End-to-End Disfluency Detection in Automatic Speech Recognition for Second Language Learners

Tudor Nicolae Mateiu

School of Science

Thesis submitted for examination for the degree of Master of Science in Technology.

Espoo 21.11.2022

Supervisor

Prof. Mikko Kurimo

Advisor

MSc Yaroslav Getman,
Dr. Tamás Grósz

Aalto University
School of Science

Aalto University
School of Science

Aalto University, P.O. BOX 11000, 00076 AALTO
www.aalto.fi
Abstract of the master's thesis

Author Tudor Nicolae Mateiu

Title End-to-End Disfluency Detection in Automatic Speech Recognition for Second Language Learners

Degree programme Computer, Communication and Information Sciences

Major Machine Learning, Data Science and Artificial Intelligence **Code of major** SCI3044

Supervisor Prof. Mikko Kurimo

Advisor MSc Yaroslav Getman, Dr. Tamás Grósz

Date 21.11.2022 **Number of pages** 60+2 **Language** English

Abstract

Second language (L2) learner's speech data is a big challenge for Automatic Speech Recognition (ASR) models. Moreover, L2 students' speech contains many grammatical errors, mispronunciations and disfluencies, depending on the person's proficiency level. Disfluency detection tasks have conventionally been carried out as an added step after an ASR pipeline, which is inconvenient, as data needs to be prepared in addition to the one used for ASR, as well as the need of finetuning a supplemental model and incorporating it into the downstream task.

Conventional ASR systems are comprised of separate model components, an acoustic model, a language model and a lexicon. End-to-end ASR introduces a simplified pipeline over traditional systems, such that the acoustic feature sequences are directly mapped to word sequences, without the need for additional modules.

As end-to-end systems streamline the ASR process, this thesis investigates the incorporation of disfluency detection into the same low-resource end-to-end ASR task, thus eliminating the need for a separate component, and ultimately resulting in reduced computations. The disfluency detection models in this work are developed for L2 speakers learning Finnish, and obtain good performance without substantially deviating from an end-to-end L2 Finnish ASR baseline. The best model's ASR performance is promising, reaching a word error rate of 30.41 % and a character error rate of 13.17 %. Moreover, for disfluency detection the model obtains a Recall of 0.5655 and a Precision of 0.6017. The results are encouraging as the models can successfully extrapolate different disfluency types from low-resource L2 Finnish speech.

Keywords End-to-end, disfluency detection, ASR, Wav2Vec2.0

Preface

I dedicate this thesis to my loving parents, for all the support and assistance they have given me during the completion of the Master's programme and my stay in Finland.

I also want to thank my significant other, Alicia, for the time spent together in Finland. For the support and motivation she gave me during hardships, for all the amazing experiences we went through together and for the loving memories I will keep from our life in Espoo.

Finally I want to thank my supervisor, professor Mikko Kurimo, and my advisors, MSc Yaroslav Getman and Dr. Tamás Grósz. I thank them for the incredible opportunity given to me, for the chance of working at Aalto University, as well as for the counsel and feedback during the completion of my thesis.

Espoo, 21.11.2022

Tudor Nicolae Mateiu

Contents

Symbols and abbreviations

Symbols

$\mathcal{L}()$	loss function
E	loss function error
φ	activation function
θ	artificial neuron weight
∂	partial derivative
$P()$	probability function

Abbreviations

AM	acoustic model
ANN	artificial neural network
ASR	automatic speech recognition
BERT	bidirectional encoder representations
CER	character error rate
CL	curriculum learning
CNN	convolutional neural network
CTC	connectionist temporal classification
DL	deep learning
DNN	deep neural network
FE	feature extractor
FNN	feedforward neural network
GRU	gated recurrent unit
HMM	hidden Markov model
L2	second language learning
LM	language model
LSTM	long short-term memory
MFCC	mel-frequency cepstral coefficient
ML	machine learning
NLP	natural language processing
RNN	recurrent neural network
WER	word error rate

1 Introduction

Second language learning (L2) is not an effortless task for humans. It is usual, in the process of learning, for speakers to produce lexical and grammatical errors, enunciate words incorrectly and generate many types of disfluencies, often correlated to their native language. These characteristic features are problematic to Automatic Speech Recognition (ASR) systems to be trained on L2 speech, they ultimately increase the complexity of learning to recognize the speech. Additionally, this intricacy is magnified when dealing with lexicogrammatically demanding languages like Finnish.

Research in ASR has achieved great strides in improving the state-of-the-art, with results reaching that of human level speech recognition. Conventional ASR systems are normally comprised of separate model components, linked between each step, as for example an ASR system with an acoustic and language models. These conventional systems have performance drawbacks and they require labeled corpora to obtain decent competitive results. However, superior performance to conventional ASR was achieved with the introduction of self-supervised end-to-end systems [1, 2, 3]. These systems gave way to the development of large pre-trained models, which make use of substantial amounts of unlabeled data to automatically train and learn the given language or information about the pronunciation. These models not only obtain state-of-the-art results in ASR [4, 5, 6, 7], but as well with low-resource language ASR like for Swedish and Finnish [8, 9].

Early systems had independent ASR and disfluency detection models in the downstream task. This led to research exploring forms of incorporating disfluency detection in the same system. However, although recent research obtains satisfactory results, these systems usually only strive at detecting one type of disfluency or disfluencies originating from native language speakers [10, 11, 12, 13]. Moreover, research shows that end-to-end systems can successfully infer knowledge from disfluent speech [14] and acceptably detect up to two disfluency types [15].

This thesis employs a pre-trained Wav2Vec2 model [3] as the basis for an integrated disfluency detector in L2 Finnish ASR. Wav2Vec2 models are chosen as they are a robust self-supervised end-to-end ASR system which obtain state-of-the-art results with native language speaker speech, and perform well for low-resource languages [8]. Additionally, at the time of writing, only minor attempts have been carried out to use Wav2Vec2, or similar approaches, as a combined ASR and disfluency detection system [13, 15]. Therefore, the objectives pursued in this work are:

- Obtain a baseline from a pre-trained Wav2Vec2 by finetuning it on newly available L2 Finnish data from the Digitala project [16]. Afterwards, compare the results to previous work [17] in L2 Finnish ASR.
- Obtain a Wav2Vec2 model finetuned for L2 Finnish ASR disfluency detection by utilizing data with certain disfluencies tagged with a general disfluency token. Can this new model reach similar ASR results to the L2 Finnish ASR baseline?
- Validate the predicted results and examine if they are adequate. Additionally,

how many disfluencies in the testing set can the model predict correctly? Are there any specific disfluencies which are harder to infer than others?

The thesis is divided in multiple sections, each, in turn, with their different subsections. First, in Section 1 a brief introduction is given. Section 2 presents relevant background for the thesis and in Section 3 the methods researched are described. Afterwards, in Section 4 all the aspects related to experimentation are explained. This includes explanations of the data used, metrics, models, finetuning and commentary on the results obtained. Finally, Section 5 presents an abridged summary for the work done and the results obtained in the thesis, and in Section 6 the conclusions for the thesis are given.

2 Background

2.1 Machine Learning

2.2 Machine Learning

Machine learning (ML) is a branch of artificial intelligence tasked with the scientific study and development of algorithms that can learn from provided data. The behaviour of traditional artificial intelligence algorithms is explicitly dictated by written programming rules in order for them to perform certain tasks. In contrast, ML algorithms build a mathematical model from training data, and their knowledge is then applied, for instance, to predict or give decisions based on a series of new inputs. There are different methods in which a ML model can be built, depending on how the available data is presented and how the learning process is conducted.

2.2.1 Supervised Learning

Supervised learning is based on the concept of introducing data that is comprised of inputs and their corresponding labeled outputs. The goal during training is to approximate the model so that for new inputs the algorithm can predict their output correctly.

2.2.2 Unsupervised Learning

Unsupervised learning, as opposed to supervised learning, is a training method which employs completely unlabeled data. The goal for unsupervised learning is to discover the underlying structure or patterns in the data. It is used for such things as data clustering tasks, and autoencoder networks which employ unsupervised learning for training.

2.2.3 Self-supervised Learning

Self-supervised learning is a training method in which an ML model learns from unlabeled data. As opposed to unsupervised learning, which is typical for tasks such as clustering, self-supervised models first learn to extract good quality hidden representations from the unlabeled data, and these representations are then used in order to learn and predict the actual outputs of the data.

2.3 Artificial Neural Networks

Deep Learning (DL) is based on employing complex architectures which transform data and extract features to perform a task. These complex artificial neural network (ANN) architectures are trained on considerably more data than normal ML algorithms in order to achieve reasonable performance. Additionally, DL models use learning methods such as supervised learning to optimize the many different hyperparameters they are comprised of.

ANNs are complex architectures, loosely inspired by the biological neuron and synapse connections that constitute an animal's brain. These networks are comprised of an input layer, a hidden layer, and an output layer. Additionally, if there are multiple hidden layers, they are also known as Deep Neural Networks (DNN). The layers, in turn, consist of connected artificial neurons that compute input data and send output data to other connected neurons. Like this, data that enters the input layer, gets passed and processed between layers until it reaches the output layer; depending on the type of ANN this can be done several times and by following different strategies.

The elementary artificial neurons, such as perceptrons [18], are comprised of one or multiple weighted inputs, which are summed to produce an output by passing them through its activation function. Therefore, given a number of $m + 1$ inputs, with signals $x_0 \dots x_m$, and weights $w_{i0} \dots w_{im}$, the neuron will sum the products of its input signals and weights, and pass it to the activation function φ, such that:

$$y_i = \varphi(\sum_{j=0}^{m} w_{ij} x j) \tag{1}$$

The neuron's output y_i can then propagate through to the input of following connected layers, or it can exit the network. Additionally, when the connections between an ANN's layers do not form cycles, they are known as Feedforward Neural Networks (FNN) [19]. In FNNs the information moves forward through the connected layers, and then to the final output layer. In contrast, a Recurrent Neural Network [20] is a network which can contain layer nodes to create cycles, thus allowing the output of some nodes to influence subsequent inputs to the same nodes.

Regardless of connection type, the common procedure by which ANNs acquire knowledge from the input signals is through backpropagation [21]. Backpropagation is a method used to efficiently train ANNs following a gradient descent approach that exploits the formula for computing the derivative of the composition of two or more functions. The main feature of backpropagation is that it enables feedforward models to become iterative, recursive and efficient at calculating the weight values to improve the network until it is able to perform the task for which it is being trained.

Before training begins, the network's weights are all preliminarily set depending on a chosen method. Habitually, weight values are randomly generated, although there exist techniques that propose better weight initialization [22, 23]. For supervised learning, each neuron gains knowledge from labeled training examples, which consist of a set of tuples of inputs and corresponding correct outputs, (x_i, t_i). Initially, the network computes an output, y_i, and is penalized or rewarded if the output differs or not from the correct one.

From this, a loss function $\mathcal{L}(t, y)$ is used for measuring the disparity, or error E, between the expected output and the predicted output. In a way, the training process can be conceptualized as the need to produce an output that exactly matches the expected output, which means the error sought after must be zero. ANN training can be thought of as an optimization problem to find a hypothesis that minimizes this error.

The network's output is ultimately obtained by the calculation of the node's activation functions, which in turn employ the values of the neurons' weights and input signals. However, as the input signals can not be modified, the error depends on the variable value of the node weights, therefore they are ultimately what needs to be changed in the network to enable learning. Thus, together with backpropagation, the gradient descent algorithm is used to find the set of weights that minimizes the error.

The gradient descent method involves calculating the derivative of the loss function $\mathcal{L}(t, y)$ with respect to the weights of the network in order to update the weights accordingly. Afterwards, the gradient descent method updates each weight according to the subtraction of the weight's original value and the multiplication of the learning rate and partial derivative, such that:

$$\theta_{ijk} = \theta_{ijk} - (lr \times \frac{\partial E}{\partial \theta_{ijk}}) \tag{2}$$

where E is the loss $\mathcal{L}(t_i, y_i)$ between the expected output t_i and the predicted output y_i, θ_{ijk} is the weight value j of neuron i from layer k, and lr is the learning rate, a value which controls the step size by which weights are updated.

Backpropagation is performed for each of the weights and for all the examples in the training dataset, although not all backpropagation implementations use the same loss function. Every pass over all the training examples is called an epoch.

2.3.1 Convolutional Neural Networks

Convolutional Neural Networks (CNN) [24] are a specialized kind of ANN for processing data that has a known grid-like topology. They are most commonly applied when required to analyze visual imagery, however there are multiple other fields where it has been applied with successful results, including text and audio processing [25, 26]. Additionally, CNNs have been utilized as the building blocks of more complex networks [60, 3]. They make use of a mathematical linear operation called convolution at least once throughout the network.

Its design, shown in Figure 1, is comprised of connected layers, each built of groups of artificial neurons. Neurons from one layer receive input from a specific group of neurons of the previous layer, this group is called a receptive field. The architecture has an input and an output layer, as well as multiple hidden layers. The hidden layers of a CNN typically consist of a series of convolutional layers that carry out linear transformations via multiplication or other scalar products.

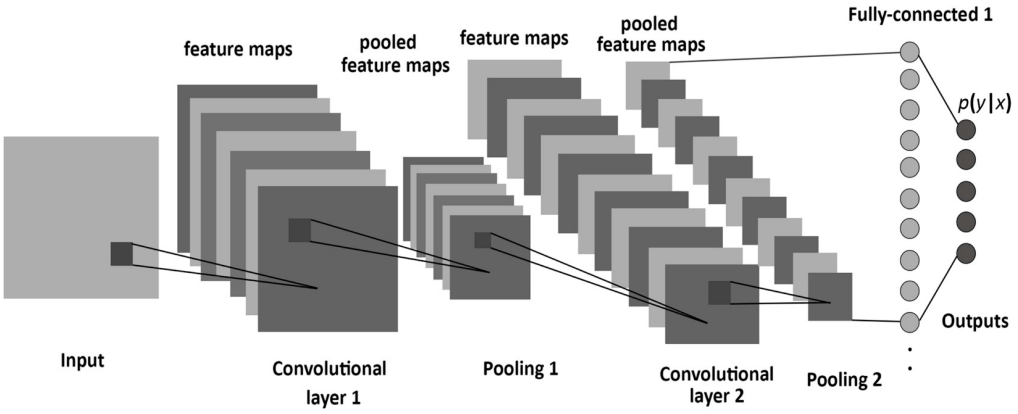

Figure 1: CNN architecture example [27]. The network contains multiple convolutional layers, pooling layers after each convolutional layer, and a final possible flattening of features passed to a fully connected layer to obtain the outputs.

Convolutional layers are the fundamental element of CNN architectures. The feature extraction is carried out through the receptive fields in each convolutional layer. In the feedfoward process of the network, the set of learnable parameters of these fields, the kernel, is moved across the input. Next, the dot product between the strides of the kernel and the input values is calculated. The result is a feature map from which the system learns when some feature is found at specific locations of the map. Because of this, the main body of convolutional layers is also known as a feature extractor.

Additionally, pooling layers are used in order to reduce the number of dimensions of the featured maps obtained from one convolutional layer to another. There are two ways of carrying out layer pooling, local and global. Local pooling merges small areas of the feature maps, while global pooling combines all of the neurons of the feature maps. Moreover, there are many techniques to calculate these pooling operations, the most famous are max pooling and average pooling. Max pooling obtains the maximum value of each of the clusters being pooled from the feature map, while average pooling instead calculates the average value.

After having passed through all the convolutional and pooling layers, it is customary to flatten the feature maps into a vector before the feature maps arrive to the final layer. The final layers are comprised of one or multiple fully connected layers, and the output is calculated depending on the task at hand. For instance, to compute the output for a binary classification task a sigmoid activation function is employed. However, for a multiple class classification task, it is common to use the softmax function.

2.3.2 Recurrent Neural Networks

A Recurrent Neural Network (RNN) [20, 28] is an augmentation of a standard feedforward neural network, which is able to handle sequential input that can vary in length. The network handles this variable sequence by having a recurrent hidden state whose activation at each pass is dependent on that of the previous iteration, as

shown in Figure 2. Unfortunately, it is often difficult to train RNNs without facing its two main problems: gradient vanishing and gradient explosion. On one hand, as the backpropagation algorithm progresses, the gradients obtained get smaller at each step. This can result in unchanged weights closer to the input layer and a convergence to local minima will be more complicated. On the other hand, occasionally the gradients obtained via backpropagation will increase in value and gradient descent will fail to converge. These issues have led to the research of recurrent gated units, which have become a dominant approach for reducing the negative impacts of the gradient problem in RNNs.

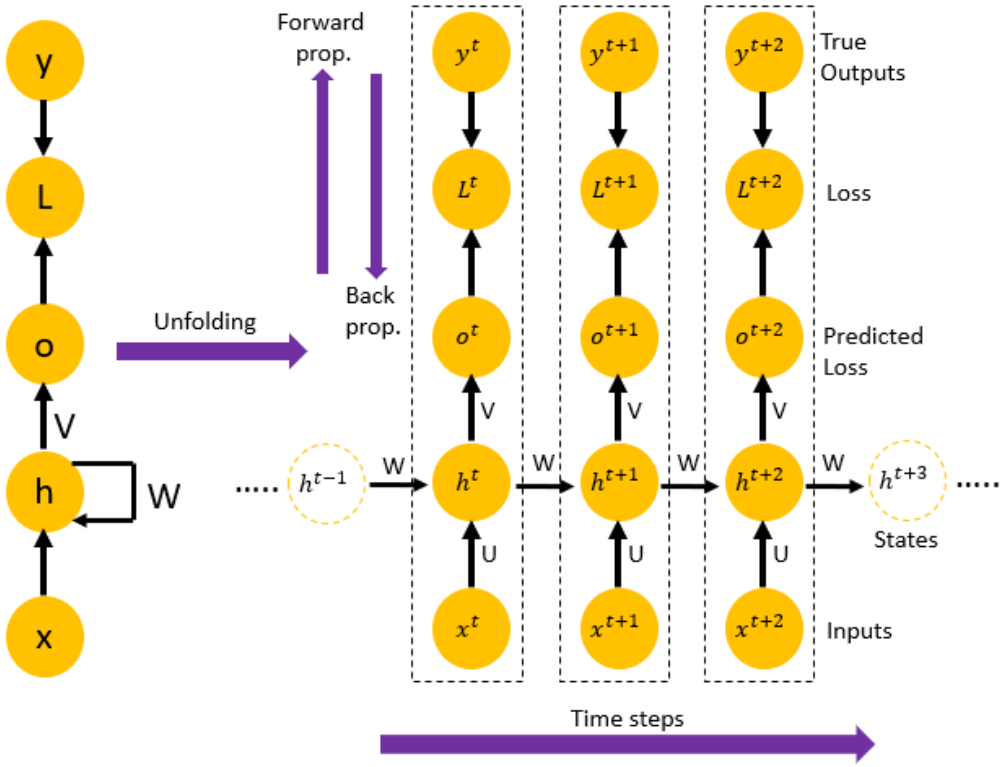

Figure 2: Illustration of a folded RNN architecture (left) and one unfolded over different time steps (right) [29].

