CHALMERS
UNIVERSITY OF TECHNOLOGY

UNIVERSITY OF GOTHENBURG

Generating subtitles with controllable length using natural language processing

Master's thesis in Computer science and engineering

Joakim Svensson
Victor Troksch

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2022

Master's thesis 2022

Generating subtitles with controllable length using natural language processing

Joakim Svensson
Victor Troksch

UNIVERSITY OF
GOTHENBURG

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Computer Science and Engineering
Chalmers University of Technology
University of Gothenburg
Gothenburg, Sweden 2022

Generating subtitles with controllable length using natural language processing

Joakim Svensson
Victor Troksch

Supervisor: Richard Johansson, Department of Computer Science and Engineering, Chalmers
Advisor: Niklas Jansson & Peter Eklund, Plint AB
Examiner: Moa Johansson, Department of Computer Science and Engineering, Chalmers

Master's Thesis 2022
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Typeset in LaTeX
Gothenburg, Sweden 2022

Generating subtitles with controllable length using natural language processing

Joakim Svensson
Victor Troksch
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg

Abstract

Creating subtitles for video content is a task that has traditionally been performed manually by subtitlers. When creating a subtitle, there are rules and guidelines for how the text should be presented to the viewer. Therefore, a subtitle, translated from one language to another, often contains linguistic compression in the form of paraphrasing or removing parts of the dialogues. With advances in natural language processing, subtitlers now have tools for machine translation and automated speech recognition to assist them in their work. This thesis aims to explore various methods for how to control the generated output length of a sequence-to-sequence model, which are typically used for text generation and therefore also for machine translation. We apply different modifications to both the model itself and the data to control the output. Furthermore, this project makes use of transfer learning and pre-trained models with the Transformer architecture. The length ratio method produced the best results, in which it was possible to effectively control the output length of a generated subtitle. We also discover that it was also possible to apply this method for a translation model. Although it is a relatively simple method, it produced the desired results with linguistic correctness.

Keywords: Natural Language Processing, NLP, Transformer, seq2seq, text generation, BART, subtitles

Acknowledgements

We want to express our thanks and gratitude to our supervisors who have helped and supported us throughout this project. At Plint, we would like to thank Niklas Jansson & Peter Eklund for their involvement and support in our daily work. We would also like to thank our supervisor Richard Johansson, from the Department of Computer Science and Engineering at Chalmers University of Technology, who has provided us with important feedback along the way. Finally, we would like to thank all of our family members and friends who have supported us during this time.

Joakim Svensson & Victor Troksch, Gothenburg, June 2022

Contents

List of Figures

List of Tables

1
Introduction

Creating subtitles to video content is a challenging task that is performed by translators and linguists working as subtitlers. The amount of content that needs subtitling is increasing due to the growth of streaming platforms and international services. There are also regulations, such as directives like the one from the European Union in 2020 [9], which says that all material proved by the public sector in the union (and therefore Sweden as well) needs subtitling.The creation of subtitles is a manual process, and partially automating it would be highly beneficial regarding the work and time resources that it requires.

1.1 Background

In our daily life, we encounter subtitles, whether we think about it or not. It can be, for example, when watching the news or our favourite TV show. With the increase of streaming services and international content comes an increased demand for subtitles. At first glance, it may seem that creating subtitles is an easy task, but there is more to it than just translating dialogues from one language to another. When subtitling, the text must be translated while following certain length rules and specifications [4]. For example, the length restriction exists to ensure that the subtitle actually fits the screen. The length of a subtitle is also affected by the reading speed, which means that it must be possible to read the subtitle in a short period of time, as new subtitles will follow. Furthermore, there are also specifications and rules involving how certain words must be preserved (or censored), when to break lines, how to handle scene cuts, and how to identify speakers (such as a narrator), to mention a few. The challenge lies in creating subtitles that follow these rules while still maintaining the essence of the dialogue.

The creation of subtitles is a task traditionally performed by subtitlers that translate spoken content into subtitles for a specific language. Subtitlers also have to handle the task of segmentation, meaning that they have to partition the subtitles to match the dialogue. Segmentation is usually performed by adding time stamps to each subtitle, which determines when the subtitle will be visible to the viewer. If the segmentation is incorrect, the subtitles can appear at the wrong time and therefore confuse and annoy the viewer.

Another challenge in creating subtitles is that a dialogue can be highly nuanced and could contain things such as colloquialisms, cultural references, and humour. Fur-

thermore, it is also hard to translate certain sayings that are typical in the source language, but do not exist in the target language. These are things that are easy for a human to spot and to find a different translation for, but the more difficult for a machine.

Recent developments of technologies within natural language processing (NLP), such as automatic speech recognition (ASR) and machine translation (MT), have affected the way subtitlers work. With ASR, they can now get a transcript of the spoken content to use as a template when creating subtitles. This transcript was previously created by listening to the content and manually writing it down. Developments like this can significantly increase the productivity of subtitlers, and this was also shown in a study by Campbell in 2019 [6]. Although subtitlers can use aids to assist them in their work, the AudioVisual Translators Europe (AVTE) association claims that fully automated machine translation models are far from taking over the work of media translators [10]. However, they agree on the point of using ASR and MT as a complement in their work as linguists and media translators. At Plint, the company where this project takes place, a software platform is developed and maintained where their large pool of freelancing subtitlers works to create subtitles that the company later can deliver to its customers.

1.2 Problem Definition

To keep a subtitle within the length limitation, the task of creating subtitles is also often the task of performing a summary of the context along with the translation. This is a concept called linguistic or semantic compression, which means that the semantics of the sentence are maintained by paraphrasing parts of the sentence or removing redundant words. Therefore, this problem can be divided into two sub-problems within NLP, namely text summarization and machine translation.

This project intends to focus on the summarization part, which means that a subtitle should be generated with a controllable length. The project intends to explore various methods to approach and solve this problem. Furthermore, this involves enabling current state-of-the-art models within NLP. These models have been trained on large amounts of data and are a suitable starting point, rather than developing a model from scratch.

1.3 Aim

The aim of this project is to generate subtitles with controllable lengths. Controlling the output length is of importance for a subtitle due to the limited amount of characters that it is allowed to have. Restricting the output length of a generated sequence would not only be beneficial for subtitle generation, it would also be beneficial in other tasks involving language modelling, such as machine translation and text summarization.

More specifically, the project will consist of exploring and implementing models with controllable output lengths through either additional tokens, length encodings, or both. These methods will be implemented with existing pre-trained models such as BART [19]. Furthermore, the aim is to evaluate the methods with both quantitative and qualitative measurements to see how the implemented methods influence the model.

1.4 Limitations

The project is limited to working only with models based on the Transformer architecture [38]. This is motivated by most of the current state-of-the-art models within various NLP tasks uses this architecture. Furthermore, the project is limited to only considering pre-trained models. Training a language model from scratch would require more computing power and resources than available. The final limitation is that the project is limited to only considering subtitle generation between English and English and from English to Swedish. The limitation of considering only two languages is motivated by the time constraint of the thesis.

1.5 Related Work

The project has taken inspiration from various articles and papers within the fields of machine translation, text summarization, and sentence simplification. This section will mention a few of the articles related to and mentioned in this thesis.

1.5.1 Neural Machine Translation with Constraints

Neural machine translation (NMT) is a well-researched field within NLP, and current state-of-the-art models are all based on the Transformer architecture [38]. However, when generating subtitles, the length of the subtitle is crucial, and hence this task is closely related to the work of constraining the output length of NMT. Many attempts to restrict the output length of a sequence-to-sequence model are inspired by the work of Takase & Okazaki [37]. To preserve a length constraint, they implemented a modified positional encoding in the decoder of the Transformer, which encodes the positional encoding with respect to how many tokens that are left to be generated in the sequence, rather than how many tokens that have been generated. This work inspired Lakew et al. [18] and Niehaus [28] who both uses this encoding in their work. They also experimented with the implementation of special tokens to represent a specific length or ratio to their models.

1.5.2 Text Summarization

Translating a wordy dialogue into a subtitle is a complex task and hence a challenge to approach with text summarization. It is the task of creating a summary that contains the most important and relevant content of the original text. The two most common methods are extractive summarization and abstractive summarization [40].

The extractive summarization works by selecting a subset of words present in the original text and stitching them together to create a summary, while abstractive summarization can paraphrase and insert new words to create a more fluent and coherent summary.

Rush et al. [33] developed a model that implements abstractive summarization to create summaries of fixed length. Their summary generation is carried out using a beam search algorithm [12] to generate summaries with fixed length. Although their model has proven to be effective, it is limited in the aspect of being only capable of working with the same language.

1.5.3 Sentence Simplifications

Text simplification is the subfield within NLP that focuses on automatic simplification of sentences. Simplifying sentences is beneficial for people who are not fluent in a language, such as children, language learners, or people with reading disorders. Simplifying a text or sentence consists of modifying the content and rewriting the structure, while still preserving the original meaning. Modifications can consist of paraphrasing words, removing redundant words, or splitting a sentence into two if it helps simplify the original sentence.

The relevant work for this thesis is related to the work of Martin et al. [23][24]. His work is based on creating Control Tokens to control the text generation of a transformer simplification model. These tokens are created to represent different features of the relation between the source- and target sentences, such as the length difference and the amount of paraphrasing between the two sentences. This method has been proven to work on Transformer models trained from scratch, but also on pre-trained models such as BART [19].

1.6 Outline

This chapter of the thesis has introduced the background of the project, as well as the definition, aim, and limitations of the project. Furthermore, this chapter also covers a short introduction to previous work related to the project. The outline of the remaining chapters of the thesis can be seen in the list below:

- Chapter 2 introduces the theory involved in this thesis. Concepts and theory specific for NLP, such as tokenization, word embeddings are described along with more common machine learning theory, such as sequence-to-sequence models and the Transformer, together with an introduction to pre-trained models. The chapter is finalised with an introduction to the NLP metrics that are used to evaluate language models.

- Chapter 3 describes the methods used in this thesis. The chapter starts by introducing the data used in the project and the associated pre-processing techniques. In addition, it explains what models and what methods were used. Finally, the chapter describes how the model is trained and evaluated.

- Chapter 4 presents the results for the evaluated methods and also shows a few examples of the generated sequences.

- Chapter 5 discusses and draws conclusions based on the methods and the results. Improvements and suggestions for future work are also presented here.

2
Theory

This chapter introduces and describes the theory and concepts used in this thesis. The chapter starts with a short introduction to Natural Languange Processing (NLP) in general and the most important concepts within the field. Following the introduction of NLP comes an in-depth explanation of some of the most influential deep learning models for NLP, like the sequence-to-sequence (seq2seq) architecture, the Transformer, and an introduction to the pre-trained models used in this thesis. Lastly, comes an explanation of the most common metrics and scores used to evaluate our methods and experiments.

2.1 Natural Language Processing

Natural language processing (NLP) is the field within machine learning that involves text and natural language. The field can be described as the interaction and bridge between a computer and its ability to process and handle natural language. Natural language is defined as the native speech of people, in contrast to artificial languages that are designed to control a computer, for example. There are many different topics and subfields within NLP and to name a few, there are tasks like machine translation, text generation, text classification, and question answering.

Modern NLP models are typically constructed and trained with neural networks. This implies that the data require special features that can be interpreted by a machine learning model. Therefore, the data has to go through a number of preprocessing steps in order to transform the raw text data into numerical data that a computer can understand. The following sections will cover the most essential methods for representing text with numbers.

2.1.1 Word Tokens

Tokenization is one of the most important preprocessing steps in NLP. Tokenization is a technique used to split up a text, sentence, or document into smaller pieces that are called tokens. The tokens can be divided into groups of words, subwords, or even characters, depending on what tokenizer is used. The tokenizer keeps track of all tokens in a vocabulary, meaning that it maps every token to a specific token-ID. To get an understanding of the different tokenization methods, consider the following sentence:

The kids are playing football.

Word-based tokenization:
[The, kids, are, playing, football]

Character-based tokenization:
[T, h, e, k, i, d, s, a, r, e, p, l, a, y, i, n, g, f, o, o, t, b, a, l, l]

Subword-based tokenization:
[The, kid, s, are, play, ing, foot, ball]

Word- and character-based tokenization is probably the easiest to interpret. However, these two methods have some issues. With the word-based method, the vocabulary tends to become very large because every word has its own token (the English language contains more than 500,000 unique words). This issue can be solved by creating a vocabulary consisting of only the most common words and then assigning an "unknown token" for words not included in the vocabulary. However, this method will lead to a loss of performance for the model since there will be a loss of information at each of the unknown tokens. Another problem is that similar words, like "bird" and "birds", will initially have completely different representations in the model due to the individual tokens.

The problem with large vocabularies is countered by having the tokenization based on characters. However, a single character does not often say much on its own compared to words in languages using the Latin alphabet (Chinese signs carry more information, for example). This means that the model has to look at several tokens to interpret the meaning of a single word. The models also have to handle larger inputs for every query. A word-based input of 5 tokens could be equivalent to more than 30 character tokens.

The subword-based tokenization is a combination of the two previous mentioned methods and is also the most common method. This method is also used by most of the current state-of-the-art models within NLP. Common character sequences, such as short words for example, are left intact while longer and more uncommon ones are split into subwords. This method has the advantage that it can build every word in the document by stitching together the said subwords. This means that the vocabulary does not need to be as big as for a word-based tokenizer, but also that the model does not have to handle as many tokens as with a character-based tokenizer. This method can also learn pre- and suffixes along with grammatical word endings, which can be seen in the example sentence above: *The, kid, s, are, play, ing, foot, ball.* This will allow the model to see the similarity between "kid" and "kids", for example. However, the partitioning does not necessarily have to result in a favorable way. It might as well be tokenized as: *Th, ki, ds, are, pla, ying, foo, tball.* The partitioning of the subword tokens depends on the data that was used to train the tokenizer.

There are several algorithms to create the subword tokens, where Byte-Pair [35], WordPiece [34] and SentencePiece [17] are the most common ones. Byte-Pair encoding (BPE) creates the vocabulary by first finding every unique word in a corpus, called pre-tokenization, where also the word frequency is saved. The next step is to create a base vocabulary consisting of every character present in the unique words. Starting with the base vocabulary, the training data are tokenized into temporary tokens consisting of neighbouring existing token pairs. The most frequent temporary token can be determined by the earlier word count and is added to the vocabulary. The process is repeated, where one token is added per iteration, until the desired size (specified hyperparameter) is reached.

WordPiece tokenization is comparable to BPE. The algorithm starts by initializing a base vocabulary consisting of every character that occurs in the training data. However, the pair selection is not based on the highest frequency, but rather on whether the pair maximizes the likelihood of the training data when it is added to the vocabulary. This implies that maximizing the likelihood of the pair, whose probability is greater than all other pairs, gives the best training data.

SentencePiece [17] is a method that also utilizes the BPE method, but includes whitespaces in the set of available characters. The tokens that are created will make up the final vocabulary, and, as mentioned earlier, every token in the vocabulary is assigned with a unique integer as ID or index.

2.1.2 Word Embeddings

Many NLP-models have the words represented by one-hot encodings, meaning a binary vector where every element is equal to zero apart from one, which represents the token-ID. However, assigning a single binary vector for each word is often not enough for a machine learning model to understand words on a deeper level. The most common way to represent words is therefore to use word embeddings. A word embedding is a mathematical representation of each word that works by assigning a vector with real-valued numbers to each word in the vocabulary. Words that have semantic similarities are also expected to have similar word embeddings in the vector space. This is not possible with one-hot encoding, since binary vectors entail the same distance between every word in the vocabulary. To create word embeddings, there are a few available methods, but the most common ones are to use methods that involve machine learning or statistics.

Word2Vec [27][26] is a statistical method that is essentially a shallow neural network consisting of one hidden layer, which can be trained with two different methods. The first is the Continuous Bag-of-Words (CBOW) method, which learns the word embeddings by implementing a context window around a word, and then the network tries to predict the said word. The weights corresponding to the word later act as the embedding. The Skip-gram method is similar to CBOW but it works the other way around, meaning that the model is trained to predict the surrounding words in the training set. In both methods, the size of the context window is speci-

fied when the model is created and will act as the embedding dimension.

GloVe (Global Vectors for Word Representation) [30] is an embedding technique for distributed word representation, which utilizes unsupervised learning to achieve embedding vectors. Unlike Word2Vec, GloVe does not only look at the local statistics, meaning the surrounding words, but also considers the word co-occurrences. The model is trained by creating a co-occurrence matrix for every word in the training corpus. For every word, the matrix stores the number of occurrences together with adjacent words in a specified window size.

Word embeddings can also be constructed by training a regular **embedding layer**, as in [11][31][19]. The embedding layer acts as a lookup table, where every word corresponds to a vector of a specified embedding dimension. The weights are initialized randomly and updated with respect to the loss function of the language model.

2.2 Artificial Neural Networks

Artificial neural networks (ANNs) are a type of computing system inspired by the human brain. ANNs can come in many different shapes and sizes depending on the task they are used to perform. This chapter will cover a short introduction to the most essential network types relevant to this thesis and how they are constructed.

2.2.1 Feed Forward Neural Network

The feedforward neural network (FFNN) was the first type of ANN to be invented and is also probably one of the simplest network architectures to date. The network is called a feedforward neural network because the information flows forward through the network without any cycles or loops.

The simplest version of a FFNN can be viewed as a single perceptron, which means that it is constructed with only an input layer and an output layer. This version is also known as a single-layer perceptron. The output is calculated according to Equation 2.1, where w represents the weights, x the input, b is the bias, and σ is the non-linear activation function. A single-layer perceptron is a linear classifier.

$$y = \sigma\left(\sum_{j=0}^{n} w_i x_i + b\right) \tag{2.1}$$

The other version of a FFNN is called the multi-layer perceptron (MLP), which is composed of many perceptrons. The MLPs are constructed of at least three layers: an input layer, a hidden layer, and an output layer. This enables the network to compute non-linearly separable functions and is hence suitable for tasks such as classification in supervised learning.

2.2.2 Recurrent Neural Networks

Recurrent neural networks (RNNs) are a type of ANNs that are used to process sequential data. Unlike ordinary FFNNs, RNNs contain directed cycles, which means that the information does not flow in a strict direct order. These cycles enable the network to handle information about previous steps in the computation, which can be seen as the network having memory. Due to the ability of RNNs to process sequential data, they are typically useful in NLP tasks.

RNNs work by receiving an input x and for each time step t calculate an output $y(t)$, based on the input of $x(t)$ and the previous output at $y(t-1)$. The output of the previous step is saved in a hidden state, denoted $h(t)$. Furthermore, a RNN can have different sizes of inputs and outputs. A one-to-many network takes one input and generates a sequence of outputs. A many-to-one network works in the opposite way, which means that it takes many inputs to generate one output. Lastly, many-to-many RNNs take an input sequence and generate a sequence as an output, hence is this type suitable for machine translation. An illustration of a many-to-many network can be seen in Figure 2.1.

