0.325	**0.538**	0.544
DC_mean (word-split)	**0.529**	0.554	**0.799**	0.851
DC_amax (word-split)	**0.309**	0.323	**0.493**	0.541

Table 4.7: This table compares the FAR when the FRR is fixed at 0.05 for both training sets. For each model and split the results of better performing training set are shown in bold.

The annotated error labels were combined into a maximum of 7 error classes (exact error labels were shown in Table 3.2): "correct", "insertion", "omission", "repetition", "substitution", "decoding" and "false start".

In order to train acoustic models, a lexicon was needed with phonetic transcriptions of all utterances. This lexicon was taken from the Corpus Gesproken Nederlands (CGN) [41]. Words and utterances that were not in the CGN lexicon where transcribed using a Phonetisaurus grapheme-to-phoneme (G2P) model that was also trained on the CGN lexicon. The resulting lexicon was checked manually for errors.

In order to train Wav2Vec 2.0 models, the Malmberg dataset was split up into single utterances. All types of utterances were used for training, including repetitions, false starts and insertions. However, during testing only the phone-level error classes "correct", "substitution" and "decoding" were included. Phonetic transcriptions were taken from the lexicon described above. Words with multiple correct pronunciation variants were used multiple times (once with every variant) during training. During testing the variant with the highest score was used. For the initial Wav2Vec 2.0 models (section 4.2.3) a clean dataset was used, where the error classes alignment boundaries were given by the annotations. Words that were annotated

with noise were left out as well as words shorter than 0.25 seconds, since these often sounded distorted out of context. To evaluate the phone-level errors of the hybrid model (section 4.2.4) the error classes and alignment boundaries were given by the baseline WFST rather than the annotation. All utterances from the phone-level error classes were included, even those shorter than 0.25 seconds. Of course, the annotations were still used to evaluate the final predictions. The same 4-fold cross validation was used as for the WFST model.

4.2.2 Baseline WFST models

This section shows the results of the baseline weighted finite state transducer (WFST) based models (also see section 3.2.2). These models detect word-level errors like insertions, omissions and repetitions. Word substitutions can also be detected as a combination of an insertion and a omission. In the baseline this is how phone-level errors are detected.

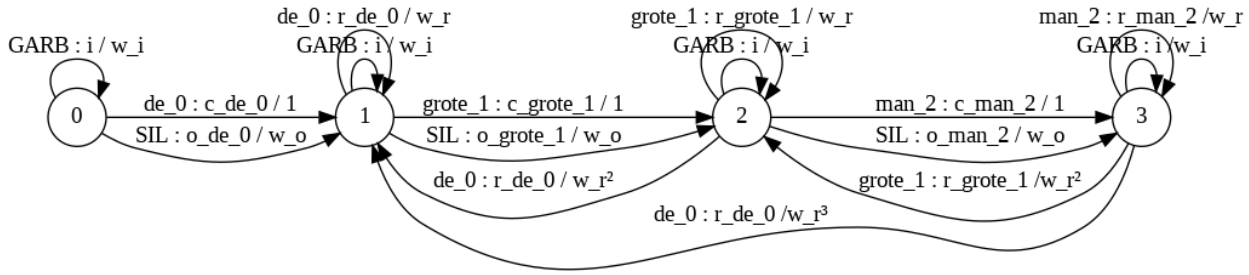
4.2.2.1 Models

Like explained in section 2.2, WFST models for RED are a composition of four WFSTs: $H \circ C \circ L \circ G$. $H \circ C$ is the WFST representation of the acoustic model, which takes in the MFCC features and outputs probabilities for a given sequence of context-dependent phones. For this research, using a neural network based acoustic model yielded better results than the simpler Gaussian mixture model. Furthermore, L is a WFST representation of the lexicon, which takes in sequences of phones and outputs words. WFSTs for ASR use very large lexica, but in the case of RED the WFSTs are prompt-specific and therefore the lexicon can only output words from the prompt. The final WFST G represents the grammar. For ASR this grammar is a language model and is only used to give a probability to a given sequence of words. However, for RED G is custom designed so that it can also output various reading errors.

Multiple custom grammar WFSTs were created for this research. In the simplest case the grammar can only output insertions, which are represented with self-loops, and omissions and correct words which both transition to the next state. Repetitions can be added to the grammar WFST by allowing words from previous states to be recognized for backwards transitions. An example of this simplest case with repetitions is shown in Figure 4.4a for the phrase "de grote man" (the big man). This WFST takes words as input as well as "SIL" for silence and "GARB" for out of vocabulary words. Words are also marked with their position so that there is no ambiguity when a word is used multiple times in the same prompt. The WFST can output whether a word was correctly pronounced, repeated or omitted by outputting the word marked with 'c', 'r', or 'o' respectively. It can also output 'i' (for insertion) if an out-of-vocabulary utterance is recognized. Every error has a weight associated with it. The weight of recognizing a correct word is always set to one. The weight of repetitions is raised to a power depending on how far back in the sentence a word is repeated. This is done because when two words in a row are repeated, the model will output it as a repetition and a correct word, instead of two repetitions.

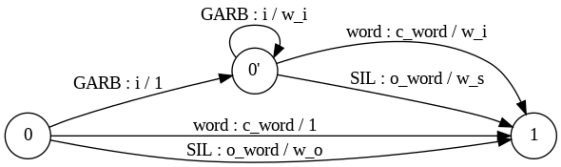
Substitutions are recognized as an omission plus an insertion. This means that the weight for substitutions in this model is effectively $w_i \times w_o$. This is not always desirable, since this means substitutions cannot be tuned separately from insertions and omissions. A grammar can be modified to include a separate weight w_s for substitutions, by replacing the self-loops that recognize insertions from Figure 4.4a with the construction shown in Figure 4.4b. For every state i this WFST has a state i', which allows a new weight w_s to be assigned to the the recognition of an insertion followed by an omission. Repetitions can be recognized as before, but additionally if there is a repetition edge from state i to j with weight w then there is an edge from i' to j with weight $w \times w_i$ (this is not shown in image 4.4b).

Finally, grammars can be modified to recognize decoding and false starts. Following [52], it was decided that these errors would be detected using syllables rather than individual phones. A consequence of this is that decoding and false start errors can only be recognized for words with more than one syllable. Similar errors for one-syllable words will be recognized as insertions or substitutions. To recognize these errors, the lexicon L was modified to only output the syllables of the words in the prompt. An extra WFST L' is used to combine these syllables into words. An example structure of L' is shown in Figure 4.4c. Of course a word can be recognized when its syllables are recognized in order. From any non-zero state i there is an edge to i' when a silence is recognized. From here either the remaining syllables can be recognized in order to output a decoding error or the first syllable can be recognized, which indicates that the student is starting over and thus made a false start error. Figure 4.4c shows only one word but in reality all words in the prompt are represented by their own loops starting from state 0. State 0 also has self-loops for all one-syllable words as well as the SIL and GARB class where the input is equal to the output. The composite lexicon WFST $L \circ L'$ now takes in phones and outputs words, similar to a regular lexicon WFST L, but in addition it can also output words marked with 'd' for decoding errors for any word with more than one syllable. Furthermore it can output a false start (marked with 'fs') before outputting any word with more than one syllable. To handle these errors G is modified so that every state i that corresponds to a word with more than one syllable has a self-loop from that can recognize and output a false start of that word and a transition from i to $i+1$ that can recognize and output a decoded word, both with their own respective weights.

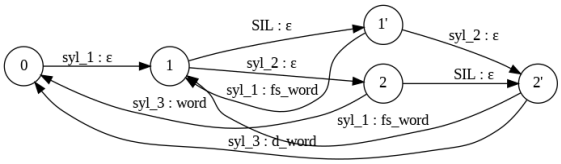


(a) Example of simple WFST.

.



(b) Example of the implementation of a substitution weight (w_s) in a WFST. For visibility repetitions are omitted from this image.



(c) Example of WFST for recognizing false starts and decoding.

Figure 4.4: Examples of various WFSTs. Edges are marked with *input_symbol:output_symbol/weight*.

Before the final result can be evaluated, the output of these WFSTs is processed from left to right and the following rule-based transformations are applied:

- Insertions neighbouring omissions are changed into substitutions:
  $i + o\_word_i \rightarrow s\_word_i$

- Any correct word after a repetition that goes back at least one state is changed to a repetition, if the correct word was already recognized earlier:
  $c\_word_i + r\_word_{i-1} + c\_word_i \rightarrow c\_word_i + r\_word_{i-1} + r\_word_i$
- A repetition of a previously omitted word is changed to a correct word, if the correct word was not yet recognized earlier:
  $o\_word_i + r\_word_i \rightarrow c\_word_i$

In early tests all weights were set to 1, but it quickly became apparent that this would not lead to good results. Instead the weights were initially set based on the number of occurrences of the error classes in the dataset. For grammars without repetitions, annotated repetitions were counted as insertions. For the grammars without w_s, annotated substitutions were counted as both omissions and insertions. Finally for grammars without decoding, annotated false starts were counted towards insertions and annotated decoding errors were counted towards substitutions or towards insertions and omissions. The same was the case for all decoding and false start errors for words with only one syllable. All weights were normalized with respect to the number of correctly pronounced words (such that w_c is always equal to 1), with the exception of w_d and w_{fs} which were normalized with respect to the number of correctly pronounced words with more than one syllable. The resulting weights are shown in Table 4.8.

	simple	+r	+s	+r+s	+d	+d+r	+d+s	+d+r+s
w_c	1	1	1	1	1	1	1	1
w_i	0.1198	0.1044	0.0817	0.0663	0.1050	0.0896	0.0703	0.0549
w_o	0.0517	0.0517	0.0136	0.0136	0.0482	0.0482	0.0136	0.0136
w_r	-	0.0154	-	0.0154	-	0.0154	-	0.0154
w_s	-	-	0.0381	0.0381	-	-	0.0346	0.0346
w_d	-	-	-	-	0.0204	0.0204	0.0204	0.0204
w_{fs}	-	-	-	-	0.0102	0.0102	0.0102	0.0102

Table 4.8: Weights for the various WFST versions.

After gathering the initial results, it was found that many error types were still predicted too often. Therefore, each weight was iteratively adjusted based on the difference between the true count and the predicted count of the corresponding error class. So for example, each iteration w_i is multiplied by the number of insertions in the dataset divided by the number of insertions predicted by the model. If models do not use certain weights, the occurrences of that class in the dataset are counted towards the class that replaces it. So for example if a model cannot recognize repetitions, all repeated words in the dataset are counted as insertions. Each iteration, all weights were updated simultaneously. For computational simplicity, each weight was normalized so that w_c was always equal to 1 after the initial adjustment. This process was repeated for 20 iterations, at which point the predicted number for each error class got close to the true number and thus the weights converged.

Of course, this is a heuristic approach to training, since the fact that the model predicts the correct amount of errors for each error class, does not mean that the predictions themselves are correct. It would be better to adjust the weights in a way that minimizes modeling errors. However, determining the number of modeling errors is not trivial for each error class. Furthermore, each weight adjustment will likely decrease the number of false positives while increasing the number of false negatives or vice versa. This makes it very difficult to determine an optimum and therefore it would not always be clear in which direction the weights should be adjusted. The total number of predicted errors on the other hand, is directly influenced by the

model weights and therefore it can always easily be determined in what direction to adjust the weights. In addition, any performance-measuring metrics can still be calculated at every iteration after which the best iteration can be selected.

4.2.2.2 Evaluation

The main metric for evaluating RED results is the word error rate (WER) described in section 2.3.1. In this case the WER is calculated between the transcript of the student's utterances and the output predicted by the model. The main goal of this evaluation was to test whether the model would report an error when one is made, but not punish the model for predicting the wrong type of error. In order to achieve this, in both the prediction and the target omissions are left out and the other errors are simply marked as "<error>". For repetitions this process was not trivial, since when a word is repeated, the annotations sometimes mark the first and sometimes the second utterance of the word as a repetition. This was changed in post-processing, so that the second word was always marked as an error and the first is correct, the same way the model would predict it. Consecutive errors are collapsed into a single error. This is to prevent the model from being punished when it, for example, predicts one insertion when two consecutive insertions are annotated.

The WER captures information on the model's performance on both word-level and phone-level errors. For the hybrid model later it is also important to evaluate specifically how well the models detect word-level errors. Therefore, the WER was recalculated in a way where phone-level errors were not penalized. To achieve this, every time a word is marked as correct, substitution or decoding in the output, it is checked whether this word is correct in the annotation. Only if this is the not the case, it is marked as "<error>". From there the WER is calculated as before. This metric will be referred to as WER*. An example of the how the WER and WER* outputs are constructed is shown in Figure 4.5.

Transcript	De	kl-	kleine	jongens	loopt
Annotation	c_de_0	fs_kleine_1	c_kleine_1	s_jongen_2	c_loopt_3
WER target	de	<error>	kleine	<error>	loopt
Output	c_de_0		c_kleine_1	c_jongen_2	s_loopt_3
WER output	de		kleine	jongen	<error>
WER* output	de		kleine	<error>	loopt

Figure 4.5: Example of how the WER and WER* hypotheses are derived from the output. In this (fictional) case the prompt was "De kleine jongen loopt". The student makes two reading errors: first there is a false start for the word "kleine" and then there is substitution error when "jongen" is substituted for the plural "jongens". The model misses the false start and erroneously thinks the word "jongen" is correctly pronounced but that "loopt" is substituted. This means that in the WER output only "loopt" is replaced with "<error>". All words in the output are correct or substitutions, so for the WER* it is checked whether the words are correctly pronounced in the annotation. This is only not the case for "jongen" so this only this word is changed to "<error>". The Levenshtein distance between the WER output and target is 3 and the target has length 5, so the WER is 0.6. The distance between the WER* output and the target is 1 so the WER* is 0.2.

The evaluation methods above were chosen to simulate a scenario where the model is used as a computer-assisted reading tutor, which can indicate to a student at which words in the prompt an error is made, but which does not show what type of error is made. Of course in an ideal scenario the model would also be able to distinguish different errors from each other. In order to assess this, we can look at each word in the prompt and see how often the output class (e.g. correct, omission, substitution) in the prediction matches that of the annotation. This information can be used to calculate the diagnosis error rate (DER) described

in section 2.3.1. One problem for this strategy is that the words from the annotation need to be aligned to the words from the prediction. This is not trivial because insertions, repetitions and false starts will cause the target words and predicted words to no longer be aligned when they are erroneously predicted or missed. To get around this, we only compare each word in the prompt once by only looking at the error classes "correct", "decoding", "omission" and "substitution", since one of these classes must be recognized at least once for each word in order to progress through the WFST. This is simply achieved by filtering the other error classes out of the annotation and the predicted output. An example is shown in Figure 4.6. Unfortunately, there are still cases where this alignment does not work out, either because of annotation errors or because repetitions in the prediction cause words to be assigned two different output classes. An example of this is shown in Figure 4.7. This occurred for 6-10% of the data points, depending on the model. When this occurred, only the predictions before the first occurrence of the ambiguous word were taken into account for calculating the DER. The same method can be used to calculate the FAR and FRR, which gives a more detailed indication of the model's performance on detecting these four error classes.