The Long Short-Term Memory (LSTM) gated unit architecture [30] consists of a set of recurrently connected memory blocks, shown in Figure 3. Each block contains self-connected memory cells and three multiplicative units. These units allow the LSTM's memory cells to store and access information over long steps of time. As a result, the units will help in mitigating the vanishing gradient problem RNNs inherently have, and enable the use of RNNs for tasks requiring longer contextual memory.

The Gated Recurrent Unit (GRU) gated unit architecture [31] is similar to that of the LSTM: it consists of an update gate and a reset gate but it does not make use

of any memory cells, shown in Figure 3. These two gates decide what information should be passed to the output and can be trained to keep contextual information or remove information that is irrelevant to the prediction sequence. Just as LSTMs, GRUs also help in mitigating the vanishing gradient problem. Moreover, they are faster to train than LSTMs, given the lack of memory cells, and obtain similar results.

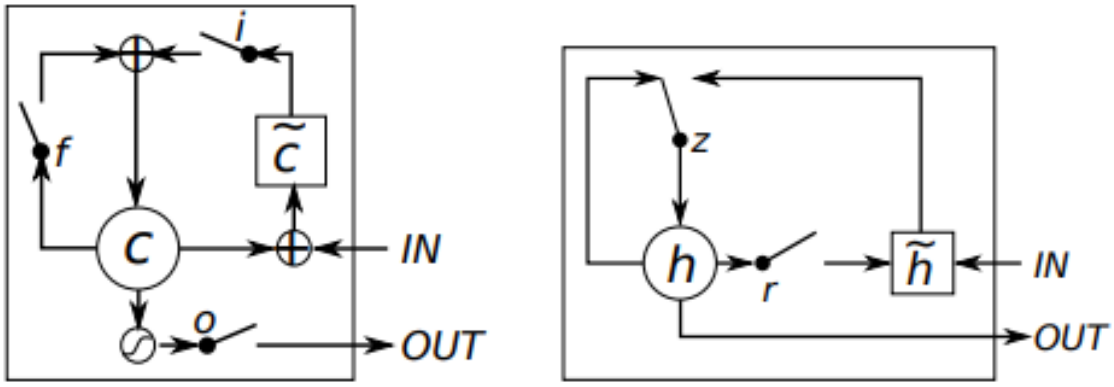

Figure 3: Illustration of Long Short-Term Memory (left) and Gated Recurrent Unit (right) [31].

These established RNN approaches have obtained good results in sequence transduction tasks, as they solve the gradient vanishing problem. Specifically, they have obtained substantially better results than older feed-forward approaches in tasks such as sequence to sequence translation [32] and text generation [33]. Regardless of obtaining better results, their sequential nature impedes parallelization during the training and evaluation process, which becomes critical if longer sequence lengths are used or needed as input or output. These memory constraints limit batching across examples and restricts the model to longer training and evaluation times. However, there have been improvements in the computational efficiency of training and evaluation via parameter reduction tricks [34] and conditional unit activation [35]. Irregardless, the sequentiality remains, and, ultimately, their efficiency remains restrained.

2.3.3 Transformer models

The Transformer model [36] is the first ANN transduction model which relies entirely on self-attention mechanisms to carry out computations without using sequence aligned RNNs or convolutions. This fact enables the model to successfully train and evaluate using parallelism, which considerably lowers the time needed for these computations, while also enabling it to keep contextual information over much larger sequences without the fading of gradients [2].

The Transformer model principally follows the same architecture as established sequence transduction encoder-decoder models. It employs stacked fully connected

encoder-decoder layers and multi-head attention layers, as can be observed in Figure 4.

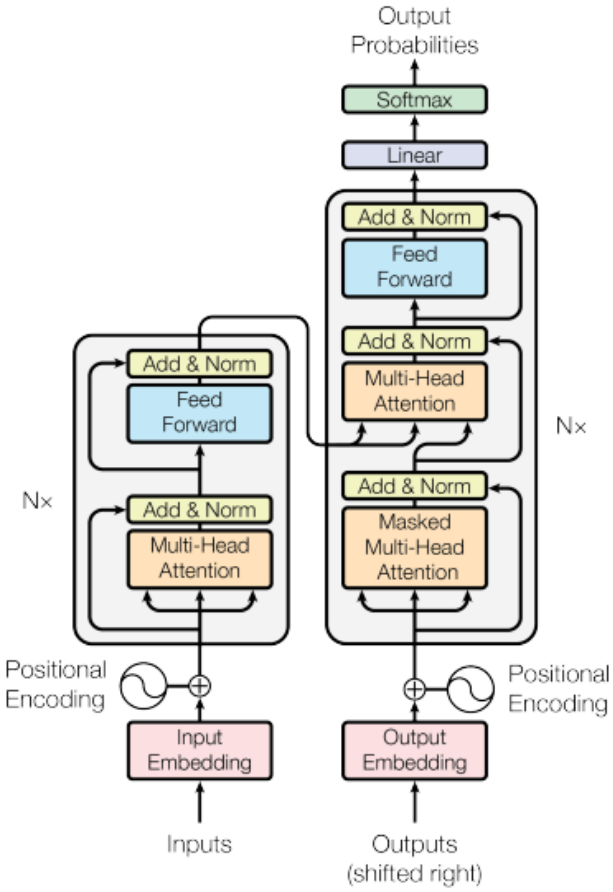

Figure 4: Illustration of a basic Transformer model architecture [36].

The standard Transformer's encoder stack is composed of six identical layers, in turn comprised of two sub-layers each. These sub-layers are composed of: first, a multi-head attention mechanism, and second, a basic position-wise feed-forward network. Additionally, they are followed by a residual connection [37] and layer normalization [38]. The residual connections help to increase accuracy, as they can help to optimize deeper networks. The layer normalization normalizes the results obtained after each layer, thus helping in reducing the computation times. After having passed all of these layers, the encoder's outputs of dimension $d_{\text{model}} = 512$ is then be passed onto the decoder.

The decoder stack is composed of six identical layers, similar to the encoder. The main difference is that there is an extra sub-layer in addition to the two present in the encoder. This other sub-layer is a masked multi-head attention mechanism which computes over the outputs obtained from the encoder. The decoder's sub-layers are also followed by a residual connection and layer normalization operations.

The Transformer models' self-attention techniques enable the decoder to handle all positions in the decoder up to and including the current position of the decoder.

A basic attention function is described as the mapping of a query and a set of key-value pairs to an output, and is calculated as a weighed sum of these values. More specifically, Transformer models use scaled dot-product attention, shown in Figure 5, as it benefits the model with faster and more space-efficient calculations.

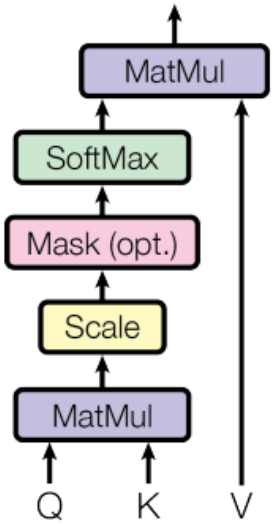

Figure 5: Illustration of a scale dot-product attention mechanism [36].

The dot product of the query, Q, and keys, K, is divided by the square root of the dimension of keys d_k. This division is done to scale the dot products, in order to keep the softmax function out of regions with extremely small gradients. Afterwards, the softmax function is applied to normalize the results and the attention weights are performed. These weights are subsequently multiplied to all the input sequences, V. These calculations [2] are done as:

$$\text{Attention}(Q, K, V) = softmax(\frac{QK^T}{\sqrt{d_k}})V \tag{3}$$

This scaled dot-product attention function is performed h times, shown in Figure 6, with different projections to the dimensions of queries, keys and values, d_k, d_k and d_v respectively, as it is more beneficial than applying only a single attention function. A total of $h = 8$ parallel attention heads are used at no additional computational cost, as each head is reduced dimensionally to $d_k = d_v = d_{\text{model}}/h = 64$. The benefit of using this multi-headed approach is that the Transformer can attend to the data from different representation subspaces at different positions.

$$\text{MultiHead}(Q, K, V) = Concat(\text{head}_1, \text{head}_2, \ldots, \text{head}_h)W^O \tag{4}$$

The multi-head attention function is calculated, as shown in Eq 4, as the multiplication of the concatenation of the attention heads and the weights matrix

$W^O \in \mathbb{R}^{hd_v \times d_{\text{model}}}$. Additionally, each attention head is calculated with the projections of the weight matrices $W_i^Q \in \mathbb{R}^{d_{\text{model}} \times d_k}$, $W_i^K \in \mathbb{R}^{d_{\text{model}} \times d_k}$ and $W_i^V \in \mathbb{R}^{d_{\text{model}} \times d_v}$, such that:

$$\text{head}_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V) \tag{5}$$

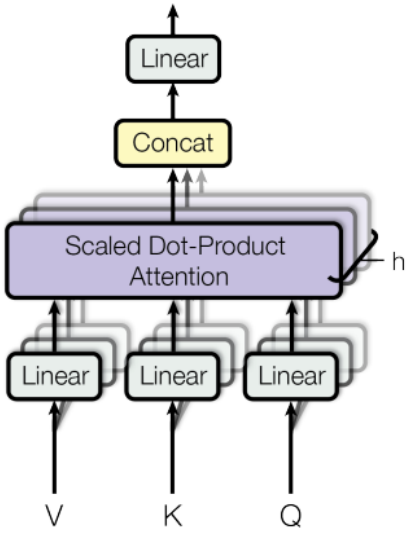

Figure 6: Illustration of a multi-headed attention mechanism [36], with h scaled dot-product attention heads.

Furthermore, besides multi-head attention layers, the encoder and decoder stacks contain fully connected position-wise feedforward networks. These networks are separately and identically applied to each position.

$$\text{FFN}(x) = \max(0, xW_1 + b_1)W_2 + b_2 \tag{6}$$

As shown in Eq 6, the output of these layers is calculated with two linear transformations and a ReLU activation. The linear transformations employ different weights from the different attention layers, and the inner-layer output results in a dimensionality of $d_{ff} = 2048$. Additionally, the inputs and outputs of both the encoder and decoder stacks are transformed into low-dimensional vectors. These transformations are done using learned embeddings to convert the input and output token vectors to a dimension of $d_{\text{model}} = 512$. Moreover, the decoder outputs are transformed into predicted next-token probabilities by applying the standard learned linear transformation and softmax function.

As a final note, the Transformer model requires the injection of information pertaining to the relative or absolute position of tokens in a given sequence. This additional information is necessary due to the lack of recurrence and convolutions in the model. The Transformer employs fixed positional encodings [39], which are calculated using sine and cosine functions, such that:

$$\mathrm{PE}_{(pos,2i)} = sin(pos/10000^{2i/d_{\mathrm{model}}}) \tag{7}$$

$$\mathrm{PE}_{(pos,2i+1)} = cos(pos/10000^{2i/d_{\mathrm{model}}}) \tag{8}$$

where *pos* is the position and i is the dimension. The model uses the sinusoidal fixed positional encoding method because it is theorized that the model may be able to extrapolate to sequence lengths longer than the ones encountered during training.

2.3.4 BERT models

The Bidirectional Encoder Representations from Transformers (BERT) [40] is a proposed language representation model that reduces the limitation of unidirectionality in prevalent techniques before this model. BERT employs a masked language model in the pre-training task, which randomly masks a number of input tokens. The task is then to predict the original vocabulary id of these masked tokens based on context. This effectively allows for the pre-training of deep bidirectional Transformer models, as the masked language model allows BERT to learn words based on their left and right contexts.

The BERT architecture is based on the original Transformer implementation and is designed as a multi-layer bidirectional Transformer encoder. There are two proposed BERT configurations [40] depending on the preferred parameter count: "$\mathrm{BERT}_{\mathrm{BASE}}$" and "$\mathrm{BERT}_{\mathrm{LARGE}}$". The base model is comprised of 12 Transformer blocks, 768 hidden nodes and 12 self-attention heads. This version has a total of 110 million parameters, and illustrated in Figure 7. On the other hand, the large model is comprised of 24 Transformer blocks, 1024 hidden nodes and 16 self-attention heads, for a total of 340 million parameters.

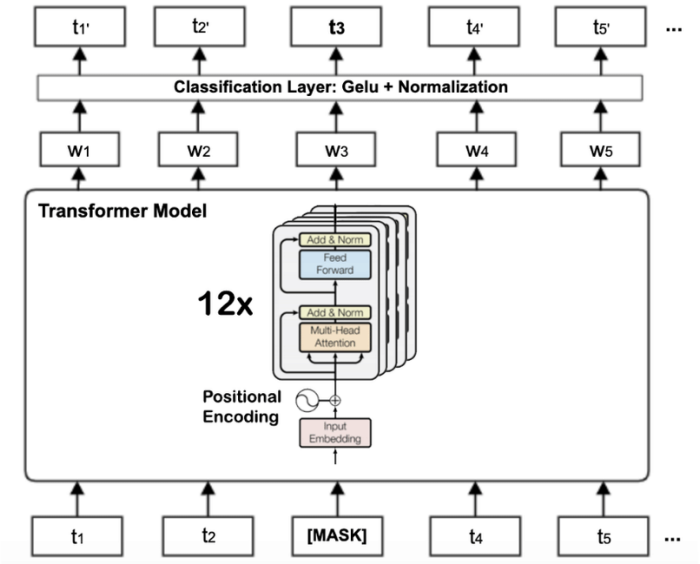

Figure 7: Illustration of an example BERT base-sized architecture [41].

The BERT model has been designed with the possibility of carrying out different tasks. The input representation can handle single sentence inputs and pair of sentences inputs in one sequence. The first token of every BERT sentence is the special token $[CLS]$, which represents sentence-level classification. However, when using sentence pairs as inputs, the two sentences are separated with the additional $[SEP]$ special token, which indicates where the sequence separates the first and second sentences. Moreover, embeddings are added to each token to indicate the position of the token and whether the token originates from sentence A or sentence B, shown in Figure 8.

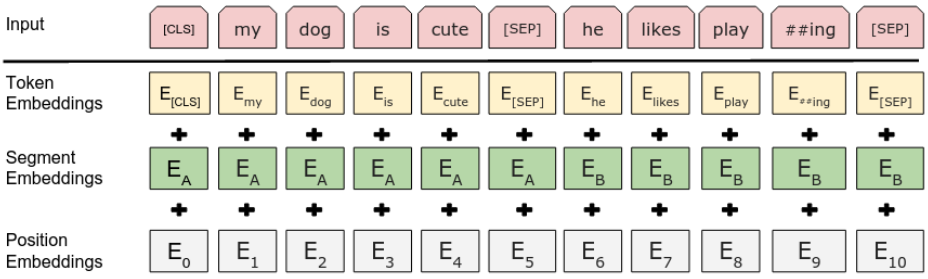

Figure 8: Illustration of an example pair of sentences as a BERT input representation [41].

Finally, the BERT pre-training phase is divided in two tasks. First, the masked language model masks 15 % of the tokens in each sentence from the training data. Afterwards, the tokens selected have an 80 % chance at being replaced with the $[MASK]$ token, a 10 % chance at being replaced with a random token and a 10 % chance at remaining unchanged. This additional procedure is necessary so that there is no inconsistency between pre-training and finetuning, given that the $[MASK]$ token is technically only found in the pre-training phase. Subsequently, the task is for the original token to be predicted with the cross-entropy loss.

The first step in pre-training ensures a robust deep bidirectional representation, however, there are many tasks which are based on sentence pair relationships. Thus, the BERT model aims at learning to capture these in its second pre-training step by carrying out a binarized next sentence prediction task. The sentences in the data are generated such that, the first sentence A is always the correct one, while the next sentence B has a 50 % probability of being a random sentence from the corpus and 50 % probability of being the correct next sentence to A. The model is then tasked with predicting if the input sentence pair is correct, whether sentence A is truly followed by sentence B.

Because of the manner in which the BERT model is pre-trained, and the knowledge it receives during said process, the model can be easily finetuned for more specific downstream tasks. These include the same ones the model has been pre-trained for, but BERT can also be modified, for example, by attaching fully-connected linear layers to the end of its architecture to encompass other tasks as well.

3 Research material and methods

3.1 Automatic Speech Recognition

Automatic Speech Recognition is an interdisciplinary field, in which human spoken language is recognized by a machine and transcribed into text. Conventional ASR performed its recognition task via a probabilistic approach that had dominated the field until recently. The many drawbacks of conventional ASR gave way to the research of end-to-end ASR systems, which solved the shortcomings of the previous research.

3.1.1 Conventional ASR systems

Conventional ASR systems are separated into different independent components, shown in Figure 9, which need to be trained separately and each perform their own task during the ASR process. These components are: the feature extraction segment (FE), the acoustic model (AM), the language model (LM), the lexicon and the decoder.

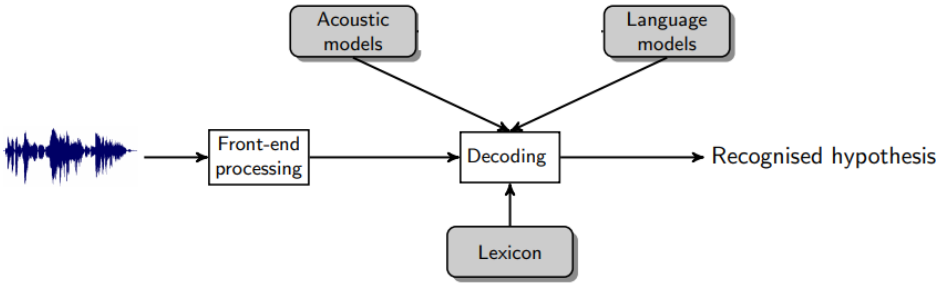

Figure 9: Overview of a conventional ASR system [42], comprised of the feature extraction segment, acoustic and language models and the lexicon.