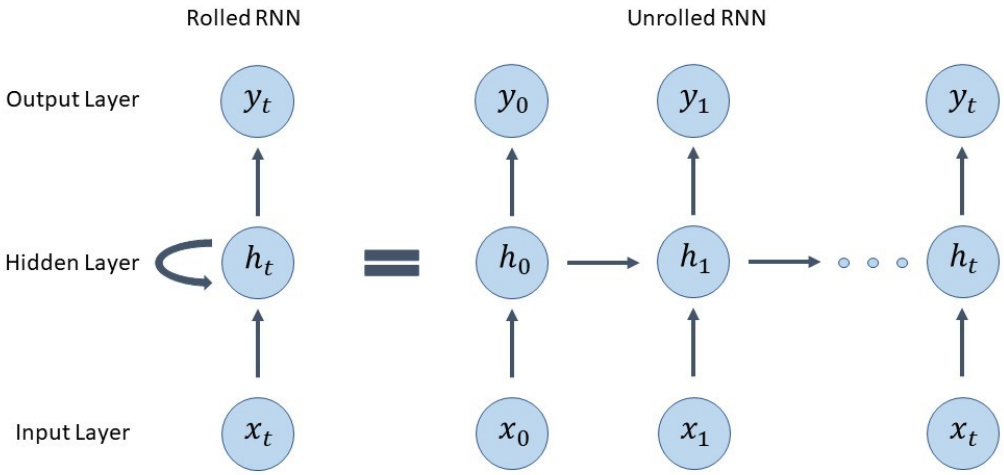

Figure 2.1: A visualization of a many-to-many RNN. The network is unrolled in the right-hand side of the figure to visualize the effect of the hidden states h.

2.3 Sequence-to-sequence Models

A sequence-to-sequence (seq2seq) model, or encoder-decoder model, is a special class of ANN architectures that takes sequential data as input and generates a new sequence as output. The seq2seq model is composed of an encoder and a decoder,

where both components are usually constructed by RNNs. The encoder is used to compress the input sequence into a context vector that the decoder can use to generate the new output.

The encoder works by encoding each word in the input sequence by computing the hidden states h_i for each time step i. The final hidden state, at time step n, is denoted h_n and this state is equivalent to the context vector that is sent to the decoder. The decoder generates an output y_i for each time step, depending on the previous state. The initial state of the decoder is the context vector h_n.

One drawback with the seq2seq models and the encoder-decoder architecture is that it has issues with long sentences. Cho et al. [8] showed that when the input sequence becomes longer, it is harder for the model to encapsulate all the important information in the context vector.

2.3.1 Attention

To solve the problem that arises with long input sequences, Bahdanau [2] introduced the attention mechanism. This mechanism is created with the intention of mimicking cognitive attention by deciding which parts of the input sequence that are of importance. It works by creating a context vector c_i from a linear combination of the hidden states h_j in the encoder and the attention weights α_{ij} for each time step j, see Equation 2.2. The context vector c_i is also influenced by the previous hidden state s_{i-1} as can be seen in Equation 2.4.

$$c_i = \sum_{j=1}^{T_x} \alpha_{ij} h_j \tag{2.2}$$

Here, the weight of each α_{ij} is equal to:

$$\alpha_{ij} = \frac{exp(e_{ij})}{\sum_{k=1}^{T_x} exp(e_{ik})} \tag{2.3}$$

Here, e_{ij} is the alignment score of a FFNN described by the function a:

$$e_{ij} = a(s_{i-1}, h_j) \tag{2.4}$$

By passing all context vectors c_i to the decoder, the model can decide what to focus on in the sequence while decoding the next step. See Figure 2.2 for an illustration of how the attention mechanism is calculated.

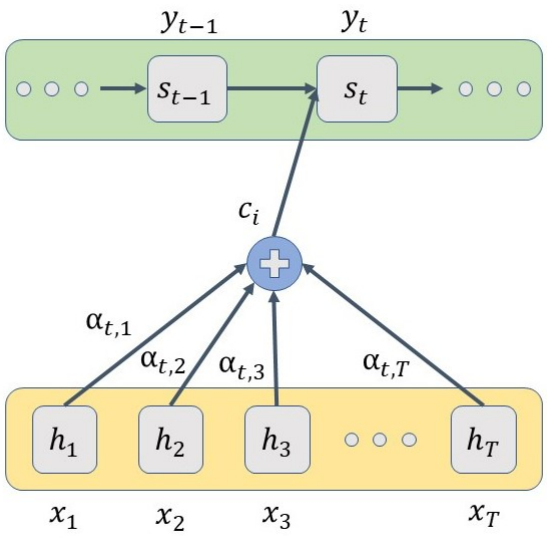

Figure 2.2: The attention mechanism visualized with an example sequence.

2.3.2 Teacher Forcing

Teacher forcing is a common method used to train seq2seq models quickly and efficiently. Consider an arbitrary seq2seq model that predicts an output $y(t)$, given an input $x(t)$. The method works by letting the next input to the model $x(t+1)$ be equal to the actual ground truth, regardless of the predicted output $y(t)$. This will enable the model to learn the next prediction from the correct input in the training data, rather than using the predicted output from the previous time step. The idea behind this method is to not let the model train on false predictions and hence waste valuable training time. An example of teacher forcing can be seen in Figure 2.3.

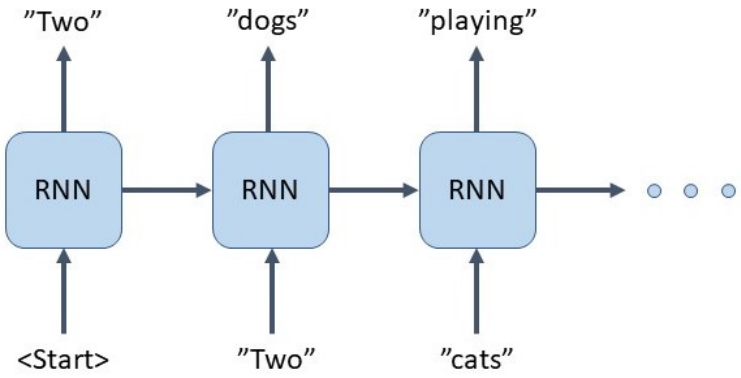

Figure 2.3: Teacher forcing exemplified with a RNN. The input at $x(3)$ is not the output from $y(2)$, instead it is the actual ground truth at the associated time step.

2.3.3 Autoregressive generation

A seq2seq model is, in fact, also an autoregressive model. This is a model that is used to describe time-varying processes, such as language generation. For a model to be autoregressive, it has to predict future values based on previous values. To put this into context, an autoregressive language model let the i-th generated word in a sequence depend on all preceding $i-1$ words. The probability distribution for a word sequence can be described by Equation 2.5, where $w_{1:T}$ is the generated text sequence, W_0 equals the initial text sequence fed to the model and $w_{1:0}$ is the empty set, implying that no sequence has been generated in advance.

$$P(w_{1:T}|W_0) = \prod_{t=1}^{T} P(w_t|w_{1:t-1}, W_0), \text{where } w_{1:0} = \emptyset \tag{2.5}$$

With Equation 2.5, text can be generated in different ways. One method is to use the **Greedy Search** algorithm. It makes the selection by selecting the word with the highest probability for the given time step, that is, $w_t = argmax(P(w|w_{1:t-1})$. This algorithm is efficient, but comes with the downside that it can miss combinations of words that have a higher probability than the predicted one. The reason for this is that a word combination with higher probability can exist where the first word has a lower probability than the generated one. An example of this can be seen in Figure 2.4. The algorithm runs until a special end-of-sequence token is generated or a specified number of words have been reached.

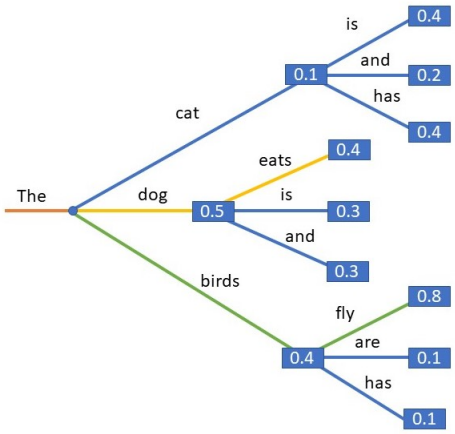

Figure 2.4: Beam- and Greedy search visualized by an example. The yellow lines represent the greedy approach, which takes the highest probability for each word and it results in a score of 0.20. The green lines represents the Beam search, which takes multiple combinations into consideration and results in a score of 0.32.

The **Beam Search** algorithm counters the problem of missing out on hidden high-probability words by keeping the n most likely sequences at each time step t. This creates beams in the search tree, hence the name, which can get past lower probabilities in order to find better predictions further down the beam. Due to the fact that the algorithm stores several hypothetical sequences and acts greedily at the

same time (since it always picks the n words with the highest probability), it is guaranteed to find a sequence at least as likely as the greedy search algorithm, at the expense of computational cost. However, the algorithm is not guaranteed to find the sequence with the highest probability, since that would require a complete search of all possible word combinations.

Both the greedy- and beam search algorithms can encounter the problem with repeating sequences of words. To avoid the problem, a n-gram penalty can be implemented [29]. This can be done in different ways, but one way is to set the probability to 0 for a word that creates a n-gram that has already appeared in the sequence.

It is also worth noting that the human language is not always as predictable. In [15], the author shows that humans seem to prefer to be surprised by a text. This means that it might not be the best practice to always pick the word with the highest probability instead of sampling from the distribution if the goal is to mimic human generated text.

2.4 Transformer

The Transformer is a deep learning model that was introduced by Vaswani et al. [38] in 2017. The model was developed for machine translation, but it quickly became state-of-the-art in many NLP related tasks. The Transformer is essentially a seq2seq model, but unlike RNNs, the Transformer does not need to process data in sequential order. Without having to process the data in order, the training can be parallelized. To capture the structure and order of a sequence, the Transformer uses positional encodings (see Section 2.4.2) and the self-attention mechanism (see Section 2.4.1).

The Transformer is composed of two blocks, one block that works as an encoder and the other block that works as a decoder. In the original paper, the Transformer is composed of 6 layers in each block, but the number of layers can be modified to serve specific tasks. In all layers, within the two blocks, there are residual layers that work as skip connections. The skip connection is an operation where some parts of the output skip one or more layers and instead it is being added to a layer deeper into the model. Residual layers prevent deep networks from losing track of input and leads to a better performing network as a result [13].

The first block is the encoder block, where each of the 6 layers works as an independent encoder with its own weights. Each layer in the encoder can be broken down into two sub-layers, a multi-head attention module and one feed-forward neural network (FFNN). After each sublayer inside the encoder, there is a residual connection followed by a layer normalization. Dropout is also applied to each sub-layer.

The second block is the decoder block, which is very similar to the encoder block, with the key difference that a multi-head attention mechanism is added on the output of the encoder stack. The decoder is also modified so that it cannot attend

to subsequent positions when decoding the output. This means that the model output can only depend on the previous positions in the generated sequence. This modification is also known as masked multi-head attention.

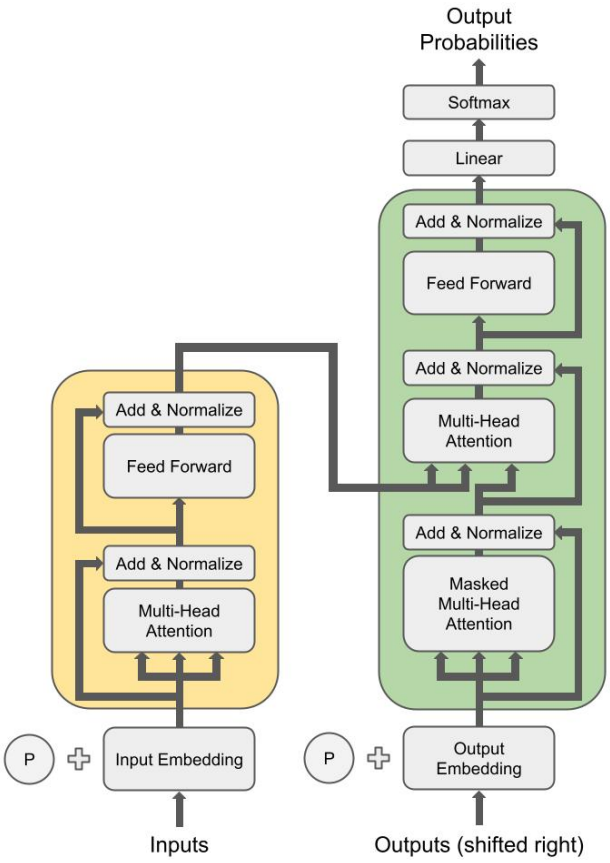

Figure 2.5: Visualization of the Transformer architecture as depicted in the original paper. The yellow part represents the encoder block and the green the decoder block.

2.4.1 Self-attention

With the introduction of the Transformer, the first transduction sequence model that relies only on attention was introduced. The implementation that enabled this new feature was to replace the recurrent layers with a self-attention module. This type of attention mechanism works by allowing the model to process attention within an input sequence. By relating the positions in a sequence with other positions in the same sequence, self-attention can create an understanding of how words relate to each other.

Self-attention is computed multiple times in each layer, in parallel and independently, through what is called multi-headed attention. This is a module that concatenates the outputs of each self-attention module before linearly transforming the outputs to create a final representation. A detailed explanation of how self-attention is calculated follows.

The three main components in calculating self-attention are the vectors of queries, keys, and values, denoted as Q, K, and V. All these vectors are retrieved from the word representation in the input sequence, by multiplying the input embedding with the corresponding matrix, denoted as W^Q, W^K, and W^V. The queries and keys have the dimension of d_k and the value vector d_v. When calculating self-attention, one can see the calculation as a mapping between a query and a set of key-value pairs to an output.

Self-attention is measured with an attention score. This score represents how much attention should be paid to other parts of the input with respect to the current position. In other words, the score represents how much attention should be paid to other words in the sequence, based on the current word. This score is computed by first taking the dot product of the query matrix with the key matrix. The score is then divided by $\sqrt{d_k}$ followed by a softmax function. Lastly, the self-attention calculation is finalized by multiplying it with the value matrix, see Equation 2.6.

$$Attention(Q, K, V) = softmax(\frac{QK^T}{\sqrt{d_k}})V \tag{2.6}$$

As mentioned above, the Transformer calculates the self-attention scores in parallel, which means that it performs the self-attention calculation multiple times. This enables the model to focus on different positions inside the sequence, and in turn, leads to better scores. The original implementation of the Transformer uses 8 different heads, which generate 8 different matrices for self-attention per sequence. To pass these values through the FFNN, the matrices for each head are concatenated into one big matrix. This matrix, which contains all the multi-headed attention scores, is then multiplied with an additional weight matrix, denoted W^O, to create the final representation, see Equations 2.7 and 2.8.

$$MultiHead(Q, K, V) = Concat(head_1, \ldots, head_h)W^O \tag{2.7}$$

where

$$head_i = Attention(QW_i^Q, KW_i^K, VW_i^V) \tag{2.8}$$

2.4.2 Positional Encodings

Unlike RNNs, the Transformer sees the input as a quantity of words instead of a sequence by default. Since a language depends on the word ordering, positional information needs to be added in order for the Transformer to function properly. One key element in the Transformer architecture is therefore the positional encoding, which takes care of the positional information. This encoding is applied to both the encoder and decoder parts of the model. An example of the importance of word ordering can be seen below, where both sentences contain the same words but mean two different things.

"He likes football but hates golf."
"He likes golf but hates football."

In [38], the authors describe two ways to encode the positions, either fixed or learned. The fixed method is based on the use of trigonometric functions to encode the positions. The positions are encoded according to Equation 2.9, where *pos* represents the position of the word in a sentence, d_{model} is the total number of embedding dimensions of the model, and i is a specific dimension according to $i = 1, \ldots, d_{model}/2$. When the frequencies of the sinusoidal functions are altered, the encoded values will be different for every position. The variation of the encodings is determined by the size of the embedding dimension. Even dimensions get the sine embedding, while odd dimensions get the cosine one. The positional encodings are simply added to the input embeddings, as they have the same dimensions. A visualization of Equation 2.9 can be seen in Figure 2.6.

$$\begin{aligned} PE_{pos,2i} &= sin\left(\frac{pos}{10000^{\frac{2i}{d_{model}}}}\right) \\ PE_{pos,2i+1} &= cos\left(\frac{pos}{10000^{\frac{2i}{d_{model}}}}\right) \end{aligned} \tag{2.9}$$

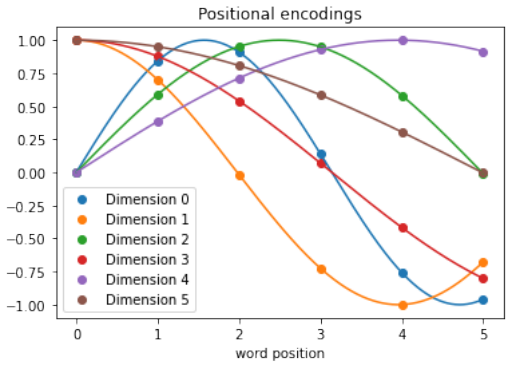

Figure 2.6: Visualization of a sequence containing 6 words, also with an embedding dimension of 6. Worth noting is that the denominator used for this visualization is altered in order to make the shapes better visible on small example sentences.

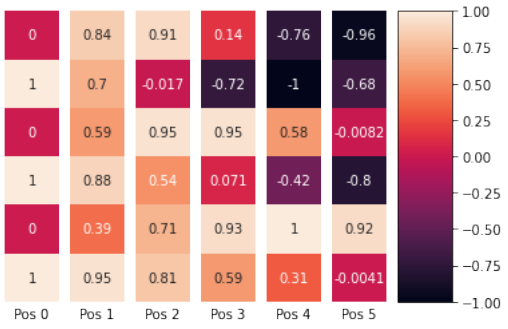

Figure 2.7: The resulting positional encoding values from Figure 2.6.

The other method is to let the model learn the positional encodings on its own. This is done using a regular embedding layer that the model learns during training, which means that the model learns a positional embedding layer. However, Vaswani showed that there was very little difference between this method and the one using sinusoidal encodings. In the original Transformer paper, they chose the sinusoidal

approach because of the hypothesis that a sinusoidal encoding would learn long dependencies better than the learned variant. More recent work, which is currently state-of-the-art, has for most cases adapted the learned approach over the static approach [19][5].

2.5 Pre-Trained Models

Transfer learning is the concept of transferring knowledge between one model to another, by reusing parts of a pre-trained model into the new model. The idea is that if there already is a model used for solving a similar to the one you are dealing with, the new model could inherit some features from the pre-trained model. By using a pre-trained model instead of training a model from scratch, one can save both time and resources.

Formally, the transfer learning task is defined as follows: Given a source domain $\mathcal{D}_S$ and a learning task $\mathcal{T}_S$ with the corresponding notation for the target domain $\mathcal{D}_T$ and $\mathcal{T}_T$. The aim is to improve the learning task of the target domain $\mathcal{T}_T$ with the help of the learning task of the source domain $\mathcal{T}_S$. Specifically, the transfer is based on improving the conditional probability distribution $P(Y_T|X_T)$ in $\mathcal{D}_T$ with the information from $\mathcal{D}_S$ and $\mathcal{T}_S$. A requirement is that either $\mathcal{D}_T \neq \mathcal{D}_S$ or $\mathcal{T}_T \neq \mathcal{T}_S$, otherwise no information would be transferred.

A common use case of transfer learning within NLP is to fine-tune a pre-trained model for a downstream task. Transformer models are often pre-trained on large text corpora, meaning that they have a good understanding of a language. This enables the model to be fine-tuned for specific NLP tasks. For instance, one can easily create a sentiment classifier by adding one or more linear layers on top of a pre-trained model. Recent development of language models such as BERT [11] and BART [19] has led to models that could perform multi-tasking, meaning that the same model can be used for different tasks. For example, the same model can be used for machine translation, question answering, and semantic analysis. Another example is the use of multilingual models, where one model can learn multiple languages at once. The following subsections cover some of the most influential NLP models and the most relevant ones for this project.