Transcript	De	kl-	kleine	jongens		loopt
Annotation	c_de_0	fs_kleine_1	c_kleine_1	s_jongen_2		c_loopt_3
Filtered annotation	c_de_0		c_kleine_1	s_jongen_2		c_loopt_3
Output	c_de_0		c_kleine_1	c_jongen_2	insertion	c_loopt_3
Filtered output	c_de_0		c_kleine_1	c_jongen_2		c_loopt_3
Evaluation	TA		TA	FA		TA

Figure 4.6: Example of how the annotation and output are filtered in order to match the words of the prompt to one prediction so that statistics like the DER, FAR and FRR can be calculated. In this case the model again misses the false start and erroneously thinks the word "jongen" is correctly pronounced but followed by an insertion. By filtering out all error classes but "correct", "decoding", "omission" and "substitution", the prompt words can be matched one to one between the filtered annotation and filtered output. In this case there is one false acceptance for the word "jongen".

Transcript	de		jongen	de	kleine	jongen	loopt
Annotation	r_de_0		r_jongen_2	c_de_0	c_kleine_1	c_jongen_2	c_loopt_3
Filtered Annotation				c_de_0	c_kleine_1	c_jongen_2	c_loopt_3
Output	c_de_0	o_kleine_1	c_jongen_2	r_de_0	c_kleine_1	r_jongen_2	c_loopt_3
Filtered output	c_de_0	o_kleine_1	c_jongen_2		c_kleine_1		c_loopt_3
Evaluation	TA	FR?	TA		TA?	TA	

Figure 4.7: Example of where the filtering approach fails. In this case the student initially omitted the word "kleine" and then started over to correct their mistake. The model finds the correct path through the WFST, where "r_de_0" goes back to the first node of the WFST. In post-processing, the second instance of "jongen" is marked as a repetition, but "kleine" is not because it was omitted earlier, which gives the final output shown in the example. In the filtered output there are two predictions for the word "kleine" and so it cannot be matched one-to-one with the filtered annotation and therefore this data point would not be counted towards the DER.

Finally, for each model a linear regression analysis was conducted between the number of annotated errors and the number of predicted errors, both as a percentage of the number of words in the prompt. The R-value of this analysis represents the model's ability to predict the amount of errors made by a student, which can be seen as an assessment of the student's reading level.

4.2.2.3 Results

The WER, WER*, DER and R-value for the initial models are shown in Table 4.9. The weights were updated using the method described above in section 4.2.2.1 for 20 iterations. The results of the final models are shown in Table 4.10. For the best performing model WFST+r+s at iteration 20, the results per grade are shown in Table 4.11.

Model	**WER**	**WER***	**DER**	**Error%**	**Predicted Error%**	***R*-value**
WFST-simple	0.208	0.159	0.344	0.111	0.203	0.115
WFST+r	0.212	0.183	0.348	0.111	0.251	**0.175**
WFST+s	**0.205**	**0.154**	**0.338**	0.111	0.198	0.110
WFST+r+s	0.212	0.179	0.339	0.111	0.247	0.148
WFST+d	0.206	0.157	0.356	0.111	0.209	0.124
WFST+d+r	0.210	0.178	0.344	0.111	0.249	0.168
WFST+d+s	**0.205**	**0.154**	0.346	0.111	0.202	0.115
WFST+d+r+s	0.214	0.179	0.356	0.111	0.247	0.134

Table 4.9: Results with initial weights.

Model	**WER**	**WER***	**DER**	**Error%**	**Predicted Error%**	***R*-value**
WFST-simple	0.177	0.143	0.391	0.111	0.147	0.097
WFST+r	0.177	0.145	0.401	0.111	0.149	0.110
WFST+s	0.152	0.098	0.349	0.111	0.112	0.131
WFST+r+s	**0.151**	**0.097**	0.347	0.111	0.112	0.155
WFST+d	0.180	0.147	0.392	0.111	0.146	0.109
WFST+d+r	0.178	0.146	0.406	0.111	0.146	0.127
WFST+d+s	0.154	0.099	0.341	0.111	0.111	0.135
WFST+d+r+s	0.152	0.099	**0.325**	0.111	0.111	**0.160**

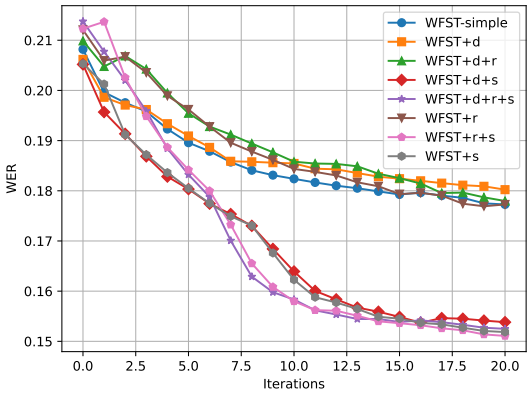
Table 4.10: Results after 20 iterations.

Grade	**WER**	**WER***	**DER**	**Error%**	**Predicted Error%**	***R*-value**
3	0.175	0.125	0.342	0.114	0.137	0.118
4	0.134	0.090	0.347	0.112	0.086	0.168
5	0.142	0.093	0.430	0.108	0.094	0.188
6-8	0.155	0.091	0.316	0.109	0.122	0.288

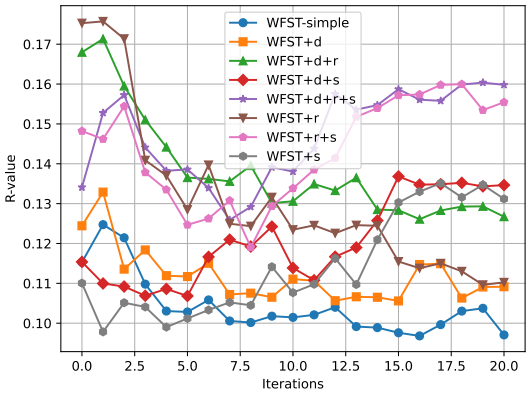
Table 4.11: Results for each of the four different grade groups (which are also the folds used for cross-validation) for WFST+r+s after 20 iterations.

To get a better overview of the training process, Figure 4.8 shows how these statistics changed over each iteration of weight adjustments. All models were initially too strict, which can be seen from the difference in the percentage of errors and the percentage of predicted errors in Table 4.9. Therefore, each iteration all the weights corresponding to each error were lowered. This also affected the balance between the FRR and

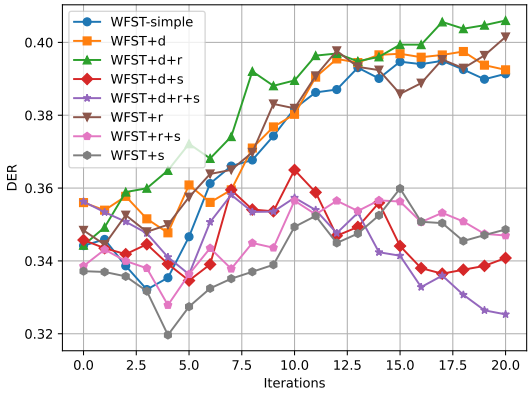
FAR, which can be seen in Figure 4.8d. As the models became more likely to predict the "correct" class, the FAR increased and the FRR decreased.



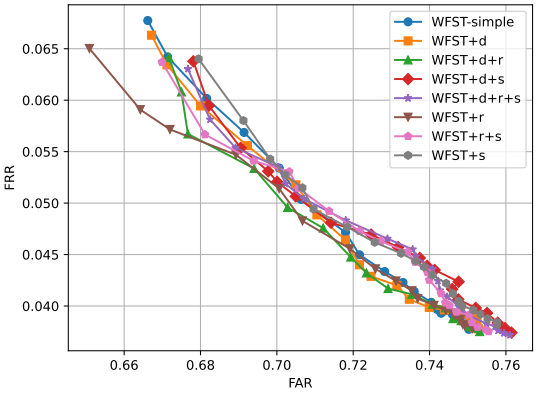
(a) Progression of WER over iterations.



(b) Progression of R-value over iterations.



(c) Progression of DER over iterations.



(d) Balance between FAR and FRR over iterations.

Figure 4.8: Graphs showing the progression of various statistics during the training process.

4.2.3 Wav2Vec 2.0

The next step was to train and evaluate Wav2Vec 2.0 models on the clean Malmberg dataset, in order to detect phone-level errors (see section 3.2.3).

4.2.3.1 Models

Like with target phone detection, two types of Wav2Vec 2.0 models were trained: a phone classification (PC) and a direct classification (DC) variant. Specifically, the DC model with adaptive max pooling (DC_amax)

was used, since this was the best performing model with the CHOREC dataset. Each type was trained either from scratch or was a fine-tuned version of the model trained with the CHOREC school-split. Each model was also evaluated in a zero-shot situation where the CHOREC-trained models were used to make predictions without training on any Malmberg data. Every model variant was trained and evaluated once for every test split and the results were combined by taking the average of the folds.

The Malmberg dataset has a similar issue as TPD in section 4.1, namely that the distribution of correctly and incorrectly pronounced utterances is very skewed, with only 4.3% of utterances being pronounced incorrectly. Therefore, the models were also trained with a balanced dataset, which contained all incorrect utterances and an equal amount of randomly selected correct utterances. These balanced dataset models were trained from scratch with randomly initialized weights.

4.2.3.2 Evaluation

The PC model was evaluated as before, meaning a prediction or a label is correct if the phone sequence matches that of the canonical pronunciation of the word. The DC model was evaluated differently from section 4.1. The DC model checks whether each phone in the canonical pronunciation scores above a certain threshold in order to make a prediction. This method does not verify the order of phones nor the amount of times each phone occurs. In section 4.1, labels were evaluated in a similar way, in order to ensure a fair comparison. However, for RED it is important that pronunciation errors caused by the order of phones or by a difference in the number of occurrences of a phone are also recognized. For this reason, labels are evaluated in the same way as the PC model. In an attempt to also enforce order of phones with the DC model, a different evaluation method was implemented. Instead of simply taking the maximum score for each phone, this implementation searches the output frames from left to right until the first phone of the canonical pronunciation scores above a pre-determined threshold. Then it searches for the next phone in the canonical pronunciation in the remaining frames. If all phones score above the threshold the model predicts the word to be correctly pronounced. Not only does this implementation check whether the phones are in the correct order, but it also enforces that when phones occur multiple times, they need to score above the threshold multiple times. The ordered and unordered versions of the DC model used separate thresholds. Both used the same threshold across all 4 folds for the cross-validation.

The models are evaluated based on accuracy, precision, recall, F-score, FRR and FAR. Because the data is so skewed, balanced versions of the precision, recall and F-score are also calculated.

4.2.3.3 Results

Results for all models are shown in Table 4.12. These results were calculated, using the cumulative results from all 4 folds of the cross-validation. The thresholds for the DC models in this table were set in order to optimize the balanced F-score.

Of course the balance between the FAR and FRR for the DC models is affected by the chosen threshold. The FAR and FRR from the results in Table 4.12 might not always be desirable in practice. Table 4.13 shows the FAR valules at various FRR cutoffs for the different DC models.

The FRR for the PC model is also not desirable in practice. Unfortunately, the PC model does not have a threshold parameter like the DC model to change the balance between the FRR and FAR. However, the balance can altered heuristically, by setting a threshold for an allowed maximum phoneme error rate (PER) between the canonical pronunciation and the prediction. The relation between the maximum allowed PER and the balanced F-score is plotted in Figure 4.9. It shows that setting a low threshold has a positive effect on the results. For the "fine-tuned" and "from scratch" models the optimal threshold was a PER of 0.143, which allows for one phone deletion, insertion or substitution for words with 7 or more phones. For the PC

Model	Accuracy	Precision	Recall	F-score	FRR	FAR
PC (zero-shot)	0.063/0.519	**0.990**/**0.808**	0.021	0.041/0.041	0.979	**0.005**
DC (zero-shot)	**0.955**/0.490	0.957/0.490	**0.998**	**0.977**/0.657	**0.002**	0.996
DC_ordered (zero-shot)	0.830/**0.566**	0.964/0.535	0.855	0.906/**0.658**	0.145	0.710
PC (fine-tuned)	0.687/**0.781**	**0.992**/**0.845**	0.678	0.806/0.752	0.322	**0.119**
DC (fine-tuned)	**0.861**/0.704	0.977/0.646	**0.876**	**0.924**/0.743	**0.124**	0.459
DC_ordered (fine-tuned)	0.840/0.753	0.982/0.705	0.849	0.911/**0.771**	0.151	0.340
PC (from scratch)	0.714/**0.786**	0.991/**0.830**	0.707	0.826/0.764	0.293	**0.138**
DC (from scratch)	**0.856**/0.713	0.978/0.656	**0.869**	**0.920**/0.747	**0.130**	0.437
DC_ordered (from scratch)	0.853/0.758	**0.982**/0.708	0.862	0.918/**0.777**	0.138	0.341
PC (balanced)	0.629/**0.764**	**0.993**/**0.861**	0.616	0.760/0.718	0.384	**0.095**
DC (balanced)	0.829/0.699	0.977/0.649	0.841	0.904/0.732	0.159	0.436
DC_ordered (balanced)	**0.854**/0.732	0.980/0.676	**0.866**	**0.919**/**0.759**	**0.134**	0.397

Table 4.12: Results of Wav2Vec 2.0 model variations. The accuracy, precision and F-score show balanced versions after the "/". The best results for each training method are shown in bold.

model trained on a balanced dataset, the optimal threshold is 0.25. To compare the effects of the threshold on the DC model and the maximum allowed PER on the PC model, Figure 4.10 shows how the balance between FAR and FRR varies for each model.

Model	FRR = 0.2	FRR = 0.1	FRR = 0.05
DC (zero-shot)	0.693	0.838	0.923
DC_ordered (zero-shot)	0.619	0.798	0.897
DC (fine-tuned)	0.361	0.513	0.659
DC_ordered (fine-tuned)	**0.284**	0.428	0.584
DC (from scratch)	0.360	0.500	0.646
DC_ordered (from scratch)	0.287	**0.407**	**0.574**
DC (balanced)	0.384	0.567	0.718
DC_ordered (balanced)	0.313	0.465	0.642

Table 4.13: The FAR for the different DC models at an FRR of 0.2, 0.1 and 0.05. The best scores are shown in bold.

4.2.4 Hybrid Model

The next step was to combine the WFST and Wav2Vec 2.0 models into one hybrid model. The WFST in the hybrid model is only responsible for detecting word-level errors. This means word omissions, repetitions, insertions and false starts. Any phone-level errors detected by the WFST model, as well as all words the WFST deems correctly pronounced, will be re-evaluated by Wav2Vec 2.0. The WFST is also used to determine at what frames these utterances are located, so that these can be used as the input for the Wav2Vec 2.0 model.

For the baseline WFST, WFST+r+s was selected. This model scored the best WER*, which indicates it has the best performance on word-level errors. This model was used to create a new dataset, containing

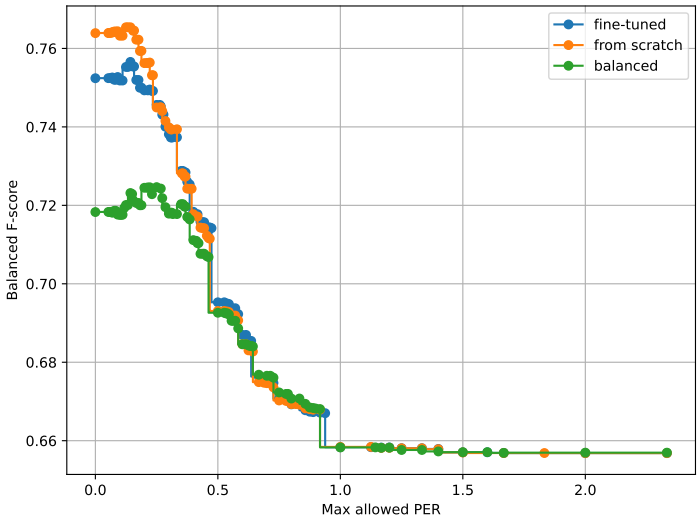


Figure 4.9: The balanced F-score as a function of the maximum allowed PER for the different PC models. The maxima are at (0.143, 0.765) (from scratch), (0.143, 0.758) (fine-tuned) and (0.250, 0.726) (balanced).