The FE portion of the pipeline receives speech signal and converts it into feature vectors. The most common type of feature vector used for conventional ASR are coefficients derived from Mel frequency cepstrum [43]. These cepstra are computed using frequency bands which closely approximate the perception of sounds by the human auditory system. From these the Mel Frequency Cepstral Coefficients (MFCC) [44] are derived. Given an audio spectrum, the usual process of deriving the MFCCs starts with magnifying high frequencies which can result from natural speech. Afterwards, the audio spectrum is divided into 20-30 ms frames and discrete Fourier transform is applied via a windowing process, usually Hamming window, to extract the signal's spectral information. The Mel spectrum is then calculated by passing the spectral information, obtained from discrete Fourier transform, through multiple triangular band pass filters, from which the logarithmic output is obtained. Ultimately, discrete cosine transform is applied to the log Mel spectrum and the MFCCs are derived.

Given the MFCCs obtained from the previous step, the AM is utilized to compute the probabilities of phoneme for each of the feature vector's split frames. The AM

contains the required statistical representations of the phonemes of the words found in the LM and the modelling is usually done by building Hidden Markov Models (HMM). The HMMs are comprised of a series of state nodes and transitions between each state given a set of probabilities. Additionally, given a phoneme, the HMM can be built according to how many phonemes around the input will be taken into consideration as additional context. Figure 10 shows a left-right HMM built with 5 hidden states to improve performance in speech with background noise.

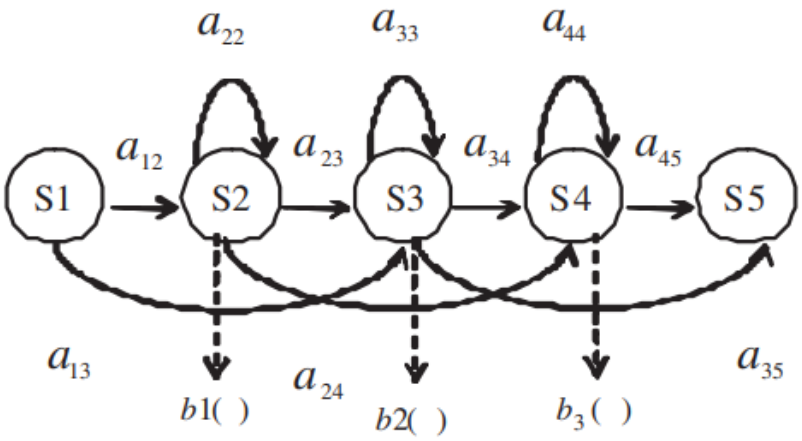

Figure 10: Illustration of left-right HMM with 5 hidden states [44]. In this particular HMM the states are depicted as s_i nodes, $b_i()$ indicate the observation probability density functions and a_{ij} are the transition probabilities from nodes i to j.

The LM is independently trained on a substantial amount of text data, and is then employed to statistically predict in a sentence what word is the next one, when taking as context the previous words. Usually these predictions are done using word frequency counts, such as n-gram LMs. These models employ a $n-1$ long context of previous words, and are developed by building a $(n-1)$-order Markov model from large text training corpora. The model contains all the possible combinations of contiguous $n-1$ words and their frequency. For example a 2-gram (or bi-gram) model, with the training sentence "Hello good, wonderful people!" will generate the following bi-grams: "hello good", "good wonderful" and "wonderful people". This type of LM modeling brings multiple drawbacks, that although have been addressed through research, are ultimately still detrimental.

On one hand, new n-gram sequences that do not appear in training data will result in errors as their resulting probability by the LM is zero. The implementation of probability smoothing processes have been employed to solve this issue. Smoothing methods like Kneser-Ney smoothing [45] recalculate the probability of the n-grams and assign some small probability to new ones. However, even these methods do not fully fix the problems which rise from the statistical aspect of LMs, as well as their failure at capturing longer context, as n-gram models with large n still fail to

capture long context with the same capacity as RNNs with recurrent gated units or Attention mechanisms, or Transformer models [40]. On the other hand, LMs can suffer from probability biases, as true correct words can be rejected in favour of stored n-grams with high probability.

The last component of the system is the lexicon. This component is in charge of storing a huge vocabulary of words with their respective phoneme pronunciation. Consequential downsides of this are the sheer size and specialized human labor required in order to have a robust phoneme to word mapping lexicon. Additionally, given a specific language, many words can be pronounced identically to each other, but one word can also have multiple pronunciations.

Conventional ASR can be expressed as the task of finding the most probable word sequence given a set of observations, such that:

$$W^* = \text{argmax}_W P(W|O) \tag{9}$$

where W^* is the most probable word sequence, O is the observation sequence and $P(W|O)$ is the conditional probability of a word sequence W given the observation sequence O. By using Bayes' rule, equation 9 can be formulated as:

$$W^* = \frac{P(O|W)P(W)}{P(O)} = \text{argmax}_W P(O|W)P(W) \tag{10}$$

where $P(O|W)$ is obtained from the AM and $P(W)$ is calculated by the LM.

All of these components, work together in the decoder to finalize the conventional ASR pipeline. First, the AM obtains the most probable phonemes given multiple MFCC feature vectors. Second, given the lexicon and the LM, a choice is made, with the phonemes obtained, as to which word should be the correct one.

3.1.2 End-to-end ASR systems

During the past decade, all the drawbacks of conventional ASR have led to the research of end-to-end ASR systems, and have consolidated their place as the state-of-the-art models for ASR tasks [15, 46, 47, 48, 49, 50].

Just like conventional ASR, end-to-end systems are comprised of multiple components, however, these are merged into one single architecture, and it enables the model to be trained as a whole. This merged model can single-handedly carry out the direct mapping of speech signal into text sequence. In contrast, conventional ASR systems require the individual training of it's components, which complicates the training process and decreases their speech recognition potential, when compared to single end-to-end systems trained on considerable amounts of data.

Ultimately, the use of multiple components can lead to conflicting predictions between themselves. Moreover, the manual creation of sufficiently large lexicons is tedious, costly and requires specialized experts. Even with expert human modeling, homophones still occur in large lexicons. Homophones lead to lexicons with word entries which possess the same pronunciation, or single entries which hold multiple pronunciations. For example, *read* is pronounced differently depending on the verb tense, and *tomato* can be pronounced *tuh-maa-tow* or *tuh-may-tow*. Moreover, the

use of rigid components can lead to out-of-vocabulary errors, which, even with probabilistic smoothing, the predicted words can be the result of high probability bias, instead of deduction based on knowledge or on the context the models can take into consideration.

Finally, end-to-end models can more easily handle the prediction of speech disfluencies, mispronunciations, make out speech from noisy samples and speaker accent [15, 48, 49, 50].

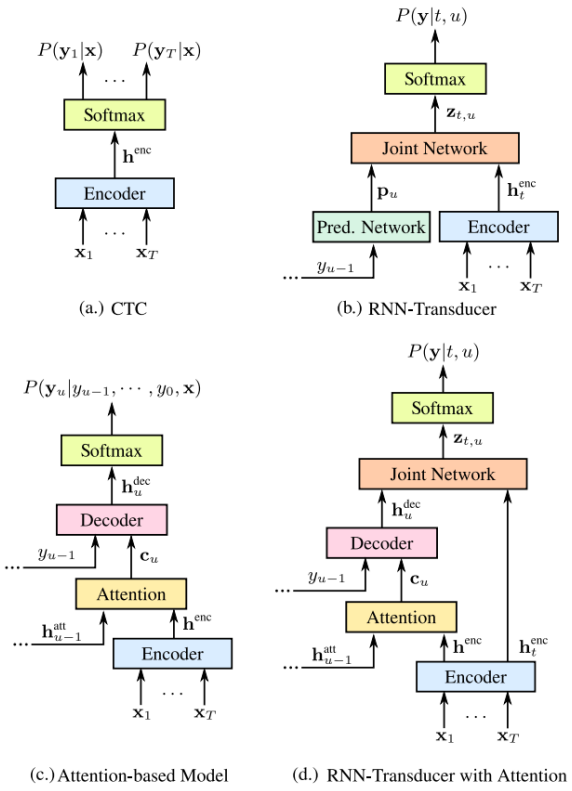

Figure 11: Basic illustrations of the different end-to-end ASR systems [51].

As shown in Figure 11, end-to-end architectures can be categorized in three different approach groups: Connectionist Temporal Classification (CTC), RNN-Transducers, and Attention-based.

The main problem with speech data is the unknown alignment between the characters in the transcript and that of the audio. CTC [1] is a way of getting around not knowing this alignment, and it is especially well suited for ASR applications. Moreover, a new token, ϵ, is introduced to the set of allowed outputs that gets around two alignment pitfalls. First, it solves the need to force every input step to align to some output, as utterances can have stretches of silence with no corresponding output. Second, the inability of producing outputs with multiple characters in a row if we assign an output to each input and then collapse the repeats. This blank ϵ token does not correspond to anything and is removed when obtaining the output.

The CTC alignments result in a more natural way of obtaining the probability of an output sequence given the probabilities at each time-step. These conditional

probabilities are calculated by CTC as:

$$p(Y|X) = \sum_{A \in A_{X,Y}} \prod_{t=1}^{T} p_t(a_t|X) \tag{11}$$

Models trained with CTC typically employ an LM, such as a RNN, to estimate $p_t(a_t|X)$, in order to also account for context. Additionally, the loss can be computed faster with a dynamic programming algorithm, where the main insight is that if two alignments have reached the same output at the same step, we can merge them.

On the other hand, the RNN-Transducer [52] was proposed to solve the two main limitations of CTC: the unawareness of CTC to the context of outputs, and the inability of CTC to map inputs to outputs larger than the inputs. RNN-Transducer can map an input to any number of outputs while also taking into account the context between input and output, thanks to its recurrent prediction network.

Finally, the Attention-based systems [2] use attention mechanisms so that the model can focus on relevant contextual information of the prediction of the next output. This subsequently aids the model to obtain better results on longer sentences, however they obtain acceptable results only when dealing with inputs that approximate the length of the training samples.

3.1.3 Wav2Vec

Wav2Vec [60] is a CNN speech encoder model trained in an unsupervised manner to learn speech representations without the need of labeled data. The model is trained to learn these representations by predicting future samples from a given audio signal context.

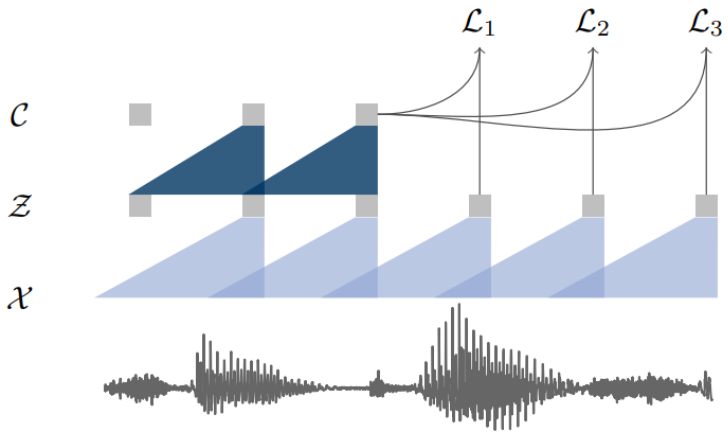

Figure 12: Illustration of Wav2Vec pre-training procedure [60].

As shown in Figure 12, the model is comprised of two CNNs: the encoder network and the context network. Both networks have a causal convolution with a group normalization layer and ReLU nonlinearity. On one hand, the encoder network is

designed as a five layer CNN. It maps the input signal $x_i \in X$ to the low-frequency latent representation $z_i \in Z$, resulting in feature vectors of 30ms of speech every 10ms, such that: $f : X \rightarrow Z$. On the other hand, the context network is designed as a nine layer CNN. It maps v encoder outputs $z_1, ..., z_{i-v} \in Z$ into a single contextualized tensor $c_i \in C$ which covers 210ms of audio, such that: $g : Z \rightarrow C$.

Instead of predicting future samples, the Wav2Vec is trained to distinguish the true sample z_{i+k} at the future step k from the distractors/negatives $\tilde{z}$. The training task is a contrastive loss calculated as:

$$\mathcal{L}_k = -\sum_{i=1}^{T-k}(log\sigma(z_{i+k}^T h_k(c_i)) + \lambda \mathbb{E}_{\tilde{z}\sim p_n}[log\sigma(-\tilde{z}^T h_k(c_i))]) \tag{12}$$

where $\sigma(x)$ is the sigmoid function $1/(1+exp(-x))$, $\sigma(z_{i+k}^T h_k(c_i))$ is the probability of the sample being the correct one, and $h_k(c_i) = W_k c_i + b_k$ is a step-specific affine transformation.

Wav2Vec obtained better results with a substantially smaller amount of data than the state-of-the-art at the time [61].

3.1.4 Wav2Vec2

Wav2Vec2 [3], known officially as Wav2Vec 2.0, is a successor of Wav2Vec. The model is a self-supervised architecture trained from raw audio to learn speech representations without the need of labeled data. Wav2Vec2 is comprised of a multi-layered CNN encoder for obtaining latent speech representations, which are then fed to a Transformer-based model to obtain contextualized representations.

The encoder network, just like Wav2Vec's encoder network, is designed of a multi-layer CNN network. It maps input signals $x_i \in X$ to the latent representations $z_i \in Z$, $z_1 \dots z_T$ for T time-steps, such that: $f : X \rightarrow Z$. However, the network uses a GELU function [62] for it's activation, as opposed to the Wav2Vec encoder which employs ReLU.

The encoder's latent representation outputs are passed to the Transformer-based contextual network as inputs. The number of time-steps of the inputs depend on the encoder's stride. Moreover, relative positional information is learned from embeddings obtained from a convolutional layer designed to learn contextual representations [63]. The context network is then followed by a GELU activation and layer normalization.

Additionally, the model contains a quantization module, where the outputs of the encoder network are discretized to a finite set of speech representations by applying product quantization [64]. To obtain the quantized outputs $q \in \mathbb{R}^f$, the quantization process concatenates chosen quantized representations from different codebooks. Moreover, Gumbel softmax is used for choosing which entry v is taken from a given codebook g, such that:

$$p_{g,v} = \frac{exp(l_{g,v} + n_v)/\tau}{\sum_{k=1}^{V} exp(l_{g,k} + n_k)/\tau} \tag{13}$$

where the encoder network outputs are mapped to $l \in \mathbb{R}^{G\times V}$, $n = -log(-log(u))$, u are samples from the uniform distribution $U(0, 1)$, and τ is a Gumbel-Softmax [65]

temperature variable.

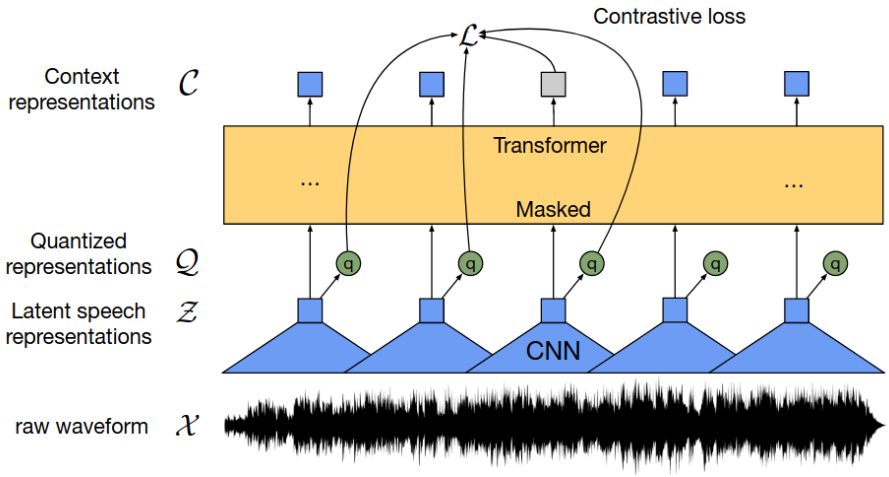

Figure 13: Illustration of Wav2Vec2 pre-training procedure [3].

As illustrated in Figure 13, the pre-training process consists of two main phases. First, similar to BERT's training process, a proportion of the encoder network inputs are masked before being passed to the contextual Transformer network.

Second, the Wav2Vec2 pre-training task solves a combined task. On one hand, the speech audio representations are first learned via a contrastive loss $\mathcal{L}_m$, where true quantized latent speech representations needs to be identified from a set of incorrect representations, or distractors. The contrastive loss is calculated as:

$$\mathcal{L}_m = -log\frac{exp(sim(c_t, q_t)/k}{\sum_{\tilde{q}\sim Q_t}(exp(sim(c_t, \tilde{q})/k)} \tag{14}$$

where q_t is the true quantized latent speech representation, with a set of $K+1$ candidates $\tilde{q} \in Q_t$. Here, Q_t includes the correct sample q_t and K other distractors.

On the other hand, in order to equally use entries in each of the codebooks, the diversity loss $\mathcal{L}_d$ is charged with increasing the amount of quantized codebook representations. The diversity loss maximizes the entropy of the averaged softmax distribution over the codebook entries for each codebook, and is calculated as:

$$\mathcal{L}_d = \frac{1}{GV}\sum_{g=1}^{G} -H(\overline{p}_g) = \frac{1}{GV}\sum_{g=1}^{G}\sum_{v=1}^{V}\overline{p}_{g,v}log\overline{p}_{g,v} \tag{15}$$

where, V is the number of entries in a codebook and G is the number of codebooks.

Therefore, the pre-training task can be described as the need to learn representations of speech by solving the contrastive loss task, and, additionally, compelling the model to use all codebook entries equally as often. This task can be formulated as:

$$\mathcal{L} = \mathcal{L}_m + \alpha\mathcal{L}_d \tag{16}$$

where α is a tuned hyperparameter.

After pre-training is done, Wav2Vec2 is finetuned for ASR by a classification task, where the classes represent the vocabulary, with the aid of a linear classifier appended to the contextual network. The optimization step is done by using the CTC loss and applying masking similar to SpecAugment [66], while keeping the CNN part frozen. By masking to time-steps and channels, these two steps help with learning on unlabeled data, delay overfitting and improve the performance of the model.