2.5.1 BERT

BERT is a language model that was introduced by Google AI Language in 2018 [11]. The name stands for **B**idirectional **E**ncoder **R**epresentations from **T**ransformers and its architecture is almost identical to the Transformer. The model originally came in two sizes, where the first, $\text{BERT}_{\text{BASE}}$, consists of 12 layers, 12 attention heads and a hidden size of 768, with a total of 110 million parameters. The second one, $\text{BERT}_{\text{LARGE}}$, consists of 24 layers, 16 attention heads, and a hidden size of 1024, resulting in a total of 340 million parameters. BERT has learned positional embeddings, which means that the embeddings are learned during pre-training.

As stated in the name, BERT works with a bidirectional encoder, which enables the model to look at positions further in the sequence when encoding. This feature is suitable for pre-training on unsupervised tasks such as masked language modelling (MLM) and next sentence prediction (NSP). MLM is a pre-training method where some percentage of all tokens in an input sequence is replaced with a masking token. Then, the model's task is to assign and predict the correct token for the mask. When pre-training BERT, 15% of all tokens were masked. NSP is another pre-training task where the model's task is to predict if two sentences are related to each other or not. It is used to pre-train for tasks that involve an understanding between sentences, such as a question answering task.

2.5.2 GPT

The first **G**enerative **P**re-trained **T**ransformer (GPT) was introduced by OpenAI in 2018 [31]. The architecture of the GPT model is similar to that of the decoder block in the Transformer model. It stacks 12 decoder layers on top of each other to create a sequential decoding block. The output tokens generated by the model are predicted autoregressively, which means that it can be used to generate sequences. When decoding, the model can only look at previously generated words, similar to the Transformer model.

At the time of its release, it achieved state-of-the-art in 9 of the 12 datasets it was evaluated on, including tasks such as question answering, semantic similarity assessment, and text classification. The original GPT model has 2 successors, namely GPT-2 [32] and GPT-3 [5], substantially increasing the number of parameters for each model.

2.5.3 BART

BART is a seq2seq model based on the Transformer architecture which was introduced by Facebook AI in 2019 [19]. The two building blocks of the BART model can be generalized as using BERT as the encoder, because of the bidirectional encoder, and GPT as the decoder, because of the autoregressive decoder. The difference between the original Transformer and BART is that the latter uses GeLU [14] instead of ReLU [1] as activation functions. Regarding the rest of the architecture, it is very similar to BERT with the two differences that the decoder layers perform cross-attention over the final hidden layer of the encoder and that BERT uses an additional FFNN to predict the next word, which BART does not. Each of the encoder and decoder blocks are made of 6 layers in the base model and 12 layers in the large model. The $\text{BART}_{\text{BASE}}$ model has approximately 140 million parameters, and the $\text{BART}_{\text{LARGE}}$ model has approximately 400 million parameters. Because of the similarity to BERT, BART also uses learned positional embeddings.

The model is pre-trained as a denoising autoencoder, which means that the model is trained on noisy and corrupted text. Training data are modified and corrupted in numerous ways. The modifications to the data are summarized in the list below.

- **Token Masking:** Random tokens in the input sequence are masked with a mask token. The model is then trained to predict the correct mask based on the rest of the sequence. Essentially, the same pre-training method as the MLM with BERT.

- **Token Deletion:** Random tokens are deleted from the input, which will lead the model to predict what content the deleted token had, based on the rest of the sequence. Furthermore, the model must predict what positions they had.

- **Text Infilling:** Similar to deletion, text filling removes several tokens in a row and replaces it with a single mask token. This implies that the model has to learn and predict the content of the missing tokens.

- **Sentence Permutation:** Performing sentence permutation, meaning that words in a sentence are shuffled. This will enable the model to learn the context of the input sentence, regardless of the order.

- **Document Rotation:** This method chooses a random word to start the sequence. This will teach the model about the beginning of documents.

After the pre-training phase, the model is ready to be fine-tuned for downstream tasks. Because of the autoregressive decoder, BART can be fine-tuned for sequence generation tasks such as text summarization and machine translation. The encoder takes an input sequence and generates outputs autoregressively through the decoder.

2.5.4 Marian & MarianMT

Marian is an NMT framework written entirely in C++, which is highly optimized for machine translation. The framework is mostly developed by Microsoft and the University of Edinburgh. The NLP group at the University of Helsinki has pre-trained over $1,000$ Marian models, and they were made public on Hugging Face after converting the models to Python. MarianMT is the name of the class provided by Hugging Face, where one can import the pre-trained Marian models.

The architecture of the MarianMT models is almost identical to the BART model, except for a few minor differences. The first difference is that the Marian models use the sinusoidal positional embedding instead of the learned positional embedding as is the case with BART. The second difference is that there is no layer normalization in the Marian models. The MarianMT models at Hugging Face are also slightly smaller than the $\text{BART}_{\text{BASE}}$ model, with a total of about 74 million parameters.

2.6 Metrics

Evaluating a language model is a complex task that requires specific metrics. To address one of the problems that could arise when measuring and comparing texts, consider the following sentences:

I enjoyed the show

I liked the concert

The contextual meaning of these sentences is essentially the same, but they are different in terms of words used to describe the situation. This is a common case for the task of sentence simplification and text summarization, where redundant words are removed or paraphrased. The problem that arises in these situations is that most metrics are looking at matching n-grams, meaning n number of matching words in a row, and do not take the contextual meaning into consideration.

Evaluation scores are often calculated between a candidate and one or more references. The candidate is the generated translation or prediction, and the references are the correct translations, made, for example, by a translator.

2.6.1 Cosine similarity

Cosine similarity is a way to measure the similarity between two vectors. It is defined as the cosine of the angle between the two vectors. If the cosine similarity score is 1, the two vectors have the same orientation. Likewise, if the cosine similarity score is 0, the two vectors are orthogonal. In NLP, cosine similarity can be used to measure the similarity of two strings.

Cosine similarity is formally defined as:

$$cosine\ similarity = cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{||A||\ ||B||} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2}\sqrt{\sum_{i=1}^{n} B_i^2}} \tag{2.10}$$

2.6.2 ROUGE-N

ROUGE stands for **R**ecall-**O**riented **U**nderstudy for **G**isting **E**valuation [20] and it is a set of metrics used to evaluate machine translations and text summarizations. With ROUGE-N, the N stands for the number of n-grams that are used in the current evaluation. For ROUGE-1 and ROUGE-2 it is the unigrams and, respectively, the bigrams that are evaluated. ROUGE-N is composed of recall, precision, and F1-scores.

Recall is calculated by first counting the number of overlapping n-grams found in both the candidate and the reference. This number is then divided by the number

of n-grams found in the reference. The recall formula can be seen in Equation 2.11, where TP stands for true positives and FP stands for false positive.

$$\textbf{Recall} = \frac{TP}{TP + FP} \tag{2.11}$$

Precision is, similarly to recall, also calculated by first counting the number of overlapping n-grams found in both the candidate and references. Precision is, however, calculated by dividing by the number of n-grams found in the candidate. The formula can be seen in Equation 2.12 where FN stands for false negatives.

$$\textbf{Precision} = \frac{TP}{TP + FN} \tag{2.12}$$

F1-Score is a combination of recall and precision. The combination results in a score that not only rely on giving high scores to sequences that cover as many words as possible, but also does so without outputting redundant words. The general formula for the F1-score can be found in Equation 2.13.

$$\textbf{F1} = 2 * \frac{precision * recall}{precision + recall} \tag{2.13}$$

2.6.3 BLEU

Bilingual **E**valuation **U**nderstudy (BLEU) [16] is an algorithm designed to evaluate the quality of machine translated text. It was developed with the idea that the closer a machine translation is to a professional human translation, the better it is. BLEU is also great at rewarding sentences that do not generate an abundance of reasonable words. This is done by calculating a modified n-gram precision score for the candidate.

The n-gram precision score is calculated using the following Equation:

$$p_n = \frac{\sum_{n\text{-gram} \in C} Count_{clipped}(n\text{-gram})}{\sum_{n\text{-gram} \in C} Count(n\text{-gram})} \tag{2.14}$$

The first function, $Count$, counts the number of matching n-grams between the candidate C and the references $R_1, \ldots, R_m$. The other function, $Count_{clipped}$, counts the number of clipped n-grams. The clipped n-grams refer to limiting the count for each correct word in the candidate to the maximum number of times it occurs in one of the reference sentences.

The BLEU score is usually computed by taking the modified precision score for n-grams up to 4, which means p_1, p_2, p_3 and p_4. To obtain the final score, a brevity penalty (BP in Equation 2.15) is also introduced, which penalizes short candidate sentences. This was implemented because shorter translations are more likely to

get a higher modified precision score, even though they might miss out on a lot of context. The resulting formula ends up being the following.

$$BLEU = BP \cdot exp(\sum_{n=1}^{N} w_n \cdot log \cdot p_n) \tag{2.15}$$

2.6.4 METEOR

Metric for **E**valuation of **T**ranslation with **E**xplicit **OR**dering (METEOR) [3] is a metric to evaluate machine translations. It was developed to counter some of the shortcomings with BLEU. The metric works by calculating the harmonic mean of the precision and recall of unigrams, where the recall is weighted higher. The algorithm creates an alignment between two sentences, just as in BLEU. The score is calculated by first counting the number of matching unigrams in the generated and reference translations, which is described as m, the number of unigrams in the generated translation, w_t, and the number of unigrams in the reference translations,w_r. With this information, the unigram- recall R and precision P can be calculated according to Equation 2.16.

$$R = \frac{m}{w_r}, \; P = \frac{m}{w_t} \tag{2.16}$$

With recall and precision, the weighted harmonic mean can be calculated according to Equation 2.17. The average is weighted with the recall being 9 times more important.

$$F_{mean} = \frac{10PR}{R + 9P} \tag{2.17}$$

So far, the calculations only account for the matching of single words and not longer segments occurring in both the generated- and reference translations. To account for these occurrences, n-grams matches are used to compute an alignment penalty. Mappings between the generation and references that are not adjacent increase the penalty. Calculation wise, this is done by first defining the chunks c as the set of adjacent unigrams in the generation and reference. The longer the mapping, the fewer chunks exist, which means that a perfect match results in 1 chunk. The penalty can be calculated according to Equation 2.18 where w_m is the number of mapped unigrams.

$$p = 0.5\left(\frac{c}{u_m}\right)^3 \tag{2.18}$$

Now, the final METEOR score can be calculated as in Equation 2.19.

$$METEOR = F_{mean}(1 - p) \tag{2.19}$$

3
Methods

This chapter aims to specify and explain the methods used in this thesis. The methods have been shaped in an iterative process, in which the results along the way have affected the chosen methods. The chapter starts with an introduction of the datasets used in this thesis, followed by how the data are pre-processed. All of the methods are based on using a pre-trained Transformer models. Our specific implementations and modifications of the models are described in Section 3.3.2 & 3.3.3. The methods are then evaluated with both quantitative and qualitative analysis.

3.1 Plint Data

Plint has been in the localisation business for many years, and during this time, thousands of hours of film have been translated and subtitled into many different languages. Some of the projects that the company has worked with contain JSON files where the original transcript and subtitles can be found. These files also contain metadata with information about the subtitle. An example of the metadata that exists for the subtitles is that the files contain time stamps that determine when and for how long a subtitle is visible to the viewer.

The transcript in the project file contains the spoken content of the video being subtitled. This was previously done manually, listening to the content and writing down the words, but lately most companies have used ASR for this task. The transcript is provided with time stamps for each word. Therefore, it is necessary to segment the ASR-output into fitting sentences. To solve this task, the supervisors at Plint made a script that formats the ASR-output to match the timestamps of the subtitle.

Plint provided us with three JSON-files per project, one containing the raw ASR-output, one containing the ASR-transcripted segmentations, and one containing the English subtitles. In total, we received 4443 files from 1481 subtitling projects. In order to make use of these data, and to construct a dataset, we had to perform a few steps of pre-processing. To begin with, we merged the ASR-segmentations and the English subtitles into a single file. We chose not to merge the raw ASR-outputs because we already had the segmented version of the ASR-transcripts. When constructing the dataset, we labeled the ASR-transcriptions as *"input"* and the subtitles as *"truth"*. We ignored all the metadata because it did not serve us any purpose.

The method to segment the ASR-outputs based on time stamps produced overall well-aligned matchings between the ASR-output and the subtitles. However, when evaluating the data, we found outliers in which single words, at the beginning or end of a sentence, had been placed incorrectly. Hence, we had to find a solution to this problem, and we came up with an algorithm that solved this by comparing the endings and beginnings of two succeeding data pairs, where the data pairs are the input - truth alignments from above. If the endings of the first data pair do not match, the endings are compared to the beginnings of the second data pair. If the algorithm then finds a match between the endings of the first pair and the beginnings of the second pair, one of the words has been placed incorrectly, and the data pairs are adjusted. If there is a further mismatch, that is, if none of the words at the beginnings and endings matches at all, the data pair is removed.

The ASR-output and the English subtitles are also compared by content, in order to filter out pairs that are not related even though they might share the first and last words. This is done by calculating the cosine similarity between the two. If the similarity is less than 0.5, the data pair is discarded. Setting the threshold too low would allow for faulty data pairs, where a too strict threshold would discard examples with linguistic compression, which would defeat the purpose of the project.

As stated in Section 1.1, there are guidelines on how to announce which person is talking. Since the transcript contains only the spoken language, the speaker indicators in the English subtitles are also filtered out. A constructed example from the filtered dataset can be seen in Table 3.1.

	Input	**Truth**
text	Our fine saint Donald Duck is much superior	Our saint Donald Duck is superior

Table 3.1: A constructed example of how an entry in the Plint dataset looks like.

3.2 Public Data

In addition to the data provided by Plint, we used two open-source datasets. This section will cover the details of each of these datasets, but also how we processed and modified the data to create more suitable datasets for our purposes.

OpenSubtitles [21] is a dataset consisting of subtitles retrieved from the web page *opensubtitles.org*. The dataset is composed of a collection of subtitles for 62 languages and 1782 bitexts. The set of subtitles from English to Swedish contained about 17 million subtitle pairs.

WikiLarge [41] is a dataset that is typically used for sentence simplification. The dataset is constructed by aligning a complex version of a sentence with its corre-

sponding simpler version. The complex sentences come from the regular English Wikipedia[1] web page, and the simple sentences come from the Simple English Wikipedia[2] web page. The dataset contains 296,402 sentences used for training, all based on the complex-simple alignment. WikiLarge also has a test and validation set based on the Turkcorpus [39], these two sets contain 2000 and 359 sentences, respectively.

3.2.1 Backtranslation with OpenSubtitles

By using backtranslation on the OpenSubtitles dataset for English to Swedish, it was possible to find a large proportion of linguistic compression and paraphrasing between the subtitles. Backtranslation was performed on the Swedish subtitles, which means that the Swedish subtitles were translated back to English. This method was inspired by a paper from Netflix [25], and the translations were made with a MarianMT model from Hugging Face. The backtranslation was a computationally heavy task, and therefore we ended up translating only a subset of the total subtitles that we had available. In total, we translated 260,000 Swedish subtitles back to English. The checkpoint we used was *"Helsinki-NLP/opus-mt-sv-en"*, which is the pre-trained checkpoint for machine translation from Swedish to English with the MarianMT model.

To verify that the translations were correct and useful, we started to analyse the translations manually. However, this method was both ineffective and time-consuming. Therefore, we needed a way to automatically analyse the translations. The manual analysis did, however, produce an important insight; we discovered that when the OpenSubtitles data had been aligned poorly, there was also a bad translation. A bad alignment meant that parts of the subtitles did not match, for example, some parts of the translation could have been offsetted to the next subtitle.

To solve this issue, we introduced a filter based on cosine similarity. In theory, this will give a low score if the translations are aligned poorly, because very few of the words will match, and a high score if the translations are aligned correctly, because more words will match between the sentences. However, this method has the downside of punishing short sentences for paraphrasing words, resulting in a very low cosine similarity score. Therefore, we had to find a sweet-spot where the bad translations were removed, but the good ones were kept. We went with a threshold for cosine similarity of 0.5, and it gave the results we were after. Furthermore, we also filtered subtitles on the basis of different length criterion, this filter had limitations in character lengths and length ratios. The applied length limitation was to only consider subtitles with character lengths between 5 and 100. The ratio criteria were applied to only consider subtitles with a ratio below 2. The resulting dataset contains 147,548 backtranslated subtitles, and a visualization of the length and ratios of the data can be seen in Figure 3.1. Further on, we refer to this dataset as *"OpenBack"*.

[1] en.wikipedia.org
[2] simple.wikipedia.org

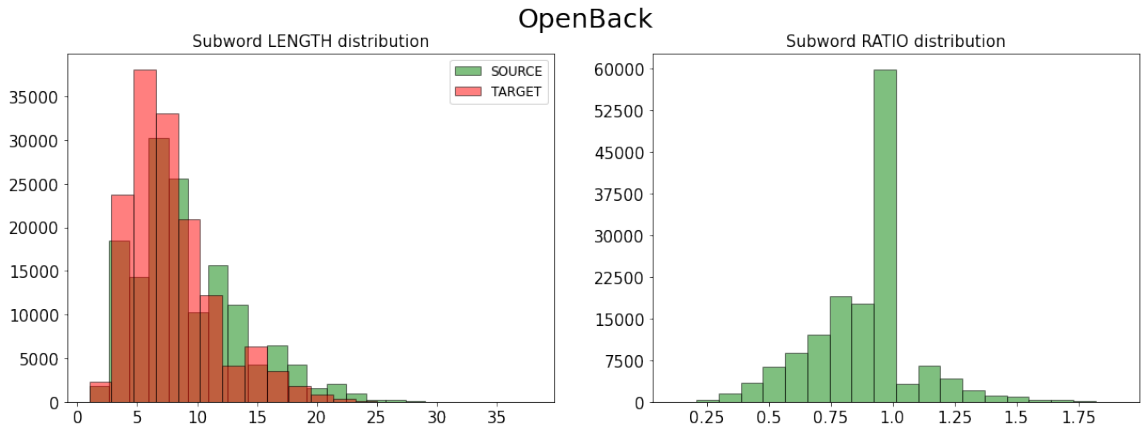

Figure 3.1: Visualization of the distributions for subword lengths and subword ratios of the OpenBack dataset.

3.2.2 OpenSubtitles - English to Swedish

To train a translation model, we needed subtitle data within two languages. Because we had already used the OpenSubtitles data for backtranslations, it was suitable to use this dataset for the purpose of translation as well. To avoid the problems of poorly aligned data, we used the same data as in the backtranslation. Using the same data would also assure us that the translations were correct, as the backtranslated data were correct according to our filter based on the cosine similarity. However, the data still needed to be processed through the length filter to make sure the lengths and ratios criterion was still fulfilled. The length filter that we applied was the same as in the previous task, meaning a threshold of 2 on the length ratio and character lengths between 5 and 100. The resulting dataset with English - Swedish subtitles consists of 144,968 subtitles.