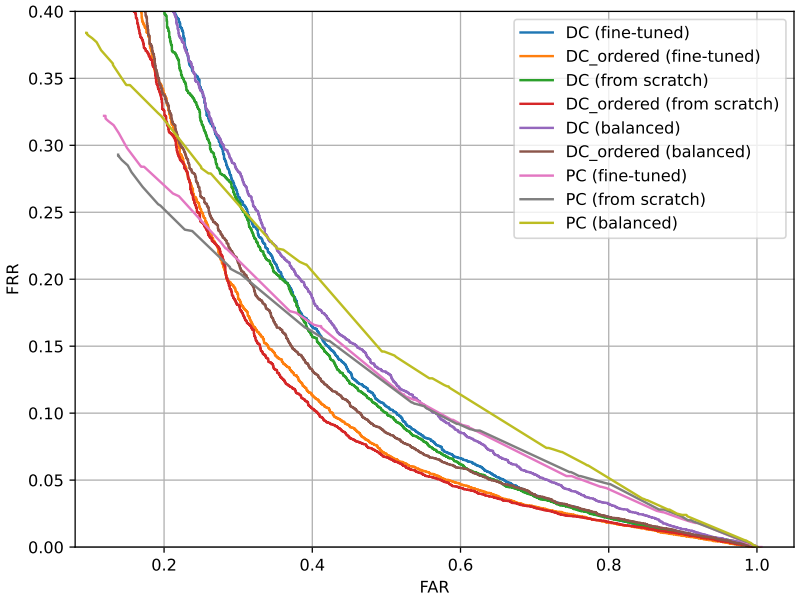


Figure 4.10: The trade-off between FAR and FRR for every model.

all utterances that the model predicted to be correct, a substitution, or a decoding error. The frames for these words were also extracted using the model, rather than the annotated boundaries. This new dataset was used to re-evaluate the Wav2Vec 2.0 models from the previous section, so that they can be compared to the results on the clean dataset. The results are shown in Table 4.14. This table also shows how the WFST baseline performed on this dataset, although the results might be hard to compare since the balance between FAR and FRR is so different from the Wav2Vec 2.0 models, which were optimized to maximize the balanced F-score. A better comparison can be found in Table 4.15, which again shows the FAR for the different DC models when the FRR is fixed at various values, this time including a FRR of 0.035, equal to that of the baseline. All models score a FAR below 0.839 at 0.035 FRR, indicating that they are an improvement on the baseline for the detection of phone-level errors.

Again, the PC models were tested with a threshold for the maximum allowable PER. The effects on the balanced F-score are shown in Figure 4.11. This time the model trained from scratch and the fine-tuned model peaked at a maximum PER of 0.333 and the balanced model peaked at 0.5. The effects of thresholds on the balance between FAR and FRR are plotted in Figure 4.12.

Model	**Accuracy**	**Precision**	**Recall**	**F-score**	**FRR**	**FAR**
WFST (Baseline)	0.928/0.553	0.960/0.523	0.965	0.963/0.679	0.035	0.839
PC (fine-tuned)	0.639/**0.752**	**0.990/0.821**	0.629	0.769/0.712	0.371	**0.131**
DC (fine-tuned)	0.815/0.720	0.979/0.675	0.825	0.895/0.742	0.175	0.380
DC_ordered (fine-tuned)	**0.820**/0.751	0.982/0.711	**0.827**	**0.898/0.764**	**0.173**	0.321
PC (from scratch)	0.649/0.753	**0.990/0.816**	0.638	0.776/0.716	0.362	**0.138**
DC (from scratch)	**0.808**/0.729	0.979/0.688	**0.816**	0.816/0.746	**0.184**	0.354
DC_ordered (from scratch)	0.805/**0.757**	0.983/0.725	0.810	**0.888/0.765**	0.190	0.293
PC (balanced)	0.573/0.726	**0.990/0.825**	0.558	0.714/0.665	0.442	**0.113**
DC (balanced)	0.827/0.705	0.977/0.654	**0.839**	0.863/0.735	**0.161**	0.423
DC_ordered (balanced)	**0.829/0.738**	0.980/0.691	0.838	**0.904/0.757**	0.162	0.359

Table 4.14: Results of Wav2Vec 2.0 model variations, using forced aligned boundaries. The accuracy, precision and F-score show balanced versions after the "/". The best results for each training method are shown in bold.

Model	**FRR = 0.2**	**FRR = 0.1**	**FRR = 0.05**	**FRR = 0.035**
WFST+r+s (baseline)	-	-	-	0.839
DC (fine-tuned)	0.343	0.536	0.726	0.798
DC_ordered (fine-tuned)	0.298	**0.483**	0.690	**0.758**
DC (from scratch)	0.335	0.542	0.730	0.811
DC_ordered (from scratch)	**0.284**	**0.483**	**0.688**	0.762
DC (balanced)	0.367	0.563	0.759	0.823
DC_ordered (balanced)	0.311	0.519	0.712	0.782

Table 4.15: The FAR for the different DC models at an FRR of 0.2, 0.1, 0.05 and 0.035. The best scores are shown in bold.

Two Wav2Vec 2.0 models were chosen to evaluate the hybrid model as a whole: the fine-tuned DC_ordered with the FRR fixed at 0.035 and DC_ordered trained from scratch with the FRR fixed at 0.05. The results

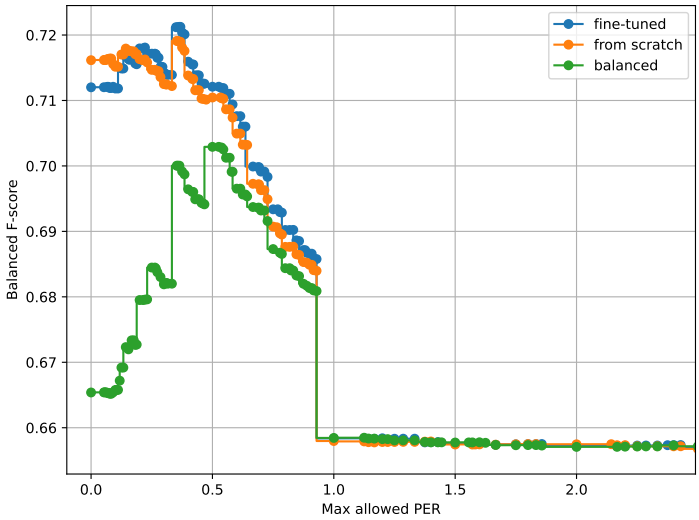


Figure 4.11: The balanced F-score as a function of the maximum allowed PER for the different PC models. The maxima are at (0.333, 0.720) (from scratch), (0.333, 0.721) (fine-tuned) and (0.500, 0.704) (balanced).

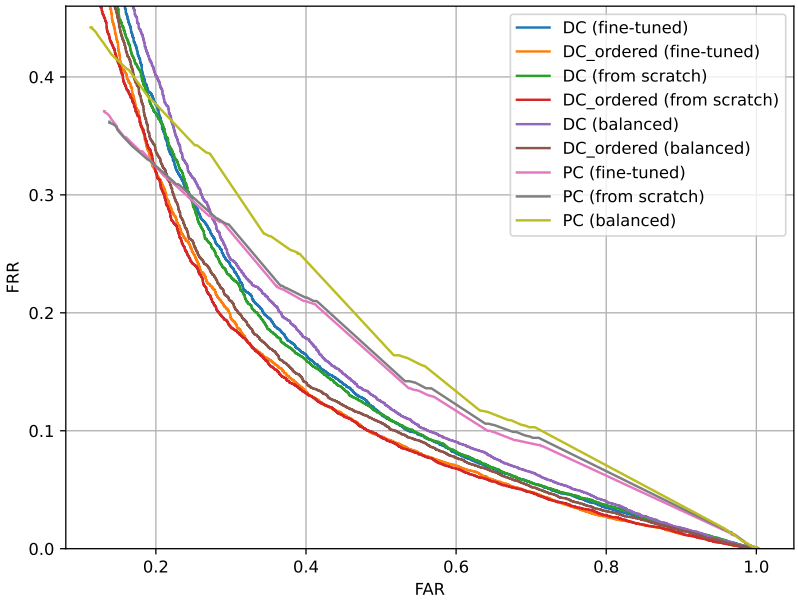


Figure 4.12: The trade-off between FAR and FRR for every model.

of these hybrid models are shown in Table 4.16.

Model	**WER**	**DER**	**Error%**	**Predicted Error%**	***R*-value**
WFST+r+s (baseline)	0.151	0.347	0.111	0.112	0.155
Hybrid 0.035	**0.141**	0.306	0.111	0.114	**0.183**
Hybrid 0.05	0.147	**0.297**	0.111	0.131	0.179

Table 4.16: Results of the hybrid models. Best results are shown in bold.

Chapter 5

Discussion

This chapter discusses the results outlined in the previous chapter. Section 5.1 discusses the target phone detection results outlined in section 4.1. Section 5.2 discusses the reading error detection results outlined in section 4.2.

5.1 Target Phone Detection

For the first part of this research, the main task was target phone detection (TPD). In TPD the goal is to predict whether a given target phone was pronounced correctly. This information can also be used to asses whether a word as a whole was pronounced correctly. This task is referred to as mispronunciation detection (MD). Two types of Wav2Vec 2.0 based models were built for these tasks: the phone classification (PC) model, which generates a phonetic transcription of the utterance, and the direct classification (DC) model, which assigns a score to every phone. Three types of pooling were used for the DC model: max pooling, mean pooling and adaptive max pooling. This resulted in three model variants, namely DC_max, DC_mean and DC_amax respectively. As a baseline, a DNN-HMM-based goodness of pronunciation (GOP) method named trans-GOP was implemented.

All models were tested with two different train-test splits of the CHOREC dataset: 1) The school-split where 6 out of 8 schools were used for training, one for validation and one for testing and 2) the word-split where 80% of the vocabulary was used for training and 10% each for validation and testing. Thresholds for TPD (and MD in the case of the DC models) were determined using the balanced F-score, which mimics the F-score for a test set where each phone was mispronounced exactly 50% of the times.

For reference, some of the results from the previous chapter are copied in Tables 5.1, 5.2 and 5.3.

5.1.1 Effectiveness of Wav2Vec 2.0 for Target Phone Detection and Mispronunciation Detection

From the school-split results it is clear that Wav2Vec 2.0 based methods can improve over more traditional hybrid model based models (or at least trans-GOP) in both MD and TPD. First looking at the TPD school-split, it seems that generally the PC model has better FRR and recall, while the DC models have better FAR and precision. When looking at balanced statistics, the number of true acceptances and false rejections decreases while the rest stays the same. As a result the precision will always increase and the accuracy will increase as long as the recall is over 0.5 (or the FRR is under 0.5). Furthermore, this increase gets larger,

Model	Accuracy	Precision	Recall	F-score	FRR	FAR
PC (school-split)	**0.968**/0.834	0.985/0.781	**0.979**	**0.982**/0.862	**0.021**	0.306
DC_max (school-split)	0.934/**0.890**	**0.995**/0.873	0.932	0.962/**0.895**	0.068	**0.155**
DC_mean (school-split)	0.933/0.867	0.992/0.843	0.935	0.962/0.879	0.065	0.201
DC_amax (school-split)	0.911/0.878	0.990/**0.860**	0.914	0.950/0.882	0.086	0.158
Trans-GOP (school-split)	0.785/0.746	0.979/0.743	0.789	0.871/0.757	0.211	0.294
PC (word-split)	0.843/0.837	**0.992**/0.882	0.843	0.896/0.836	0.157	**0.130**
DC_max (word-split)	**0.866**/**0.838**	0.989/0.853	**0.868**	**0.915**/0.843	**0.132**	0.161
DC_mean (word-split)	0.824/0.757	0.982/0.774	0.828	0.878/0.764	0.172	0.268
DC_amax (word-split)	0.843/0.836	0.991/**0.867**	0.842	0.902/**0.855**	0.158	0.140
Trans-GOP (word-split)	0.813/0.714	0.967/0.677	0.822	0.885/0.735	0.178	0.391

Table 5.1: TPD performance of the different models. All statistics are averaged over all phones. For accuracy, precision and F-score, the balanced version is shown after a '/'. The best scores for each split are shown in bold. Copy of Table 4.3.

Model	Accuracy	Precision	Recall	F-score	FRR	FAR
PC (school-split)	0.925	0.943	0.940	0.941	0.060	**0.102**
PC* (school-split)	**0.934**	0.969	**0.956**	**0.963**	**0.044**	0.237
DC_max (school-split)	0.891	**0.980**	0.895	0.940	0.105	0.144
DC_mean (school-split)	0.915	0.978	0.927	0.951	0.073	0.173
DC_amax (school-split)	0.865	0.975	0.870	0.920	0.130	0.178
Trans-GOP (school-split)	0.757	0.787	0.815	0.800	0.185	0.328
PC (word-split)	0.620	0.927	0.344	0.502	0.656	**0.034**
PC* (word-split)	0.614	**0.976**	0.560	0.712	0.440	0.078
DC_max (word-split)	0.796	0.944	0.808	0.871	0.192	0.273
DC_mean (word-split)	0.776	0.902	0.826	0.862	0.174	0.513
DC_amax (word-split)	**0.825**	0.947	0.841	**0.890**	0.159	0.265
Trans-GOP (word-split)	0.778	0.747	**0.852**	0.796	**0.148**	0.300

Table 5.2: MD performance of the different models. The best scores for each split are shown in bold. Copy of Table 4.4.

the higher the recall. This explains why the difference between regular and balanced precision is larger for the PC model than the DC models and also why the F-score is better for the PC model, but the balanced F-score is worse.

For the MD school-split the same conclusions hold. To fairly compare the PC and DC model it is better to look at the PC* model since these models have the same task. The task of the trans-GOP model is slightly different from both models, since it assigns scores to each pronunciation variant. However, from analyzing the results it became apparent that the trans-GOP model gives low scores to all force aligned phones when the target word is pronounced correctly but there are utterances before or after the word. Therefore the decision was made to count an annotated label as correct only when it exactly matched one of the canonical pronunciation variants. This truth criterion makes the trans-GOP model more comparable to the PC model than the PC* and DC model.

Model	Accuracy	Precision	F-score
PC* (school-split)	0.853	0.781	0.860
DC_max (school-split)	**0.874**	**0.846**	0.870
DC_mean (school-split)	**0.874**	0.826	**0.873**
DC_amax (school-split)	0.845	0.813	0.841
PC* (word-split)	0.755	**0.849**	0.678
DC_max (word-split)	0.764	0.716	0.759
DC_mean (word-split)	0.643	0.578	0.680
DC_amax (word-split)	**0.783**	0.729	**0.781**

Table 5.3: Balanced accuracy, precision and F-score for MD. The best scores for each split are shown in bold. Copy of Table 4.5.