3.2 Disfluency detection

Conversational speech is an inherently difficult type of speech for ASR, due to spontaneous errors [53]. For native speaker speech these errors usually are: disfluencies occurring in the moment (repetitions, stuttering), which are usually not edited out from the audio, correct but uncommon word pronunciations, and prosodic variability. All these errors are exacerbated when dealing with unedited L2 conversational speech data, as well as with L2 read-aloud data. For instance, disfluencies and mispronunciations occur more frequently, and usually tend to be linked to the speakers' native language.

In the past, only a small number of research was done with the aim of developing robust models with good performance at detecting speech disfluencies, as a result of a lack of labeled data for a conventional statistical ASR task as well as for its insignificant application potential. This is in contrast to more mainstream ASR tasks such as straightforward speech recognition and speaker identification. Additionally, conventional ASR approaches applied to disfluency detection lacked significant statistical findings and remained moderately stagnant, as neither HMM approaches [54, 55] nor more nuanced probabilistic models [56, 57] reached prominent results to advance the field.

In recent years interest has increased in developing applications for the treatment of stuttering disfluencies as well as tools aimed for L2 self-learning. The intent of these L2 tools is for displaying what pronunciation errors the speaker uttered and what disfluencies were detected, with the aim of correcting and decreasing their occurrence. This new research wave in disfluency detection incorporated into ASR is linked to two factors: first, society's rising regard in recent years at aiding people with speech impediments and easier integration of non-native speakers, and second, the normalization of end-to-end self-supervised systems as the state-of-the-art in ASR tasks.

Forays have been made with end-to-end systems by attempting to detect disfluencies in stuttering therapy [13]. This work obtains good results by using an end-to-end approach as a disfluency detection system combined with ASR, however it only strives at detecting one type of disfluency with a CTC Transformer-based model, the Wav2Vec2, trained on native speech. Moreover, new research expanded the detection task to two disfluency types, namely fillers and hesitations [15], and showed that these systems can also obtain acceptable results for tasks with very different disfluency properties. Their method also outperformed other approaches intended to reduce disfluency related errors. Specifically, a CTC-based Transformer model for

disfluency detection outperformed another CTC-based Transformer approach [58], as well as individual approaches for disfluency removal [10] and hesitation labeling [59].

This thesis will employ the Wav2Vec2 CTC Transformer-based model, in view of the fact that self-supervised end-to-end systems have broken through the disfluency detection bottleneck and show prominent results when working with multiple disfluencies.

4 Experimentation

In this section, the speech datasets and metrics employed are discussed, the experiments carried out are presented and the results obtained are examined. The experiments are tailored to a Finnish L2 ASR approach with an additional disfluency detection. Therefore, a pre-trained multilingual Uralic Wav2Vec2[1] model is employed for the models in this thesis, which will be fine-tuned with Finnish data tailored to L2. Additionally, a supplemental Wav2Vec2 without disfluency detection will be employed as a baseline for comparison analysis of the results.

The word error rate (WER) and character error rate (CER) are used to evaluate the performance of the ASR disfluency detector model. Additionally, Precision and Recall are used to validate the results of the disfluency detector.

4.1 Datasets

The speech recordings used are a subset of the Finnish speech data which has been collected for the DigiTala project [16]. The data from the DigiTala project are used to develop ASR tools that can aid in the examination of L2 Finnish and Swedish skills, and is primarily intented to reduce the human workload in evaluations of Finnish high school examinations. The collected L2 Finnish data are sourced on read-aloud and free-form speaking tasks from high school and university students, and are of gold standard as they have been manually transcribed. In addition to the transcripts, the tasks are graded by multiple human examiners in accordance to L2 language skills.

Data	Total	Fluent	Non-fluent
Lukio	2966	730	2236
Aalto	1883	551	1332
YKI	386	0	386

Table 1: Fluent, non-fluent and total amount of recordings in Lukio, Aalto and YKI compilations.

The DigiTala project's data has been expanded during 2021 and 2022, and the L2 Finnish data currently consists of 5235 transcribed speech samples, where the recordings have a total duration of approximately 1822 minutes. The average duration of a sample is of 20.8 seconds, but they can vary from simple tasks formed of a few words to long sentences. The recordings, as shown in Table 1, have been obtained from speakers of Lukio (higher education high-schools), from Aalto university and from YKI examinations; the latter are certificates of language proficiency for adults. The number of Lukio and Aalto recordings, 2966 and 1883 respectively, is much higher than YKI at only 386, but the recordings from YKI are considerably longer than the rest, as they come from lengthy oral examinations. Moreover, the transcripts

[1] https://huggingface.co/Finnish-NLP/wav2vec2-large-uralic-voxpopuli-v2-finnish

of the recordings can contain the errors and disfluencies the speaker may have generated while talking. YKI is unique because it contains only non-fluent samples, samples where there are errors or disfluencies, in view of the fact that they are long examination recordings and the speaker is bound to make some mistake. On the other hand, the Lukio recordings have 730 fluent and 2236 non-fluent samples, and, the Aalto recordings have 551 fluent and 1332 non-fluent samples. Additionally, the non-fluent transcripts contain the following tags:

- Hesitations are marked with the *<hesitation>* tag along with the hesitation's text. Some resemble the form: *öö<hesitation>*, *mm<hesitation>*, *aa<hesitation>*. Additionally, irregardless of their length, the hesitations are transcribed with only two letters: *ööööööö* is transcribed as *öö<hesitation>*.
- Paralinguistic events such as laughter, coughing, sighing, exclamations, etc., are marked with the *<paral>* tag.
- Translinguistics, or code-switching, words employed by the speaker from any language other than Finnish, are marked as *<trans>*. Moreover, if the transcriber recognizes the language the word originates from, then it must be tagged accordingly: if the word is in English, then it is tagged as *<transen>*; if the word is in Swedish, then it is tagged as *<transsv>*, etc.
- Unclear words and non-words are marked under the general *<garbage>* tag.
- Names of people, places and things are marked with a following *<name>* tag when the name is known by the transcriber, for example: *Mallorca<name>*, *Aapo<name>*; otherwise, if the name is not understood then with a white-space followed by *<name>*.
- Partially correct words are annotated between asterisks and the incorrect form must be maintained, for example: the correct word is *muusikko*, the person pronounces it incorrectly as *musikko* and the transcriber marks it as **musikko**.
- Repetitions and stutterings are marked with a hyphen, for example: *jalkapalloa ja* ***sul-*** ***sul****kapalloa*.

However, only a handful of tags are actual disfluencies, and, as such, relevant to the L2 ASR disfluency detector. Therefore, the following disfluencies will be used and the rest will have their tags removed from the transcripts:

- Hesitations followed by *<hesitation>*.
- Paralinguistics marked as *<paral>*.
- Translinguistics of any language will be changed to the general tag *<trans>*, and used as is.
- Partially correct words marked with asterisks.

- Repetitions marked with hyphen.

As shown in Table 2, the number of *<hesitation>* tags in the data is of 1346 for Lukio and 581 for Aalto. The number of *<paral>* tags is of 933 and 14 for Aalto. The number of *<trans>* tags is of 72 for Lukio and 29 for Aalto. The number of partially correct and repetition tags is of 3069 and 351 respectively for Lukio, 3127 and 118 respectively for Aalto, and, 5835 and 1428 respectively for YKI. None of the previously mentioned disfluency types, besides partially correct words and repetitions, are found in the YKI data. This is as a result of the transcriber tasks not containing the same instructions as for the Lukio and Aalto tasks, only *<garbage>* tags are found in YKI and these are not usable.

Disfluency	Lukio	Aalto	YKI
`<hesitation>`	1346	581	-
`<paral>`	933	14	-
`<trans>`	72	29	-
`*partial*`	3069	3127	5835
`-repetition-`	351	118	1428

Table 2: Number of disfluency types in the Lukio, Aalto and YKI transcripts. Partially correct words are shown as **partial**, and repetitions are shown as *-repetition-*.

As a note, the YKI data is not be employed for the model, as the data contains 11327 *<garbage>* tags and the recordings are exceedingly long to be used as inputs. Additionally, processing the transcripts, as well as the recordings, in order to remove these tags and also splitting the recordings might not have been a viable solution. Moreover, the L2 ASR baseline used to compare to the L2 ASR disfluency detector does not make use of the YKI data, therefore the comparisons are of two models more alike to eachother. By using only the Lukio and Aalto recordings, the data has a total of 320 unique speakers.

Additional processing of the different disfluency tags has been carried out, as there are many errors commited by the transcribers when marking these tags and they had to be correctly cleaned. For example, many *<hesitation>* tags were transcribed incorrectly as *<hesitate>*, *<hesittate>* or *<hesitatae>*, among others. Finally, all of the disfluency tags are changed to the general tag *<disfluency>*, as the L2 ASR disfluency model is not multi-class. This has also been done to tags of partially correct words by removing the asterisks, for instance **musikko** will change to *musikko<disfluency>*. And with repetitions by removing the hyphens, *jalkapalloa ja sul- sulkapalloa* will change to *jalkapalloa ja sul<disfluency> sulkapalloa*.

The audio recordings are converted to monophonic (mono) audio and resampled at a rate of 16000 Hz with 16 bits per channel. On the other hand, the transcripts have been preprocessed by converting all letters to lowercase and removing all punctuation except apostrophe. Finally, the Finnish character *å* has been added, special tags for unknown tokens and padding tokens have been inserted in the dataset's vocabulary, and all other characters not found in the Finnish alphabet have been removed.

The common approach for the process of building the L2 ASR disfluency models has been followed, and the data has been split into three unique sets to be used in the three stages of building the model: training, validating and testing. This division of the data is a straightforward random split where 80% will be used for training, 10% will be used for validating the training process, and 10% will be used for testing the model once the training has concluded. The validation will be mentioned as the development (dev) set. Although, a unique condition has to be upheld in order for the three processes to be independent between eachother, that is, none of the speakers found in any one set, should be found in any other, for example: *speaker_156* is found in the training split, thus, it is not present neither in the dev or the testing sets. This stipulation was taken into consideration when performing the random split mentioned above, and, as shown in Table 3, from the total of 320 speakers, the training set contains 256, the validation set 32, and the training set 32. Having unique speakers in each of the sets will increase the confidence of the results obtained, as the testing process will not be adulterated by having the results influenced by the model having learned on speakers found also in the testing set. The total duration, the number of samples and disfluencies in the dev and test sets are approximately the same: 1.98 hours and 2.06 hours respectively, 485 samples and 484 samples respectively, and, 348 disfluencies and 362 disfluencies respectively. On the other hand, the training set contains 256 unique speakers, a duration of 15.39 hours, 3862 samples and 2741 disfluencies.

Split	Speakers	Duration, h	Samples	Disfluencies
`Train`	256	15.39	3862	2741
`Dev`	32	1.98	485	348
`Test`	32	2.06	484	362

Table 3: Amount of speaker, duration in hours, number of samples and *<disfluency>* tags for the data splits.

All in all, the training, dev and testing sets have a proportional number of speakers, total duration of samples, number of samples and of disfluency tags. Finally, the distribution of sample lengths for these splits has been examined in order to further validate their correct use. As shown in Figures A1, A2 and A3, the sample lengths for each of the sets are similarly distributed, with most of the data being short length samples of less than 25 words per transcript, a smaller spread of samples with larger lengths as well as a few outlier samples of more than 125 words.

4.2 Metrics

To evaluate the ASR performance of the system, the WER and CER metrics are used. WER is a popular metric used for performance assessment of ASR and NLP systems. It is based on the Levenshtein distance [67], and measures the performance at the word level. Additionally, there is an inherent difficulty produced by the fact that generated sequences can have different lengths from the reference sequence, the

metric works around this by first aligning the predicted sequence with the reference using dynamic string alignment. The WER metric calculates the percentage of predicted words that are incorrectly produced when comparing the sequence to that of the original reference sequence. It computes the percentage by taking into consideration the sum of the substitutions, insertions and deletions in the predicted sequence and dividing it by the number of words in the reference. Although the WER metric is used as the main performance indicator, the CER metric is also employed to analyze the ASR results. CER is similar to WER, and it calculates the percentage in the same way but on the character level. It is a useful metric characterized by a robustness when comparing words which might differ from the reference, as a slight deviation in the characters predicted means only a slight increase in the CER. Although this seems as an apparent advantage over WER, as a single character difference would increase the WER significantly more, both metrics are complementary when analyzing the performance of the ASR and lower values of both indicate good performance for ASR.

Moreover, the Precision and Recall metrics are employed to analyze the performance of the disfluency detection model by taking into consideration the disfluencies predicted by the model. In this work, the Precision is the proportion of tokens predicted as disfluencies by the model that are labeled correctly as such, it is thus obtained as the true positives divided by the sum of true positives and false positives. In turn, the Recall is the proportion of tokens correctly labeled as disfluencies which are correctly predicted as such by the model, and is obtained as the true positives divided by the sum of true positives and false negatives. For these two metrics, higher values indicate a better performance.

4.3 Pre-trained model and baseline

The base model employed for the L2 ASR disfluency detector will be a pre-trained, publicly available Facebook Wav2Vec2 model, namely *wav2vec2-large-uralic-voxpopuli-v2*[2]. This model contains approximately 317 million parameters and weighs roughly 1.27 GB. As shown in Table 4, it has been pre-trained on a total of 42.5k hours of Uralic language family unlabeled data: 14.2k hours of Finnish, 10.6k hours of Estonian and 17.7k hours of Hungarian speech, sampled at 16000 Hz from the VoxPopuli v2 dataset. The VoxPopuli dataset [68] is currently the largest publicly available multilingual dataset, gathered and processed by Facebook, for the purpose of unsupervised and semisupervised learning. Most importantly, it is comprised of 400k hours of untranscribed speech from 23 languages, from which the Uralic languages are used for the models in this work.

Additionally, an L2 ASR Wav2Vec2 model implemented by Yaroslav Getman [69] is used in order to compare the results obtained by the L2 ASR disfluency detector models in this project. This model, labeled *comparison_model*, is also pre-trained on all the Uralic languages from the VoxPopuli v2 dataset, including the 14.2k hours of Finnish. Moreover, it is finetuned for L2 ASR on the YKI data, but only on the

[2]https://huggingface.co/Finnish-NLP/wav2vec2-large-uralic-voxpopuli-v2-finnish

wav2vec2-large-uralic-voxpopuli-v2				
	Finnish	**Estonian**	**Hungarian**	**Total**
`Hours`	14.2k	10.6k	17.7k	42.5k

Table 4: Unlabeled data from Finnish, Estonian and Hungarian the base model has been pre-trained on from Facebook's VoxPopuli v2 dataset.

first data collection trial of Lukio data, as opposed to the models in this work, which use the second data collection trial as well.

4.4 Finetuning

4.4.1 Finetuning environment and hyper-parameters

Model finetuning has been carried out using Aalto University's Triton high-performance computing cluster. This service provides Aalto researchers and students with powerful environments on which to run GPU-intensive computing processes such as the finetuning operations of this work's L2 ASR disfluency detector models.

For finetuning, Triton's *gpu[1-10]* nodes and *dgx[1-7]* nodes have been used, which are comprised of Nvidia Tesla V100 graphic cards with 5120 Nvidia CUDA cores, and, 32GB and 16GB of GPU memory respectively. Moreover, the *dgx[1-7]* nodes provide a priority assignment of resources for the researchers of the ELEC department. Additionally, Triton provides a work folder with an approximate minimum of 200GB of storage, which have been used in this work for the storing of code, model checkpoints and data.

In regard to the model hyperparameters and fitting settings, both the baseline and the L2 ASR disfluency detector models employ the same parameters. The models are trained for a total of 40 epochs, and carry out WER and CER evaluation after each epoch. Additionally, up to 15 checkpoints are saved at a time, and after the training process is finished the best model with smallest WER is chosen from the saved checkpoints. Given the immense memory requirements of the Wav2Vec2 model, gradient checkpointing [70] is employed in order to reduce memory usage. Gradient checkpointing achieves this memory performance boost by trading off more computational time required, as certain layers' activations are cleared and recomputed during backward passes. Moreover, a small batch size of 2 is used for both the training and evaluation batches, in order to prevent inevitable memory issues that appear during the training process of large models which additionally make use of large data inputs. Lastly, a learning rate of 0.0003 and warmup ratio of 0.1 is used.

4.4.2 L2 Finnish ASR baseline finetuning

In the process of obtaining an L2 ASR disfluency detector, initial experiments consist of obtaining a new baseline L2 Finnish ASR model with comparable or better performance than the L2 Finnish *comparison_model*. The previously mentioned

model was trained for a duration of 70-80 epochs, on a first compilation of L2 Lukio data collected before Autumn of 2021 and on the YKI data, and the new baseline, denominated *disfl_baseline*, will be trained on the full Aalto L2 data and on the full Lukio L2 data collected to date as seen on Table 1. From the current data, the same splits as seen in Table 3 will be employed for training. Moreover, the <*disfluency*> tags will not be utilized, with the purpose of first obtaining a reliable ASR baseline to compare with the *comparison_model*. Both models have been evaluated on the same testing set from Table 3.

Model Name	Train size, h	WER, %	CER, %
comparison_model	≈ 14.60	35.47	11.09
disfl_baseline	≈ 15.39	29.09	9.08

Table 5: Results for the comparison model and the baseline used for the disfluency detector.

As seen in Table 5, the baseline developed for the disfluency detector obtains a WER of 29.09 %, with an absolute improvement of 6.38 % over the *comparison_model*, which obtains 35.47 %. This increase in WER efficiency is most likely correlated with the longer total duration of the data which the baseline has been trained with, as it is an increase of 0.79 hours over the comparison model. Similarly the improvement in WER, the baseline model obtains a lower CER of 9.08 %, while the comparison model results in a CER of 11.09 %. The baseline results are satisfactory when compared to the *comparison_model*, consequently, the same pre-trained model and it's settings will be employed for the finetuning of the L2 ASR disfluency detector.

Figure 14 shows the progress of the model after each epoch of the finetuning process. The training for the baseline model stops after the specified 40 epochs, which is where the model reaches the WER and CER scores of Table 5. By observing the graph, the likely possibility is that if the number of epochs was higher, the baseline model could have achieved better results than the current ones, but the purpose of these comparisons was to simply validate the use of the chosen model as the disfluency detector.