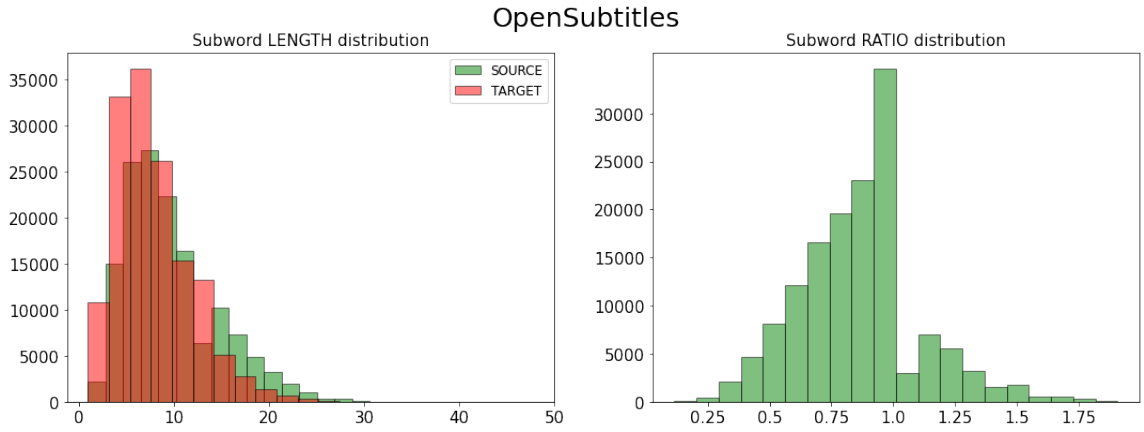

Figure 3.2: Visualization of the distributions for subword lengths and subword ratios of the OpenSubtitles dataset.

3.2.3 WikiOpen

The use of the WikiLarge dataset was inspired by the sentence simplification methods of Martin et al. [24][23]. The dataset contains sentences of various lengths, where the shortest are only a few characters long, and the longest are up to 500 characters long. To fit our purpose, we had to filter the data so that it consisted only of sentences that could work as a subtitle.

Our first step in filtering out the sentences was to manually evaluate the dataset and just as we found in the OpenSubtitles data, there were instances of poorly aligned sentence pairs. Therefore, our first step was to implement the same cosine filter with a threshold of 0.5. We also applied the same length and ratio restrictions, which means that we only consider data with character length in a range between 5 and 100 and a ratio between 0 and 2. To create a larger dataset, we ended up combining these data with the backtranslated OpenSubtitles data. We named this dataset *"WikiOpen"* and it contains a total of 202,718 instances, where 147,723 of them are subtitles from OpenBack and 54,995 are sentences from WikiLarge. The subword length and subword ratio distributions in this dataset can be seen in Figure 3.3.

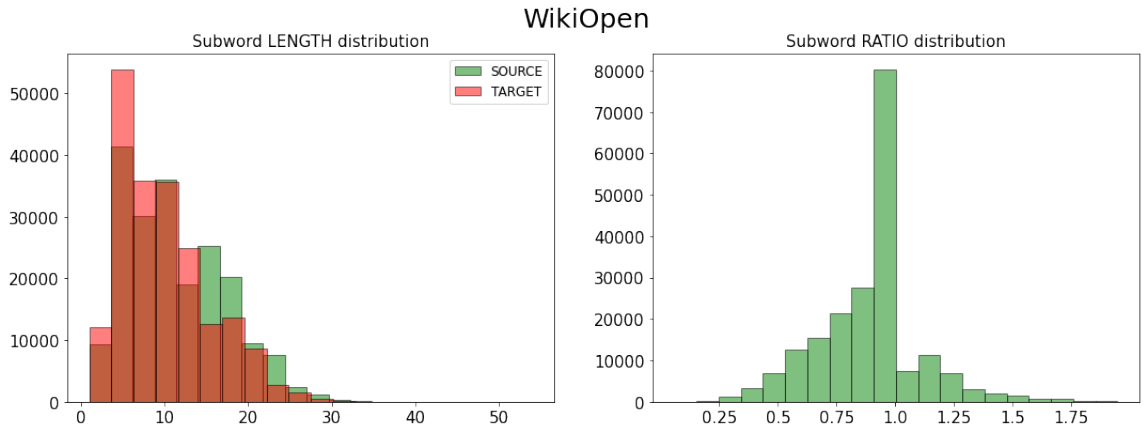

Figure 3.3: Visualization of the distributions for subword lengths and subword ratios of the WikiOpen dataset.

3.3 Model

All the methods in this thesis are based on using a Transformer model to generate the subtitles and we found that the BART model was suitable for the task of English to English and the Marian model for Swedish to English.

For most parts, the project utilizes pre-trained models as a starting point. This means that our methods are based on models that are fine-tuned on the transferred weights from a pre-trained model. When inheriting the weights of the pre-trained model, the tokenizer and vocabulary follow as well. However, in most cases, changes or additions to the tokenizer were necessary.

3.3.1 Hugging Face

Hugging Face is a company that provides open-source NLP technologies. The research leaders such as Facebook AI, Microsoft, and Google AI are all supporting and uploading their models on Hugging Face for public use. This is beneficial for both research and educational purposes, where smaller research institutes and universities can access state-of-the-art models for free. The community around Hugging Face has also contributed with thousands of fine-tuned models.

When downloading a model from Hugging Face, it is possible to download only the architecture of a model and then pre-training it from scratch. However, it is also possible to download a pre-trained model if the user defines a checkpoint. The checkpoints are essentially the current state of the model's weights and tokenizer. This enables the user to access the freely available state-of-the-art models that have been released onto Hugging Face. Likewise, the user can also access the models that have been fine-tuned for downstream tasks by the community. All of the pre-trained models that we use in this thesis are all retrieved from Hugging Face.

3.3.2 Transformer with Length Encoding

To create a position-aware encoding, we introduce the positional encoding method proposed by Sho & Takase [37]. They train a Transformer model to constrain the output by implementing the length encodings seen in Equation 3.1, which is a slight modification of Equation 2.9. These new encodings look at the current position with respect to the remainder of the sequence, instead of the beginning of the sequence, as is the case with the original implementation.

$$\begin{aligned} PE_{len,pos,2i} &= sin\Big(\frac{len - pos}{10000^{\frac{2i}{d_{model}}}}\Big) \\ PE_{len,pos,2i+1} &= cos\Big(\frac{len - pos}{10000^{\frac{2i+1}{d_{model}}}}\Big) \end{aligned} \tag{3.1}$$

Here, *len* represents the specified length of the output sequence, which means the length of the final subtitle, and *pos* represents the current position in the sequence. The other parameters remain the same as in the original implementation (see Section 2.4.2). BART is based on a BPE tokenizer, and therefore we experimented with two different methods on how to count the *len* and *pos* parameters. During training, *len*, is the length of the target sequence and during inference, it is the desired length of the output.

Subword count was the first method that we used to represent the different lengths. It is based on the tokenizer's decoding ability, meaning that we based the length parameters on the number of subwords from the tokenizer. This implies that all tokens are treated equally in length, hence all have the length of 1.

Character count was the second method that we planned to use to represent the different lengths. In contrast to the subword count, this method takes each individual subword length into consideration. Therefore, the position *pos*, is calculated by counting the character length of each of the preceding subword tokens. Similarly, the length, *len*, is simply the character length of the target sequence.

3.3.3 Transformer with Length Token

In addition to the modified positional encoding, we introduced new tokens that would represent different lengths to the model. The length token is prepended to each source sentence and is meant to represent the relation between the source sentence and the target sentence. The tokens were created by adding a token that represented some value to the tokenizer. We used an approach with angle brackets in combination with the corresponding value that we wanted to represent, the tokens therefore had the shape of "<value>". We experimented with a few different methods on how to assign the value when creating our length tokens, but the two main concepts were to create the tokens based on length or by ratio.

The first method was to prepend the length of the target sentence to the source sentence. The length was retrieved by tokenizing the target sentence with the tokenizer. Therefore, the length tokens would look like "<5>" for sequences with five subwords.

The second method was to make the length tokens aware of the length ratio between the source sentence and the target sentence. To calculate the ratio, we divided the subword length of the target sentence with the subword length of the source sentence (see Equation 3.2). For this method, we used two versions of tokens, a category-based version and a value-based version.

$$\frac{subword\_length\_target}{subword\_length\_source} = ratio \tag{3.2}$$

The first version of the ratio method was to use only three tokens to represent the ratios. The three tokens that we created were "<shrt>", "<norm>" and "<long>", which indicates the lengths of short, normal, and long. The calculation to get these tokens was carried out according to Equation 3.2, and the tokens were assigned using the following rules:

$ratio < 0.95 \implies <shrt>$
$1.05 > ratio > 0.95 \implies <norm>$
$ratio > 1.05 \implies <long>$

The second version used value-based ratio tokens. This implied that we created the ratio tokens based on the actual value between the target and source sentences rather than putting them into categories. We created 20 tokens between 0 and 2, with an interval of 0.10. The source sentences were then assigned with a token for the value-based ratio based on the closest ratio.

All of the tokens that we created were added to the tokenizer, where their weights were initialized randomly. This enabled the model to distinguish each token individually during training. However, this also means that tokens that are close to each other in value, such as "<11>" and "<12>" for example, might not be related to each other at all in the embedding dimensions.

3.4 Training

In this project, we made use of cloud-based computing to train our models. The company provided us with an instance on the EC2 platform at Amazon Web Services (AWS) where we had access to a NVIDIA T4 Tensor Core GPU. When training on the Plint data, it was important that it did not leave the AWS servers due to legal rights. However, when training the models on the data from the public datasets, we were able to use free computing resources. Therefore, these models were trained on GPUs at Google Colab and Kaggle.

3.5 Evaluation

To evaluate our models, we used multiple metrics combined with manual analysis. The metrics we used to evaluate our models were BLEU, ROUGE-N, and METEOR. These metrics were evaluated with an implementation of the Jury library [7]. The manual analysis was based on evaluating the models with focus on the linguistic performance, which is difficult to measure with metrics. This analysis was performed by picking random samples in the corresponding test sets and evaluating the impact of each token.

4
Results

This chapter presents the results and evaluations obtained in this project. The results are based on experiments with different combinations of the methods described in Chapter 3. The performed experiments can be seen in the list below:

- **BART** with Length Encodings and Length Tokens
  - Fine-tuned with Plint data
- **BART** with Category-based Ratio Tokens
  - Fine-tuned with WikiOpen data
  - Fine-tuned with OpenBack data
- **BART** with Value-based Ratio Tokens
  - Fine-tuned with WikiOpen data
  - Fine-tuned with OpenBack data
- **Marian** with Category-based Ratio Tokens
  - Fine-tuned with OpenSubtitles data
- **Marian** with Value-based Tokens
  - Fine-tuned with OpenSubtitles data

The results are presented with tables and figures showing the metrics and length ratios for each method. Furthermore, this chapter contains examples generated from each of the evaluated models. A further discussion of the results can be found in Chapter 5.

4.1 BART with Length Encodings and Tokens

Our first experiments consisted of fine-tuning a BART model with the length encodings introduced in Section 3.3.2, together with a length token based on the subword length of the target sentence. This method utilizes the pre-trained checkpoint "*facebook/bart-base*", when initializing the BART model. For fine-tuning the model, we used the Plint dataset where we split 90% of the data into a training set and evenly split the remaining data into a validation and test set equal to 5%. Furthermore, we used a batch size of 64 and the learning rate was set at $3 \cdot 10^{-5}$. Cross-entropy loss was used as the loss function.

4.1.1 Baseline

The baseline model for this method was created using the BART model for the pre-trained checkpoint without any modifications, which means without any length encodings or length tokens. The model is fine-tuned for 5 epochs with an evaluation of the validation set at epochs 0, 2, and 4. The graph of average (over batch) training and validation loss can be seen in Figure 4.1.

A few generated examples from the baseline model can be seen in Table 4.1. **Gen** declares an unconstrained beam search, with beam size of 5, while $\textbf{Gen}_{pen}$ has the same amount of beams but utilizes the n-gram penalty (bi-gram) mentioned in Section 2.3.3 (the probability for a word that creates a bi-gram that has already appeared in the sequence is set to 0). The generated sequences are similar to the input with the exception that the model is adding the "
" token, which is the token for line breaks, when the sequences tend to get longer. The model also misses the last couple of tokens in the last example.

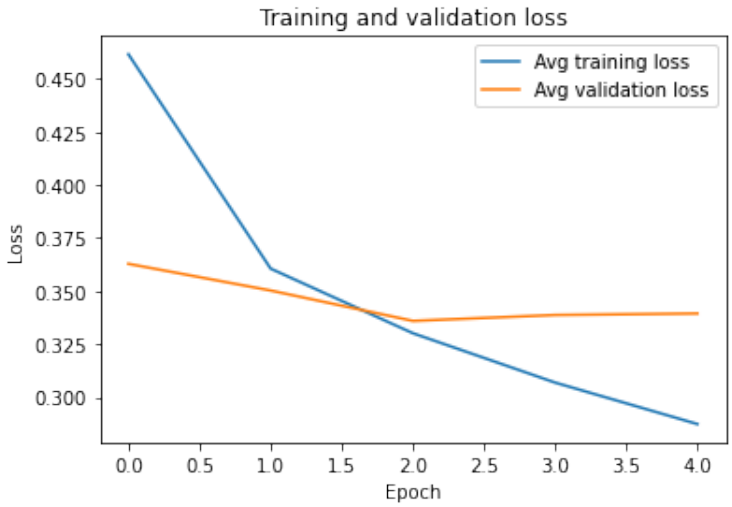

Figure 4.1: Average training loss plot of the initial baseline model.

4.1.2 Length Encodings and Length Tokens

To evaluate the method with length encodings and length tokens, we modified the BART model to use these as an additional input. Looking at the target length distribution of the Plint dataset in Figure 4.2, we figured that a maximum length of 50 would be more than sufficient for our task. This means that tokens up to "<50>" could be used in our model.

This model was fine-tuned for 11 epochs, where the evaluation of the validation set was performed every 5 epochs. The graph of average (over batch) training and validation loss can be seen in Figure 4.3.

Baseline - Epoch 4	
Input	That was her idea, not mine.
Gen	That was her idea, not mine.
Gen$_{pen}$	That was her idea, not mine.
Target	It was her idea.
Input	You were planning to marry Mrs. Van Dorn, weren't you?
Gen	You were planning to marry Mrs. Van Dorn, weren't you
Gen$_{pen}$	You were planning to marry Mrs. Van Dorn, weren't you
Target	You were going to get married, weren't you?
Input	I've just had time to think things out put myself in your position.
Gen	I've just had time to think things out, put myself in your
Gen$_{pen}$	I've just had time to think things out, put myself in your
Target	I've put myself in your position

Table 4.1: Generated sequences by the BART baseline model fine-tuned on the Plint dataset for 5 epochs. **Gen** corresponds to a beam search with a beam size of 5 and **Gen**$_{pen}$ to a bi-gram penalised beam search with beam size of 5.

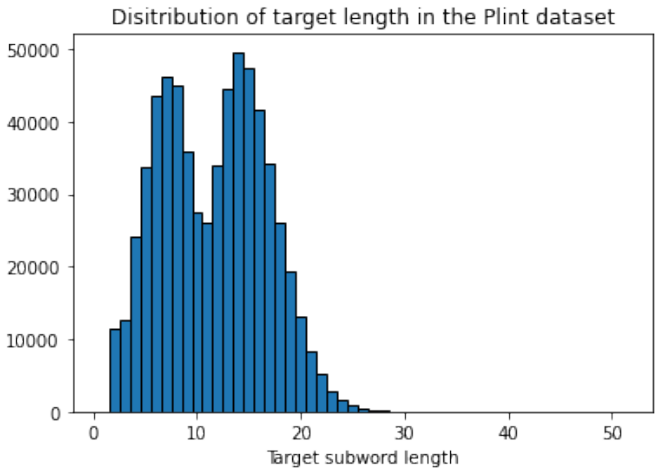

Figure 4.2: Distribution of target subword length in the dataset from Plint.

Although a reasonable loss graph, and in contrast to the similar implementation in [37], the approach did not work, thus resulting in the model generating gibberish. Some examples can be seen in Table 4.2 where the model seems to begin to stutter and repeat itself. Generation was carried out in the same way as with the baseline model, which means a beam search with 5 beams, but also with a variant using a bi-gram penalty.

We also experimented with freezing parts of the model. The experiment consisted of partially freezing the encoder, which means that the weights within the encoder were prevented from being updated. The first approach was to freeze the entire model except for the word embeddings, and the second was to also freeze the word embeddings except for our new length tokens. However, these methods did not result in any success and are therefore not presented.

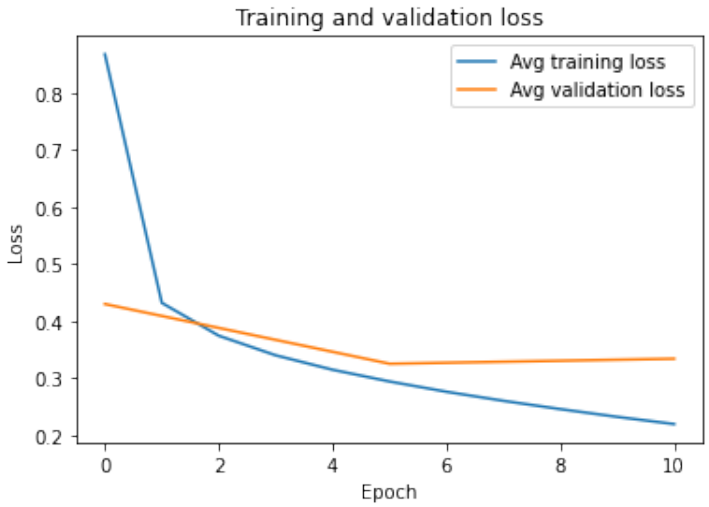

Figure 4.3: Average training loss plot of the the initial model.

To gain a better understanding of the failed implementation, a visualization of the added positional encodings compared to the pre-trained BART position embeddings can be seen in Figure 4.4. As stated in Equation 3.1, the sinusoidal approach ranges from -1 to 1, while the learned embeddings are mostly centered around 0 (with the exception of a single value being approximately -3.9, which is not visible but explains the colour bar). From this it is possible to see that the sinusoidal encodings have a much higher variance than the learned ones, and this could possibly cause problems when they are added together in the decoder.