Comparing the PC to the PC* model shows that the PC* model scores higher on all metrics except for the FAR. This indicates that the PC model in general is good at predicting the data points that are labeled as incorrect with the PC truth criterion and correct with the PC* criterion, so utterances that contain the canonical pronunciation but with utterances before or after. This mostly consists of utterances with hesitations and decoding.

When comparing the word-split to the school-split results, the DC models score worse on all metrics and the PC model scores worse on all metrics except the FAR which improves significantly. On the other hand, the results of the trans-GOP model are less affected. When analyzing the errors of the PC model in the word-split, it is found that many errors occur when a word in the test set is similar to a word in the train set and the model predicts the exact word from the train set. For example the test set contains the pseudo-word "sooi" which is canonically pronounced as "/s o j/". The train set has the pseudo-word "zwoei" which is pronounced as "/z w u j/". According to the annotated labels, the word "sooi" is pronounced correctly 308 times in the test set. Out of these instances, 142 times the phone sequence "/z w u j/" was the exact prediction and 291 times the first phone was predicted to be a "/z/" as opposed to merely 8 times a "/s/". This clearly highlights one of the major flaws of the PC model, namely that it overfits with respect to the vocabulary it was trained on. This explains why the model performs worse overall when the word-split is used. It also explains why the FAR improves for the PC models, since predictions in general are more likely to result in a rejection. This difference between the school-split and word-split version of the PC model is especially noticeable in the MD results. The reason that the difference is smaller in TPD is because the scores are averaged over all phones. In Figure 4.2b it can be seen however, that the word-split has a few phones that score significantly worse than the median. These are mostly phones like "/s/", which are in words from the test set that are similar to words in the train set, with these phones being the main difference.

The DC models also achieve worse overall results and more per phone variance when the word-split is used, indicating that these models also overfit to some extent. However, the FAR does not improve for these models. This is likely due to the fact that the threshold was set for the school-split and word-split separately in order to maximize the balanced F-score. For all three model variants, the threshold was set significantly lower for the word-split, which results in less rejections. These lower thresholds might indicate that the models on average "less confidence" in their predictions in the word-split, however I want to be careful when comparing the absolute threshold values of different models. That is because these absolute scores of different models also vary due to different loss functions and stochastic elements of training, and in the end the only relevant aspect for predictions is the relative scores that one given model assigns to different phones.

Comparing the different DC models to each other, the DC_mean model achieves significantly worse results with the word-split than DC_max and DC_amax. This is due to the pooling strategy used. Mean pooling encourages the model to assign high scores to the pronounced phones at every frame, whereas max pooling allows pronounced phones to get low scores at some frames and adaptive max pooling forces the model to give lower scores to all but the pronounced phone at each frame. The latter causes the context-sensitive embeddings to mostly carry information about the phones pronounced at that frame, most likely only using a small range of surrounding frames as context. With mean pooling on the other hand, the context-sensitive embeddings need context from all other frames, in order to carry information about all phones used. In doing so, the model might inadvertently learn to predict what word was said and consequently give high scores to all phones that are part of that word, instead of predicting the phones directly. This obviously makes the model more sensitive to the vocabulary it was trained on.

5.1.2 Balanced Training

The discrepancy between the school-split and the word-split, could partially be caused by the fact that the majority of the data is correctly pronounced words. This encourages the models to predict what word a student is attempting to pronounce and then predict phone sequence corresponding to the target word. Since most words are correctly pronounced, this will usually give a perfect score. For this reason the models were retrained with a subset of the original training set, such that half of the words contained one pronunciation error. The results are copied from the previous chapter in Tables 5.4 and 5.5.

Model	**TPD F-score**		**MD F-score**	
Training set	**Full**	**Balanced**	**Full**	**Balanced**
PC (school-split)	0.982/0.862	0.970/0.883	**0.941**	0.906
PC* (school-split)	-	-	**0.963/0.860**	0.921/0.858
DC_max (school-split)	0.962/0.895	0.956/0.898	**0.940/0.870**	0.918/0.862
DC_mean (school-split)	**0.962/0.879**	0.935/0.877	**0.951/0.873**	0.924/0.857
DC_amax (school-split)	0.950/0.882	**0.964/0.888**	0.920/0.841	**0.921/0.858**
PC (word-split)	0.896/0.836	**0.916/0.861**	0.502	**0.573**
PC* (word-split)	-	-	0.712/0.678	**0.725/0.695**
DC_max (word-split)	0.915/0.843	0.836/0.870	0.871/0.759	0.852/0.797
DC_mean (word-split)	0.878/0.764	**0.913/0.779**	**0.862/0.680**	0.837/0.669
DC_amax (word-split)	0.902/0.855	0.944/0.849	0.890/0.781	0.872/0.791

Table 5.4: This table compares the original results to the results from the models train with the balanced training set. Balanced F-scores are shown after a '/'. Thresholds for the DC models were set to optimize the balanced F-score. PC* does not show TPD stats, since these would be the same as the PC model. For each model, split and task the results of better performing training set are shown in bold, as long as both the regular and balanced F-score are improved upon. Copy of Table 4.6.

The main finding of interest is that the results for the PC model in the word-split improved, as hypothesized. Also, if we again look at the predictions for the word "sooi", we see that 187 times the first phone was predicted to be a "/z/" and only 28 times the sequence "/z w u j/" was predicted (as opposed to 291 and 142 times respectively, when using the full training set). This shows that using a balanced training set did mitigate the "target word learning" behavior for the PC model.

That being said, for no other model did all results clearly improve. For DC_amax in the school-split, all

Model	TPD FAR		MD FAR	
Training set	**Full**	**Balanced**	**Full**	**Balanced**
DC_max (school-split)	**0.190**	0.200	**0.243**	0.358
DC_mean (school-split)	**0.196**	0.251	**0.236**	0.296
DC_amax (school-split)	0.251	**0.215**	**0.348**	0.354
DC_max (word-split)	0.325	0.325	**0.538**	0.544
DC_mean (word-split)	**0.529**	0.554	**0.799**	0.851
DC_amax (word-split)	**0.309**	0.323	**0.493**	0.541

Table 5.5: This table compares the FAR when the FRR is fixed at 0.05 for both training sets. For each model and split the results of better performing training set are shown in bold. Copy of Table 4.7.

results improved with the exception of the MD FAR in Table 5.5. The fact that the TPD FAR improved but the MD FAR did not, indicates that there are a few outlier phones, which achieve significantly worse scores. In fact for many models the difference between the scores for MD is more exaggerated than for TPD, for the same reason. These outliers are likely phones for which there already was not much data available in the full training set.

It is also interesting that DC_max scores a better balanced F-score for TPD in both the school-split and the word-split. Since the threshold was set to optimize the balanced F-score, it could be argued that DC_max also performs better with a balanced training set. The fact that the balanced F-score is better but the regular F-score is worse indicates that the models perform better on incorrectly pronounced phones but worse on correctly pronounced phones. This is also likely the case for all models since the difference between balanced F-scores is often smaller than the difference between regular F-scores. This finding is of course logical, considering that incorrectly pronounced phones make up a much larger portion of the training set.

Of course, the balanced training set contains on average only a quarter of the data from the full training set, so if the use of a balanced training set has no effect, the expectation would be that the results deteriorate since less data is used. This makes it somewhat difficult to interpret the results. It is possible that the effects found, would be more pronounced if the balanced and regular training set contained the same amount of data. For future research, when collecting data for TPD training, it is important to pay attention to the relative amount of pronunciation mistakes in the dataset. When working with a skewed dataset, one could try to prune the correctly pronounced words, especially when using a PC model with unknown vocabulary or when the performance on incorrectly pronounced words needs improving.

5.1.3 Student Variance

Figure 4.3 shows how the F-score varies between students. It is important to analyze this variance, since if a model consistently performs worse for certain students, this could be an issue for fairness.

In the school-split only the trans-GOP model shows a relatively large variance, although the PC model has a few outliers with lower F-scores. In the word-split, all models show larger variance, however the trans-GOP, PC and PC* models vary more than the others. It should be noted that a higher variance for the word-split is expected to some degree, because the word-split test set contains both more students and less data points per student.

A linear regression was performed to test whether the variance in F-score can be explained by a student's performance. The results are shown in Table 5.6. For all models except for PC and PC* in the word-split, there was a positive correlation between the F-score and the percentage of words that was correctly

pronounced. This indicates that all these models perform worse when students make more errors. This is of course related to the fact that these models have a higher FAR than FRR. The more interesting result is that the PC and PC* models do not show a significant negative correlation, as would be expected from them having much lower FAR than FRR. This means that other factors are the cause of the large variance for these models.

Model	R**-value**	**P-value**
PC (school-split)	**0.563**	$\mathbf{8 \times 10^{-6}}$
PC* (school-split)	**0.578**	$\mathbf{4 \times 10^{-6}}$
DC_max (school-split)	**0.302**	**0.025**
DC_mean (school-split)	**0.656**	$\mathbf{6 \times 10^{-8}}$
DC_amax (school-split)	**0.475**	**0.0002**
trans-GOP (school-split)	**0.712**	$\mathbf{1 \times 10^{-9}}$
PC (word-split)	0.000	0.973
PC* (word-split)	0.079	0.198
DC_max (word-split)	**0.279**	$\mathbf{4 \times 10^{-6}}$
DC_mean (word-split)	**0.400**	$\mathbf{1 \times 10^{-11}}$
DC_amax (word-split)	**0.196**	**0.001**
trans-GOP (word-split)	**0.451**	$\mathbf{8 \times 10^{-15}}$

Table 5.6: Results of linear regression between the F-score and correctly pronounced words (as a percentage of total words). Statistically significant results are shown in bold ($p < 0.05$).

When analyzing to the recordings of the lowest outliers, nothing in particular stood out about the audio or annotations. It was found that for the PC models in the word-split, the students with the lowest F-scores always had a very high FRR but near perfect FAR, which is expected when you look at the overall results for these models. For all models a t-test was done to test whether the mean F-score was significantly different for different genders, but no significant results were found.

These tests are somewhat superficial when it comes to determining the fairness level of these models. No data was available to test for ethnicity or accent. We can conclude however that the PC model trained with the word-split will get significantly more false rejections for some students than others. Furthermore, the other models do perform slightly worse for students that make more pronunciation errors, which is something potential future users should keep in mind.

5.1.4 Limitations and Future Research

There are two main conclusions from this first research: 1) Wav2Vec 2.0 models can improve over the trans-GOP baseline for TPD and MD and 2) Wav2Vec 2.0 models overfit with respect to the vocabulary they were trained on. This second conclusion holds especially true for the PC model, however this is mostly due to the lack of a threshold. It can be seen in Figure 4.1 that the PC* performs similarly to the DC models at the same FRR values for both the school-split and the word-split. The difference between splits is also more noticeable for MD than for TPD. That is because the majority of errors in the word-split are caused by erroneous predictions of a small amount of outlier phones.

One limitation of this research is the relatively small number of pronunciation errors made in the CHOREC dataset. This was a problem for evaluation and choosing thresholds, since the percentage of errors directly affects the F-score. The results on the balanced training set suggest that it might also have

affected training. When the majority of the words in the training set are correctly pronounced, this enables the model to get good results by always predicting the correct words, rather than the pronounced phones. This is likely one of the reasons that the models overfit with respect to the training vocabulary. For future research, it might be beneficial to ensure that the training data has a sufficient relative number of pronunciation errors, as well as a more varied vocabulary. On the other hand, the difference in results of the school-split and the word-split shows that the models perform better on the vocabulary they are trained on. So alternatively, one could try to ensure that the vocabulary of the training data will match with that of the final application and accept that the models will be dependent on the training vocabulary. In this case, it could be interesting to see how much data the models need to learn a vocabulary, since many studies have shown that Wav2Vec 2.0 can achieve good results with few-shot learning [6, 19, 64].

Another issue is the different truth criteria for the annotated labels in MD. This makes it harder to compare the DC models to the baseline. For example in the word-split, the F-score for the trans-GOP model is lower than the DC models, however the FAR and FRR scores seem comparable to the best models as can be seen in Figure 4.1b. It is difficult to draw any conclusions from these results, since the models are technically predicting different things.

The difference in truth criteria also means that the PC model has more predictive power than the DC models. The PC model can predict whether the phones are in correct order and whether any phones are repeated. This is not the case for the DC model in its current implementation. However, the DC model might hold information about phone order. Specifically, the DC_amax model is trained to predict the phones in the correct order. It is possible that DC_max uses a similar strategy. For the current research the order of the phones is less important. For TPD the order of phones is not relevant, and the most common order-related errors in MD are decoding errors and hesitations, which in future applications could also be detected by different models like WFSTs (as is done in the reading error detection models from sections 4.2 and 5.2). However, for future research it is an option to implement the DC models for MD with the additional condition that the phones have to be in the correct order. A simple implementation would be to search the output frames from left to right until the first phone of the canonical pronunciation scores above a pre-determined threshold and then search for the next phone in the canonical pronunciation in the remaining frames. If all phones score above the threshold the model predicts the word to be correctly pronounced. This implementation is similar to trans-GOP in that it tries to match the input directly to a canonical pronunciation. However unlike the trans-GOP model, it would not be sensitive to hesitations or decoding since it does not try to exactly force align the phones to specific frames. This version of the DC model was also implemented into the reading error detection models of sections 4.2 and 5.2 as DC_ordered.

An alternative direction could be to alter the PC model such that it can also use a threshold. Thresholds give users more control and it can help mitigate the effects of overfitting as indicated by the differences between the results of the PC and DC model in the word-split. A simple option would be to set a threshold on the phone error rate (PER) and accept word attempts if the PER between the canonical and predicted pronunciation falls below that threshold (again this was later tested in sections 4.2 and 5.2). Alternatively in future work, the PC model could be altered such that it no longer outputs the most likely string of phones but instead assigns a score to a the canonical string, for example by summing the score of all sequences of phones and padding tokens that would collapse into this string, similarly to how this happens during training with the Connectionist Temporal Classification algorithm (CTC).

5.2 Reading Error Detection

The second task in this research was reading error detection (RED). For this task the goal was to detect various types of reading errors made by children reading texts out loud in the Malmberg dataset. Errors

could be made on the word-level or on the phone-level. Word-level errors are errors where words from the prompt are either skipped (omissions) or added (insertions). Phone-level errors are errors where words from the prompt are pronounced wrong (substitutions). Initially all error classes were detected using a baseline WFST models (section 5.2.1). Then Wav2Vec 2.0 models were used to detect phone-level errors and re-implemented into the WFST baseline as a hybrid model (section 5.2.2).

5.2.1 Weighted Finite State Transducer (WFST) Models

First various models were built based on weighted finite state transducers (WFST). These models are able to detect both word-level and phone-level errors. The different variations are based on which types of errors it distinguishes. The simplest WFST can only predict the classes "omission", "insertion" and "correct". Substitutions are derived after the initial prediction, by combining insertions and omissions. Since each type of error has its own weight assigned to it, this means that the weight for substitutions in this model is equal to $w_o \times w_i$. To give substitutions their own weight, some WFSTs (which are marked with "+s") explicitly incorporate a combination of an insertion plus an omission into their structure. Furthermore, some models (marked with "+r") allow repetitions to be recognized, which are insertions from words earlier in the prompt. Finally, some WFSTs (marked with "+d") process the prompt in syllables, which allows for the recognition of false starts and decoding. The former occurs when a student starts reading a word, stops halfway and then starts over and reads the full word (e.g. "gro- grote"). The latter occurs when a student reads a word in sections (either phones or syllables) with pauses in between (e.g. "gro- te" or "g-r-o-t-e").