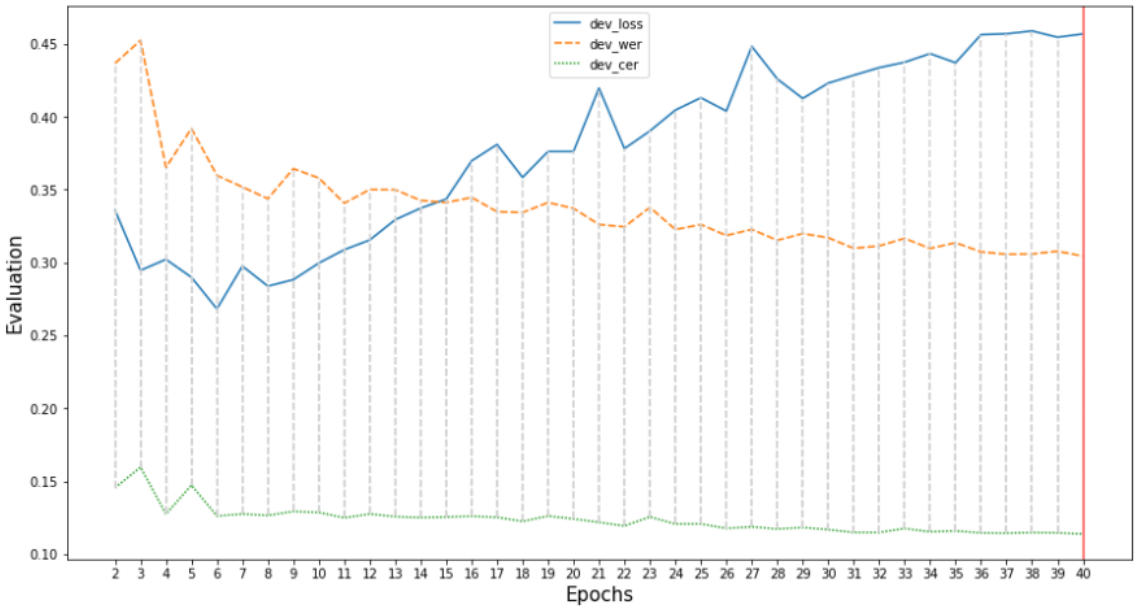

Figure 14: Time series graph indicating the model's loss, WER and CER on the dev set while finetuning.

Moreover, the baseline model required approximately 20 hours of finetuning for 40 epochs, an additional increase in number of epochs would not be necessary, as with the current amount the comparisons already indicate that the baseline is a sufficiently good choice for the disfluency detector.

4.4.3 L2 Finnish ASR disfluency detector finetuning

The L2 Finnish ASR disfluency detector has been trained by following the baseline's training hyperparameters, employing 40 epochs for a total duration of approximately 21 hours. The training set used is mentioned in Table 3, and contains the *<disfluency>* tags indicating the disruptions in speech declared prior.

The results of the L2 Finnish ASR disfluency detector are shown in Table 6. The disfluency detector, *disfl_detector*, reaches 30.41 % WER and 13.17 % CER after 40 epochs. Moreover, in order to examine whether the model might achieve better results with more epochs, another model has been finetuned with 60 epochs for a duration of approximately 31 hours. The results for this model, *disfl_detector_60ep*, are very similar to the latter. For the WER, it obtains an insignificant improvement of 0.03 %, with a WER of 30.38 %. And, in contrast, it slightly worsens for the CER as it results in 13.36 % CER, an increase of 0.19 % over the model trained for 40 epochs. Therefore, the results obtained by the model trained for 60 epochs does not justify the use of this model, particularly when also taking into account the rise in total training time.

Model Name	WER, %	CER, %
disfl_detector	30.41	13.17
disfl_detector_60ep	30.38	13.36

Table 6: Results obtained by the L2 Finnish ASR disfluency detector.

The progress after each epoch during finetuning can be observed in Figure 15 for the model trained for 40 epochs, and in Figure 16 for the model trained for 60 epochs.

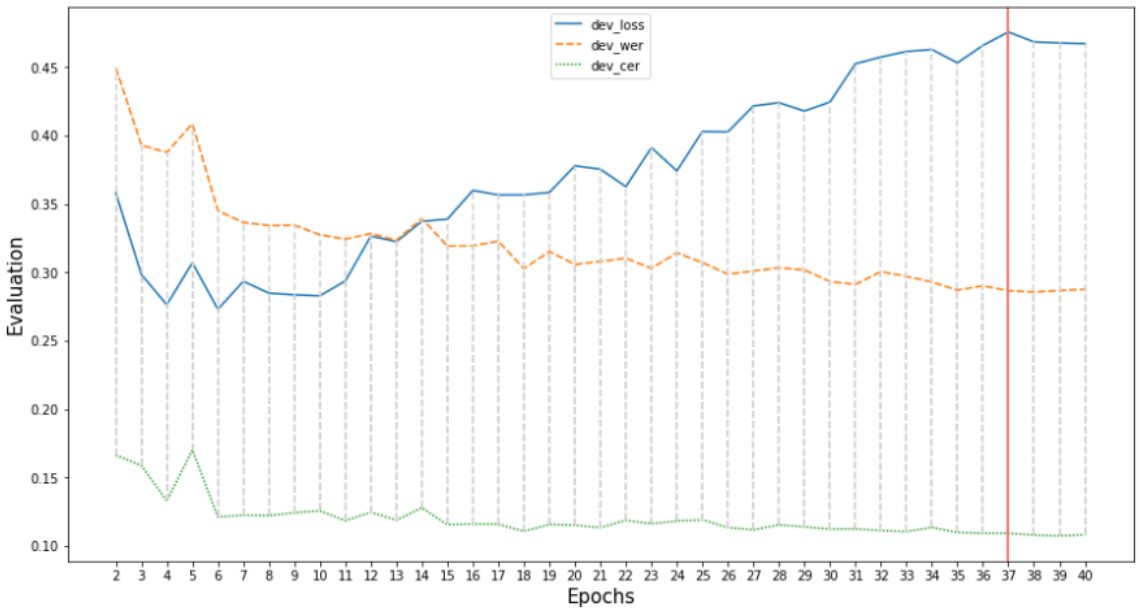

Figure 15: Time series graph indicating the disfluency detector model's loss, WER and CER on the dev set while finetuning for 40 epochs.

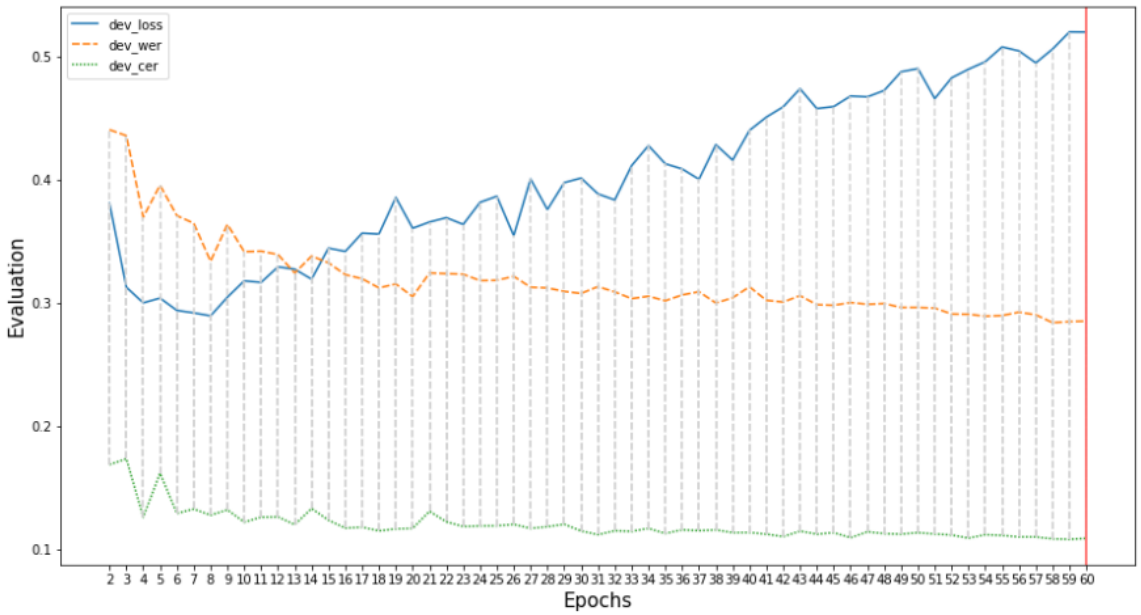

Figure 16: Time series graph indicating the disfluency detector model's loss, WER and CER on the dev set while finetuning for 60 epochs.

4.4.4 L2 Finnish ASR Disfluency Detector with Curriculum Learning

In addition to the standard finetuning which employs randomly shuffled data to train, this work also studies the possibility of improving the performance of the model via a Curriculum Learning (CL) [71, 72] strategy. The CL strategy is based on the educational process humans undergo in order to reach a certain level of expertise needed to perform in their line of work. This educational process gradually introduces new concepts with an increasing level of difficulty, often in relation to previously learned ones, in order to successfully guide a person through a meaningful learning regime. In ML, the basic idea of applying CL to the training process is to start with data which might be easier for the model to understand, and then gradually increase the difficulty as the model learns.

Model Name	WER, %	CER, %
disfl_detector_cl	30.77	13.79

Table 7: Results obtained by the L2 Finnish ASR disfluency detector when using CL strategy.

There are multiple types of CL strategies one might employ depending on the task and the angle of choosing the difficulty. In the context of ASR such strategies include: first starting with short length audio samples, then increasing the length, or, beginning with clearer samples and gradually introducing samples with more noise present.

In this project, the CL strategy implemented is to progressively feed the model increasingly disfluent samples. For this strategy, the same training, dev and test sets as mentioned in Table 3 are used. However, instead of randomly sampling the inputs from the training set, the split is first sorted according to how fluent the samples are: the first samples contain no disfluency tags, while latter samples contain the most. Therefore, the model learns fluent speech before introducing harder samples where the speaker produced disfluency mistakes.

The results for the L2 Finnish ASR disfluency detector model using the CL strategy can be observed in Table 7. The model obtains 30.77 % WER and 13.79 % CER after 40 epochs of finetuning.

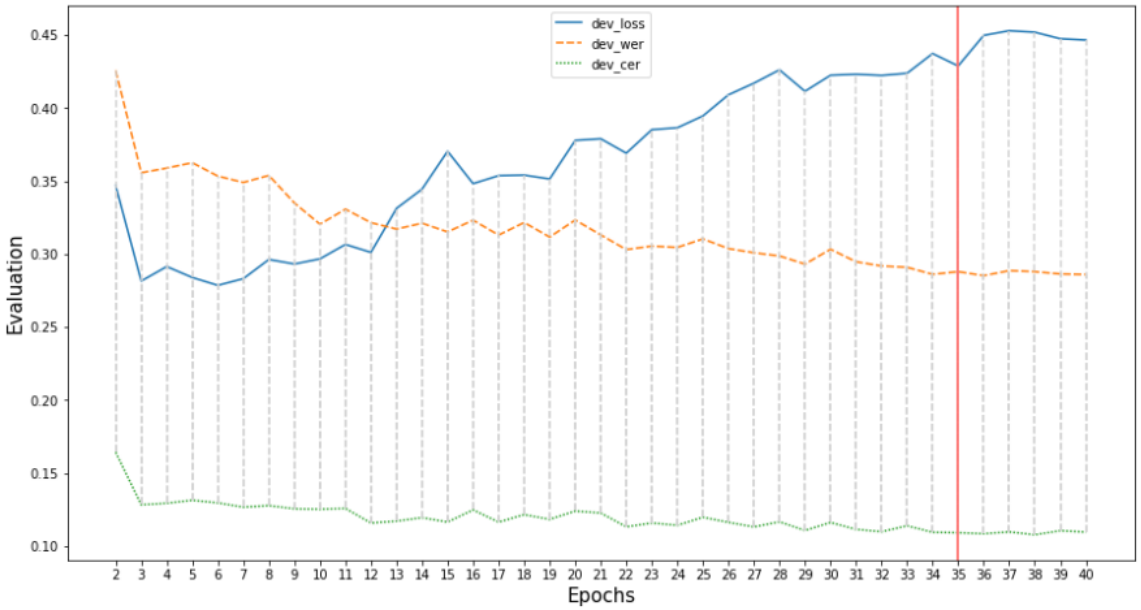

Figure 17: Time series graph indicating the disfluency detector model's loss, WER and CER on the dev set while finetuning, when using CL strategy.

Moreover, the progress after each epoch during finetuning can be observed in Figure 17. Compared to the other finetuned models, shown in Figures 14 and 15, the one employing CL seems to stabilize faster during the first few epochs, and reaches the best results in less iterations, for example, *disfl_detector_cl* reaches the best model in epoch 35, while *disfl_detector* reaches it in epoch 37.

4.5 Results

4.5.1 Combined results comparison

Considering the number of different models that have been finetuned, their respective results are abridged in Table 8. Moreover, the baseline model, and the comparison model *comparison_model*, are included as well.

Model Name	WER, %	CER, %
comparison_model	35.47	11.09
disfl_baseline	29.09	9.08
disfl_detector	30.41	13.17
disfl_detector_60ep	30.38	13.36
disfl_detector_cl	30.77	13.79

Table 8: Combined results of models finetuned for the L2 Finnish ASR disfluency detector task, as well as the comparison and the baseline models.

The baseline comparison model *disfl_baseline* offers great improvement in results over those of *comparison_model*. It reaches 29.09 % WER and 9.08 % CER, when compared to the latter's results of 35.47 % WER and 11.09 % CER. As has been mentioned, these results are a good indication of the model's ASR capabilities, and has been used as the base for the finetuning of the L2 Finnish ASR disfluency detector models.

There are three models that have been finetuned based on the baseline. Models *disfl_detector*, and *disfl_detector_60ep*, have been finetuned with randomly sampled data from the training set, for a duration of 40 epochs for the first and 60 epochs for the second. The 40 epoch model reaches a WER of 30.41 % and CER of 13.17 %, indicating a slight increase of 1.32 % in WER and a more tangible rise of 4.09 % in CER. The 60 epoch model obtains a WER of 30.38 % and CER of 13.36 %, a respective rise of 1.29 % in WER and 4.28 % in CER. Moreover, *disfl_detector_cl*, trained with a CL strategy for input sampling, has obtained minimally worst results. It achieves a WER of 30.77 % and CER of 13.79 %, an increase of 1.68 % in WER and 4.71 % in CER, when compared to the baseline results.

Model *disfl_detector*, henceforth identified simply as the disfluency detector, has been chosen as the best out of the three finetuned detectors. This is in view of the fact that, on the one hand, it obtains similar results as the one trained for 60 epochs, with considerably less time required for finetuning. And, on the other hand, it obtains better results than the CL strategy model, which, although only slightly worse, does succeed in stabilizing the model faster during the first few iterations of training.

As mentioned, the disfluency detector obtains a negligible increase of 1.32 % WER, but, although it is the best model out of the three finetuned, it nonetheless achieves a substantial increase of 4.09 % in CER. This rise in WER and CER develops as the result of the same problem: the detection of *<disfluency>* tags from speech and the consequent unalignment of original transcript with the predicted one. This issue is discussed in subsequent sections; as a summary, when the disfluency detector incorrectly recognizes a *<disfluency>* tag, the following predicted transcript becomes unaligned to some degree and the WER and CER increase. The CER rises more than the WER, as it calculates the correct alignment of each character instead of word.

4.5.2 WER and CER analysis

The WER and CER metrics of the disfluency detector have been analyzed to verify their validity by means of different exploratory graphs.

Figures 18 and 19 show the distribution of WER and CER scores respectively in two separate violin graphs obtained from the test set. The CER distribution is heavily skewed to the left, close to the average CER score of 0.1317. This is the desired and correct result, and only a few CER scores are outliers. The WER score distribution is also heavily skewed towards the left, close to the WER score mean of 0.3041. However, there are more outliers present than in the CER distribution, and some are concentrated around a score of 1. This represents a small number of test samples comprised only of sentences containing one or a few words. Here, the WER score is 1 when the model has incorrectly predicted the word, or it obtains a higher than 1 WER score when it has inserted additional words. Such behaviour is expected, as even one erroneous character in a sample with one word results in a WER of 1.

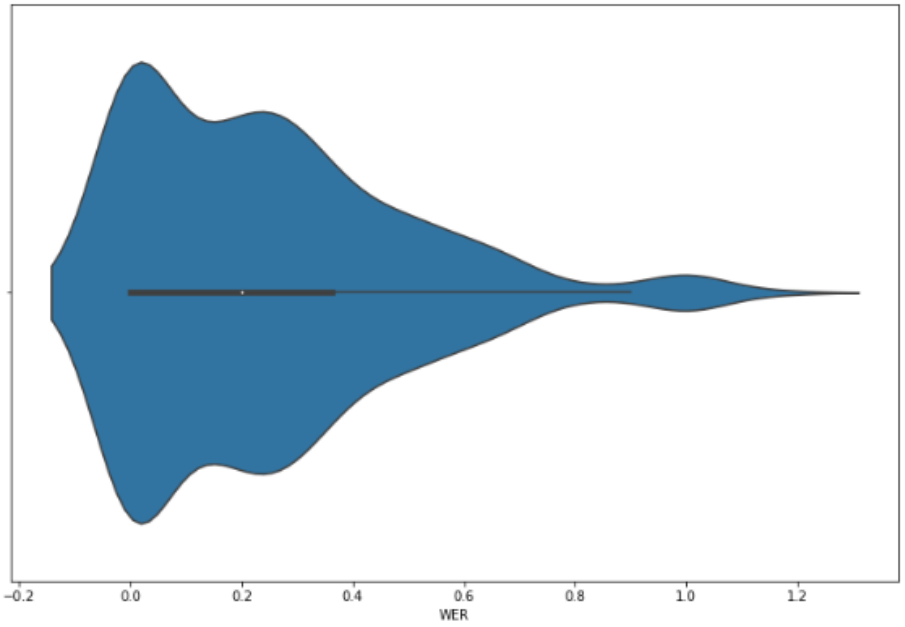

Figure 18: WER scores violin graph. Vertical axis depicts the amount of samples for each WER value. Negative horizontal axis values are due to the figure generation.