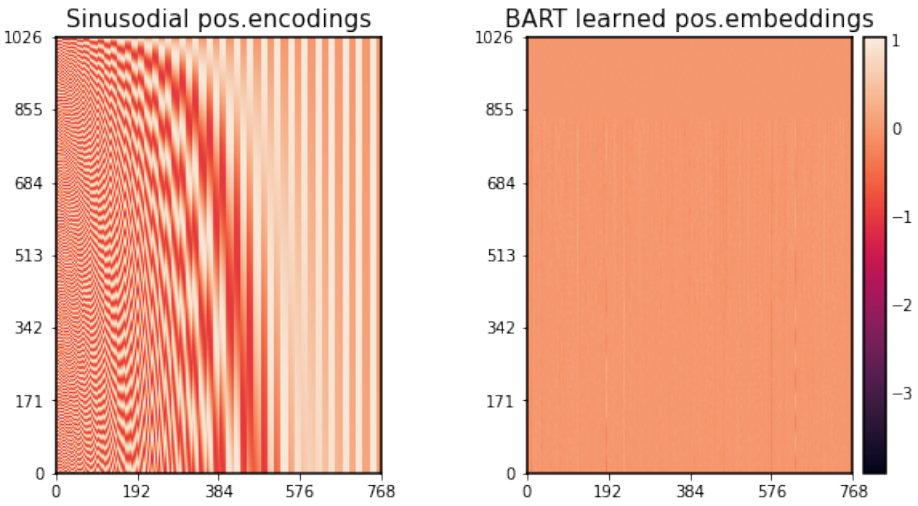

Figure 4.4: Visualization of the implemented sinusodial positinal encodings and the original learned positinal embeddings of the BART-checkpoint. The sinusodial positional encodings range from -1 to 1 while the original learned embeddings are mostly centered around 0

Length Encodings and Length Tokens - Epoch 5	
Input	<7> That was her idea, not mine.
Gen	That was her idea, not mine not mine. That was my idea. That
Gen$_{pen}$	That was her idea, not mine I'm sorry. That was my idea.
Target	It was her idea.
Input	<13> You were planning to marry Mrs. Van Dorn, weren't you?
Gen	You were planning to marry Mrs. Van Dorn Van Dorn Van D
Gen$_{pen}$	You were planning to marry Mrs. Van Van Dorn Van. Van You
Target	You were going to get married, weren't you?
Input	<10> I've just had time to think things out put myself in your position.
Gen	I've just had time to think things out. Put myself in to think things
Gen$_{pen}$	I've just had time to think things out. Put myself in out to think
Target	I've put myself in your position
Length Encodings and Length Tokens - Epoch 10	
Input	<7> That was her idea, not mine.
Gen	That was her thatThatThatThat ThatThatThatIt wasThatThat thatThat was
Gen$_{pen}$	That was her That thatThatThat ThatThatIt was That That wasThat that That
Target	It was her idea.
Input	<13> You were planning to marry Mrs. Van Dorn, weren't you?
Gen	You were planningYou wereYou planningYou planning planningYou wanted planningYou would marryYou
Gen$_{pen}$	You were planningYou planning planningToYou,You would marryYou wanted planning toYou
Target	You were going to get married, weren't you?
Input	<10> I've just had time to think things out put myself in your position.
Gen	I just had time time timeHadIIIHad time timeI timeII
Gen$_{pen}$	I just had time timeIIHad timeTime toITo think timetoI
Target	I've put myself in your position

Table 4.2: Generated sequences by the BART model with additional length encodings and length tokens, fine-tuned on the Plint dataset for 5 epochs. **Gen** corresponds to a beam search with a beam size of 5 and **Gen**$_{pen}$ to a bi-gram penalised beam search with beam size of 5.

4.2 BART with Ratio Tokens

Based on the methods in Section 3.3.3, the ratio tokens are created to represent different length ratios between the source and target sentences. To evaluate this method, we fine-tuned the BART model using the WikiOpen and OpenBack datasets. The datasets were divided into train, validation, and test sets with a ratio of 80/15/5. Fine-tuning was performed with a batch size of 64 and a learning rate

of $3.10 \cdot 10^{-4}$. The checkpoint used to initialize our model prior to fine-tuning was *"facebook/bart-base"*. All models had converged after 5 epochs, meaning no improvement in validation loss, and hence this was used for all models.

At inference time, we create a test set by backtranslating new data, similar to Section 3.2.1. This set contains 1000 sentences, and the distribution of the category-based and value-based ratio tokens can be seen in Figure 4.5 & 4.6.

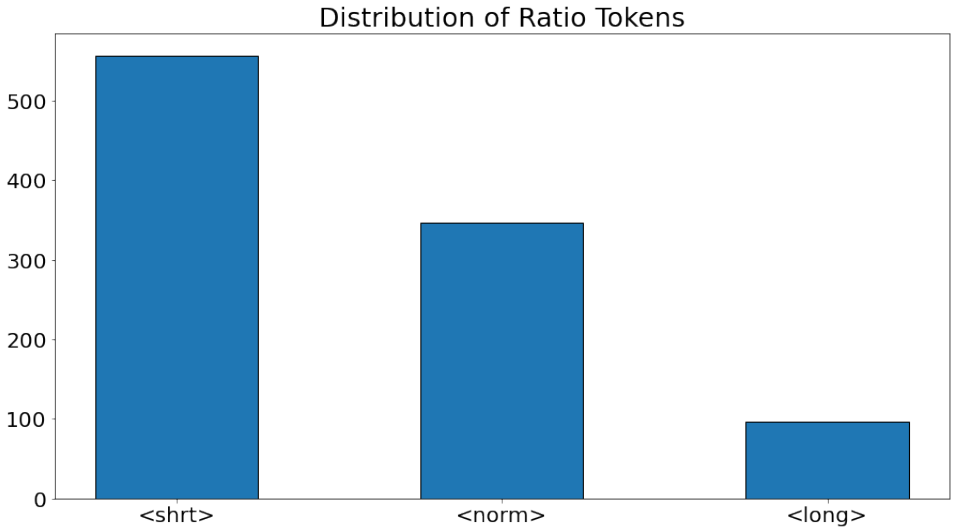

Figure 4.5: Distribution of the test set based on ratio categories.

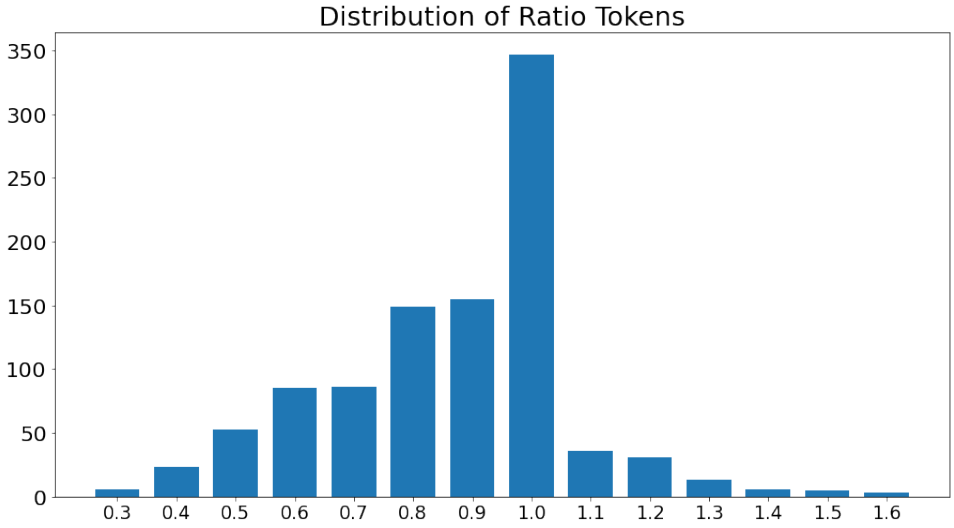

Figure 4.6: Distribution of the test set based on ratio values.

By prepending each of the sentences in the test set with the different ratio tokens, we could generate evaluation data to analyse the impact of each token. To generate the evaluation data, we used a beam search with a beam size of 3. The evaluation was constructed in two ways, where the first method was to prepend all sentences

in the test set with each token. The second method was to divide the test set into groups of length tokens, which means that the token was evaluated in the group to which it belongs based on its ratio with the corresponding target sentence. This implied that we created new sets within the test set to evaluate the performance of each token.

The evaluation is based on measuring the impact of each token by first analysing the generated length ratio compared to the source sentence. This is denoted as LR^{src} and was done for all sentences within the test set and therefore corresponds to the first evaluation strategy. Furthermore, the evaluation and analysis are based on the metrics of Section 3.5 and this is carried out for each token group, hence this corresponds to the second evaluation strategy. We use ROUGE-2_{recall} to evaluate the ROUGE-N metric. Lastly, there is a human evaluation of random samples of the generated sentences.

4.2.1 Baseline

The baseline models were created by fine-tuning BART on each of the two datasets without any length tokens, meaning that it was fine-tuned only on the source - target aligned sentences. This method was motivated by the need to be able to measure the impact of the tokens. The metrics evaluated for the baseline models can be seen in Table 4.3.

BART - BASELINE

Token	WikiOpen LR^{src}	OpenBack LR^{src}
	0.88	0.88

Table 4.3: The mean length ratios against the source (LR^{src}) for each dataset with the baseline models are presented in the columns of this table. The results were evaluated from a fine-tuned BART model on the WikiOpen and OpenBack datasets without any ratio tokens.

BART - BASELINE

	WikiOpen			OpenBack		
Token	BLEU	ROUGE-2	METEOR	BLEU	ROUGE-2	METEOR
	50.15	62.74	76.28	50.39	62.26	75.65

Table 4.4: The metrics evaluated for each dataset with the baseline models. The results were evaluated from a fine-tuned BART model on the WikiOpen and OpenBack datasets without any ratio tokens.

4.2.2 Category-based Ratio Tokens

The first version of the method with ratio tokens was to use three categories to represent short, normal, and long sentences. The distribution of the three tokens for each dataset can be found in Figure 4.7. The resulting length ratios can be found in Table 4.5 & 4.6.

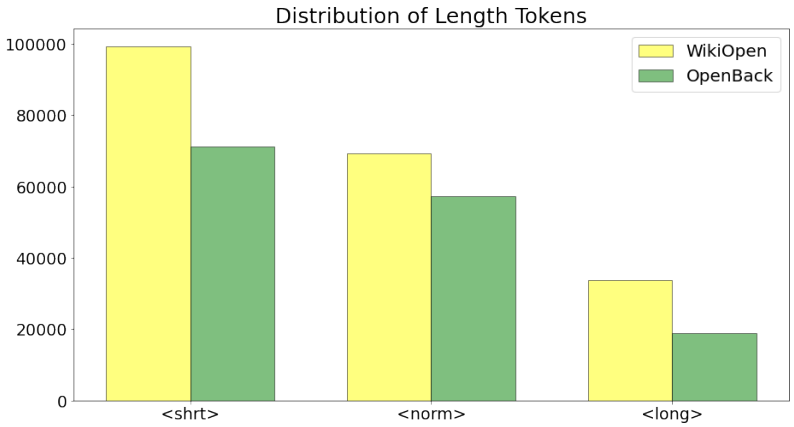

Figure 4.7: The distribution of the category-based ratio tokens in WikiOpen and OpenBack

BART - CATEGORIES

Token	WikiOpen LR^{src}	OpenBack LR^{src}
<shrt>	0.76	0.78
<norm>	0.98	0.98
<long>	1.13	1.15

Table 4.5: The resulting mean length ratios against the source (LR^{src}) with category-based ratio tokens on the WikiOpen and OpenBack datasets. The left-most column contains the evaluated token.

BART - CATEGORIES

	WikiOpen			OpenBack		
Token	BLEU	ROUGE-2	METEOR	BLEU	ROUGE-2	METEOR
<shrt>	46.42	57.22	71.25	46.14	56.83	70.81
<norm>	74.07	80.84	87.76	74.80	81.75	88.25
<long>	45.22	54.66	68.89	46.95	56.21	69.35

Table 4.6: The evaluated metrics for each dataset with the category-based tokens. The results are evaluated from a fine-tuned BART model on the WikiOpen and OpenBack datasets with tokens representing short, normal, and long sentence ratios.

4.2.3 Value-based Ratio Tokens

The second version of the method using ratio tokens was to use 20 value-based tokens ranging from 0 to 2, with an interval of 0.1. The distribution of the tokens in the data can be seen in Figure 4.8. The results of this method can be seen in Table 4.7 & B.1. Due to the unbalance of the value-based ratio tokens we only present the metrics for these tokens in the range of 0.5 to 1.2. For a full presentation of the results for each token we refer to Appendix B.

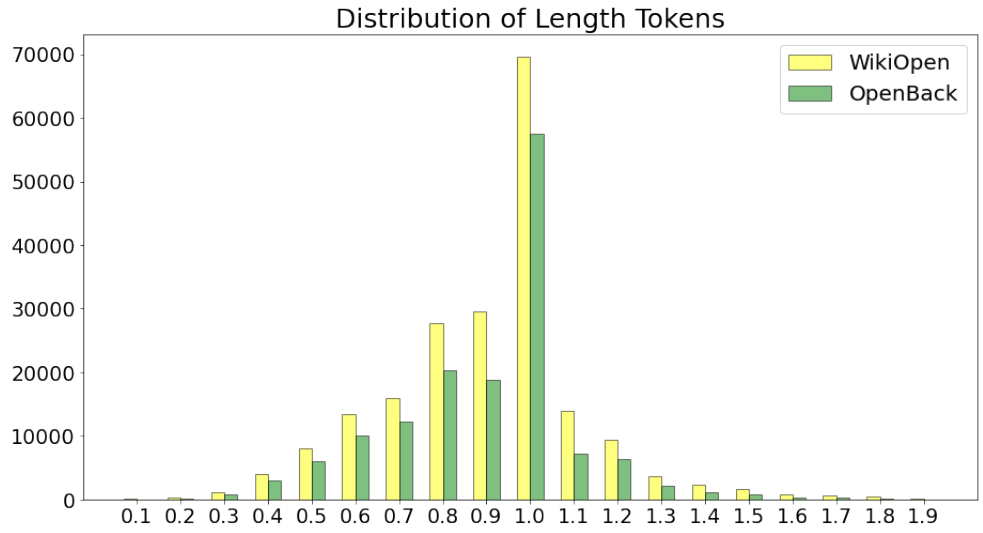

Figure 4.8: The distribution of each value-based ratio token for the WikiOpen and OpenBack datasets. The x-axis can be interpreted as the corresponding ratio token.

4.2.4 Manual Token Evaluation

To fully understand the effect of the tokens, we also carried out a manual evaluation. The results are evaluated by picking a subset of the test set to analyse the individual tokens' affect on the linguistics, with a focus on the fluency and adequacy. In Table 4.9-4.12 are a few examples of the analysis, but the entire set can be found in Appendix A. From this analysis we can see that the model generates the sentences with respect to each token and that the semantic meaning of each sentence is maintained. It is also possible to see that the two datasets generates slightly different results but overall they are very similar. For a further discussion about the result and our findings, see Chapter 5.

BART - VALUES

Token	WikiOpen LR^{src}	OpenBack LR^{src}
<ratio_0.20>	0.30	0.36
<ratio_0.30>	0.38	0.42
<ratio_0.40>	0.45	0.48
<ratio_0.50>	0.54	0.56
<ratio_0.60>	0.62	0.64
<ratio_0.70>	0.71	0.71
<ratio_0.80>	0.83	0.81
<ratio_0.90>	0.94	0.90
<ratio_1.00>	0.99	0.98
<ratio_1.10>	1.13	1.04
<ratio_1.20>	1.21	1.13
<ratio_1.30>	1.30	1.25
<ratio_1.40>	1.35	1.30
<ratio_1.50>	1.40	1.34
<ratio_1.60>	1.46	1.41
<ratio_1.70>	1.50	1.42

Table 4.7: The resulting mean length ratios against the source (LR^{src}) with value-based ratio tokens on the WikiOpen and OpenBack datasets. The leftmost column contains the evaluated tokens and in the other two columns are the corresponding mean length ratios for each dataset.

BART - VALUES

	WikiOpen			OpenBack		
Token	BLEU	ROUGE-2	METEOR	BLEU	ROUGE-2	METEOR
<ratio_0.50>	44.48	58.45	68.58	43.72	58.27	69.87
<ratio_0.60>	48.52	58.45	72.85	46.67	62.14	74.40
<ratio_0.70>	50.39	62.68	74.64	47.67	61.14	74.26
<ratio_0.80>	48.24	57.78	71.72	46.98	56.43	70.69
<ratio_0.90>	48.66	58.95	75.97	49.29	59.11	74.94
<ratio_1.00>	75.46	81.30	87.87	74.88	80.88	87.84
<ratio_1.10>	54.43	62.34	77.62	55.41	64.64	77.96
<ratio_1.20>	41.32	57.51	72.15	43.18	55.90	70.24

Table 4.8: The evaluated metrics for each dataset with the value-based ratio tokens. The results are evaluated from a fine-tuned BART model on the WikiOpen and OpenBack datasets with 20 tokens representing length ratios between 0 and 2, with an interval of 0.1. The leftmost column show the token used to evaluate the model.

Example 1 - BART - WikiOpen	
Source:	No, that doesn't make any difference.
Target:	No. It doesn't make any difference.
Target Token:	<norm> & <ratio_1.00>
Baseline	
	No, it doesn't matter.
Categories	
<shrt>	It doesn't matter.
<norm>	No, it doesn't matter at all.
<long>	No, that doesn't make a difference.
Values	
<ratio_0.50>	It doesn't matter.
<ratio_0.60>	It doesn't matter.
<ratio_0.70>	It doesn't matter.
<ratio_0.80>	That doesn't make a difference.
<ratio_0.90>	No, it doesn't matter.
<ratio_1.00>	No, it doesn't make a difference.
<ratio_1.10>	No, it doesn't make a difference.
<ratio_1.20>	No, that doesn't make a difference at all.

Table 4.9: Sentences generated with the BART model fine-tuned on WikiOpen

Example 1 - BART - OpenBack	
Source:	No, that doesn't make any difference.
Target:	No. It doesn't make any difference.
Target Token:	<norm> & <ratio_1.00>
Baseline	
	No, it doesn't matter.
Categories	
<shrt>	It doesn't matter.
<norm>	No, that doesn't make any difference.
<long>	No, that doesn't make any difference.
Values	
<ratio_0.50>	It doesn't matter.
<ratio_0.60>	It doesn't matter.
<ratio_0.70>	It doesn't matter.
<ratio_0.80>	No, that doesn't matter.
<ratio_0.90>	No, that doesn't matter.
<ratio_1.00>	No, that doesn't make any difference.
<ratio_1.10>	No, that doesn't make any difference.
<ratio_1.20>	No, it doesn't make any difference at all.

Table 4.10: Sentences generated with the BART model fine-tuned on OpenBack

Example 2 - BART - WikiOpen	
Source:	I'm going to stay with you.
Target:	I'll stay with you.
Target Token:	<shrt> & <ratio_0.80>
Baseline	
	I'll stay with you.
Categories	
<shrt>	I'll stay with you.
<norm>	I'm going to stay with you.
<long>	I'm going to stay with you for a while.
Values	
<ratio_0.50>	I'll stay.
<ratio_0.60>	I'm staying.
<ratio_0.70>	I'll stay with you.
<ratio_0.80>	I'll stay with you.
<ratio_0.90>	I'm coming with you.
<ratio_1.00>	I'm going to stay with you.
<ratio_1.10>	I'm going to stay with you for a while.
<ratio_1.20>	I'm going to stay with you for a while.

Table 4.11: Sentences generated with the BART model fine-tuned on WikiOpen

Example 2 - BART - OpenBack	
Source:	I'm going to stay with you.
Target:	I'll stay with you.
Target Token:	<shrt> & <ratio_0.80>
Baseline	
	I'll stay with you.
Categories	
<shrt>	I'll stay with you.
<norm>	I'm going to stay with you.
<long>	I'm going to stay with you for a while.
Values	
<ratio_0.50>	Stay with you.
<ratio_0.60>	I'll stay with you.
<ratio_0.70>	I'll stay with you.
<ratio_0.80>	I'll stay with you.
<ratio_0.90>	I'll stay with you.
<ratio_1.00>	I'm going to stay with you.
<ratio_1.10>	I'm gonna stay with you for a while.
<ratio_1.20>	I'm going to stay with you for a while.