5.2.1.1 Analysis of Baseline WFST Results

The results from the previous chapter are copied in Table 5.7 for reference. The main metric is the word error rate (WER) which gives an indication of how many errors the model makes; either when an error is predicted to be correct or a correct utterance is predicted to be an error. The best performing model is WFST+r+s which has a WER of 0.151, which basically means that for 15.1% of all labeled utterances a mistake is made by the model. This is too large of a margin of error for any practical application. Additionally, the Malmberg dataset is very skewed, with only a small percentage of the utterances being an error while the rest is correctly pronounced words from the prompt. Therefore, a baseline that predicts every text to be completely correct scores a WER of 0.089, which outperforms all models. While this is obviously another indication that the models would have no practical use for this dataset, it is not sufficient to say that the models hold no merit at all. It is not uncommon for a simple baseline to perform excessively well when working with skewed data. Because the majority of utterances is correct, the WER for this dataset is mostly affected by a model's recall and does not give as good of an indication of a model's ability to correctly find reading errors.

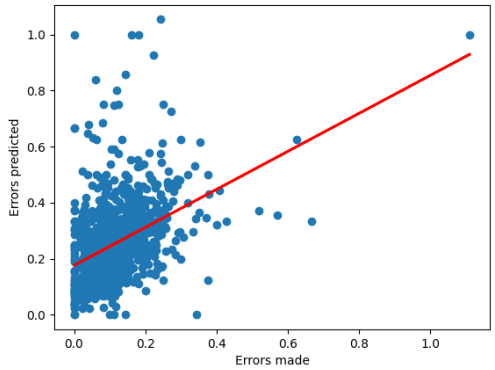
Table 5.7 also shows the diagnosis error rate (DER), which indicates how often the correct error class is predicted in cases where there is a true rejection. In order to line up the words of the annotation with those of the prediction, insertions, repetitions and false starts are not considered. The lowest DER scored is 0.325, which is too high to reliably diagnose what error is made. This result is both expected and not very relevant, given that the models could already not reliably distinguish errors from correct utterances in general.

The R-value in Table 5.7 shows the strength of the relation (as calculated by a linear regression) between the number of errors made by the student and the number of errors predicted by the model as a percentage of the number of the words in the prompt. Based on the assumption that the number of errors a student makes is indicative of their reading level, this metric can be used as an indication of the model's capability to asses the reading level of a student. Again for none of the models, a strong relationship was found. This means that the models are not suited for estimating how many errors a student makes while reading, and

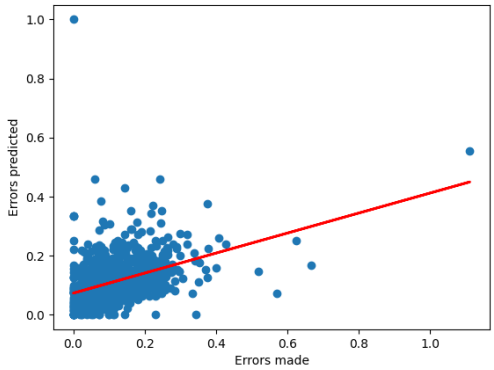
Model	WER	WER*	DER	Error%	Predicted Error%	R-value
WFST-simple	0.177	0.143	0.391	0.111	0.147	0.097
WFST+r	0.177	0.145	0.401	0.111	0.149	0.110
WFST+s	0.152	0.098	0.349	0.111	0.112	0.131
WFST+r+s	**0.151**	**0.097**	0.347	0.111	0.112	0.155
WFST+d	0.180	0.147	0.392	0.111	0.146	0.109
WFST+d+r	0.178	0.146	0.406	0.111	0.146	0.127
WFST+d+s	0.154	0.099	0.341	0.111	0.111	0.135
WFST+d+r+s	0.152	0.099	**0.325**	0.111	0.111	**0.160**

Table 5.7: WFST results after 20 iterations of training.

by assumption not suited for assessing a student's reading level. The graph of the progression of the R-value over the training iterations (shown in Figure 4.8b) shows that, for the models WFST+r and WFST+d+r, the R-value was at its highest at iteration 0, even though these models predicted over twice as many errors as were actually in the dataset. In fact many for many models the R-value dipped after iteration 0 and then climbed back up. This means that, while early on these models predicted too many errors in general, they did consistently predict more errors when more errors were made. To illustrate this, the scatter plots in Figure 5.1 show the plots for the highest R-value at 0 iterations (WFST+r) and at 20 iterations (WFST+d+r+s). The plot of the model at iteration 0 on the left has significantly more outliers with a high amount of predicted errors than the plot on the right, however many of these outliers do have a ratio of errors made that is above the average of 0.111.



(a) Scatter plot of WFST+r at iteration 0 (R-value=0.175).



(b) Scatter plot of WFST+d+r+s at iteration 20 (R-value=0.160).

Figure 5.1: Scatter plots for the predicted number of errors versus the true number of errors made, as a ratio of number of words in the prompt. The red line is the best fitted line corresponding to the linear regression.

Overall, if we look at the differences between the models, we can conclude that the inclusion of +s leads to significant improvements in the results in all metrics. This is most likely a consequence of the training method. For the +s models, the weights for omissions (w_o) for example are adjusted at each iteration in

proportion to the number of omissions in the dataset over the number of omissions predicted:

$$w_o \textit{ adjustment factor} = \frac{\textit{omissions in the dataset}}{\textit{predicted omissions}} \tag{5.1}$$

Equivalent calculations are used for substitutions and insertions. However, models that do not have a dedicated weight for substitutions, can only predict substitutions as a combination of an insertion and an omission. Therefore the weight for omission is adjusted as follows:

$$w_o \textit{ adjustment factor} = \frac{\textit{omissions in the dataset} + \textit{substitutions in the dataset}}{\textit{predicted omissions} + \textit{predicted substitutions}} \tag{5.2}$$

Again an equivalent calculation was used for the insertion weight.

The idea behind this approach was that the non-+s models would have a higher likelihood to output omissions and insertions, which would then contract into substitutions. However, as the likelihood of predicting omissions and insertions decreases over iterations, the likelihood of predicting substitutions decreased significantly faster. After 20 iterations, the number of predicted substitutions is around 10% of the true number, while the number of predicted omissions and insertions is significantly higher than the true number. The +s models avoid this problem by having a dedicated substitution weight and therefore the number of omissions, insertions and substitutions ends up being approximately the same to the true number.

The inclusion of +d or +r does not seem to make a significant difference in the overall statistics, but that is also due to the fact that these errors make up only a small part of the dataset (see Table 5.8). The performance of the models on individual error types is discussed further in section 5.2.1.3.

5.2.1.2 Outlier Analysis

Figure 5.1 shows that there are many outliers in these results, mostly when too many errors are predicted. This is even more evident when looking at the spread of the WER per sample, which is plotted on the left in Figure 5.2. The best performing model WFST+r+s scored a WER of 0.151. The mean WER for this model is 0.158 and the standard deviation is 0.142, meaning that almost all WER values from zero to double the mean all fall within one standard deviation. Listening to some of these highest WER outliers reveals that many occur when students make a few mistakes early on (usually either omissions or insertions), which are misinterpreted by the model. When the model misses omissions or insertions, it ends up in the wrong state for predicting the subsequent words and it inevitably needs to make more errors to make sense of the rest of the sentence.

Many of these examples are related to the title of the prompts. Every text has a title which is sometimes read out loud by the student and sometimes not. Unfortunately, the annotators were not consistent on whether or not this is a mistake. Prompts used in the Malmberg dataset were not available separately, but were instead extracted from the annotations, so if the student reads the title and the annotator marks it as insertions, the prompt will not include the title and therefore the WFST will not expect it. Conversely, if the student does not read the title and the annotator marks it as omissions, the WFST will exect a title. Because of the low error weights, the model does not like predicting many insertions or omissions in a row, especially not if words in the title match with or closely resemble words in the first sentence. This is a side effect of beam search decoding. Because of the weights, generally errors get a lower score than correct words, so the optimal path through the WFST does not get a good score until the first correct word is said. The longer it takes for the first correct word to appear, the more likely the correct path falls outside of the beam. In order to predict many errors in a row at the start of the text, the model essentially needs to sacrifice a good score early on to achieve a higher score later and since beam search decoding is a breadth-first search, this is unlikely to happen.

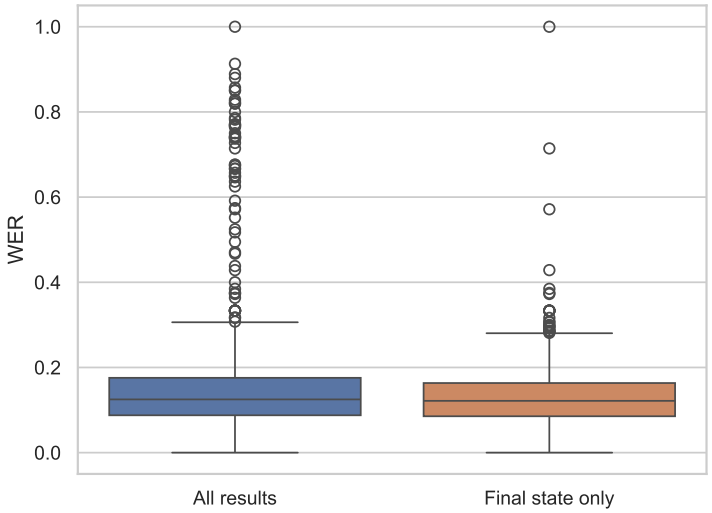


Figure 5.2: Boxplots showing the spread of the WER for the best performing model (WFST+r+s). The left plot shows the full results and the right shows only the results for which the final state was reached.

One potential solution for this problem would be to change the WFST topology so that repeated omissions or insertions all have the same weight as a singular omission or insertion. For omissions this could lead to a different issue though, because the model would then have a cheap way to predict any word in the sentence from any node. If an utterance is erroneously classified as a word further ahead in the sentence, this would likely cause the model to make many other errors, since it would now be expecting the wrong words.

Another finding was that for many of the outliers, decoding terminated too early. In theory, decoding with a WFST should only terminate when the final state is reached, which in the case of the WFSTs in this research can only happen after every word in the prompt has been classified. However, for around 7% of the data decoding ended before the final state was reached. This happens because of a parameter called the minimum activation. This represents the minimum score that a path through the WFST must have for it to be considered during decoding and it is meant to make decoding more efficient. When no remaining paths score above the minimum activation threshold, decoding is terminated early. When this happens, the model outputs the best scoring path up to that point and no prediction is made for the remaining words in the sentence, which is equivalent to predicting the remaining words to be omissions for the WER calculation.

Decoding in this research was done with the default minimum activation used in Kaldi. Decreasing the minimum activation would prevent termination before reaching the final state, however it would also decrease the efficiency of decoding. Furthermore, early termination only happens when the model deems the audio to not be a close enough match to any possible path through the WFST. As far as I have seen, this only occurs when either the audio is distorted, or the student's utterance deviates significantly from the prompt (with multiple severe errors in a row) and the partial prediction for that text usually already contained a lot of errors. Therefore, it would be better for practical implementations to leave the minimum activation as is, and instead ask the student to read again when the final state cannot be reached during decoding. If we ignore the data points for which the final state was not reached, this only significantly impacts the WER,

which drops by around 2.5% for every model. For the best performing model WFST+r+s (the right plot in Figure 5.2) the WER drops from 0.151 to 0.126, the mean WER drops from 0.158 to 0.131 and the standard deviation drops from 0.142 to 0.075. It should be noted that, while the results are significantly better for data points that reach the final state, they still cannot improve the basic baseline that predicts every word to be correct.

An alternative strategy towards preventing early termination of decoding is to increase the beam size. A larger beam size means that more paths will be considered and thus it is more likely for a path with initially low scores to stay in the beam until the score increases. To test this the data was decoded with WFST+r+s with an increased beam size from 15 to 30. With a beam size of 15, 114 data points did not reach the final state and with a beam size of 30, this number was reduced to 80. The WER dropped from 0.151 to 0.134. Listening to the audio of these 114 files revealed that the higher beam size mostly helped decode files were the title was skipped and the first sentence was not similar to the title. The increased beam-size did not help if the title was skipped and words in the first sentence matched those in the title or if the final state was not reached due to excessive noise. Furthermore, the WER of the files already that reached the final state with a beam size of 15 stayed the same at 0.126. This indicates that increasing the beam size has no additional benefits besides decoding these specific types of data points, while it does exponentially decrease the speed and efficiency of decoding. Future users should determine for themselves whether this trade-off is worth it.

5.2.1.3 Deeper Analysis of Individual Error Types

Next, we try to get an indication of how the models perform on each individual error type. Therefore, we examine the results WFST+d+r+s at iteration 0 and iteration 20 of training. It is useful to first examine the total error class counts, since these counts were responsible for determining the change in model weights during training. The error class counts are listed in Table 5.8.

	Correct	**Substitution**	**Omission**	**Decoding**	**Repetition**	**Insertion**	**False Start**
True	61853	2292	879	154	1005	2232	342
$i = 0$	58088	2453	3302	152	4570	4405	652
$i = 20$	59912	2374	860	154	994	2275	361

Table 5.8: Error class counts for WFST+d+r+s at iteration 0 and iteration 20 of training compared to true counts.

It is difficult to say exactly what percentage of the errors are false accepts and what percentage are false rejections. However, we can at least see how often the error classes "correct", "omission", "substitution" and "decoding" are confused with one another, using the same word alignment strategy used to derive the DER (described in section 4.2.2.2). The balance between the FAR and FRR for these classes varies over iterations, as can be seen in Figure 4.8d. Focusing specifically on WFST+d+r+s, from the start to the end of training the FAR for this model increased from 0.677 to 0.761, while the FRR decreased from 0.063 to 0.037. Table 5.9 shows how much each error type contributes to the FAR and FRR for the model WFST+d+r+s at the start and at the end of training. To get a better insight of the FAR, we look at the percentage of false acceptances (FA) and the class-specific recall. The latter is the recall when looking only at that error class, so for example the number of substitutions correctly predicted over the total number of substitutions in the dataset. From the class-specific recall, we can conclude that the model is significantly worse at finding substitution and decoding errors than it is at finding omissions. Because substitutions are over twice as common in the dataset as omissions and almost 15 times as common as decoding errors, this means that the majority of the FAR is attributed to the false acceptance of substitutions.

It is to be expected that substitutions and decoding are easier for the model to miss than omissions. Many substitution errors are phonetically similar to the correct word (e.g. "takje" vs "takjes", "gewichtloosheid" vs "gewichtsloosheid"). In these cases, utterances will of course still get a relatively high score from the acoustic model and since the grammar encourages correct words to be recognized over substitutions, it is very unlikely that the model will catch these errors. In the case of decoding, the phone sequence often exactly matches that of the correct word and instead the accuracy depends on whether the model can detect pauses between the syllables. However both the since the phones match exactly the non-decoded version of the word still gets a good score from the model and since decoding has a lower weight than the correct word in the grammar, decoding errors are also more likely to be missed. Omissions are the only error that do not have this problem where the erroneous utterance (or rather lack of an utterance) is often significantly different from the correct pronunciation and therefore will consistently receive a lower score from the acoustic model. It is also unsurprising that most of the false acceptances of omissions are for omissions of short words like articles, since these are the most "phonetically similar" to silence. These words also tend to be less clearly pronounced by a fast reading student.