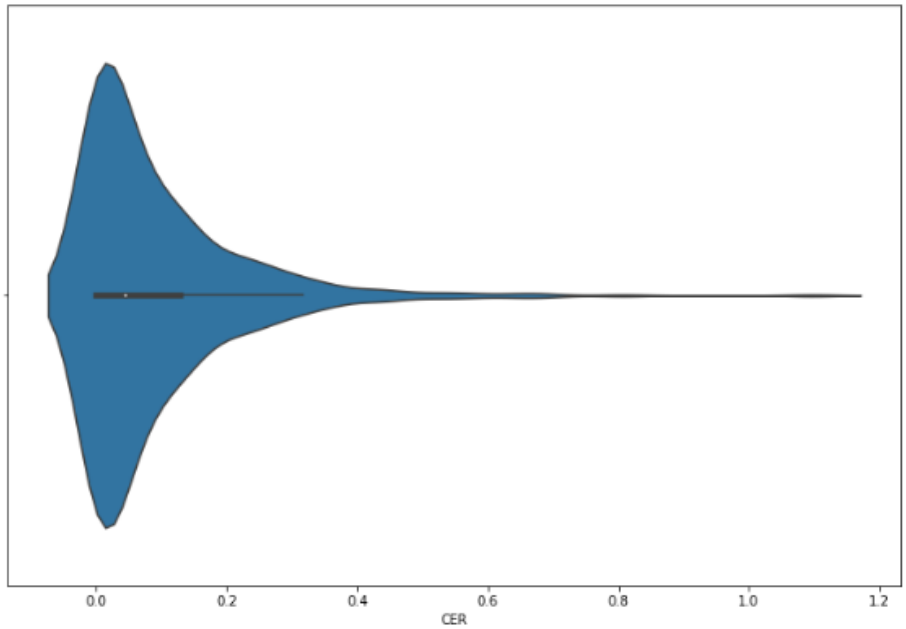

Figure 19: CER scores violin graph. Vertical axis depicts the amount of samples for each CER value. Negative horizontal axis values are due to the figure generation.

The previous characteristic is further proven in Figures 20 and 21, which respectively indicate the WER and CER scores as a function of test set sample lengths.

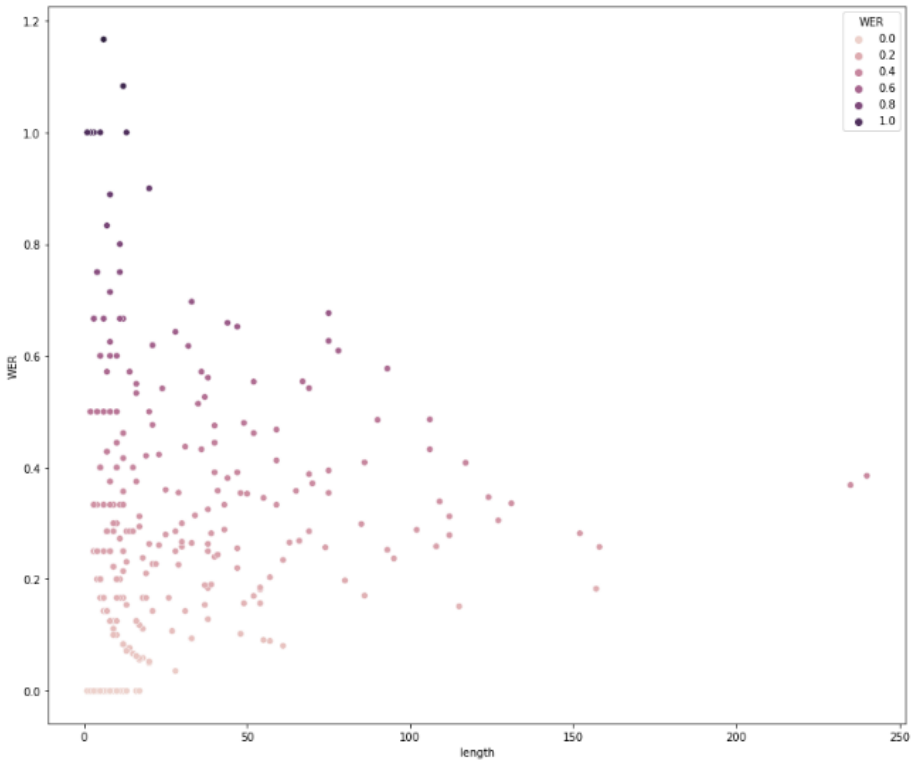

Figure 20: Scatter plot of WER scores as a function of sample length.

As can be observed, the model performs uniformly, close to the WER and CER means, independently of the sample lengths. It obtains great results with smaller samples, when the samples are longer than 100 words, and surprisingly well for samples of more than 200 words. However, there are outliers for both WER and CER scores when dealing with very short utterances, as a consequence of incorrectly predicting even a single character. Nevertheless, the results are more than satisfactory, with only a few outliers of such nature, which does not indicate the model as incapable of performing for the ASR task.

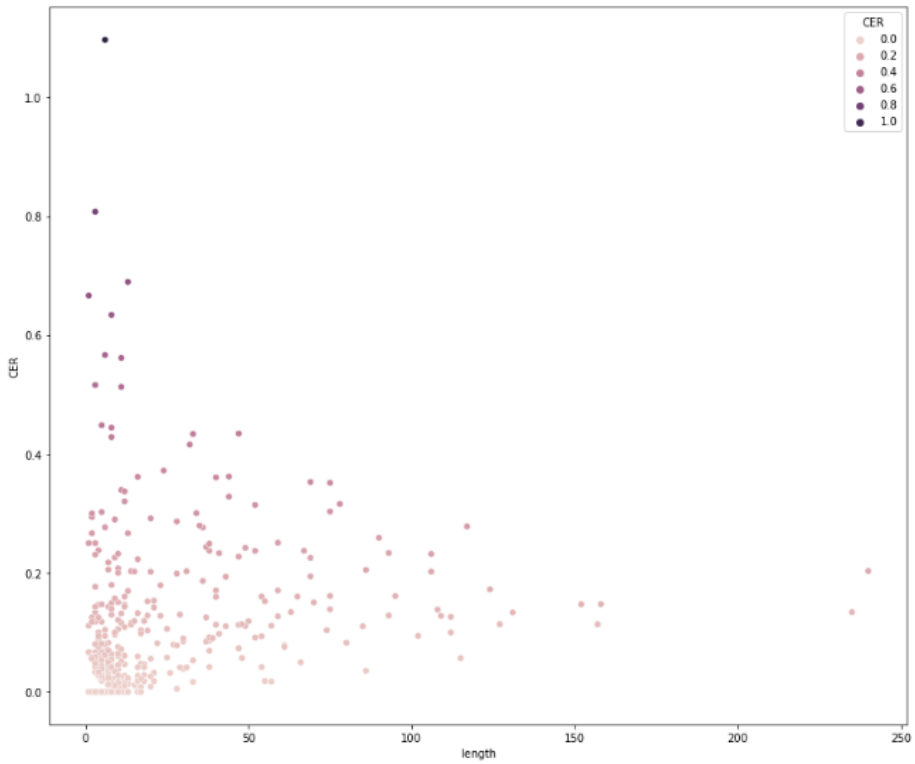

Figure 21: Scatter plot of CER scores as a function of sample length.

The ASR capability of the model has been demonstrated with the previous examinations. However, another analysis performed examines the possible relation between a speaker's proficiency level and the WER score obtained for the samples of said speakers. In addition to the audio and transcripts, the Lukio and Aalto L2 Finnish datasets contain the proficiency of each speaker as evaluated in accordance to official CEFR language skills. There are usually multiple proficiency scores per speaker done by different raters, and they have been averaged to ease representation. The relation between the WER score and the proficiency of the speakers is shown in Figure 22. In the data, the proficiency levels are marked in an ascending order as A1, A2, B1, B2, C1, C2, however in the illustration, they are equivalently labeled 1, 2, 3, 4, 5, 6, with label 0 reserved for speakers without any proficiency score assigned.

It must be noted that a proficiency of C2 is the maximum proficiency one can obtain via CEFR standards, while A1 is considered the beginner level.

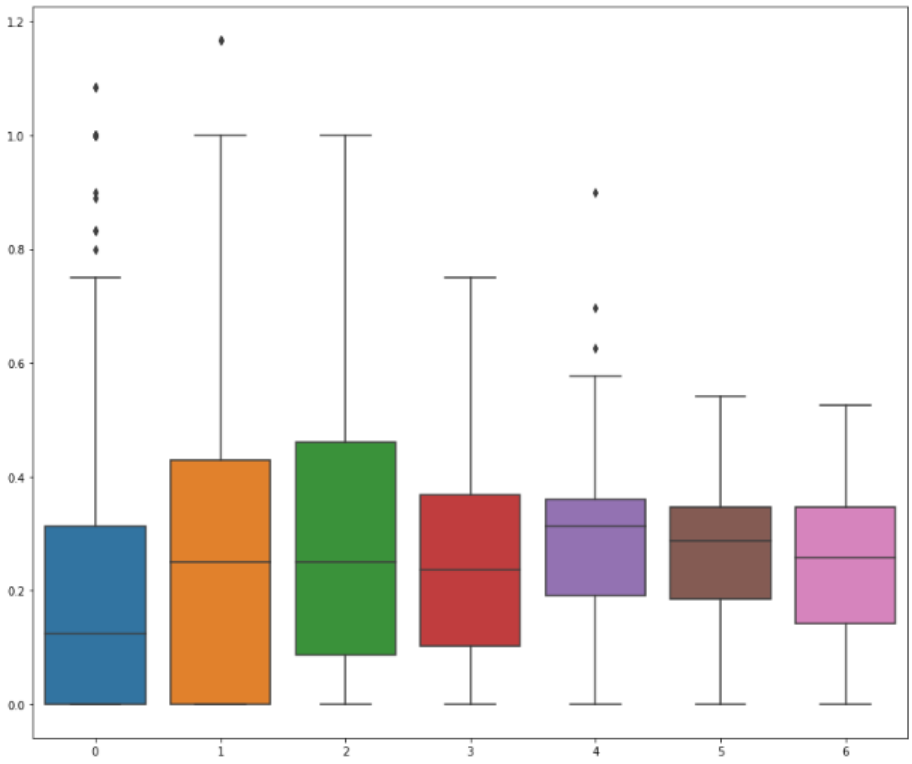

Figure 22: Box plot indicating the relation between the WER score (y-axis) and speaker proficiency (x-axis).

From the representation in Figure 22, we can observe that the WER goes in line with the proficiency of speakers, which was the expected relation between the two attributes. While most categories have a similar distribution of WER errors in their respective samples, there are a few indications that the speakers improve in higher categories. Category 0, 1 and 2 have a wider distribution, with scores mainly ranging from 0 % and reaching $\approx$ 30 %. This can be due to the tasks being shorter and easier, as they are tailored to beginner speakers. Nonetheless, these proficiencies have longer tails towards higher error scores, as well as more detrimental outliers. In contrast, the intermediate proficiencies 3 and 4 have a narrower distribution, shorter tails and less outliers, when compared to the previous categories. This can be seen as evidence of skill improvement, as they tend to make less mistakes while also undergoing more difficult and longer evaluations. Finally, the same pattern occurs with categories 5 and 6, which incidentally are narrower than all the other proficiencies, contain no outliers and have the shortest tails.

4.6 Validation of Disfluency Detector predictions

4.6.1 Kaldi metrics

Although the WER and CER scores for the disfluency detector are satisfactory, the generated results need to be scrutinized to guarantee they fulfill acceptable expectations. Therefore, the validity of the results has been analyzed with tools

such as Kaldi [73], the Precision and Recall metrics, and other visual exploration techniques.

The WER and CER metrics are derived from the Levenshtein distance, and before calculating the error score, the algorithms first align the two transcripts. However, in order to manually analyze and validate the predicted transcripts, they must first be aligned to the original gold transcripts. Therefore, Kaldi has been used to examine these results, given that this toolkit employs a distinct method for sentence alignment.

Kaldi [73] is an open-source toolkit designed for ASR development, and offers features such as: code-level integration with Finite State Transducers, extensive linear algebra support, matrix library, metrics, acoustic modeling, alignment, among others. Kaldi's alignment library has been employed for result validation, and it makes use of human-readable representations of sequences of Hidden Markov Model states taken by the best-path Viterbi [74] algorithm alignment of an utterance.

Additionally, before inspecting the model's alignments using Kaldi, the CSID has been obtained employing Kaldi's alignment algorithm. The CSID of the alignment of a predicted utterance and original transcript represents the correctly (C) predicted words, the substitutions (S), the insertions (I), and the deletions (D). For a robust model, the desirable CSID contains substantially more correctly predicted words, than substitutions, insertions or deletions. Moreover, as a general rule, the number of insertions and deletions must be considerably smaller than the number of correctly predicted words and substitutions.

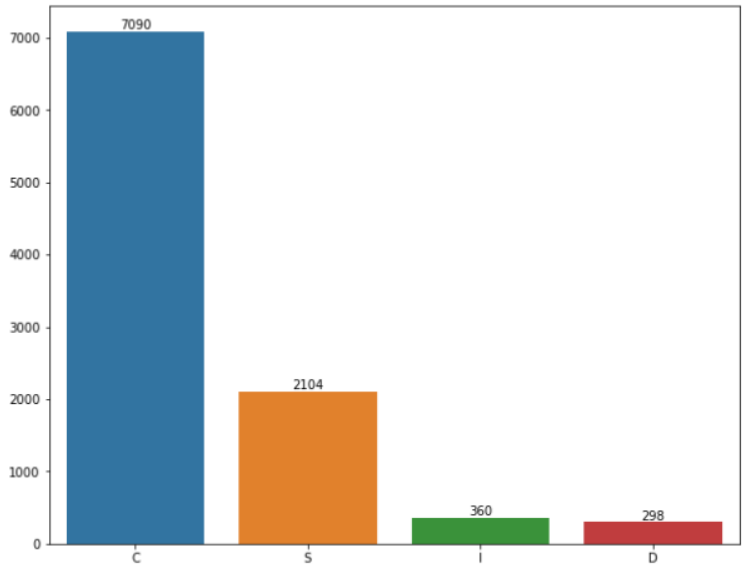

Figure 23: Kaldi obtained CSID for the L2 ASR baseline model.

Figures 23, 24 and 25 show the CSID's of the L2 ASR baseline model, the L2 ASR disfluency detector model and L2 ASR disfluency model with CL strategy respectively. As can be seen, the CSID trend for each of the models is practically identical, further indicating that the disfluency detector models do not deviate from

the baseline model. The figures show a substantial difference between the number of correctly predicted words and substitutions, insertions and deletions. Moreover, the insertions and deletions comprise a small percentage of predicted words.

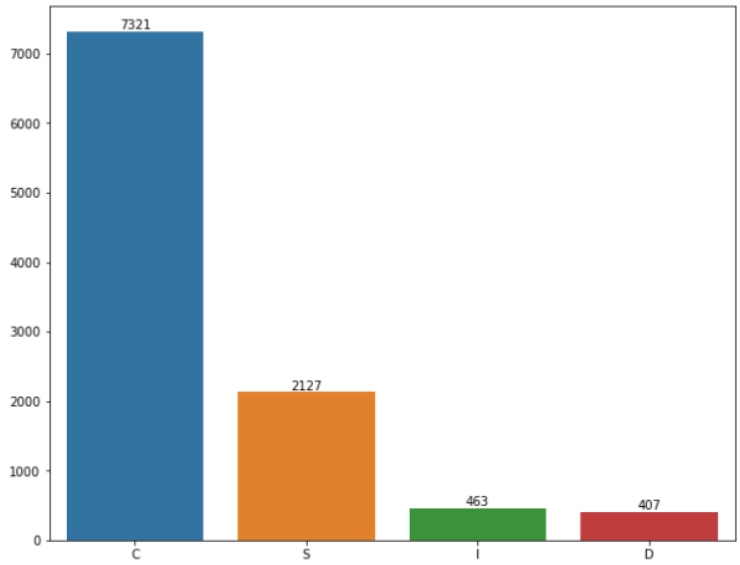

Figure 24: Kaldi obtained CSID for the L2 ASR Disfluency Detector model.

There is a considerable number of substitutions, higher than insertions and deletions, and they are an indication of the models generating errors at the character level. Furthermore, many insertions and deletions happen as a consequence of misalignment between the original transcript and the predicted one because of human errors in transcribing or the model not being able to differentiate between complex words, for example an insertion can happen when the original transcript has the word “vuonnatuhatyhdeksänsataakahdeksankymmentäyksi” and the model inserts an additional word such that it predicts “vuonna tuhatyhdeksänsataakahdeksankymmentäyksi”.

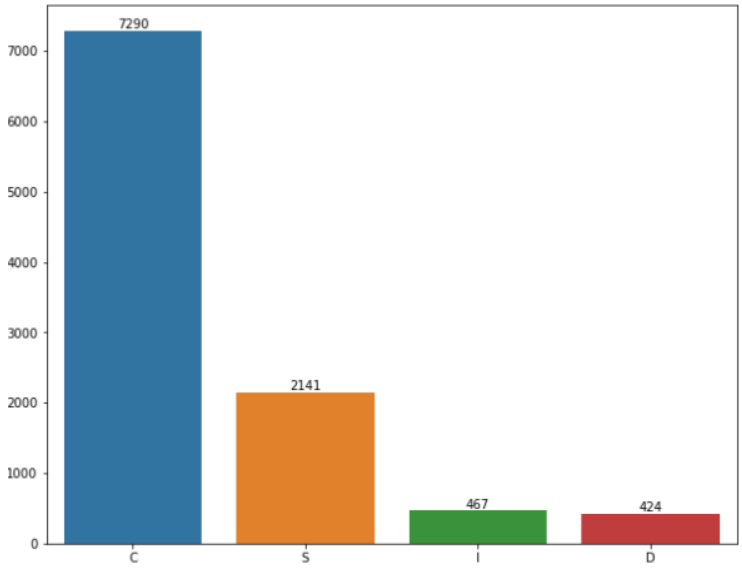

Figure 25: Kaldi obtained CSID for the L2 ASR Disfluency Detector model using CL strategy.

4.6.2 Kaldi alignment examples

Some examples of Kaldi alignments are shown and commented below. An important note is that these examples are predicted by the best disfluency detector model chosen from the previous section.