Table 4.12: Sentences generated with the BART model fine-tuned on OpenBack

4.3 Marian with Ratio Tokens

In addition to the monolingual BART model, we also implemented the same methods as in the previous section with the Marian model (see Section 2.5.4). Using the same implementation also implies that we used the same training procedure, hence a learning rate of $3.10 \cdot 10^{-4}$, batch size of 64, and that we trained for 5 epochs. Furthermore, we used the OpenSubtitles dataset with data from English to Swedish to fine-tune this model. This dataset was also divided into train, validation, and test sets according to a ratio of 80/15/5. The checkpoint used to initialize the model prior to fine-tuning was *"Helsinki-NLP/opus-mt-en-sv"*.

During inference, we used the associated test set to generate 1000 sentences for evaluation. The data was generated with the same methods as in the previous section, meaning a beam search with a beam size of 3. Furthermore, we used the same methods to measure the impact of each token, meaning that we performed analysis on the impact of each length token but also on the linguistic metrics. The distribution for the category- and ratio-based ratio tokens can be seen in Figure 4.9 & 4.10.

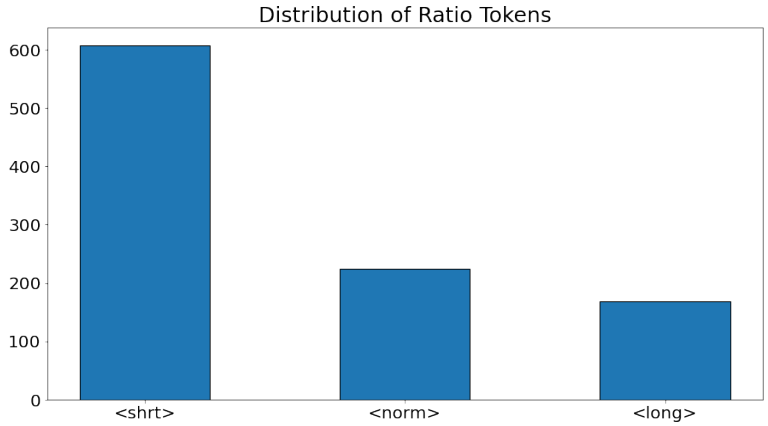

Figure 4.9: Distribution of the test set based on ratio categories

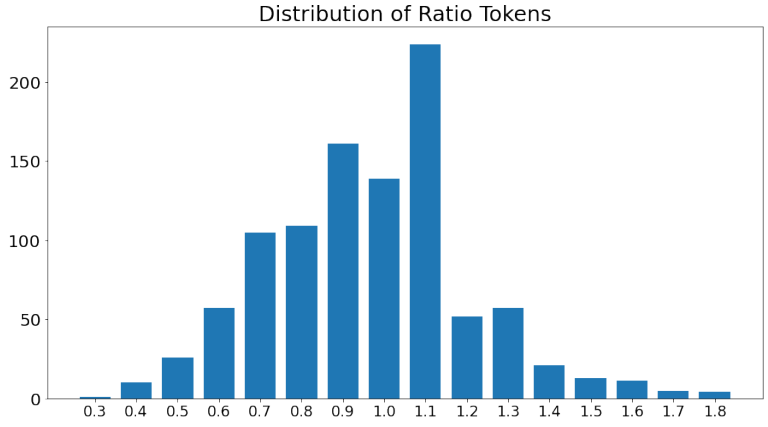

Figure 4.10: Distribution of the test set based on ratio values

4.3.1 Baseline

The baseline model was trained by fine-tuning the Marian model without any length tokens. Similarly to the BART model, we had to measure the impact of each token by having a baseline that we could compare with. The resulting length ratio of the baseline model can be found in Table 4.13.

Marian - BASELINE

Token	OpenSubtitles LR^{src}
	0.87

Table 4.13: The mean length ratio against the source (LR^{src}) for the Marian baseline model is presented in the columns above. It was created by fine-tuning the Marian model on OpenSubtitles data without any additional length tokens.

4.3.2 Category-based Ratio Tokens

Following the methods described in 3.3.3, we applied the three categories of short, normal and long sentence ratios to each of the source subtitles. The distribution of these tokens can be found in Figure 4.11, and the resulting length ratios can be found in Table 4.14.

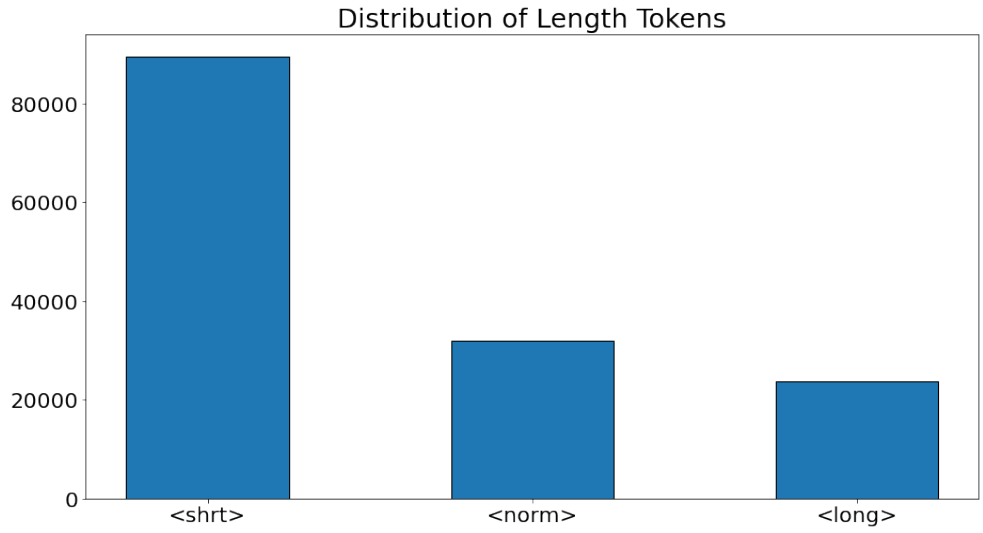

Figure 4.11: Distribution of the training set for category-based tokens

Marian - CATEGORIES

Token	OpenSubtitles LR^{src}
<shrt>	0.80
<norm>	0.96
<long>	1.12

Table 4.14: The metrics were evaluated with category-based tokens on the OpenSubtitles data using a fine-tuned Marian model. The left column contains the evaluated tokens and the right corresponds to their value.

Marian - CATEGORIES

Token	OpenSubtitles BLEU	ROUGE-2	METEOR
<shrt>	36.85	50.11	63.43
<norm>	52.86	61.55	69.86
<long>	40.52	50.52	62.24

Table 4.15: The metrics evaluated for each dataset with category-based tokens. The result is evaluated from a Marian model fine-tuned on the OpenSubtitles dataset with tokens representing short, normal, and long sentence ratios.

4.3.3 Value-based Ratio Tokens

The second ratio method was to implement the value-based ratio tokens, ranging from 0 to 2. We used the same method for an interval of 0.1, resulting in 20 tokens. The distribution of these tokens can be seen in Figure 4.12. The results of the fine-tuned model based on length ratios and metrics can be found in Table 4.16 & B.2.

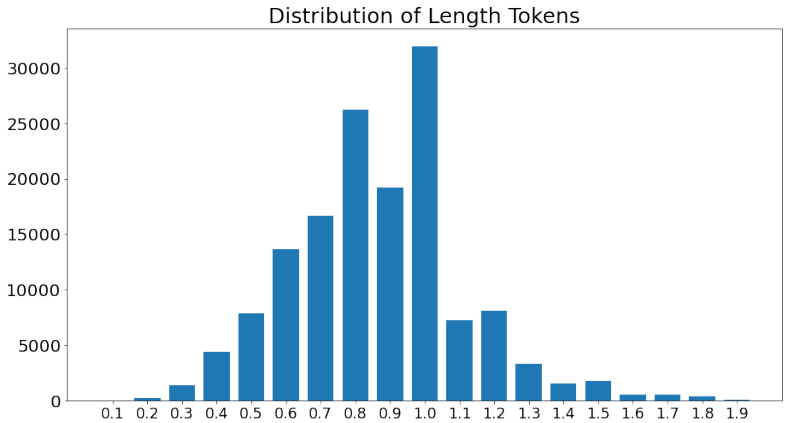

Figure 4.12: Distribution of the training set for value-based tokens

Marian - VALUES

Token	OpenSubtitles LR^{src}
<ratio_0.20>	0.35
<ratio_0.30>	0.36
<ratio_0.40>	0.44
<ratio_0.50>	0.54
<ratio_0.60>	0.65
<ratio_0.70>	0.74
<ratio_0.80>	0.84
<ratio_0.90>	0.91
<ratio_1.00>	0.99
<ratio_1.10>	1.09
<ratio_1.20>	1.19
<ratio_1.30>	1.31
<ratio_1.40>	1.44
<ratio_1.50>	1.51
<ratio_1.60>	1.59
<ratio_1.70>	1.61

Table 4.16: The resulting mean length ratios against the source (LR^{src}) with value-based ratio tokens on the OpenSubtitles dataset. The leftmost column contains the evaluated tokens and the right column corresponds to the mean length ratio for each token.

Marian - VALUES

Token	OpenSubtitles BLEU	ROUGE-2	METEOR
<ratio_0.50>	35.68	43.72	53.92
<ratio_0.60>	38.20	49.88	62.05
<ratio_0.70>	32.28	48.09	61.07
<ratio_0.80>	41.18	55.94	67.45
<ratio_0.90>	39.38	50.20	63.62
<ratio_1.00>	52.37	62.82	71.00
<ratio_1.10>	43.65	56.96	70.17
<ratio_1.20>	44.43	56.54	69.40

Table 4.17: The evaluated metrics with the value-based ratio tokens. The results are evaluated from a fine-tuned Marian model on the OpenSubtitles dataset with 20 tokens representing length ratios between 0 and 2, with an interval of 0.1. The leftmost column show the token used to evaluate the model.

4.3.4 Manual Token Evaluation

Manual token analysis is based on picking random samples of the generated text for each token used for evaluation. Some of these examples can be seen in Table 4.18 & 4.19. From this evaluation, we can see that the model generates output with respect to the ratio token. Both the length ratio and the semantics are generated accordingly, and the overall impression is that it performs satisfying results. More examples of the generated sentences can be seen in Appendix A and further comments about the result can be found in Chapter 5.

Example 1 - Marian - OpenSubtitles	
Source:	Watch that cigarette, lieutenant.
Target:	Akta cigaretten!
Target Token:	<shrt> & <ratio_0.70>
Baseline	
	Se upp med cigaretten, löjtnant.
Categories	
<shrt>	Akta cigaretten.
<norm>	Akta cigaretten, löjtnant.
<long>	Se upp med cigaretten, löjtnant.
Values	
<ratio_0.50>	Akta cigaretten.
<ratio_0.60>	Akta cigaretten.
<ratio_0.70>	Akta cigaretten.
<ratio_0.80>	Akta cigaretten, löjtnant.
<ratio_0.90>	Akta cigaretten, löjtnant.
<ratio_1.00>	Akta cigaretten, löjtnant.
<ratio_1.10>	Se upp med cigaretten, löjtnant.
<ratio_1.20>	Se upp med cigaretten, löjtnant.

Table 4.18: Examples of sentences generated with the Marian model

Example 2 - Marian - OpenSubtitles	
Source:	Who else except yourself has access to this apartment?
Target:	Vem mer har tillgång till rummet?
Target Token:	<shrt> & <ratio_0.70>
Baseline	
	Vem mer än du har tillgång till den här lägenheten?
Categories	
<shrt>	Vem annars har tillgång till lägenheten?
<norm>	Vem annars har tillgång till den här lägenheten?
<long>	Vem mer än du själv har tillgång till den här lägenheten?
Values	
<ratio_0.50>	Har ni tillgång till lägenheten?
<ratio_0.60>	Vem annars har tillgång till lägenheten?
<ratio_0.70>	Vem annars har tillgång till lägenheten?
<ratio_0.80>	Vem mer än ni har tillgång till lägenheten?
<ratio_0.90>	Vem mer än ni har tillgång till lägenheten?
<ratio_1.00>	Vem mer än du har tillgång till den här lägenheten?
<ratio_1.10>	Vem mer än du har tillgång till den här lägenheten?
<ratio_1.20>	Vem mer än ni själv har tillgång till den här lägenheten?

Table 4.19: Examples of sentences generated with the Marian model

4.4 Summary

A short summary of the results can be seen in the list below:

- The BART models with length- encodings and tokens fine-tuned on the Plint dataset did not achieve any successful results because the model started to stutter and repeat itself. However, the baseline model fine-tuned with the Plint dataset did generate sentences that were almost identical to the input and for long sequences it also inserted the break token at reasonable positions.

- The BART models with ratio tokens was successful in controlling the length of the generated sentences. The value-based method generated a more fine-grained control of the length than the category-based, and the model fine-tuned on the WikiOpen dataset performed slightly better based on the desired length.

- The Marian models with ratio tokens produced successful results, similar to the BART models with the same method. This model is also a translation model and, in general, produced good translations based on our manual analysis. Again, the value-based ratio tokens produced a more fine-grained control of the length than the category-based ratio tokens.

5
Discussion and Conclusion

In this chapter we will discuss the results and reason about the most interesting findings in our project. In addition, we will also discuss the choice of methods, the different datasets, our limitations, and the evaluation of our results. The chapter is finalized with a section of conclusions followed by a section with suggestions on further improvements and future work.

5.1 Discussion

The goal of this project was to find a way to generate subtitles of controllable length. In the following subsections, we will reason about the chosen methods, the datasets, the way the evaluation was performed, and the results themselves.

5.1.1 Methods of choice

The Transformer is the most dominant architecture used for seq2seq models, and hence it was the obvious choice when choosing the model type to work with. Furthermore, the Transformer enables transfer learning and pre-trained models, which has multiple benefits. First of all, the amount of resources required to train a language model is extensive, therefore, a pre-trained model has a far less impact with regard to energy consumption [36]. Secondly, enabling a pre-trained state-of-the-art model will guarantee that the model has good linguistic knowledge from start. Thirdly, Lakew et al. [18] showed that their methods of controlling the generated output length of neural machine translation worked both with pre-training a model from scratch, but also with fine-tuning a baseline model. Lastly, the sheer amount of time and resources needed to train a Transformer model from scratch is so large that it would not be feasible for us in this thesis.

Our methods are all based on using subwords as length measurements for our length tokens and positional encodings. This is hence considered a limitation in our project. With only considering subwords, the act of controlling the number of characters in a subtitle is harder to achieve, since not every subword contains the same number of characters. A mean of all subword lengths could be calculated to create an estimate of how many characters the final sequence will contain. However, the task of controlling the sequence length, on a character level, can not be solved perfectly with our specified subword length tokens, without letting the model have knowledge of every specific subword length. Since the original implementation of

the positional encodings in [38] is added to each token, we figured we could do a similar implementation with the modified length encoding (see Section 3.3.2) with the calculations based on the subword tokens.

5.1.2 Reasoning about the data

The data from Plint were composed of ASR-transcripts together with a human-made reference, used as English subtitles. These data were constructed into a dataset (see Section 3.1) that we initially used to train our models. However, this dataset did not contain as much linguistic compression as we initially anticipated. By analysing the data, we found that for most parts, the target and source sentences differed with only a few words or characters. This misconception could have been avoided if a more extensive data analysis had been performed during the earlier phases of the project, during the time of retrieving and constructing the dataset from the Plint servers, for example.

To find linguistic compression, we turned to public datasets. This was initially done using the WikiLarge dataset [41], but we reasoned that using all of these data would be misleading for our purposes, so we made use of the filtering methods implemented in Section 3.2.3. Furthermore, we performed backtranslation on the OpenSubtitles dataset to create more training data from subtitles with linguistic compression. As mentioned above, we discovered that linguistic compression was not very common in the Plint dataset, and therefore we turned to subtitles from English to Swedish where we found the compression rate to be higher. The task of generating translations with the Marian model was computationally heavy, and hence we only translated a subset of the data. However, we can argue, based on the result, that the generated data was still enough to prove our concept.

5.1.3 Evaluating the models

Evaluating the performance of a language model is a complex task and hence there is a need for multiple performance metrics and methods. It is arguably difficult to only rely on metrics, and therefore we used a combination of metrics together with a manual analysis of our models. The manual analysis was performed by analysing randomly selected examples of generated subtitles from the test data. Furthermore, this analysis was focusing on the semantics of the generated sequences, specifically how well the adequacy and fluency of the sentences were. Since we are evaluating subtitles, it is also important that the subtitle maintains the essence of the original input.

5.1.4 Analysing the results

BART with Length Encodings and Tokens
Our first experiment consisted of fine-tuning the BART model with both length encodings and length tokens using the Plint dataset. This model did not deliver any useful results even though similar methods were implemented in [37][18]. The problem found during the evaluation was that the model started to stutter, which

is obvious from Table 4.2. When troubleshooting the cause of the stuttering, we inspected the positional embeddings in our fine-tuned model and compared them with the original BART model (see Figure 4.4). From this we could see that the new positional encodings, and therefore also the word embeddings, have a much higher variance. We hypothesised that the old embeddings are slowly vanishing and the new embeddings are taking over. This hypothesis was strengthened by analysing different epochs of the model, where we could see that the longer we trained, the worse the results (see Table 4.2). For this reason, we believe that the positional encodings were the issue. We hypothesise that the model still knew most of the words that should be part of the correct output, but the positional encoding was the faulty part. We also hypothesise that this model probably would have performed better if the length encodings were part of the pre-training, but that would require the model to be trained from scratch, which would not be feasible due to our limited resources.

The baseline model corresponding to the initial method was only the BART model fine-tuned on the Plint dataset. The loss graph in Figure 4.1 shows very small movement which is anticipated, given that the model is pre-trained and since we found that the dataset's amount of sentence pairs with meaningful linguistic compression to be minimal. This means that the model was left to pick up on the insertions of line break tokens ("
") and some small text alterations. Looking at the generated examples in Table 4.1, one can see that the model inserts line breaks around the middle which is quite common for longer subtitles. It should also be noted that since the baseline model does not have a separate token for the line break,
 actually corresponds to 3 separate tokens.

Due to the lack of useful results from the subword method, the length encoding method with a character-based count was never fully realised with a proper training procedure. This was due to the time limitations of the project. Given the stage of the project in which we were in, the results that were achieved, and the limited resources, we figured other approaches were better to prioritise.

BART with Ratio Tokens

Motivated by our findings in the first experiment, we were looking for another method that did not involve modifying the positional embeddings. Inspired by the Control Tokens from Martin et al. [24] we implemented our ratio tokens. Despite the simplicity of this method, it produced surprisingly good results. From evaluating the length ratios of the generated sequences we could see that the length tokens are very good at following the desired lengths. Furthermore, it is interesting to see how much impact each token actually has, considering that it is a completely new token that has been fine-tuned for only a few epochs.

The idea of having two different methods on the length ratios was motivated by experimenting with how close you can control the generated outputs. When comparing the two methods of category-based tokens and value-based tokens, we can see that value-based tokens produce a more fine-grained control of the sequence

lengths. However, a more fine-grained control of the length is probably hard to achieve because the subtitles are rather short, and an additional step in between the 20 tokens would probably not have that big of an impact. This is motivated by the fact that many of the subsequent value-based ratio tokens repeat the same output as the previous token. An example of this can be seen in Table 4.12 where the output is repeated for four consecutive value-based tokens. Hence, we do not believe that adding another token between these would generate any different output. Furthermore, it would also give less training data for each category since the data will be divided more.