Furthermore, we can conclude that the increase in FAR during training is exclusively caused by an increase in falsely accepted omissions, since the class-specific recall of the other errors only increases during training. This is most likely due to the fact that at iteration 0, the model predicted over three times as many omissions as were actually in the dataset, while for substitutions and decoding these numbers were approximately equal (as can be seen in Table 5.8). Because of this, the weight for omissions in the final model is significantly lower which leads to more false acceptances of omissions.

To get a better insight into the FRR, we look at the class-specific precision. This is calculated as the number of true rejections for a class, divided by the total number of times that the class was predicted by the model. It should be noted that this precision will rarely be high in a skewed dataset, since there is more opportunity for the model to make false rejections than true rejections. We see that the precision improves for all error classes during training, albeit only a by a small amount for substitutions. The biggest improvement comes from omissions, which now only make up 24.5% of the FRs instead of 54.0%. This means that mostly the omissions are responsible for the total drop in FRR. This is again likely caused by the fact that the weight for omissions was simply lowered the most during training.

	Substitutions	**Omissions**	**Decoding**
Class-specific Recall at $i = 0$	0.103	0.639	0.109
Class-specific Recall at $i = 20$	0.113	0.341	0.154
% of FAs at $i = 0$	87.5	7.1	5.4
% of FAs at $i = 20$	82.4	12.3	5.3
Class-specific Precision at $i = 0$	0.119	0.147	0.117
Class-specific Precision at $i = 20$	0.134	0.274	0.207
% of FRs at $i = 0$	43.4	54.0	2.6
% of FRs at $i = 20$	71.8	24.5	3.7

Table 5.9: This table shows how many times each error was falsely accepted (FA) and how many times each error was falsely recognized instead of a correct utterance (FR) for the model WFST+d+r+s after 0 iterations and after 20 iterations of training.

To analyse repetitions, we count a word as a true rejection if it is marked as a repetition in both the prediction and the annotation. A true acceptance then occurs for every word that is not marked as a repetition in either the annotation or the prediction. The same can be done for false starts. With this the

FAR and FRR can be calculated, shown in Table 5.10. We see that the FAR and FRR improve for both error types. It is interesting that even the FAR dropped even though in general the odds of acceptance became much lower during training. This is most likely due to the fact that confusion between errors decreased as the weights for other errors got lowered. For example, repetitions that were erroneously classified as insertions at iteration 0, are correctly classified at iteration 20 because the weight for insertions decreased.

	Repetitions	**False Start**
FAR at $i = 0$	0.831	0.725
FAR at $i = 20$	0.761	0.605
FRR at $i = 0$	0.045	0.006
FRR at $i = 20$	0.011	0.003

Table 5.10: FAR and FRR of repetitions and false starts for WFST+d+r+s at iteration 0 and 20.

Insertions are more difficult to analyse. Since they are not linked to words in the prompt, it is difficult to asses whether an insertion was predicted in the correct location. Initially, it was tested whether this could be achieved by coupling insertions to the spaces between the words of the prompt. However, in many cases it was too difficult to determine between which two words to check, mostly due to annotated and (erroneously) predicted omissions and repetitions. Instead, we can compare the number of predicted insertions to the number of insertions in the annotation. Even if these numbers are equal, this does not guarantee that the insertions are placed in the same location in the sentence, however the analysis still gives some indication of the model's performance related to insertions. The analysis is shown in Table 5.11. It shows that the insertion predictions did get better during training, as indicated by both the mean differences and the regression results. However, even at iteration 20 the model will still make on average approximately one insertion-related error every 33 words, or between 1 and 2 per prompt. Also, the regression results show no strong relations, especially when you take away the length of the prompt as a confounding variable. And again this analysis does not account for errors where insertions are predicted at the wrong location in the sentence.

		$\lvert\textbf{pred} - \textbf{true}\rvert$	$\textbf{pred} - \textbf{true}$	R-**value**
$i = 0$	#	2.75	2.03	0.151
	%	0.050	0.037	0.047
$i = 20$	#	1.57	-0.056	0.187
	%	0.029	4.77e−4	0.056

Table 5.11: This table shows both the absolute and regular mean differences between the number of insertions predicted and the true annotated number of insertions. This is done both as absolute numbers (#) and as a percentage of the number of words in the prompt (%). It also shows R-values as calculated by a linear regression.

Overall we see that for all error types the model performs poorly, specifically when it comes to false acceptances. Because the weights are based on the number of occurrences of each error in the dataset, the weights for errors are set very low and in doing so the model automatically sacrifices its accuracy on errors to maximize its accuracy on correctly pronounced words, which make up the majority of the dataset. There are a few reasons why low error weights lead to worse results. You could compare the error weights to a threshold like that of the DC model. Take for example the weight for substitutions. As the substitution weight gets lowered, fewer utterances will get rejected as substitutions and instead accepted as correct,

starting with the ones the model has the least confidence in that they are substitutions. Ideally, the model has less confidence in correct words than substitutions, so as the weights gets lowered this will mostly result in more true acceptances, rather than false acceptances. However in practice, there are always a certain amount of correct words that the model is very confident are substitutions. This could happen for a variety of reasons, for example annotation errors, unorthodox pronunciations, or simply imperfections in the acoustic model leading to certain phones getting lower scores. Because of this, we see that more and more of the new acceptances are false acceptances, as we lower the weight. As a result, the more the FRR is lowered, the steeper the corresponding increase in the FAR, as is clearly showcased by the shape of the curve for DC models in figures like Figure 4.12. Because of the low weights, this model is already in the lower range of the FAR-FRR trade-off even at iteration 0. We see this with the phone-level errors where the FRR starts at 0.063 and during training the FRR is decreased by only 0.026 which leads to an increase in the FAR of 0.084. It is likely, that something similar happens with the other error types. Additionally, the lower the weights of the grammar are, the less of an influence the acoustic model has on the final prediction comparatively. This means that models with low weights rely less on the input audio and more on the heuristic of average error probabilities.

5.2.1.4 Differences in Folds

The results for the best performing model WFST+r+s were also analysed for each of the four folds individually. These results are copied in Table 5.12. The most notable finding from these results is that the performance for grade 3 is significantly worse than for the other grades. The difficulty of texts scaled with the grades, meaning the prompts for grade 3 were shorter and contained easier words than the other grades. As a result, the percentage of errors made is close to that of the other grades, meaning this is not the main cause for these results.

Grade	WER	WER*	DER	Error%	Predicted Error%	R-value	FRR	FAR
3	0.175	0.125	0.342	0.114	0.137	0.118	0.035	**0.702**
4	**0.134**	0.090	0.347	0.112	0.086	0.168	**0.023**	0.775
5	0.142	0.093	0.430	0.108	0.094	0.188	0.028	0.786
6-8	0.155	0.091	**0.316**	0.109	0.122	**0.288**	0.050	0.749

Table 5.12: Results for each of the four different grade groups (which are also the folds used for cross-validation) for WFST+r+s after 20 iterations. This is a copy of Table 4.11, with the FRR and FAR added to it.

Instead, the main problem seems to be a lack of fluency. Students from grade 3 are generally first-time readers aged 6 or 7. They read significantly less fluent than even grade 4 and in the dataset this manifests itself mostly as short insertions and false starts when a student hesitates on a word. The total number of insertions is not even much higher than the other grades. For grade 3, insertions make up 4.3% of total utterances, whereas for grade 6-8 (which has the fewest), insertions make up 3.0%. However, the insertions that are made are often only one or two phones long and therefore easy for the model to miss. On the other hand hesitations and longer pauses between words, give room for the model to falsely predict insertions that are really just background noise.

This can be seen in the results, where grade 3 scores a significantly higher WER* than the other models, while the differences between WER and WER* are smaller, which indicates that the model makes more mistakes on word-level errors and less on phone-level errors. To further confirm this, a linear regression was done on the insertion percentage made and predicted, like the analysis shown in Table 5.11. As expected,

the R-value was by far the lowest at 0.037, while the other R-values were 0.120, 0.094 and 0.244 for grades 4, 5 and 6-8 respectively.

After grade 3, grade 6-8 scored the lowest WER. This grade in a way has the opposite problem of grade 3, where it performs the best on insertions but it makes the most mistakes on predicting phone-level errors. That being said, grade 6-8 is not strictly worse than the grades 4 or 5. Rather, it predicts too many errors instead of too few and because of the skewed dataset, this leads to a higher WER. As a result, grade 6-8 has a higher FRR and a lower FAR than grades 4 and 5. The extra errors predicted are mostly omissions and substitutions. This is perhaps due to the faster reading speed. Students that read faster tend to "swallow" certain phones, which makes them harder for the model to pick up.

Finally, I want to note that grade 6-8 scores a significantly higher R-value than seen previously and that generally the R-value trends upwards as students get older. Of course, an R-value of 0.288 is not a very strong effect on its own, but it might suggest that in future work a model such as this could be a part of a larger reading level assessment tool that also takes into account other variables such as reading speed. This would especially be the case if this upwards trend continues into middle school.

5.2.2 Analysis of Wav2Vec 2.0 and Hybrid Models

The second step was to build Wav2Vec 2.0 models like the ones from section 5.1 that could evaluate phone-level errors, and then use these to re-evaluate all utterances that the WFST model predicted to be a substitution or correct. Two types of Wav2Vec 2.0 models were used from section 5.1: the phone classification (PC) model and the direct classification (DC) model, specifically the DC model that is trained with max adaptive pooling (DC_amax). During evaluation, a version of the DC model was implemented that enforced that all phones were detected in the correct order (DC_ordered). In all instances, this resulted in improvements over the regular DC model.

Furthermore, there were four different testing variations: zero-shot where the best performing model from the CHOREC dataset was used without further training, fine-tuned where that model was used as a starting point for training, from scratch which used a randomly initialized Wav2Vec 2.0 model as a starting point, and balanced which was also trained from scratch but with a dataset with an equal number of correct words and substitutions. The zero-shot models performed the worst. The zero-shot PC model rejected nearly all pronunciations. This is in line with the conclusions from section 5.1, where it was found that especially the PC model trained on CHOREC was very biased towards predicting the words from the vocabulary it was trained on. The zero-shot DC models on the other hand accepted nearly every pronunciation, because the threshold was set to maximize the balanced F-score. DC_ordered also set the threshold very low so that nearly all pronunciations were accepted, but further from the minimum threshold than the regular DC model, indicating that at least the utterances with the lowest scores were mostly substitutions. For the most part however, we can conclude that in this case the models did not generalize well between datasets. This is most likely due to the differences in vocabulary and the limited vocabulary that the CHOREC model was trained with in general (only 240 words including pseudo-words), as well as to the fact that the CHOREC dataset only contained audio from Flemish students, whereas the Malmberg dataset contained a large variation in accents, with only 3% originating from Flanders.

The fine-tuned and from scratch models performed the best, with no significant difference between them. This shows that the initial weights of Wav2Vec 2.0 models before training do not matter much. This result is in line with findings like those from [49], who found that pre-training a model for ASR did not make a significant difference in phone classification results. Unlike with CHOREC, using a balanced dataset for training did not lead to better results. There are a few likely reasons for this. Firstly, the Malmberg dataset has a much larger vocabulary than the CHOREC dataset, with over 7000 unique target words as opposed

to CHOREC's 240. The main problem with CHOREC was that models started learning and predicting the words rather than just the phones. This is what made balanced training an effective measure in some cases. Because of the large variety of words, many of which only have a few occurrences, the Malmberg dataset does not suffer from this same problem. Secondly, in the Malmberg dataset 3.5% of words was not correctly pronounced and therefore the balanced dataset contained only 7% of the data, which is a little over an hour. The balanced CHOREC dataset still contained a quarter of its data, which was close to 10 hours. This is most likely the reason why the balanced models for Malmberg underperformed compared to the fine-tuned and from scratch models. If anything, the fact that the results are not much worse is a testament to the performance of Wav2Vec 2.0 in scenarios where little data is available, which has been shown multiple times in previous research [6, 19, 64].

The Wav2Vec 2.0 models were first trained and tested on a dataset that used the annotations to find correct words and substitutions and their corresponding time boundaries. Then they were re-tested on a dataset that used the best performing WFST model (WFST+r+s after 20 iterations) to assign error labels and time boundaries to the utterances, like would be done in a hybrid model pipeline. The results do get slightly worse going from annotated boundaries to the hybrid model. This is to be expected. As we saw in section 5.2.1, there were many outliers in the WFST results, where the model completely misinterpreted the sentence. In sentences like these, it is common for certain target words to be attributed to the wrong utterance, which in turn makes it impossible for the Wav2Vec 2.0 model to make a good prediction.

If we compare the PC to the DC model, we see that the PC model is much stricter, meaning it is more likely to reject words. This is showcased by the fact that the FRR is much higher than the FAR. For a dataset of which the majority of words is correctly pronounced, this behavior leads to more erroneous predictions than being too lenient. Both DC and DC_ordered can be made "less strict" by lowering the threshold. Because of its structure, multiple pronunciations can be accepted for the same utterance, whereas the PC model only outputs and accepts one specific pronunciation. When listening to some of the wrong predictions made by the PC model, it becomes apparent how this can be a problem. Many words (especially shorter words) can be interpreted in multiple ways when heard in isolation even to human ears and often the context of the rest of the sentence is needed to interpret what word is heard [48]. In many cases the output of the PC model does not match the annotated label, but still sounds like an almost equally probable transcription. This is why in ASR a language model is often used alongside Wav2Vec 2.0.

The PC model does seem to have more predictive power than the DC models. This is showcased by the fact that the PC model has lower FAR, when the threshold of the DC models is set in such a way that the FRR is equal. That being said, an FRR of around 0.3 is too high for any practical applications. In an attempt to make the PC model less strict, experiments were conducted with accepting pronunciations that scored under a maximum phone error rate (PER) threshold. Setting this threshold to 0.2 for example, is equivalent to saying that the model can make one error for every five phones, where this error can be a phone insertion, deletion or substitution. At lower PER values this resulted in slightly better F-scores. However, at higher values it quickly becomes worse than the DC models, as can be seen from graphs like in Figure 4.12, where the graph of the PC models ends up to the right of that of the DC models.

Therefore only the DC models were used for the hybrid model. The WFST that was used as a baseline for the hybrid model scored a FRR of 0.035 and a FAR of 0.839 on phone level errors. For the hybrid model, the best performing DC model was chosen with its threshold set such that it also scored a FRR of 0.035. This was the fine-tuned DC_ordered model which scored a FAR of 0.758, so an improvement of 0.081. In the hybrid model this improved the WER from 0.151 to 0.141 and the R-value from 0.155 to 0.183. A second hybrid model was created with a DC model with a FRR of 0.05 and a FAR of 0.688, meaning it sacrificed 0.015 in FRR for an improvement of 0.080 in FAR compared to the first hybrid model. This model scored a WER of 0.147 and a R-value of 0.179, meaning it did still improve over the baseline but not over the first

hybrid model. Therefore, it did not seem necessary to test models with even higher FRRs.

While the hybrid models did manage to improve over the baseline, the results are not very impressive. The initial gain in FAR is quite small. At low FRR values, the FAR differences between models become smaller, as can be seen when comparing the different DC models in graphs like in Figure 4.12, so this is part of the reason why it is difficult to make large improvements. Unsurprisingly, the small FAR gain also did not lead to a large improvements in the WER, considering that the best achievable WER given the base WFST model was 0.097, since this is equal to WER*. However, for this dataset pursuing an optimal WER might not be ideal. Since the WER is affected by both false acceptances and false rejections equally, it is strongly related to accuracy. If we optimize the DC model for accuracy the threshold would be set so low that the DC model accepts nearly everything, which also means that any reading error results in a false acceptance. For practical applications, it is more recommended to set the threshold based on a maximum acceptable FRR, chosen by the user. Since the FRR and FAR are percentages of the correct and incorrect utterances respectively, they are mostly unaffected by the balance of the dataset.