Example	Text
(I)	*** tärkeitä arvoja ovat asiakaslähtöisyys laadukkuus ja yhteistyö
	yritykselle tärkeitä arvo ovat asiakaslähtöisyys laadukkuus ja yhteistyö
(II)	yritykselle tärkeitä arvoja ovat asiakaslähtöisyys laadukkuus ja yhteistyö
	yritykselle tärkeitä arvoja ovat asiakaslähtöisyys laadukkuus ja yhteistyö
(III)	(...) perustettu *** *** vuona tuhatyhdeksänsataakaheksankymmentäyksi (...)
	(...) perustettu vuonna tuhat yheksänsataa kaheksakymmentäyks (...)
(IV)	(...) perustettu vuonna tuhat yheksänsataa kaheksakytyks (...)
	(...) perustettu vuonna *** *** tuhatyheksänsataakaheksankytyks (...)

Table 9: Kaldi alignment examples. Each example is comprised of a pair of texts, where the first sentence is the human transcript and the second sentence is the predicted one. Long sentences are shortened with "(...)". Insertions and deletions are marked with "***" by Kaldi.

Table 9 shows some insertion and deletion errors, which occur either without the model's culpability, or because the model fails to differentiate when words are simple or complex.

Examples (I) and (II) indicate such an insertion error. The human transcript from example (I) begins with “tärkeitä arvoja”, whereas the model’s prediction inserts an additional initial word such that it outputs: “yritykselle tärkeitä arvo”, besides the substitution in words “arvoja” and predicted word “arvo”. Without prior analysis, the error is that of a simple insertion, but the model correctly predicts a word that in this case the transcriber failed to write. Example (II) proves this human error, as it originates from the same speaking task, but uttered by a different speaker. In this case, the transcriber has written this supposed insertion that occurs in example (I). Although there are not many errors like this, there are still a few present, and the number of insertions should be minimally lower than Kaldi indicates.

Examples (III) and (IV) show the difficulty that the model experiences with complex words. The original transcript in example (III) contains the complex word “tuhatyhdeksänsataakaheksankymmentäyksi”. Meanwhile, the model predicts this word separately as three: “tuhat yheksänsataa kaheksakymmentäyks”. Conversely, in example (IV) the transcriber divides the complex word from example (III) in three: “tuhat yheksänsataa kaheksakytyks”, while the model predicts it as one complex word: “tuhatyheksänsataakaheksankytyks”.

Moreover, the human error problem when transcribing is fairly common in the original transcripts when comparing them manually. Although there are not many examples where there are insertions or deletions between original transcripts, there are numerous instances where there are obvious differences between the original transcripts of two different speakers from the same task, some can be seen in examples (I) and (II), and (III) and (IV). From those specific examples, the errors are the non-transcribed “yritykselle” word by the transcriber in example (I), but also the incorrectly transcribed “vuona” in example (III), which the model predicts as “vuonna”, what should be the correct word as seen in the original transcript from example (IV). Other examples from the data where the model predicted the word correctly but the original transcript is incorrect are: “fazer” incorrectly transcribed as “faser”, “yrityksellä” transcribed as “yrityksel”, “laadukkuus” transcribed as “laadukuus”, “vienti” as “vieti”, “huimasti” transcribed as “humasti” or “ohumasti”, among many others.

These errors ultimately worsen the WER, CER and CSID scores of the model, both of substitutions and of insertions and deletions. However, this is an understandable consequence of human labeling in such tasks, which is compounded by having multiple transcribers for instances of the same speaking sample. Human labeling is time consuming, tedious and monotonous, and many errors such as the previously mentioned can occur. Regretfully, this is subsequently conveyed to the calculations of the WER and CER, the CSID counts. Thus, it should be taken into consideration that the model’s error metric results would unsurprisingly be lower by a few percentages.

4.6.3 Recall, Precision and performance of disfluency tags

The Recall and Precision metrics have been used to validate the prediction of *<disfluency>* tags by the model. They have been calculated from the Kaldi alignments by taking a binary approach to the transcripts. Both original and predicted transcripts

have been coded in a binary strategy, by replacing non-*<disfluency>* tags with 0 and *<disfluency>* tags with 1.

Model Name	**Recall**	**Precision**
`disfl_detector`	0.5655	0.6017

Table 10: Recall and Precision results of finetuned L2 Finnish ASR disfluency detector, employing Kaldi aligned transcripts.

As shown in Table 10, the disfluency detector model obtains a Recall score of 0.5655 and a Precision score of 0.6017. Although these results are not perfect, they are satisfactory given the task and the amount of data that was used.

Disfluency type	Detected	Not Detected	Total
<paral>	5	22	27
<hesitation>	44	23	67
<trans>	0	5	5
-repetition-	7	10	17

Table 11: Number of predicted disfluencies by type.

Morever, Table 11 shows the number of correctly predicted specific disfluency tags. Additionally, the previously mentioned issue of wrong insertions and deletions has to be taken into account, so these values can ultimately become slightly higher.

The model correctly predicts 44 hesitations and fails to predict 23. Although the number of failed predictions is high, the model can predict approximately 65.67 %, which means the model can extrapolate knowledge for these predictions. Paralinguistics and repetitions are sometimes predicted correctly, however the model fails to predict most of them correctly. This can be as a consequence of these disfluencies occurring in a variable and context driven manner. While hesitations occur when the speaker utters expressions such as “öö” or “ää”, paralinguistics only occur when certain sounds are uttered, such as laughter, coughing, etc., and repetitions when a part of the next word is previously repeated. Although the results for these two disfluencies are less than satisfactory, they indicate that the model has the ability to detect them, and the use of more data for finetuning could result in an increase in performance when predicting them. Finally, there are no translinguistics predicted correctly. This is a problem caused by an obvious lack of translinguistic tags in the data, and the fact that the model has been finetuned for L2 Finnish ASR and may lack the knowledge to differentiate when words are non-Finnish, specifically as there are translinguistics from multiple languages.

5 Summary

Multiple Wav2Vec2 models have been built in the pursuit of an L2 Finnish ASR Disfluency Detector. First, a Wav2Vec2 baseline was established for L2 Finnish ASR by finetuning the pretrained *wav2vec2-large-uralic-voxpopuli-v2* model available, and obtained satisfactory results with a WER of 29.09 % and a CER of 9.08 %. After corroborating its performance when compared to a previous model, the pretrained baseline was used to train two approaches to L2 Finnish ASR Disfluency Detection.

The disfluency detectors have been finetuned employing Lukio and Aalto L2 Finnish data from the Digitala project. This data is comprised of audios from language examination tasks, which have been subsequently transcribed by human transcribers, and the speakers evaluated by human raters which recorded the scores. The number of tasks in the Lukio data is 2966, and 1883 in the Aalto data, for a total of 19.45 hours. The base data contains different types of disfluencies, a useful and consequential subset of which have been coded into a universal *<disfluency>* tag. There are a total of 3451 disfluency tags in the data used.

The first approach is applied for training two models. These models have been trained on a randomly sampled training set (see Table 3). One model was finetuned for 40 epochs, while another for 60 epochs. The first model obtains overall the best results in these experiments, with a WER of 30.41 % and a CER of 13.17 %. Meanwhile, the model trained for 60 epochs, *disfl_detector_60ep*, obtains a WER of 30.38 % and a CER of 13.36 %.

The second approach, follows a Curriculum Learning strategy. The CL strategy used aims at progressively feeding the model increasingly disfluent samples. The training data is shuffled and sorted from least to most disfluent, therefore the model learns easy and fluent speech before being introduced to much harder samples. This strategy aims at simulating human learning on the model. Model *disfl_detector_cl* obtains a WER of 30.77 % and a CER of 13.79 %. Although it results in minor deterioration of these scores, the model does stabilize faster during the first epochs, and reaches the best result in less iterations (see Figure 17).

Results for L2 Finnish ASR Disfluency Detection were, as expected, worse than for L2 Finnish ASR, although they are good nonetheless. The best disfluency detector model, *disfl_detector*, differs only in a slight increase of 1.32 % WER and a more tangible increase of 4.09 % in CER. Additionally, the analysis of the L2 ASR Disfluency Detector's WER and CER scores validate the correct finetuning of the model. The WER and CER distributions for the test tasks are compressed around the respective score's means, and there are only a few outliers encountered. These outliers happen as a consequence of samples comprised of one or very few words, where even one incorrect character in a word results in a high WER for that specific sample.

Moreover, the Recall and Precision metrics as well as Kaldi's CSID scores were used to validate the predicted results and correct disfluency detection. Kaldi's CSID scores indicate that the model works correctly in it's ASR function, as there are considerably more correctly predicted words than there are insertions and deletions. Furthermore, there is a small number of substitutions when compared to the correctly

predicted words, and they generally occur because of small character errors.

Additionally, when manually analyzing the Kaldi aligned samples, a considerable number of human errors can be observed in the gold transcripts (see Table 9). This problem must be taken into account, as the substantial number of errors ultimately influences the final result of the WER, CER and CSID scores, as well as the Recall and Precision metric results.

Further result analysis has shown that the model correctly predicts many hesitations, and has potential to learn how to correctly predict paralinguistics and repetitions. However, the model has difficulties in predicting translinguistics, due to the multilingual domain of these disfluencies and low number of examples containing translinguistics.

6 Conclusions

This work focuses on the research of the use of disfluency detection and tagging in the context of L2 Finnish ASR. Second language learner ASR is in itself a demanding task, given likely mistakes such as grammatical, pronunciation and fluency errors. In this thesis different pretrained publicly available Wav2Vec2 ASR models have been finetuned to incorporate L2 Finnish ASR with the detection of certain disfluencies in speech. In contrast to the usual method of employing disfluency detection as a posterior step, this approach is carried out after an ASR system performs its task.

The models have been trained on 19.45 hours of L2 Finnish data from high-school and university students, obtained from official second language examinations. Training the model was approached in two different methods: randomly sampling the training data and a curriculum learning strategy. The randomly sampled disfluency detector model provides the best performance, and obtains 30.41% WER and 13.17% CER.

Further analysis of the results has proven that incorporating the disfluency detection step inside the ASR system is a viable solution. The L2 ASR disfluency detector model is only slightly worse than the L2 ASR baseline, with a slight increase of 1.32 % WER and a more tangible increase of 4.09 % CER. However, while the model predicts correctly a substantial number of disfluencies, not all of the disfluency types are predicted correctly and equitably.

As was observed when comparing the baseline model to previous work, more data resulted in a more accurate model for L2 Finnish ASR. Therefore it is a logical assumption that utilizing additional data, transcribed with a higher number of disfluency tags, will produce a better built L2 Finnish ASR Disfluency Detector. One feasible future work is the use of proven *silver data* generation methods [75] to obtain samples with disfluencies from the large YKI dataset, as it will mean a possible increase to 30.37 hours from the 19.45 hours currently used. This type of strategy uses a trained disfluency detector model to obtain samples with newly tagged disfluencies each epoch. The use of this *silver data* could improve the model, without the need of costly wait time for new human transcribed data, by making use of readily available untagged data. Additionally, a supplementary step could be performed by employing native speaker data to finetune the ASR part of the model before commencing finetuning of the L2 Finnish ASR Disfluency Detector system as a whole, this step may improve the model's ASR performance.

References

[1] Graves, A., Fernández, S., Gomez, F. & Schmidhuber, J. Connectionist temporal classification: Labelling unsegmented sequence data with recurrent neural 'networks. *ICML 2006 - Proceedings Of The 23rd International Conference On Machine Learning*. 2006 pp. 369-376 (2006,1)

[2] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A., Kaiser, L. & Polosukhin, I. Attention Is All You Need. *CoRR*. abs/1706.03762 (2017), http://arxiv.org/abs/1706.03762

[3] Baevski, A., Zhou, H., Mohamed, A. & Auli, M. Wav2vec 2.0: A Framework for Self-Supervised Learning of Speech Representations. *Proceedings Of The 34th International Conference On Neural Information Processing Systems*. (2020)

[4] Graves, A. & Jaitly, N. Towards End-To-End Speech Recognition with Recurrent Neural Networks. *Proceedings Of The 31st International Conference On Machine Learning*. 32, 1764-1772 (2014,6,22), https://proceedings.mlr.press/v32/graves14.html

[5] Hannun, A., Case, C., Casper, J., Catanzaro, B., Diamos, G., Elsen, E., Prenger, R., Satheesh, S., Sengupta, S., Coates, A. & Ng, A. Deep Speech: Scaling up end-to-end speech recognition. *CoRR*. abs/1412.5567 (2014), http://arxiv.org/abs/1412.5567

[6] Prabhavalkar, R., Rao, K., Sainath, T., Li, B., Johnson, L. & Jaitly, N. A Comparison of Sequence-to-Sequence Models for Speech Recognition. *Proc. Interspeech 2017*. pp. 939-943 (2017)

[7] Chiu, C., Sainath, T., Wu, Y., Prabhavalkar, R., Nguyen, P., Chen, Z., Kannan, A., Weiss, R., Rao, K., Gonina, K., Jaitly, N., Li, B., Chorowski, J. & Bacchiani, M. State-of-the-art Speech Recognition With Sequence-to-Sequence Models. *CoRR*. abs/1712.01769 (2017), http://arxiv.org/abs/1712.01769

[8] Al-Ghezi, R., Getman, Y., Rouhe, A., Hildén, R. & Kurimo, M. Self-Supervised End-to-End ASR for Low Resource L2 Swedish. *Proc. Interspeech 2021*. pp. 1429-1433 (2021)

[9] Stoian, M., Bansal, S. & Goldwater, S. Analyzing ASR Pretraining for Low-Resource Speech-to-Text Translation. *ICASSP 2020 - 2020 IEEE International Conference On Acoustics, Speech And Signal Processing (ICASSP)*. pp. 7909-7913 (2020)

[10] Jamshid Lou, P. & Johnson, M. End-to-End Speech Recognition and Disfluency Removal. *Findings Of The Association For Computational Linguistics: EMNLP 2020*. pp. 2051-2061 (2020,11), https://aclanthology.org/2020.findings-emnlp.186

[11] Park, S., Shin, D., Paik, S., Choi, S., Kazakova, A. & Lee, J. Improving Distinction between ASR Errors and Speech Disfluencies with Feature Space Interpolation. *CoRR*. abs/2108.01812 (2021), https://arxiv.org/abs/2108.01812

[12] Wang, F., Chen, W., Yang, Z., Dong, Q., Xu, S. & Xu, B. Semi-Supervised Disfluency Detection. *Proceedings Of The 27th International Conference On Computational Linguistics*. pp. 3529-3538 (2018,8), https://aclanthology.org/C18-1299

[13] Bayerl, S., Wagner, D., Noeth, E. & Riedhammer, K. Detecting Dysfluencies in Stuttering Therapy Using wav2vec 2.0. *Proc. Interspeech 2022*. pp. 2868-2872 (2022)

[14] Salesky, E., Sperber, M. & Waibel, A. Fluent Translations from Disfluent Speech in End-to-End Speech Translation. *Proceedings Of The 2019 Conference Of The North American Chapter Of The Association For Computational Linguistics: Human Language Technologies, Volume 1 (Long And Short Papers)*. pp. 2786-2792 (2019,6), https://aclanthology.org/N19-1285

[15] Horii, K., Fukuda, M., Ohta, K., Nishimura, R., Ogawa, A. & Kitaoka, N. End-to-End Spontaneous Speech Recognition Using Disfluency Labeling. *Proc. Interspeech 2022*. pp. 4108-4112 (2022)

[16] Karhila, R., Rouhe, A., Smit, P., Mansikkaniemi, A., Kallio, H., Lindroos, E., Hildén, R., Vainio, M. & Kurimo, M. Digitala: An Augmented Test and Review Process Prototype for High-Stakes Spoken Foreign Language Examination. *Proc. Interspeech 2016*. pp. 784-785 (2016)

[17] Getman, Y. End-to-End Low-Resource Automatic Speech Recognition for Second Language Learners. (Aalto University. School of Electrical Engineering,2021), http://urn.fi/URN:NBN:fi:aalto-202110249766

[18] Rosenblatt, F. The perceptron: a probabilistic model for information storage and organization in the brain.. *Psychological Review*. 65 6 pp. 386-408 (1958)

[19] Zell, A. Simulation neuronaler Netze. (Oldenbourg Wissenschaftsverlag)

[20] Rumelhart, D., Hinton, G. & Williams, R. Learning internal representations by error propagation. (1986)

[21] Rumelhart, D., Hinton, G. & Williams, R. Learning representations by back-propagating errors. *Nature*. 323, 533-536 (1986,10,1), https://doi.org/10.1038/323533a0

[22] Koturwar, S. & Merchant, S. Weight Initialization of Deep Neural Networks(DNNs) using Data Statistics. *CoRR*. abs/1710.10570 (2017), http://arxiv.org/abs/1710.10570

[23] Boulila, W., Driss, M., Al-Sarem, M., Saeed, F. & Krichen, M. Weight Initialization Techniques for Deep Learning Algorithms in Remote Sensing: Recent Trends and Future Perspectives. *CoRR*. abs/2102.07004 (2021), https://arxiv.org/abs/2102.07004

[24] Fukushima, K. Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological Cybernetics*. 36, 193-202 (1980,4,1), https://doi.org/10.1007/BF00344251

[25] Kim, Y. Convolutional Neural Networks for Sentence Classification. *Proceedings Of The 2014 Conference On Empirical Methods In Natural Language Processing (EMNLP)*. pp. 1746-1751 (2014,10), https://aclanthology.org/D14-1181

[26] Han, W., Zhang, Z., Zhang, Y., Yu, J., Chiu, C., Qin, J., Gulati, A., Pang, R. & Wu, Y. ContextNet: Improving Convolutional Neural Networks for Automatic Speech Recognition with Global Context. *Proc. Interspeech 2020*. pp. 3610-3614 (2020)

[27] Albelwi, S. & Mahmood, A. A Framework for Designing the Architectures of Deep Convolutional Neural Networks. *Entropy*. 19 (2017), https://www.mdpi.com/1099-4300/19/6/242

[28] Jordan, M. Serial order: a parallel distributed processing approach. Technical report, June 1985-March 1986. (1986,5), https://www.osti.gov/biblio/6910294

[29] Rautela, M. & Gopalakrishnan, S. Deep Learning frameworks for wave propagation-based damage detection in 1D-waveguides. (2020,1)

[30] Hochreiter, S. & Schmidhuber, J. Long Short-Term Memory. *Neural Computation*. 9, 1735-1780 (1997,11), https://doi.org/10.1162/neco.1997.9.8.1735