The resulting length ratios of the baseline models show a length ratio of 0.88 for both datasets. Because of BART's ability to be effectively fine-tuned for text summarization, we believe that it learns to represent the distribution of the dataset. The datasets are biased towards a slightly compressed version of the original length, and hence the baseline is biased towards generating shorter sentences even without any tokens.

The tokens that implies a longer length, such as "<long>" and the tokens with a ratio above 1, are probably not ideal when creating subtitles. However, it was still interesting to see the behaviour and results of these tokens. For most parts, the model starts to generate reasonable and longer outputs based on the input sequence, but sometimes the model starts to generate unreasonable text. A common case for the longer outputs was that the model started to fill in with redundant words such as "and", "the", "to", and so on.

Another insight from our experiments is that the model fine-tuned on the WikiOpen dataset performs better on following the desired lengths than the model fine-tuned on the OpenBack dataset. We believe that the reason is that we have more training data for each token with the WikiOpen dataset. Regarding the semantic analysis between the two datasets, it is very hard to distinguish any differences. The only notable difference from our analysis is that the two models paraphrase words differently. However, we do not believe that we can draw any conclusions from this because the models are still very similar. To do this, we would probably have to perform a more extensive human evaluation, but that is not feasible in this thesis.

The manual analysis also shows that the model overall produces good semantic sentences for most tokens, with the exception of tokens that are in the outer ranges. However, the model is not perfect, and some of the sentences produced have bad semantics. From the analysis, we can find that this is more common for shorter sentences where the compression ratio is harder to apply. Furthermore, there are also occasions where the models do not take the compression into account. An example of this can be seen in Table 4.10, where the generated sequence is the same for the two tokens "<norm>" and "<long>".

In all experiments regarding the ratio tokens, we also tried different learning rates during fine-tuning. However, it quickly became clear to us that the learning rate

used in [23] gave the best results. With a higher learning rate, we achieved overfitting and no improvement in validation loss. Using a lower learning rate, we also received a lower validation loss.

Marian with Ratio Tokens
Since our experiments with the BART model turned out to be successful, we decided to see if it was possible to do the same experiments with a translation model. The choice of going with the Marian model instead of mBART [22], which is a multilingual BART model, was that the Marian model is much smaller and therefore more suitable for fine-tuning experiments. It was also motivated by the fact that the Marian model is very similar to the BART model, as mentioned in Section 2.5.4.

The Marian model produced results that are very similar to the BART model in terms of length ratios. The results for the baseline model and the model with category-based tokens are almost identical to those of the BART model and differ by only a few decimals. With value-based tokens, the results are also very similar, but for tokens with a longer ratio, meaning a ratio greater than 1, the Marian model actually performs better than both BART models. In fact, the Marian model is very close to following the length, which can be seen in Table 4.16.

From the manual analysis of the Swedish translations, it becomes clear that most of the translations are linguistically correct while still preserving the essence of the dialogue. Two good examples of this can be seen in Table 4.18 & 4.19. However, there are still a few occasions where the model fails to compress and paraphrase sentences with regard to the ratio token, similar to the BART models using the same methods.

5.2 Conclusions

In this thesis, we explore various methods to control the length of subtitle generation. The project has focused on implementing different methods for fine-tuning pre-trained language models with subtitling data. This work shows that it is possible to effectively generate subtitles with controllable lengths by creating and implementing length tokens based on subword ratio.

Furthermore, the results show that it is possible to achieve more fine-grained control of the output by implementing more precise tokens. Despite the simplicity of the methods, the ratio- and category-based tokens gave surprisingly good results in generating the desired length ratio while still maintaining the linguistic parts. This was also shown for a translation model with similar results. We also performed experiments that were based on the use of positional encoding to add more information to the positional embedding layer.

However, we can conclude that it was not possible to fine-tune the BART model using this method. The reason for the failed implementation is probably related to

the fact that the added encodings make the original positional embeddings drown out, thus ruining the output.

5.2.1 Future work

To continue the work with controlling the length of subtitle generation, it would be interesting to see how character-based encodings would perform using a pre-trained model, such as BART. Character length-based methods should be better suited for the subtitling business, with regard to length constraints, compared to our subword-based methods.

Looking at how well the ratio token methods performed despite the unbalanced dataset, more work could be put into exploring how the ratio tokens can perform on more balanced data, to possibly increase the length adjustability. This also implies experimenting with different token intervals to see how it would affect the results. Furthermore, more tokens could be of interest, such as tokens based on reading speed that could possibly be relevant for subtitle generation.

Since the method with length tokens and additional length encodings failed, future work could also consist of training the model from scratch with the additional encodings present and comparing it to our successful purely token-based methods. One could also continue to work with the fine-tuning, by experimenting with the scale of the added encodings in order to not drown out the pre-trained weights.

Bibliography

[1] Abien Fred Agarap. Deep learning using rectified linear units (relu), 2018.

[2] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate, 2014.

[3] Satanjeev Banerjee and Alon Lavie. METEOR: An automatic metric for MT evaluation with improved correlation with human judgments. In *Proceedings of the ACL Workshop on Intrinsic and Extrinsic Evaluation Measures for Machine Translation and/or Summarization*, pages 65–72, Ann Arbor, Michigan, June 2005. Association for Computational Linguistics.

[4] BBC. Subtitle guidelines, 2021.

[5] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners, 2020.

[6] Mara Campbell. Automatic speech recognition softwares: Will they replace audiovisual translators in the near future? (part 2). *Deep Focus, Issue 3*, 2019.

[7] Devrim Cavusoglu, Fatih Cagatay Akyon, Ulas Sert, and Cemil Cengiz. Jury: Comprehensive NLP Evaluation toolkit, feb 2022.

[8] KyungHyun Cho, Bart van Merrienboer, Dzmitry Bahdanau, and Yoshua Bengio. On the properties of neural machine translation: Encoder-decoder approaches. *CoRR*, abs/1409.1259, 2014.

[9] Council of European Union. Directive (eu) 2016/2102, 2016. `https://eur-lex.europa.eu/legal-content/EN/TXT/?uri=uriserv:OJ.L_.2016.327.01.0001.01.ENG`.

[10] Max Deryagin, Miroslav Pošta, and Daniel Landes. Machine translation manifesto, 2021.

[11] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding, 2018.

[12] Markus Freitag and Yaser Al-Onaizan. Beam search strategies for neural machine translation. In *Proceedings of the First Workshop on Neural Machine Translation*. Association for Computational Linguistics, 2017.

[13] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015.

[14] Dan Hendrycks and Kevin Gimpel. Bridging nonlinearities and stochastic regularizers with gaussian error linear units. *CoRR*, abs/1606.08415, 2016.

[15] Ari Holtzman, Jan Buys, Li Du, Maxwell Forbes, and Yejin Choi. The curious case of neural text degeneration, 2019.

[16] Todd Ward Wei-Jing Zhu Kishore Papineni, Salim Roukos. Bleu: a method for automatic evaluation of machine translation. *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 311–318, 2002.

[17] Taku Kudo and John Richardson. Sentencepiece: A simple and language independent subword tokenizer and detokenizer for neural text processing, 2018.

[18] Surafel Melaku Lakew, Mattia Di Gangi, and Marcello Federico. Controlling the output length of neural machine translation, 2019.

[19] Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Ves Stoyanov, and Luke Zettlemoyer. Bart: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension, 2019.

[20] Chin-Yew Lin. Rouge: A package for automatic evaluation of summaries. *Association for Computational Linguistics*, page 74–81, 2004.

[21] Pierre Lison and Jörg Tiedemann. OpenSubtitles2016: Extracting large parallel corpora from movie and TV subtitles. In *Proceedings of the Tenth International Conference on Language Resources and Evaluation (LREC'16)*, pages 923–929, Portorož, Slovenia, May 2016. European Language Resources Association (ELRA).

[22] Yinhan Liu, Jiatao Gu, Naman Goyal, Xian Li, Sergey Edunov, Marjan Ghazvininejad, Mike Lewis, and Luke Zettlemoyer. Multilingual denoising pre-training for neural machine translation, 2020.

[23] Louis Martin, Angela Fan, Éric de la Clergerie, Antoine Bordes, and Benoît Sagot. Muss: Multilingual unsupervised sentence simplification by mining paraphrases, 2020.

[24] Louis Martin, Benoît Sagot, Éric de la Clergerie, and Antoine Bordes. Controllable sentence simplification, 2019.

[25] Sneha Mehta, Bahareh Azarnoush, Boris Chen, Avneesh Saluja, Vinith Misra, Ballav Bihani, and Ritwik Kumar. Simplify-then-translate: Automatic preprocessing for black-box machine translation, 2020.

[26] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space.

[27] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, and Jeffrey Dean. Distributed representations of words and phrases and their compositionality, 2013.

[28] Jan Niehaus. Machine translation with unsupervised length-constraints. *Proceedings of the 14th Conference of the Association for Machine Translation in the Americas*, pages 21–35, 2020.

[29] Romain Paulus, Caiming Xiong, and Richard Socher. A deep reinforced model for abstractive summarization, 2017.

[30] Jeffrey Pennington, Richard Socher, and Christopher D Manning. Glove: Global vectors for word representation. In *EMNLP*, volume 14, pages 1532–1543, 2014.

[31] Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. Improving language understanding by generative pre-training. 2018.

[32] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever.

[33] Alexander M. Rush, Sumit Chopra, and James Weston. A neural attention model for abstractive sentence summarization. *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, page 379–389, 2015.

[34] Mike Schuster and Kaisuke Nakajima. Japanese and korean voice search. In *International Conference on Acoustics, Speech and Signal Processing*, pages 5149–5152, 2012.

[35] Rico Sennrich, Barry Haddow, and Alexandra Birch. Neural machine translation of rare words with subword units, 2015.

[36] Emma Strubell, Ananya Ganesh, and Andrew McCallum. Energy and policy considerations for deep learning in NLP. *CoRR*, abs/1906.02243, 2019.

[37] Sho Takase and Naoaki Okazaki. Positional encoding to control output sequence length. *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)*, page 3999–4004, 2019.

[38] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2017.

[39] Wei Xu, Courtney Napoles, Ellie Pavlick, Quanze Chen, and Chris Callison-Burch. Optimizing Statistical Machine Translation for Text Simplification. *Transactions of the Association for Computational Linguistics*, 4:401–415, 07 2016.

[40] Divakar Yadav, Jalpa Desai, and Arun Kumar Yadav. Automatic text summarization methods: A comprehensive review, 2022.

[41] Xingxing Zhang and Mirella Lapata. Sentence simplification with deep reinforcement learning. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 584–594, Copenhagen, Denmark, September 2017. Association for Computational Linguistics.

A

Appendix 1

Example 1 - WikiOpen

Source:	You mean where the dog was sniffing around?
Target: Target Token:	Where the dog nosed? <shrt> & <ratio_0.60>
Baseline	
	You mean where the dog snooped around?
Categories	
<shrt>	You mean where the dog snooped?
<norm>	You mean where the dog was snooping around?
<long>	You mean where the dog was snooping around?
Values	
<ratio_0.20>	Where?
<ratio_0.30>	Where?
<ratio_0.40>	Where was the dog?
<ratio_0.50>	Where was the dog?
<ratio_0.60>	Where was the dog?
<ratio_0.70>	You mean where the dog sniffed?
<ratio_0.80>	You mean where the dog was sniffing?
<ratio_0.90>	You mean where the dog was sniffing?
<ratio_1.00>	You mean where the dog was snooping around?
<ratio_1.10>	You mean where the dog was snooping around?
<ratio_1.20>	Do you mean where the dog was snooping around?
<ratio_1.30>	Do you mean where the dog was snooping around in the woods?
<ratio_1.40>	You mean where the dog was sniffing around, didn't you?
<ratio_1.50>	You mean where the dog was sniffing around, didn't you?
<ratio_1.60>	You mean where the dog was sniffing around, didn't you?
<ratio_1.70>	Do you mean where the dog was snooping around in the woods?

Example 2 - WikiOpen

Source:	We're riding together.
Target:	We ride together.
Target Token:	<shrt> & <ratio_0.80>
Baseline	
	We're riding together.
Categories	
<shrt>	We ride together.
<norm>	We're riding together.
<long>	We're gonna ride together.
Values	
<ratio_0.20>	Together.
<ratio_0.30>	Together.
<ratio_0.40>	Together.
<ratio_0.50>	Together.
<ratio_0.60>	We ride.
<ratio_0.70>	We ride together.
<ratio_0.80>	We're riding.
<ratio_0.90>	We're riding together.
<ratio_1.00>	We're riding together.
<ratio_1.10>	We're riding on each other.
<ratio_1.20>	We're riding on each other.
<ratio_1.30>	We're riding on each other.
<ratio_1.40>	We're on our way together.
<ratio_1.50>	Yeah, we're riding together.
<ratio_1.60>	Yeah, we're riding together now.
<ratio_1.70>	Yeah, we're riding together now.

Example 3 - WikiOpen

Source:	All right, dear.
Target:	All right.
Target Token:	<shrt> & <ratio_0.60>
Baseline	
	All right, dear.
Categories	
<shrt>	All right.
<norm>	All right, dear.
<long>	All right, my dear.
Values	
<ratio_0.20>	Dear...
<ratio_0.30>	Dear...
<ratio_0.40>	All right.
<ratio_0.50>	All right.
<ratio_0.60>	All right..
<ratio_0.70>	All right.
<ratio_0.80>	All right, dear.
<ratio_0.90>	All right, honey.
<ratio_1.00>	All right, honey.
<ratio_1.10>	All right, my dear.
<ratio_1.20>	All right, my dear.
<ratio_1.30>	All right, my dear.
<ratio_1.40>	All right, my dear.
<ratio_1.50>	That's all right, my dear.
<ratio_1.60>	That's all right, my dear.
<ratio_1.70>	That's all right, my dear.

Example 4 - WikiOpen

Source:	I'll teach you to fish and trap.
Target:	I'll teach you how to fish and set traps.
Target Token:	<long> & <ratio_1.20>
Baseline	
	I'll teach you how to fish and trap.
Categories	
<shrt>	I'll teach you how to fish.
<norm>	I'll teach you how to fish and trap.
<long>	I'll teach you how to fish and how to trap.
Values	
<ratio_0.20>	Fish
<ratio_0.30>	Fish and trap.
<ratio_0.40>	Learn to fish.
<ratio_0.50>	Learn to fish.
<ratio_0.60>	Learn to fish and trap.
<ratio_0.70>	I teach you to fish.
<ratio_0.80>	I teach you to fish and trap.
<ratio_0.90>	I'll teach you how to fish.
<ratio_1.00>	I'll teach you how to fish and trap.
<ratio_1.10>	I'll teach you how to fish and trap.
<ratio_1.20>	I'll teach you how to fish and how to trap.
<ratio_1.30>	I'll teach you how to fish and how to trap.
<ratio_1.40>	I'll teach you how to fish and how to trap.
<ratio_1.50>	I'll teach you how to fish and how to trap.
<ratio_1.60>	I'll teach you how to fish and trap, and you can do that.
<ratio_1.70>	I'll teach you how to fish and trap, and you'll learn to fish.

Example 5 - WikiOpen

Source:	Might have to go to Woodstock soon as it stops snowing.
Target: Target Token:	I think I'll have to go to Woodstock when it stops snowing. <long> & <ratio_1.10>
Baseline	
	I'll have to go to Woodstock as soon as it stops snowing.
Categories	
<shrt>	I have to go to Woodstock as soon as it stops snowing.
<norm>	I might have to go to Woodstock as soon as it stops snowing.
<long>	I'm gonna have to go to Woodstock as soon as it stops snowing.
Values	
<ratio_0.20>	Woodstock
<ratio_0.30>	Woodstock.
<ratio_0.40>	We have to go to Woodstock.
<ratio_0.50>	We have to go to Woodstock.
<ratio_0.60>	We have to go to Woodstock.
<ratio_0.70>	We have to go to Woodstock as soon as possible.
<ratio_0.80>	I have to go to Woodstock as soon as it stops snow.
<ratio_0.90>	I might have to go to Woodstock as soon as it stops snowing.
<ratio_1.00>	I might have to go to Woodstock as soon as it stops snowing.
<ratio_1.10>	I might have to go to Woodstock as soon as it stops snowing.
<ratio_1.20>	I think we'll have to go to Woodstock as soon as it stops snowing
<ratio_1.30>	I think we'll have to go to Woodstock as soon as it stops snowing
<ratio_1.40>	I think we're gonna have to go to Woodstock as soon as it stops snow
<ratio_1.50>	I think we're gonna have to go to Woodstock as soon as it stops snow
<ratio_1.60>	I think we're gonna have to go to Woodstock as soon as it stops snow
<ratio_1.70>	I think we're gonna have to go to Woodstock as soon as it stops snow

Example 6 - WikiOpen

Source:	You can do anything, as long as you're away from Johnny Friendly.
Target: Target Token:	You can do anything, as long as you stay away from Friendly. <short> & <ratio_0.90>
Baseline	
	You can do anything, as long as you're away from Johnny Friendly.
Categories	
<shrt>	You can do anything as long as you're away from Johnny Friendly.
<norm>	You can do anything, as long as you're away from Johnny Friendly.
<long>	You can do anything, as long as you're away from Johnny Friendly.
Values	
<ratio_0.20>	Johnny Friendly.
<ratio_0.30>	You can do anything.
<ratio_0.40>	You can do anything.
<ratio_0.50>	You can do anything you want.
<ratio_0.60>	You can do anything when you're free.
<ratio_0.70>	You can do anything when you're away from Johnny Friendly.
<ratio_0.80>	You can do anything as long as you're free of Johnny Friendly.
<ratio_0.90>	You can do anything, as long as you're away from Johnny Friendly.
<ratio_1.00>	You can do anything, as long as you're away from Johnny Friendly.
<ratio_1.10>	You can do anything, as long as you're away from Johnny Friendly.
<ratio_1.20>	You can do anything, as long as you don't run away from Johnny Friendly.
<ratio_1.30>	You can do anything, as long as you don't get caught up in Johnny Friendly
<ratio_1.40>	You can do whatever you want, as long as you're out of Johnny Friendly's
<ratio_1.50>	You can do whatever you want, as long as you don't run away from Johnny
<ratio_1.60>	You can do whatever you want, as long as you're out of Johnny Friendly's
<ratio_1.70>	You can do whatever you want, as long as you're out of Johnny Friendly's

Example 1 - OpenBack

Source:	You mean where the dog was sniffing around?
Target:	Where the dog nosed?
Target Token:	<shrt> & <ratio_0.60>
Baseline	
	You mean where the dog was sniffing around??
Categories	
<shrt>	You mean where the dog was?
<norm>	You mean where the dog was sniffing around?
<long>	Did you mean where the dog was snooping around?
Values	
<ratio_0.20>	Where?
<ratio_0.30>	Where was he?
<ratio_0.40>	Where was the dog?
<ratio_0.50>	Where was the dog?
<ratio_0.60>	Where was the dog sniffing?
<ratio_0.70>	You mean where the dog sniffed?
<ratio_0.80>	You mean where the dog sniffed?
<ratio_0.90>	You mean where the dog was sniffing around?
<ratio_1.00>	You mean where the dog was sniffing around?
<ratio_1.10>	You mean where the dog was snooping around?
<ratio_1.20>	You mean where the dog was snooping around in the woods?
<ratio_1.30>	You mean where the dog was snooping around in the woods?
<ratio_1.40>	You mean where the dog was snooping around, didn't you?
<ratio_1.50>	You mean where the dog was snooping around, didn't you?
<ratio_1.60>	You mean where the dog was snooping around, didn't you?
<ratio_1.70>	You mean where the dog was snooping around, don't you?