5.2.3 Limitations and Future Work

The main goal for the WFST models in this research was not to get the perfect score on word-level errors, but rather to create a baseline for testing the potential benefits of a Wav2Vec 2.0 hybrid model. That being said, the results of the WFSTs were somewhat underwhelming, especially considering that some of the best performing literature reports a WER of under 0.03 [38, 52]. There are some steps that could be taken towards improving the WFST results in future work.

During this research, the approach was to keep models general so that all annotated utterances could in theory be predicted. However, there is a large disparity between the severity of annotated errors. "Less severe" errors are simultaneously less important and more difficult for the model to recognize. Common examples are inserted sounds that students make while reading (e.g. saying "um" while thinking), small phonetic changes in longer words (e.g. saying "gewichtloosheid" instead of "gewichtsloosheid") or unorthodox pronunciations of common words (e.g. saying "un" instead of the indeterminate article "een", which happens in 2% of all prompt occurrences of the word)[8]. These are now all counted as errors but in practice they would probably not be given as feedback by a tutor. At the same time errors like these are so similar to their correct version, that they are difficult for the model to classify as errors. The results would likely improve, if these types of mistakes would not be counted as errors. This might seem counterintuitive, since this would make the dataset even more imbalanced and the imbalance of the current dataset already causes problems because it leads to low error weights. However, in this case this would not be an issue, since these errors are probably not predicted as errors currently. This means that removing them does not decrease the number of predicted insertions or substitutions and the therefore the weights would not have to be decreased further, even though the data technically becomes more unbalanced.

Another approach would be to model errors like these directly by adding their phonetic transcriptions to the lexicon. Currently, the model can only predict insertions or substitutions with the pseudo-phone "GARB", which essentially means the model deems the utterance more similar to the average phone sequence, than to the expected word. For the examples mentioned above this is unlikely to happen, since they are phonetically similar to the expected word. By adding common insertions and mispronunciations of words to the lexicon, the model can more accurately judge whether an utterance is the correct word or the

[8]The Malmberg does not contain phonetic transcriptions, so it is not immediately clear what exactly is said when "un" is annotated. In a few cases I have heard the annotated "un" being pronounced as "/y n/", however it is also possible that it is sometimes used to indicate that a schwa is used ("/@ n/"). If this is the case, this would be even more problematic, since this pronunciation is in the lexicon as a valid pronunciation of "een" and therefore the model would likely predict this to be correct even though it is annotated as an error.

mispronounced variant. These substitutions or insertions could be given their own weight or share their weight with the original "GARB"-based errors. This is one of the main methodological differences between the current research and that of [38], of which the best performing model scored a WER of 2.06%. In their research no garbage class is used and instead insertions and mispronunciations of each word can be recognized by the lexicon directly. Their research did use the CHOREC dataset, which has phonetic transcriptions of all insertions and mispronunciations. This is not always realistic in a practical application, because students can make novel insertion or substitution errors that would not be in the lexicon. However, adding common insertions and mispronunciations to the lexicon should still improve results to some extent. It also allows the model to recognize when an error is one of the less severe errors above, so the user can decide whether to give these errors as feedback or not. In the case of thinking sounds or other noises, which can be hard to transcribe phonetically, a special pseudo-phone can be added so that these can be recognized by the acoustic model as one class.

One of the main findings of this research is that the models sacrifice their ability to detect errors in order to improve their ability to recognize correct words. This is a side effect of the dataset being unbalanced. It would be easy to suggest that for future research a more balanced dataset with a higher percentage of errors needs to be used, but if students on average do simply not make many reading errors, such a dataset would not be realistic, nor would models based on this data function optimally in practice. Instead, errors that the current model performs poorly on and that occur infrequently could be removed from the WFST completely. This means that the model would no longer be able to classify these errors and therefore any occurrence of such an error would result in a false acceptance. For example in [52] the best performing model scored a WER of 2.17%[9]. One of the main differences between [52] and the current research is that they opted not to model insertions and omissions (but still model all other classes from WFST+d+r+s). It should be noted that while their dataset contained twice as many errors per correct word, omissions and insertions only made up 0.3% and 0.7% of the total utterances respectively compared to 1.2% and 3.3% in the Malmberg dataset.

That being said, it is still quite possible that not modeling certain errors would improve the results even for this dataset. Take for example repetitions, which scored an FAR of 0.761 and an FRR of 0.011, as shown in Table 5.10. The total number of repetitions is 1005 and the total number of remaining utterances is 67825, as is shown in Table 5.8. This means that currently repetitions are the cause of 746 false rejections and 765 false acceptances, which is a total of 1511 errors. If repetitions would not be modeled, this would of course result in an FAR of 1 and an FRR of 0, so the maximum 1005 false acceptances and 0 false rejections which gives only 1005 total errors. This analysis does not account for the "false acceptances" which are actually repetitions being recognized as other error types, however this would still happen in both scenarios.

At first glance, this might seem similar to the models that do not include +r, but for these models the insertion weight was increased proportionally so that repetitions could be modeled as insertions. This is different from always accepting repetitions. The more this method is applied, the more the model moves towards the baseline that predicts every utterance to be correct, which should of course not be an ultimate goal. However, if this is only done for some error classes with poor performance and a low occurrence rate it will not only reduce the number of mistakes the model makes with that error class, but also reduce subsequent mistakes caused by initial wrong predictions. For future research, I would recommend testing this out with various error classes to see if any variation scores below the baseline where all utterances are correct. Another potential benefit of this strategy, is that the non-modeled error classes could instead be detected using Wav2Vec 2.0 in a hybrid model. For example if omissions are not modeled, the WFST will likely assign the word to frames that include a silence between words and then in turn Wav2Vec 2.0 will recognize that these frames do not include the target word. For insertion-like errors something similar can

[9]In this paper the WER was calculated at the first stage of the hybrid model where substitutions and correct words are considered to be one class, so it is essentially equivalent to the WER* from the current research.

happen, however this would require some changes to either the method of scoring or the interpretation of the Wav2Vec 2.0 output because currently it would be difficult to tell whether the insertion comes before or after the analyzed word.

There are a few ways to potentially improve the results of the Wav2Vec 2.0 models themselves as well. Firstly, one of the main findings from the CHOREC experiments is that Wav2Vec 2.0 models perform better when they are used on vocabulary that they have been trained on, as was indicated by the difference between the school-split and word-split results. The Malmberg dataset contains over 10 000 unique words, most of which only have a few occurrences and are not repeated between the test and train set. While it would of course be preferred to have a model that works well for any prompt, it is also not unimaginable that future users know which prompts will be used to in their final application. If these prompts are also included in the training dataset, this will likely improve the accuracy of the models for these prompts.

Finally, as mentioned the PC models perform better than the DC models at the same FRR, but are too strict for any practical application. The main reason is that the current implementation of the PC model only accepts one pronunciation variant, while the DC model can accept any variant of which the phones score above the threshold. Setting a threshold on the PER attempted to mitigate this problem by accepting variants that were within a specified phonetic distance of the predicted phone sequence. This improved the results at low thresholds, but at higher thresholds quickly led to too many false acceptances. As also suggested in section 5.1.4, a potentially better solution that can be researched in future work is to alter the PC model so that it assigns a score to the phone sequence corresponding to the canonical pronunciation, rather than predicting the most likely sequence of phones. Like with the DC model, this model could then choose to accept or reject a pronunciation by comparing its score to a predetermined threshold. After training the architecture of the PC model is almost identical to the DC model, with the main exception being that the PC model, besides assigning a score to every phone at every frame, also assigns a score to a padding token. If the scores of these tokens are ignored, the final score could be determined similarly to DC or DC_ordered. Alternatively, a dynamic programming approach could be used to sum over the scores of all potential sequences of phones and padding tokens that would result in the desired pronunciation.

5.3 Overall Conclusions

In this section I will answer the research questions and give recommendations for future work.

t-statistic: -0.8239536213816965 p-value: 0.41570864101699445 t-statistic: -3.3555080592808615 p-value: 0.001959420911953357 t-statistic: 0.24934119639749777 p-value: 0.8045970992272645 t-statistic: -2.341100623840167 p-value: 0.02522819379674312

t-statistic: -0.7357486354310304 p-value: 0.4669293886950402 t-statistic: 1.0877431177963959 p-value: 0.28435951383772556 t-statistic: -0.3158880193395692 p-value: 0.7540185457293843

5.3.1 Answers to Research Questions

This section answers the research questions posed in the introduction.

1 Can a Wav2Vec 2.0 based model significantly improve on the average TPD F-score of a trans-GOP based model, where significance will be measured by a paired samples t-test on the F-scores for each phone?

Answer: Yes, in both splits Wav2Vec 2.0 based models significantly managed to improve the TPD F-score of the trans-GOP baseline. In the school-split the baseline trans-GOP model scored an F-score

of 0.871 and the F-scores of the Wav2Vec 2.0 models ranged from 0.950 to 0.982. All differences were significant according to a paired samples t-test, with p-values much lower than 0.01. In the word-split the baseline trans-GOP model scored an F-score of 0.885 and the Wav2Vec 2.0 models again all scored higher with F-scores ranging from 0.878 to 0.915. However, only the scores of DC_max (0.915) and DC_amax (0.902) were significantly higher at a significance level of 0.05, with p-values of 0.0019 and 0.025 respectively.

1.1 Is there a significant difference between the average F-scores of the phone classification and the direct classification model for TPD?

Answer: In the school-split, the PC model scores slightly higher than the DC models. The average F-score was 0.982 for the PC model, while the highest for the DC models was 0.962. While the differences were small, all were significant with p-values much lower than 0.01. In the word-split, DC_mean is clearly the worst, but DC_max and DC_amax both outperform the PC model on F-score. However, none of the differences found were significant at a significance level of 0.05.

2 Can Wav2Vec 2.0 based model significantly improve on the phone-level reading error detection accuracy of the current ITSLanguage RED model, where significance will be measured by McNemar's test?

Answer: It can, albeit only by a small amount. The current WFST model scored a FRR of 0.035 and a FAR of 0.839 for phone-level errors. At the same FRR, the best performing DC model scored an FRR of 0.758. When implemented into the hybrid model the WER dropped from 0.151 to 0.141. The difference in accuracy was also significant according to McNemar's test, with a p-value much lower than 0.01. When the FRR was fixed at 0.05 the best performing model scored a FAR of 0.688, which resulted in a WER of 0.147. This version was also significantly more accurate than the baseline at the 0.05 significance level, with a p-value of 0.023.

2.1 Is there significant difference between the accuracy of the phone classification and the direct classification model for the detection of phone-level reading errors?

Answer: The final models could not be compared properly because of the imbalance in FAR and FRR and thus significance was not tested. That being said, the DC models seem more practical than the PC models. On the clean dataset the best performing PC model scored an F-score of 0.826 and a balanced F-score of 0.764 and the best performing DC model scored an F-score of 0.918 and a balanced F-score of 0.777. When the boundaries were determined by the output of a WFST, the best performing PC model scored an F-score of 0.776 and a balanced F-score of 0.716 and the best performing DC model scored an F-score of 0.888 and a balanced F-scored of 0.765. That being said, the FAR-FRR graphs (Figures 4.10 and 4.12) clearly show that the PC models have significantly lower FAR at their initial FRR values. However, these FRR values are too high for practical applications and since there is no good way to lower them without greatly sacrificing the overall performance, the DC models are more useful in their current implementation.

2.2 Can the ITSLanguage RED model's word-level error detection accuracy be significantly improved by delegating the detection of phone-level errors to a different model?

Answer: For this question the null hypothesis could not be rejected. Initially the idea was that parameters could be varied for the hybrid model, such that the word-level error detection (measured by WER*) was optimized while sacrificing the WFST's phone-level error detection performance, since

phone-level errors would be reassessed by Wav2Vec 2.0 anyway. The WFSTs parameters were altered in two main ways: by combining error classes and by testing out various weights during training. While the training process and the inclusion of +s did lead to improvements in the WER*, these improvements were not at the cost of phone-level error detection and therefore would have been implemented, whether a hybrid model was used or not.

After the initial WFST tests, the best WFST version was chosen as a baseline for the hybrid model. To further test for this research question, I could have considered lowering only the substitution weight further and seeing if this potentially improved the WER*. However, this would have also lowered the already low FRR on phone-level errors and because of the unbalanced dataset it would likely have improved the overall results. This would have made it harder to test the impact of the hybrid model and answer main research question 2.

5.3.2 Future Work Recommendations

In this section gives recommendations for future work, based on the findings of this research.

1. One of the main findings during the TPD research was that all Wav2Vec 2.0 models performed significantly worse in the word-split than in the school-split. This indicates that the models overfit with respect to the vocabulary that they were trained on. This was further confirmed when it was found that, when words in the test set are similar to words from the training set, the PC model sometimes outputs the word from the training set in its entirety. The DC models had the same problem to some extent, however this was slightly mitigated by the fact that the threshold could be set lower so that the correct phones could still be found, even if they were not the highest scoring phones.

   Because the models have a good performance if the test set vocabulary matches that of the test set, I would recommend trying to ensure that the vocabulary of the training data will match with that of the final application for future research.

   If the vocabulary for the final application is not known, or if the user explicitly wants a model that can be used with any vocabulary, then the focus should be on avoiding overfitting on the training vocabulary. The results with the balanced training data suggest that having a large enough percentage of pronunciation errors in the dataset might prevent overfitting, especially when the amount of training data does not have to be trimmed down to achieve this. When gathering their own TPD data, future users should try to ensure that a substantial proportion of the data is mispronounced. When working with skewed pre-existing data, one could experiment with purposefully leaving out correctly pronounced data points, to see if this improves the results on the test set. Besides the imbalance of the dataset, overfitting probably also occurred due to the relatively small vocabulary size (240 words total). It is likely that simply using a larger vocabulary with enough phonetic variation will also reduce overfitting.

2. The best mispronunciation detection results were achieved with the DC model that used adaptive max pooling and enforced that the phones scored above the threshold in the correct frame order. That being said, the PC model often achieved comparable if not better results, but at impractically high false rejection rates. The lack of a threshold variable meant that the PC model in many cases was too strict for detecting mispronunciations on the word level. During this research, slight improvements were found by setting a low threshold on the maximum allowable PER. However, the improvements found were small and the results quickly deteriorated at higher thresholds. For the calculation of the PER, every phone insertion, omission and substitution is weighed equally. One potential improvement could be to weigh these errors differently based on a phone confusion matrix, like [67] do with their

kaldi-based model for phone classification. This method works as follows: Let's for example say the phone "/s/" is mistaken for a "/z/" 20% of the time in the confusion matrix. If a word that contains a "/s/" is then predicted to have a "/z/", instead of adding 1, 0.8 is added to the total number of substitutions, insertions and omissions. This total is then still divided by the number of phones in the canonical pronunciation, like is done for the PER calculation. Like in [67], the confusion matrix can be pruned so that only phones with enough occurrences in the dataset are weighed differently. This method caters specifically to the model's weaknesses and seems promising for future research.

Alternatively, models could be explored that use the same structure as the PC model, but that output phone scores like the DC model. The simplest method would be to ignore the padding token scores and then derive the score of a word following the method of DC_ordered. Alternatively, a dynamic programming approach could be used to sum the scores of all strings of phones and padding tokens that collapse into the canonical string of phones.