[31] Chung, J., Gülçehre, Cho, K. & Bengio, Y. Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling. *CoRR*. abs/1412.3555 (2014), http://arxiv.org/abs/1412.3555

[32] Sutskever, I., Vinyals, O. & Le, Q. Sequence to Sequence Learning with Neural Networks. *Advances In Neural Information Processing Systems*. 27 (2014)

[33] Cho, K., Merriënboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H. & Bengio, Y. Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation. *Proceedings Of The 2014 Conference On Empirical Methods In Natural Language Processing (EMNLP)*. pp. 1724-1734 (2014,10), https://aclanthology.org/D14-1179

[34] Kuchaiev, O. & Ginsburg, B. Factorization tricks for LSTM networks. *CoRR*. abs/1703.10722 (2017), http://arxiv.org/abs/1703.10722

[35] Shazeer, N., Mirhoseini, A., Maziarz, K., Davis, A., Le, Q., Hinton, G. & Dean, J. Outrageously Large Neural Networks: The Sparsely-Gated Mixture-of-Experts Layer. *CoRR*. abs/1701.06538 (2017), http://arxiv.org/abs/1701.06538

[36] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A., Kaiser, Ł. & Polosukhin, I. Attention is All you Need. *Advances In Neural Information Processing Systems*. 30 (2017)

[37] He, K., Zhang, X., Ren, S. & Sun, J. Deep Residual Learning for Image Recognition. *2016 IEEE Conference On Computer Vision And Pattern Recognition (CVPR)*. pp. 770-778 (2016)

[38] Ba, J., Kiros, J. & Hinton, G. Layer Normalization. (arXiv,2016), https://arxiv.org/abs/1607.06450

[39] Gehring, J., Auli, M., Grangier, D., Yarats, D. & Dauphin, Y. Convolutional Sequence to Sequence Learning. *Proceedings Of The 34th International Conference On Machine Learning - Volume 70*. pp. 1243-1252 (2017)

[40] Devlin, J., Chang, M., Lee, K. & Toutanova, K. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. *Proceedings Of The 2019 Conference Of The North American Chapter Of The Association For Computational Linguistics: Human Language Technologies, Volume 1 (Long And Short Papers)*. pp. 4171-4186 (2019,6), https://aclanthology.org/N19-1423

[41] Khalid, U., Beg, M. & Arshad, M. RUBERT: A Bilingual Roman Urdu BERT Using Cross Lingual Transfer Learning. (2021,2)

[42] Bäckström, T. Introduction to Speech Recognition. , https://speechprocessingbook.aalto.fi/Recognition/Speech_Recognition.html

[43] Sahidullah, M. & Saha, G. Design, analysis and experimental evaluation of block based transformation in MFCC computation for speaker recognition. *Speech Communication*. 54, 543-565 (2012), https://www.sciencedirect.com/science/article/pii/S0167639311001622

[44] Xu, M., Duan, L., Cai, J., Chia, L., Xu, C. & Tian, Q. HMM-Based Audio Keyword Generation. *Advances In Multimedia Information Processing - PCM 2004*. pp. 566-574 (2005)

[45] Kneser, R. & Ney, H. Improved backing-off for M-gram language modeling. *1995 International Conference On Acoustics, Speech, And Signal Processing*. 1 pp. 181-184 vol.1 (1995)

[46] Chan, W., Jaitly, N., Le, Q. & Vinyals, O. Listen, attend and spell: A neural network for large vocabulary conversational speech recognition. *2016 IEEE International Conference On Acoustics, Speech And Signal Processing (ICASSP)*. pp. 4960-4964 (2016)

[47] Hollands, S., Blackburn, D. & Christensen, H. Evaluating the Performance of State-of-the-Art ASR Systems on Non-Native English using Corpora with Extensive Language Background Variation. *Proc. Interspeech 2022*. pp. 3958-3962 (2022)

[48] Zhang, Z., Wang, Y. & Yang, J. End-to-end Mispronunciation Detection with Simulated Error Distance. *Proc. Interspeech 2022*. pp. 4327-4331 (2022)

[49] Vazhenina, D. & Markov, K. End-to-End Noisy Speech Recognition Using Fourier and Hilbert Spectrum Features. *Electronics*. 9 (2020), https://www.mdpi.com/2079-9292/9/7/1157

[50] Li, S., Ouyang, B., Liao, D., Xia, S., Li, L. & Hong, Q. End-To-End Multi-Accent Speech Recognition with Unsupervised Accent Modelling. *ICASSP 2021 - 2021 IEEE International Conference On Acoustics, Speech And Signal Processing (ICASSP)*. pp. 6418-6422 (2021)

[51] Prabhavalkar, R., Rao, K., Sainath, T., Li, B., Johnson, L. & Jaitly, N. A Comparison of Sequence-to-Sequence Models for Speech Recognition. *Proc. Interspeech 2017*. pp. 939-943 (2017)

[52] Graves, A. Sequence Transduction with Recurrent Neural Networks. *CoRR*. abs/1211.3711 (2012), http://arxiv.org/abs/1211.3711

[53] Goldwater, S., Jurafsky, D. & Manning, C. Which words are hard to recognize? Prosodic, lexical, and disfluency factors that increase speech recognition error rates. *Speech Communication*. 52 pp. 181-200 (2010,3)

[54] Wiśniewski, M., Kuniszyk-Jóźkowiak, W., Smołka, E. & Suszyński, W. Automatic Detection of Disorders in a Continuous Speech with the Hidden Markov Models Approach. *Computer Recognition Systems 2*. pp. 445-453 (2007)

[55] Wiśniewski, M., Kuniszyk-Jóźkowiak, W., Smołka, E. & Suszyński, W. Automatic detection of prolonged fricative phonemes with the Hidden Markov Models approach. (2007,1)

[56] Yang, L., Shriberg, E., Stolcke, A. & Harper, M. Comparing HMM, maximum entropy, and conditional random fields for disfluency detection. *9th European Conference On Speech Communication And Technology*. pp. 3313-3316 (2005,9)

[57] Alharbi, S., Hasan, M., Simons, A., Brumfitt, S. & Green, P. A Lightly Supervised Approach to Detect Stuttering in Children's Speech. *Proc. Interspeech 2018*. pp. 3433-3437 (2018)

[58] Watanabe, S., Boyer, F., Chang, X., Guo, P., Hayashi, T., Higuchi, Y., Hori, T., Huang, W., Inaguma, H., Kamo, N., Karita, S., Li, C., Shi, J., Subramanian, A. & Zhang, W. The 2020 ESPnet update: new features, broadened applications, performance improvements, and future plans. (arXiv,2020), https://arxiv.org/abs/2012.13006

[59] Mendelev, V., Raissi, T., Camporese, G. & Giollo, M. Improved Robustness to Disfluencies in Rnn-Transducer Based Speech Recognition. *ICASSP 2021 - 2021 IEEE International Conference On Acoustics, Speech And Signal Processing (ICASSP)*. pp. 6878-6882 (2021)

[60] Schneider, S., Baevski, A., Collobert, R. & Auli, M. wav2vec: Unsupervised Pre-Training for Speech Recognition. *Proc. Interspeech 2019*. pp. 3465-3469 (2019)

[61] Amodei, D., Anubhai, R., Battenberg, E., Case, C., Casper, J., Catanzaro, B., Chen, J., Chrzanowski, M., Coates, A., Diamos, G., Elsen, E., Engel, J., Fan, L., Fougner, C., Han, T., Hannun, A., Jun, B., LeGresley, P., Lin, L., Narang, S., Ng, A., Ozair, S., Prenger, R., Raiman, J., Satheesh, S., Seetapun, D., Sengupta, S., Wang, Y., Wang, Z., Wang, C., Xiao, B., Yogatama, D., Zhan, J. & Zhu, Z. Deep Speech 2: End-to-End Speech Recognition in English and Mandarin. *CoRR*. abs/1512.02595 (2015), http://arxiv.org/abs/1512.02595

[62] Hendrycks, D. & Gimpel, K. Bridging Nonlinearities and Stochastic Regularizers with Gaussian Error Linear Units. *CoRR*. abs/1606.08415 (2016), http://arxiv.org/abs/1606.08415

[63] Mohamed, A., Okhonko, D. & Zettlemoyer, L. Transformers with convolutional context for ASR. *CoRR*. abs/1904.11660 (2019), http://arxiv.org/abs/1904.11660

[64] Jégou, H., Douze, M. & Schmid, C. Product Quantization for Nearest Neighbor Search. *IEEE Transactions On Pattern Analysis And Machine Intelligence*. 33, 117-128 (2011)

[65] Jang, E., Gu, S. & Poole, B. Categorical Reparameterization with Gumbel-Softmax. (arXiv,2016), https://arxiv.org/abs/1611.01144

[66] Park, D., Chan, W., Zhang, Y., Chiu, C., Zoph, B., Cubuk, E. & Le, Q. SpecAugment: A Simple Data Augmentation Method for Automatic Speech Recognition. *Proc. Interspeech 2019*. pp. 2613-2617 (2019)

[67] Levenshtein, V. Binary Codes Capable of Correcting Deletions, Insertions and Reversals. *Soviet Physics Doklady*. 10 pp. 707 (1966,2)

[68] Wang, C., Riviere, M., Lee, A., Wu, A., Talnikar, C., Haziza, D., Williamson, M., Pino, J. & Dupoux, E. VoxPopuli: A Large-Scale Multilingual Speech Corpus for Representation Learning, Semi-Supervised Learning and Interpretation. *Proceedings Of The 59th Annual Meeting Of The Association For Computational Linguistics And The 11th International Joint Conference On Natural Language Processing (Volume 1: Long Papers)*. pp. 993-1003 (2021,8), https://aclanthology.org/2021.acl-long.80

[69] Getman, Y. End-to-End Low-Resource Automatic Speech Recognition for Second Language Learners. (Aalto University. School of Electrical Engineering,2021), http://urn.fi/URN:NBN:fi:aalto-202110249766

[70] Griewank, A. & Walther, A. Algorithm 799: Revolve: An Implementation of Checkpointing for the Reverse or Adjoint Mode of Computational Differentiation. *ACM Trans. Math. Softw.*. 26, 19-45 (2000,3), https://doi.org/10.1145/347837.347846

[71] Bengio, Y., Louradour, J., Collobert, R. & Weston, J. Curriculum Learning. *Proceedings Of The 26th Annual International Conference On Machine Learning*. pp. 41-48 (2009), https://doi.org/10.1145/1553374.1553380

[72] Wang, X., Chen, Y. & Zhu, W. A Survey on Curriculum Learning. *IEEE Transactions On Pattern Analysis And Machine Intelligence*. 44, 4555-4576 (2022)

[73] Povey, D., Ghoshal, A., Boulianne, G., Burget, L., Glembek, O., Goel, N., Hannemann, M., Motlicek, P., Qian, Y., Schwarz, P., Silovsky, J., Stemmer, G. & Vesely, K. The Kaldi Speech Recognition Toolkit. *IEEE 2011 Workshop On Automatic Speech Recognition And Understanding*. (2011,12), IEEE Catalog No.: CFP11SRW-USB

[74] Forney, G. The viterbi algorithm. *Proceedings Of The IEEE*. 61, 268-278 (1973)

[75] Rocholl, J., Zayats, V., Walker, D., Murad, N., Schneider, A. & Liebling, D. Disfluency Detection with Unlabeled Data and Small BERT Models. *Proc. Interspeech 2021*. pp. 766-770 (2021)

A Appendix

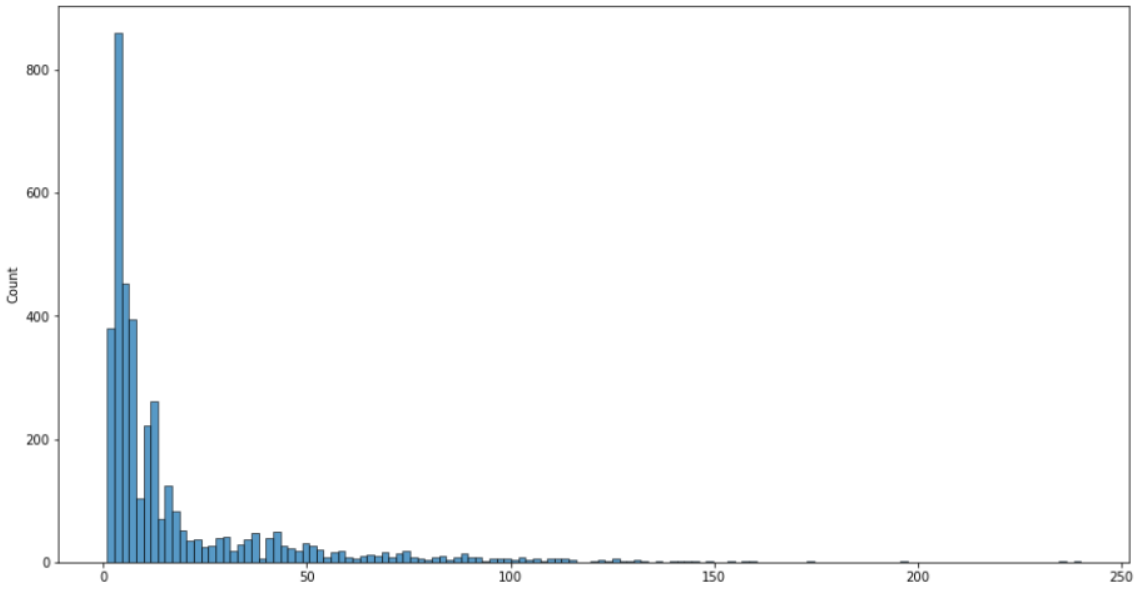

Figure A1: Distribution of sample lengths for the training split. X axis refers to the sample duration in word count.

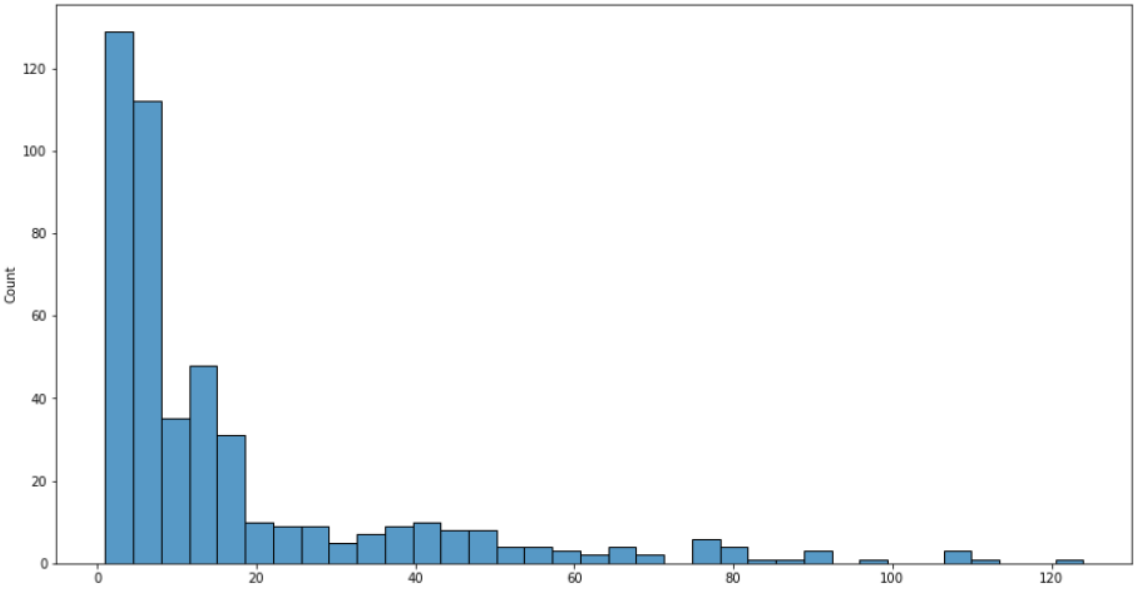

Figure A2: Distribution of sample lengths for the dev split. X axis refers to the sample duration in word count.

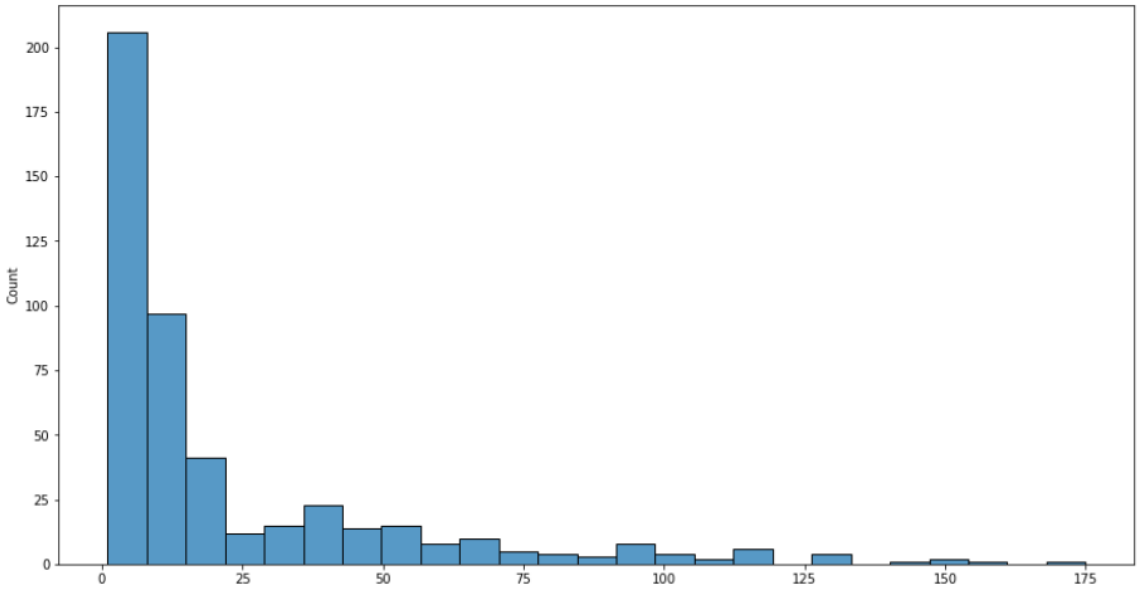

Figure A3: Distribution of sample lengths for the test split. X axis refers to the sample duration in word count.