Example 2 - OpenBack

Source:	We're riding together.
Target:	We ride together.
Target Token:	<shrt> & <ratio_0.80>
Baseline	
	We're riding together.
Categories	
<shrt>	We ride together.
<norm>	We're riding together.
<long>	We're going to ride together.
Values	
<ratio_0.20>	Together.
<ratio_0.30>	Together.
<ratio_0.40>	Together.
<ratio_0.50>	We ride.
<ratio_0.60>	We ride.
<ratio_0.70>	We ride.
<ratio_0.80>	We ride together.
<ratio_0.90>	We ride together.
<ratio_1.00>	We're riding together.
<ratio_1.10>	We're riding together.
<ratio_1.20>	We're riding together now.
<ratio_1.30>	We're riding around together.
<ratio_1.40>	We're riding around together.
<ratio_1.50>	We're going to ride together.
<ratio_1.60>	We're going to ride together.
<ratio_1.70>	We're going to ride together.

Example 3 - OpenBack

Source:	All right, dear.
Target:	All right.
Target Token:	<shrt> & <ratio_0.60>
Baseline	
	All right, dear.
Categories	
<shrt>	All right.
<norm>	All right, dear.
<long>	All right, my dear.
Values	
<ratio_0.20>	Good.
<ratio_0.30>	All right.
<ratio_0.40>	All right.
<ratio_0.50>	All right.
<ratio_0.60>	All right..
<ratio_0.70>	All right.
<ratio_0.80>	All right.
<ratio_0.90>	All right, dear.
<ratio_1.00>	All right, dear.
<ratio_1.10>	All right, my dear.
<ratio_1.20>	All right, my dear.
<ratio_1.30>	All right, my dear.
<ratio_1.40>	All right, my dear.
<ratio_1.50>	All right, my dear.
<ratio_1.60>	It's all right, my dear.
<ratio_1.70>	It's all right, my dear.

Example 4 - OpenBack

Source:	I'll teach you to fish and trap.
Target:	I'll teach you how to fish and set traps.
Target Token:	<long> & <ratio_1.20>
Baseline	
	I'll teach you how to fish and trap.
Categories	
<shrt>	I'll teach you how to fish.
<norm>	I'll teach you how to fish and trap.
<long>	I'll teach you how to fish and how to trap.
Values	
<ratio_0.20>	To trap.
<ratio_0.30>	To fish.
<ratio_0.40>	To fish and trap.
<ratio_0.50>	Learn to fish.
<ratio_0.60>	Learn to fish and trap.
<ratio_0.70>	Learn to fish and trap.
<ratio_0.80>	I teach you to fish and trap.
<ratio_0.90>	I'll teach you how to fish.
<ratio_1.00>	I'll teach you how to fish and trap.
<ratio_1.10>	I'll teach you how to fish and trap.
<ratio_1.20>	I'll teach you how to fish and trap.
<ratio_1.30>	I'll teach you how to fish and how to trap.
<ratio_1.40>	I'll teach you how to fish and how to trap.
<ratio_1.50>	I'll teach you how to fish and how to trap.
<ratio_1.60>	I'll teach you how to fish, how to trap and how to play.
<ratio_1.70>	I'll teach you how to fish, how to trap and how to play.

Example 5 - OpenBack

Source:	Might have to go to Woodstock soon as it stops snowing.
Target: Target Token:	I think I'll have to go to Woodstock when it stops snowing. <long> & <ratio_1.10>
Baseline	
	We'll have to go to Woodstock as soon as it stops snowing.
Categories	
<shrt>	I have to go to Woodstock as soon as it stops snowing.
<norm>	I might have to go to Woodstock as soon as it stops snowing.
<long>	I think we'll have to go to Woodstock as soon as it stops snowing
Values	
<ratio_0.20>	Woodstock soon.
<ratio_0.30>	Woodstock soon.
<ratio_0.40>	Woodstock as soon as possible.
<ratio_0.50>	We have to go to Woodstock.
<ratio_0.60>	I have to go to Woodstock.
<ratio_0.70>	We'll have to go to Woodstock.
<ratio_0.80>	I have to go to Woodstock when it stops snowing.
<ratio_0.90>	I might have to go to Woodstock as soon as it stops snowing.
<ratio_1.00>	I might have to go to Woodstock as soon as it stops snowing.
<ratio_1.10>	I might have to go to Woodstock as soon as it stops snowing.
<ratio_1.20>	I think we'll have to go to Woodstock as soon as it stops snowing
<ratio_1.30>	I think we'll have to go to Woodstock as soon as it stops snowing
<ratio_1.40>	I think we're gonna have to go to Woodstock as soon as it stops snow
<ratio_1.50>	I think we're gonna have to go to Woodstock as soon as it stops snow
<ratio_1.60>	I think we're gonna have to go to Woodstock as soon as it stops snow
<ratio_1.70>	I think we're gonna have to go to Woodstock as soon as it stops snow

Example 6 - OpenBack

Source:	You can do anything, as long as you're away from Johnny Friendly.
Target: Target Token:	You can do anything, as long as you stay away from Friendly. <short> & <ratio_0.90>
Baseline	
	You can do anything, as long as you're away from Johnny Friendly.
Categories	
<shrt>	You can do anything as long as you're away from Johnny Friendly.
<norm>	You can do anything, as long as you're away from Johnny Friendly.
<long>	You can do anything, as long as you're away from Johnny Friendly.
Values	
<ratio_0.20>	Johnny Friendly.
<ratio_0.30>	Johnny Friendly can do anything.
<ratio_0.40>	You can do anything.
<ratio_0.50>	You can do anything, Johnny Friendly.
<ratio_0.60>	As long as you're with Johnny Friendly.
<ratio_0.70>	As long as you're away from Johnny Friendly.
<ratio_0.80>	You can do anything as long as you're away from Johnny.
<ratio_0.90>	You can do anything as long as you're away from Johnny Friendly.
<ratio_1.00>	You can do anything, as long as you're away from Johnny Friendly.
<ratio_1.10>	You can do anything, as long as you're away from Johnny Friendly.
<ratio_1.20>	You can do anything, as long as you're away from Johnny Friendly.
<ratio_1.30>	You can do anything, as long as you don't run away from Johnny Friendly.
<ratio_1.40>	You can do anything, as long as you don't run away from Johnny Friendly.
<ratio_1.50>	You can do anything, as long as you don't run away from Johnny Friendly.
<ratio_1.60>	You can do anything, as long as you don't run away from Johnny Friendly.
<ratio_1.70>	You can do anything, as long as you don't run away from Johnny Friendly.

Example 1 - Marian - OpenSubtitles

Source:	I don't want you men to be fooled about what's coming up for you.
Target: Target Token:	Jag vill inte att ni ska vara ovetande om vad som väntar. <shrt> & <ratio_0.70>
Baseline	
	Jag vill inte att ni ska luras för det som kommer upp för er.
Categories	
<shrt>	Jag vill inte att ni ska luras för det som kommer upp för er.
<norm>	Jag vill inte att ni män ska luras för det som kommer upp för er.
<long>	Jag vill inte att ni män ska luras för det som kommer upp för er.
Values	
<ratio_0.20>	Ni ska inte luras.
<ratio_0.30>	Ni ska inte luras.
<ratio_0.40>	Ni ska inte luras.
<ratio_0.50>	Jag vill inte att ni ska luras.
<ratio_0.60>	Jag vill inte att ni ska luras för vad som väntar.
<ratio_0.70>	Jag vill inte att ni ska luras för vad som väntar er.
<ratio_0.80>	Jag vill inte att ni ska luras för det som kommer upp för er.
<ratio_0.90>	Jag vill inte att ni män ska luras för vad som kommer upp för er.
<ratio_1.00>	Jag vill inte att ni män ska luras för vad som kommer upp för er.
<ratio_1.10>	Jag vill inte att ni män ska luras för vad som kommer upp för er. - Vad är det?
<ratio_1.20>	Jag vill inte att ni män ska låta er luras för det som kommer upp för er. - Vad är det?
<ratio_1.30>	Jag vill inte att ni män ska bli lurade över vad som kommer upp för er. - Vad är det?
<ratio_1.40>	Jag vill inte att ni män ska bli lurade över vad som kommer upp för er. - Nej, sir.
<ratio_1.50>	Jag vill inte att ni män ska låta er luras för det som kommer upp för er. - Nej, sir.
<ratio_1.60>	Jag vill inte att ni män ska bli lurade över vad som kommer upp för er. - Vad är det som kommer upp för er?
<ratio_1.70>	Jag vill inte att ni män ska bli lurade över vad som kommer upp för er. - Vad är det för fel på er?

Example 2 - Marian - OpenSubtitles

Source:	I didn't even know I was gonna do it.
Target:	Jag visste inte att jag skulle göra det.
Target Token:	<shrt> & <ratio_0.80>
Baseline	
	Jag visste inte ens att jag skulle göra det.
Categories	
<shrt>	Jag visste inte att jag skulle göra det.
<norm>	Jag visste inte ens att jag skulle göra det.
<long>	Jag visste inte ens att jag skulle göra det.
Values	
<ratio_0.20>	Jag visste inte.
<ratio_0.30>	Jag visste inte.
<ratio_0.40>	Jag visste inte det.
<ratio_0.50>	Jag visste inte att jag skulle.
<ratio_0.60>	Jag visste inte att jag skulle göra det.
<ratio_0.70>	Jag visste inte att jag skulle göra det.
<ratio_0.80>	Jag visste inte att jag skulle göra det.
<ratio_0.90>	Jag visste inte ens att jag skulle göra det.
<ratio_1.00>	Jag visste inte ens att jag skulle göra det.
<ratio_1.10>	Jag visste inte ens att jag skulle göra det.
<ratio_1.20>	Jag visste inte ens att jag skulle göra det. - Nej.
<ratio_1.30>	Jag visste inte ens att jag skulle göra det, sa jag.
<ratio_1.40>	Jag visste inte ens att jag skulle göra det. - Nej då.
<ratio_1.50>	Jag visste inte ens att jag skulle göra det. - Nej då.
<ratio_1.60>	Jag visste inte ens att jag skulle göra det. - Nej, det visste jag inte.
<ratio_1.70>	Jag visste inte ens att jag skulle göra det. - Nej, det visste jag inte.

Example 3 - Marian - OpenSubtitles

Source:	We're going to war.
Target:	Vi drar ut i krig.
Target Token:	<shrt> & <ratio_0.90>
Baseline	
	Vi ska ut i krig.
Categories	
<shrt>	Vi ska ut i krig.
<norm>	Vi är på väg ut i krig.
<long>	Vi är på väg ut i krig.
Values	
<ratio_0.20>	Kriget.
<ratio_0.30>	Kriget.
<ratio_0.40>	Ut i krig.
<ratio_0.50>	Ut i krig.
<ratio_0.60>	Vi ska ut i krig.
<ratio_0.70>	Vi ska ut i krig.
<ratio_0.80>	Vi ska ut i krig.
<ratio_0.90>	Vi ska ut i krig.
<ratio_1.00>	Vi ska ut i krig.
<ratio_1.10>	Vi är på väg ut i krig.
<ratio_1.20>	Vi är på väg ut i krig.
<ratio_1.30>	Vi är på väg ut i krig.
<ratio_1.40>	Vi är på väg ut i krig. - Ja, sir.
<ratio_1.50>	Vi är på väg ut i krig. - Ja, sir.
<ratio_1.60>	Vi är på väg ut i krig. - Ja, sir.
<ratio_1.70>	Vi är på väg ut i krig. - Ja, sir.

Example 4 - Marian - OpenSubtitles

Source:	She was the only one in town who understood a poor guy like me.
Target: Target Token:	Hon var den enda här i stan som förstod en stackars sate. <shrt> & <ratio_0.90>
Baseline	
	Hon var den enda i stan som förstod en stackare som jag.
Categories	
<shrt>	Hon var den enda i stan som förstod en stackare som jag.
<norm>	Hon var den enda i stan som förstod en stackare som jag.
<long>	Hon var den enda i stan som förstod sig på en stackare som jag.
Values	
<ratio_0.20>	Hon förstod mig.
<ratio_0.30>	Bara hon förstod mig.
<ratio_0.40>	Hon förstod en stackare.
<ratio_0.50>	Hon förstod en stackare som jag.
<ratio_0.60>	Hon var den enda i stan som förstod mig.
<ratio_0.70>	Hon var den enda i stan som förstod mig.
<ratio_0.80>	Hon var den enda i stan som förstod en stackare som jag.
<ratio_0.90>	Hon var den enda i stan som förstod en stackare som jag.
<ratio_1.00>	Hon var den enda i stan som förstod en stackare som jag.
<ratio_1.10>	Hon var den enda i stan som förstod sig på en stackare som jag.
<ratio_1.20>	Hon var den enda i stan som förstod sig på en stackare som jag.
<ratio_1.30>	Hon var den enda i stan som förstod sig på en stackare som jag.
<ratio_1.40>	Hon var den ende i stan som förstod sig på en stackare som jag. - Vem då?
<ratio_1.50>	Hon var den ende i stan som förstod sig på en stackare som jag. - Vem då?
<ratio_1.60>	Hon var den ende i stan som förstod sig på en stackare som jag. - Vem då?
<ratio_1.70>	Hon var den ende i stan som förstod sig på en stackare som jag. - Vem då?

Example 5 - Marian - OpenSubtitles

Source:	Somewhere on the state highway.
Target:	Nånstans efter motorvägen.
Target Token:	<norm> & <ratio_1.00>
Baseline	
	Någonstans på motorvägen.
Categories	
<shrt>	Nånstans på motorvägen.
<norm>	Nånstans på motorvägen.
<long>	Någonstans på den statliga motorvägen.
Values	
<ratio_0.20>	Någonstans.
<ratio_0.30>	Någonstans.
<ratio_0.40>	Någonstans.
<ratio_0.50>	Någonstans.
<ratio_0.60>	Någonstans.
<ratio_0.70>	Nånstans på motorvägen.
<ratio_0.80>	Nånstans på motorvägen.
<ratio_0.90>	Nånstans på motorvägen.
<ratio_1.00>	Någonstans på motorvägen.
<ratio_1.10>	Någonstans på statsvägen.
<ratio_1.20>	Någonstans på riksvägen.
<ratio_1.30>	Någonstans på den statliga motorvägen.
<ratio_1.40>	Någonstans på den statliga motorvägen.
<ratio_1.50>	Någonstans på den statliga motorvägen.
<ratio_1.60>	Någonstans på den statliga motorvägen.
<ratio_1.70>	Någonstans på den statliga motorvägen.

Example 6 - Marian - OpenSubtitles

Source:	A lot depends on who's in the saddle.
Target:	En hel del beror på den som sitter i sadeln.
Target Token:	<long> & <ratio_1.20>
Baseline	
	Mycket beror på vem som sitter i sadeln.
Categories	
<shrt>	Mycket beror på vem som sitter i sadeln.
<norm>	Mycket beror på vem som sitter i sadeln.
<long>	Mycket beror på vem som sitter i sadeln.
Values	
<ratio_0.20>	Vem sitter i sadeln?
<ratio_0.30>	Vem sitter i sadeln?
<ratio_0.40>	Vem som sitter i sadeln.
<ratio_0.50>	Vem som sitter i sadeln.
<ratio_0.60>	Det beror på vem som sitter i sadeln.
<ratio_0.70>	Mycket beror på vem som sitter i sadeln..
<ratio_0.80>	Mycket beror på vem som sitter i sadeln..
<ratio_0.90>	Mycket beror på vem som sitter i sadeln.
<ratio_1.00>	Mycket beror på vem som sitter i sadeln.
<ratio_1.10>	Det beror mycket på vem som sitter i sadeln.
<ratio_1.20>	Det beror mycket på vem som sitter i sadeln.
<ratio_1.30>	Det beror mycket på vem som sitter i sadeln.
<ratio_1.40>	Det beror mycket på vem som sitter i sadeln. - Vad då?
<ratio_1.50>	Det beror mycket på vem som sitter i sadeln. - Ja.
<ratio_1.60>	En hel del beror på vem som sitter i sadeln. - Nej.
<ratio_1.70>	Det beror mycket på vem som sitter i sadeln. - Ja.

B

Appendix 1

BART - VALUES

	WikiOpen			OpenBack		
Token	BLEU	ROUGE-2	METEOR	BLEU	ROUGE-2	METEOR
<ratio_0.30>	60.47	50.00	51.58	30.92	16.67	0.1868
<ratio_0.40>	47.25	61.19	73.74	48.15	63.64	72.78
<ratio_0.50>	44.48	58.45	68.58	43.72	58.27	69.87
<ratio_0.60>	48.52	58.45	72.85	46.67	62.14	74.40
<ratio_0.70>	50.39	62.68	74.64	47.67	61.14	74.26
<ratio_0.80>	48.24	57.78	71.72	46.98	56.43	70.69
<ratio_0.90>	48.66	58.95	75.97	49.29	59.11	74.94
<ratio_1.00>	75.46	81.30	87.87	74.88	80.88	87.84
<ratio_1.10>	54.43	62.34	77.62	55.41	64.64	77.96
<ratio_1.20>	41.32	57.51	72.15	43.18	55.90	70.24
<ratio_1.30>	42.66	38.93	58.23	38.98	48.08	64.88
<ratio_1.40>	59.98	70.05	72.62	51.02	66.05	70.37
<ratio_1.50>	40.57	62.14	70.46	34.94	59.79	66.30
<ratio_1.60>	-	24.60	43.56	-	31.39	47.01
<ratio_1.70>	50.15	62.74	76.28	50.39	62.26	75.65

Table B.1: The evaluated metrics for each dataset with the value-based ratio tokens. The results are evaluated from a fine-tuned BART model on the WikiOpen and OpenBack datasets with 20 tokens representing length ratios between 0 and 2, with an interval of 0.1. The leftmost column show the token used to evaluate the model.

Marian - VALUES

	OpenSubtitles		
Token	BLEU	ROUGE-2	METEOR
<ratio_0.30>	59.03	66.67	75.63
<ratio_0.40>	40.37	56.08	60.14
<ratio_0.50>	35.68	43.72	53.92
<ratio_0.60>	38.20	49.88	62.05
<ratio_0.70>	32.28	48.09	61.07
<ratio_0.80>	41.18	55.94	67.45
<ratio_0.90>	39.38	50.20	63.62
<ratio_1.00>	52.37	62.82	71.00
<ratio_1.10>	43.65	56.96	70.17
<ratio_1.20>	44.43	56.54	69.40
<ratio_1.30>	40.91	59.88	69.77
<ratio_1.40>	17.62	41.53	54.25
<ratio_1.50>	36.80	21.21	33.65
<ratio_1.60>	17.95	32.84	42.79
<ratio_1.70>	50.15	62.74	76.28

Table B.2: The evaluated metrics with the value-based ratio tokens. The results are evaluated from a fine-tuned Marian model on the OpenSubtitles dataset with 20 tokens representing length ratios between 0 and 2, with an interval of 0.1. The leftmost column show the token used to evaluate the model.