3. While not the main focus of this research, several changes have been suggested for improving the WFST models built for RED. Most importantly, these are:

   - not counting less severe but hard to detect errors as errors in the annotation
   - including common substitution and insertion errors in the lexicon so that they can be recognized explicitly
   - experimenting with not modelling uncommon errors (so that they always lead to false acceptances)

   One potential extra benefit of this last method is that more error types can be recognized using Wav2Vec 2.0 instead of the WFST in a hybrid model. In general, it could be an interesting direction for future work to test to what extent Wav2Vec 2.0 can be relied on for RED. The most extreme test case would be to only allow the WFST to recognize correct words and then let Wav2Vec 2.0 detect miscues within the forced alignment range of each word (a similar approach to [13]). This would make Wav2Vec 2.0 at least partially responsible for the detection of all types of reading errors. However as a trade-off, such a model would lose many diagnosis capabilities and would probably require a new method for scoring errors like insertions, since these could be attributed as an error to either the word before or after it.

Bibliography

[1] A. Abaskohi, F. Mortazavi, and H. Moradi. Automatic speech recognition for speech assessment of preschool children. *arXiv preprint arXiv:2203.12886*, 2022.

[2] N. Abdul-Kadir, R. Sudirman, and N. Safri. Modelling of the Arabic plosive consonants characteristics based on spectrogram. In *2010 Fourth Asia International Conference on Mathematical/Analytical Modelling and Computer Simulation*, pages 282–285. IEEE, 2010.

[3] A. Alwan, Y. Bai, M. Black, L. Casey, M. Gerosa, M. Heritage, M. Iseli, B. Jones, A. Kazemzadeh, S. Lee, et al. A system for technology based assessment of language and literacy in young children: the role of multiple information sources. In *2007 IEEE 9th Workshop on Multimedia Signal Processing*, pages 26–30. IEEE, 2007.

[4] N. Arivazhagan, A. Bapna, O. Firat, D. Lepikhin, M. Johnson, M. Krikun, M. X. Chen, Y. Cao, G. Foster, C. Cherry, et al. Massively multilingual neural machine translation in the wild: Findings and challenges. *arXiv preprint arXiv:1907.05019*, 2019.

[5] P. Baay, M. Buisman, and W. Houtkoop. Laaggeletterden: achterblijvers in de digitale wereld. *Vaardigheden van burgers en aanpassingen door overheden*, 2015.

[6] A. Baevski, Y. Zhou, A. Mohamed, and M. Auli. Wav2Vec 2.0: A framework for self-supervised learning of speech representations. *Advances in Neural Information Processing Systems*, 33:12449–12460, 2020.

[7] Y. Bai, F. Hubers, C. Cucchiarini, and H. Strik. ASR-based evaluation and feedback for individualized reading practice. *INTERSPEECH*, pages 3870–3874, 2020.

[8] Y. Bai, F. Hubers, C. Cucchiarini, and H. Strik. An ASR-based reading tutor for practicing reading skills in the first grade. Improving performance through threshold adjustment. *IberSPEECH*, 2021.

[9] S. Bannò and M. Matassoni. Proficiency assessment of L2 spoken English using Wav2Vec 2.0. *arXiv preprint arXiv:2210.13168*, 2022.

[10] I. Baumann, D. Wagner, S. Bayerl, and T. Bocklet. Nonwords pronunciation classification in language development tests for preschool children. *arXiv preprint arXiv:2206.08058*, 2022.

[11] S. P. Bayerl, D. Wagner, E. Nöth, and K. Riedhammer. Detecting dysfluencies in stuttering therapy using Wav2Vec 2.0. *arXiv preprint arXiv:2204.03417*, 2022.

[12] J. Benesty, J. Chen, and Y. Huang. *Automatic speech recognition: A deep learning approach*. Springer Berlin Heidelberg, New York, USA, 2008.

[13] M. P. Black, J. Tepperman, and S. S. Narayanan. Automatic prediction of children's reading ability for high-level literacy assessment. *IEEE Transactions on Audio, Speech, and Language Processing*, 19(4): 1015–1028, 2010.

[14] P. Boersma and D. Weenink. Praat: doing phonetics by computer, 2023. URL https://www.fon.hum.uva.nl/praat/.

[15] I. Christoffels, P. Baay, I. Bijlsma, and M. Levels. Over de relatie tussen laaggeletterdheid en armoede. *Expertisecentrum Beroepsonderwijs & Stichting Lezen & Schrijven*, 2016.

[16] I. Christoffels, A. Groot, C. Clement, and J. Fond Lam. Preventie door interventie. *Literatuurstudie naar lees-en schrijfachterstanden bij kinderen en jongeren: prevalentie, relevante factoren en mogelijke interventies*, 2017.

[17] L. Cleuren. *Elements of speech technology based reading assessment and intervention.* PhD thesis, KU Leuven, 2009.

[18] L. Cleuren, J. Duchateau, P. Ghesquiere, et al. Children's oral reading corpus (CHOREC): description and assessment of annotator agreement. In *LREC 2008 Proceedings*, pages 998–1005. European Language Resources Association (ELRA); Parijs, 2008.

[19] A. Conneau, A. Baevski, R. Collobert, A. Mohamed, and M. Auli. Unsupervised cross-lingual representation learning for speech recognition. *arXiv preprint arXiv:2006.13979*, 2020.

[20] C. Cucchiarini, J. Driesen, H. V. Hamme, and E. Sanders. Recording speech of children, non-natives and elderly people for HLT applications: the JASMIN-CGN corpus, 2008.

[21] S. Deniz and M. Yavuz. Investigation of reading error types, reading levels and reading speeds of students with special learning difficulties. *Cypriot Journal of Educational Sciences*, 15(4), 2020.

[22] J. Duchateau, L. Cleuren, P. Ghesquière, et al. Automatic assessment of children's reading level. In *Proceedings of the European Conference on Speech Communication and Technology*, pages 1210–1213, 2007.

[23] J. Duchateau, Y. O. Kong, L. Cleuren, L. Latacz, J. Roelens, A. Samir, K. Demuynck, P. Ghesquière, W. Verhelst, et al. Developing a reading tutor: Design and evaluation of dedicated speech recognition and synthesis modules. *Speech Communication*, 51(10):985–994, 2009.

[24] Z. Fan, M. Li, S. Zhou, and B. Xu. Exploring Wav2Vec 2.0 on speaker verification and language identification. *arXiv preprint arXiv:2012.06185*, 2020.

[25] D. Fouarge and R. van der Velden. *Laaggeletterdheid in Nederland: Resultaten van de Adult Literacy and Life Skills Survey (ALL).* ECBO/ROA, 2011.

[26] M. Gerosa, D. Giuliani, S. Narayanan, and A. Potamianos. A review of ASR technologies for children's speech. In *Proceedings of the 2nd Workshop on Child, Computer and Interaction*, pages 1–8, 2009.

[27] B. Gold, N. Morgan, and D. Ellis. *Speech and audio signal processing: processing and perception of speech and music.* John Wiley & Sons, 2011.

[28] A. Graves, S. Fernández, F. Gomez, and J. Schmidhuber. Connectionist temporal classification: labelling unsegmented sequence data with recurrent neural networks. In *Proceedings of the 23rd international conference on Machine learning*, pages 369–376, 2006.

[29] W. Hu, Y. Qian, F. K. Soong, and Y. Wang. Improved mispronunciation detection with deep neural network trained acoustic models and transfer learning based logistic regression classifiers. *Speech Communication*, 67:154–166, 2015.

[30] S. Khurana, A. Laurent, and J. Glass. Magic dust for cross-lingual adaptation of monolingual Wav2Vec-2.0. In *ICASSP 2022-2022 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 6647–6651. IEEE, 2022.

[31] W. Labov and B. Baker. What is a reading error? *Applied Psycholinguistics*, 31(4):735–757, 2010.

[32] logopedie.nl. Spraak, May 2021. URL https://www.logopedie.nl/kennis/spraak/.

[33] B. Luo. Evaluating a computer-assisted pronunciation training (CAPT) technique for efficient classroom instruction. *Computer assisted language learning*, 29(3):451–476, 2016.

[34] M. Mohri, F. Pereira, and M. Riley. Weighted finite-state transducers in speech recognition. *Computer Speech & Language*, 16(1):69–88, 2002.

[35] M. Mommers, L. Verhoeven, and S. Van der Linden. Veilig leren lezen, 1990.

[36] J. Mostow, J. Nelson-Taylor, and J. E. Beck. Computer-guided oral reading versus independent practice: Comparison of sustained silent reading to an automated reading tutor that listens. *Journal of Educational Computing Research*, 49(2):249–276, 2013.

[37] National Reading Panel (US) and National Institute of Child Health and Human Development (US). *Teaching children to read: An evidence-based assessment of the scientific research literature on reading and its implications for reading instruction: Reports of the subgroups*. National Institute of Child Health and Human Development, 2000.

[38] M. Nicolao, M. Sanders, and T. Hain. Improved acoustic modelling for automatic literacy assessment of children. In *Proceedings of Interspeech 2018*, pages 1666–1670. ISCA, 2018.

[39] J. R. Novak, N. Minematsu, and K. Hirose. Phonetisaurus: Exploring grapheme-to-phoneme conversion with joint n-gram models in the wfst framework. *Natural Language Engineering*, 22(6):907–938, 2016.

[40] M. Novak. Evolution of the ASR decoder design. In *Text, Speech and Dialogue: 13th International Conference, TSD 2010, Brno, Czech Republic, September 6-10, 2010. Proceedings 13*, pages 10–17. Springer, 2010.

[41] N. Oostdijk. Het Corpus Gesproken Nederlands, 2000.

[42] D. S. Park, W. Chan, Y. Zhang, C.-C. Chiu, B. Zoph, E. D. Cubuk, and Q. V. Le. Specaugment: A simple data augmentation method for automatic speech recognition. *arXiv preprint arXiv:1904.08779*, 2019.

[43] V. Peddinti, D. Povey, and S. Khudanpur. A time delay neural network architecture for efficient modeling of long temporal contexts. In *Sixteenth annual conference of the international speech communication association*, 2015.

[44] L. Peng, K. Fu, B. Lin, D. Ke, and J. Zhang. A study on fine-tuning wav2vec2. 0 model for the task of mispronunciation detection and diagnosis. In *Interspeech*, pages 4448–4452, 2021.

[45] L. Pennington, J. Goldbart, and J. Marshall. Speech and language therapy to improve the communication skills of children with cerebral palsy. *Cochrane Database of Systematic Reviews*, 2004.

[46] S.-G. Pentiuc, I. Tobolcea, O. A. Schipor, M. Danubianu, and D. Schipor. Translation of the speech therapy programs in the logomon assisted therapy system. *Advances in Electrical and Computer Engineering*, 10(2):48–52, 2010.

[47] L. Pepino, P. Riera, and L. Ferrer. Emotion recognition from speech using Wav2Vec 2.0 embeddings. *arXiv preprint arXiv:2104.03502*, 2021.

[48] D. B. Pisoni and C. T. McLennan. Spoken word recognition: Historical roots, current theoretical issues, and some new directions. In *Neurobiology of language*, pages 239–253. Elsevier, 2016.

[49] J. Poncelet et al. Comparison of self-supervised speech pre-training methods on Flemish Dutch. In *2021 IEEE Automatic Speech Recognition and Understanding Workshop (ASRU)*, pages 169–176. IEEE, 2021.

[50] A. Potamianos, S. Narayanan, and S. Lee. Automatic speech recognition for children. In *Fifth European Conference on Speech Communication and Technology*, 1997.

[51] D. M. Powers. Evaluation: from precision, recall and F-measure to ROC, informedness, markedness and correlation. Technical report, Flinders University, 2020.

[52] J. Proença, C. Lopes, M. Tjalve, A. Stolcke, S. Candeias, and F. Perdigão. Mispronunciation detection in children's reading of sentences. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 26(7):1207–1219, 2018. doi: 10.1109/TASLP.2018.2820429.

[53] K. Reeder, J. Shapiro, J. Wakefield, and R. D'Silva. Speech recognition software contributes to reading development for young learners of English. *International Journal of Computer-Assisted Language Learning and Teaching (IJCALLT)*, 5(3):60–74, 2015.

[54] M. L. Rice, M. A. Sell, and P. A. Hadley. Social interactions of speech, and language-impaired children. *Journal of Speech, Language, and Hearing Research*, 34(6):1299–1307, 1991.

[55] C. M. Schuele. The impact of developmental speech and language impairments on the acquisition of literacy skills. *Mental retardation and developmental disabilities research reviews*, 10(3):176–183, 2004.

[56] P. J. Schwanenflugel and M. R. Kuhn. Reading fluency skill and the prosodic marking of linguistic focus. In P. Afflerbach, editor, *Handbook of individual differences in reading: Text and context.* Routledge, 2015.

[57] P. H. Seymour, M. Aro, and J. M. Erskine. Foundation literacy acquisition in european orthographies. *British Journal of psychology*, 94(2):143–174, 2003.

[58] S. Sudhakara, M. K. Ramanathi, C. Yarra, and P. K. Ghosh. An improved goodness of pronunciation (GoP) measure for pronunciation evaluation with DNN-HMM system considering HMM transition probabilities. In *INTERSPEECH*, pages 954–958, 2019.

[59] V. Vielzeuf and G. Antipov. Are E2E ASR models ready for an industrial usage? *arXiv preprint arXiv:2112.12572*, 2021.

[60] Y. Wang, A. Boumadane, and A. Heba. A fine-tuned Wav2Vec 2.0/huBERT benchmark for speech emotion recognition, speaker verification and spoken language understanding. *arXiv preprint arXiv:2111.02735*, 2021.

[61] S. M. Witt and S. J. Young. Phone-level pronunciation scoring and assessment for interactive language learning. *Speech communication*, 30(2-3):95–108, 2000.

[62] M. Wu, K. Li, W.-K. Leung, and H. Meng. Transformer based end-to-end mispronunciation detection and diagnosis. In *Interspeech*, pages 3954–3958, 2021.

[63] X. Wu, X. Liu, and L. Chen. Reducing EFL learners' error of sound deletion with ASR-based peer feedback. In *Emerging Technologies for Education: 6th International Symposium, SETE 2021, Zhuhai, China, November 11–12, 2021, Revised Selected Papers*, pages 178–189. Springer, 2022.

[64] Q. Xu, A. Baevski, and M. Auli. Simple and effective zero-shot cross-lingual phoneme recognition. *arXiv preprint arXiv:2109.11680*, 2021.

[65] X. Xu, Y. Kang, S. Cao, B. Lin, and L. Ma. Explore Wav2Vec 2.0 for mispronunciation detection. In *Interspeech*, pages 4428–4432, 2021.

[66] M. Yang, K. Hirschi, S. D. Looney, O. Kang, and J. H. Hansen. Improving mispronunciation detection with Wav2vec2-based momentum pseudo-labeling for accentedness and intelligibility assessment. *arXiv preprint arXiv:2203.15937*, 2022.

[67] E. Yilmaz, J. Pelemans, et al. Automatic assessment of children's reading with the FLaVoR decoding using a phone confusion model. *Proceedings Interspeech 2014*, pages 969–972, 2014.

[68] P. Zhan and A. Waibel. Vocal tract length normalization for large vocabulary continuous speech recognition. Technical report, Carnegie-Mellon University Pittsburgh PA School of Computer Science, 1997.