GRS: Combining Generation and Revision in Unsupervised Sentence Simplification

by

Mohammad Dehghan

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Mathematics
in
Computer Science

Waterloo, Ontario, Canada, 2022

Author's Declaration

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Abstract

Text simplification is a task in the natural language processing field that alters a given text to reduce the structural and lexical complexity of the text while preserving the underlying meaning.

We can classify existing text simplification approaches into generative and revision-based methods. Through explicit edit operations such as word deletion and lexical substitution, revision-based strategies iteratively simplify a given text in multiple steps. However, generative approaches generate simplified sentences from a complex sentence in one step. Generative models do not have explicit edit operations but learn implicit edit operations from data. Revision-based methods are more controllable and interpretable than generative models. On the other hand, generative models can apply more complex edits (such as paraphrasing) to a given text compared to the revision-based method.

We propose GRS: an unsupervised approach to sentence simplification that combines text generation and text revision. We start with an iterative framework in which an input sentence is revised using explicit edit operations such as word deletion and add paraphrasing as a new edit operation. This allows us to combine the advantages of generative and revision-based approaches. Paraphrasing captures complex edit operations, and the use of explicit edit operations in an iterative manner provides controllability and interpretability.

We demonstrate the advantages of GRS compared to existing methods. To evaluate our model, we use Newsela and ASSET datasets that contain high-quality complex-simple sentence pairs and are commonly used in the literature. The Newsela dataset contains 1,840 news articles re-written for children at five different readability standards. The ASSET dataset comprises 2,359 sentences from English Wikipedia. GRS outperforms all unsupervised methods on the Newsela dataset and bridges the gap between revisions-based and generative models on ASSET datasets.

Acknowledgements

I thank my supervisor, Professor Lukasz Golab, for his support, patience, and guidance throughout my research. I thank Dhruv Kumar for his help and advice. I also thank Professors Robin Cohen and Charles Clarke, who served as readers for this thesis.

Dedication

I dedicate this thesis to my family.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

Text simplification is the task of reducing the complexity and improving the readability of text while preserving its meaning. This is beneficial for persons with reading disabilities [12], non-native speakers, people with low literacy, and children. Furthermore, other natural language processing (NLP) tasks can use simplification as a pre-processing step, such as summarization [13], parsing [14], and machine translation [15].

Intuitively, humans simplify a sentence in multiple ways. One may reduce the complexity of text by replacing the difficult words with simpler synonyms, removing unimportant information, paraphrasing, or splitting the sentence. We can also change the structure of the sentence to make it simpler. Sometimes even elaborating on some parts of the sentence can help the reader to understand the text better. Figure 1.1 illustrates an example from the Newsela corpus [16] in which professional editors simplify sentences. In this example, the phrase "*with exceptions for the infirm*" is considered peripheral information and has been removed by the editor. In addition, the phrase "*are required to offer*" is paraphrased and replaced with "*should do*". Also, the word "*daily*" is changed to "*a day*".

Sentence simplification models can be categorized into *generative* and *revision-based* models. Generative approaches produce a simple sentence from a complex sentence in one step, in an auto-regressive way [17–21]. Revision-based methods iteratively edit a given sentence using a sequence of explicit edit operations such as word deletion, lexical substitution, and reordering [11, 22–24]. While generative models learn complex edit operations implicitly from data, the explicit edit operations performed by revision-based approaches can provide more control and interpretability. By learning from data, generative models can perform phrase-level complex substitutions by paraphrasing different constituents of a given sentence; however, revision-based approaches can not paraphrase, and they can only substitute difficult words with simpler ones without considering the context.

Simplification methods can also be categorized as supervised or unsupervised. Su-

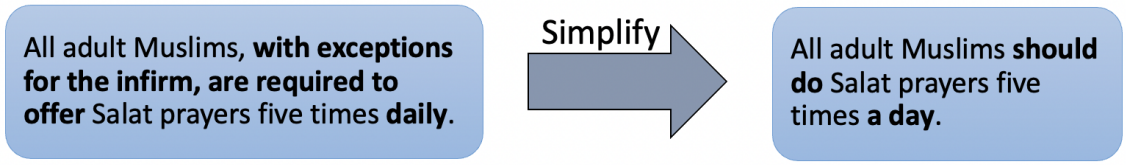

Figure 1.1

pervised methods tend to have better performance, but require aligned complex-simple sentence pairs for training [17–19, 21, 25, 26]. Unsupervised methods do not need such training data but do not perform as well [11, 20, 27].

Recently, Kumar et al. [11] proposed an unsupervised revision-based model that iteratively revises the complex sentence by applying edit operations (such as deletion, lexical simplification, and reordering) on different constituents of the sentence. This iterative simplification makes their approach more interpretable since we can obtain the intermediate simplified sentences and trace the modifications applied to a sentence. On the other hand, Martin et al. [25] proposed an unsupervised generative model that uses a BART [28] model. BART (Bidirectional Auto-Regressive Transformers) model is a state-of-the-art pre-trained language model that can generate text. Martin et al. [25] fine-tuned the BART model on paraphrase sequences. Using the BART model, they can perform more complex simplification edits and obtain better results than revision-based models.

Motivated by the shortcomings of generative and revision-based approaches, our goal is to design a hybrid simplification system with both directions' advantages. In essence, we want to have a model that is interpretable, controllable, unsupervised, and able to perform phrase-level complex substitutions by paraphrasing.

In this work we propose GRS: a new approach to bridge the gap between generative and revision-based methods for unsupervised sentence simplification. The insight is to introduce *paraphrasing* as an edit operation within an iterative revision-based framework, which was proposed by Kumar et al. [11]. For paraphrasing, we use a fine-tuned BART model [28] with lexically-constrained decoding [29–31]. This decoding technique allows us to put strict constraints on certain words in the input sentence such that they are not generated in the paraphrased sentence. Using this technique, we put these constraints on the complex fragments of a sentence, thus simplifying the sentence through paraphrasing in a controlled manner. Treating paraphrasing as an edit operation applied to a sentence

fragment introduces computation overhead since there may be many candidate fragments to choose from. To avoid the computational overhead of repeatedly performing constraint-based decoding using various combinations of words to paraphrase, GRS includes a complex component detector to identify the most appropriate words to paraphrase.

GRS is unsupervised in the sense that it does not require aligned complex-simple sentence pairs, but it uses supervised models. The paraphrasing model requires paraphrasing corpora, and the complex component detector requires two unlabeled corpora, one containing more complex sentences than the other. However, collecting paraphrasing data and unaligned simplification data is simpler than collecting aligned complex-simple pairs. The GRS model only supports the English language. This thesis is based on our work [32] published at the Findings of the 60th Annual Meeting of the Association for Computational Linguistics (ACL 2022).

To summarize, we present the following contributions:

- We design an unsupervised, controllable, and interpretable text simplification system that combines revision-based and generative text simplification approaches by integrating a paraphrasing edit operation into an iterative revision-based framework.
- We evaluate our method on ASSET [33] and Newsela [16] datasets using SARI [34] evaluation metrics. Newsela and ASSET datasets are composed of complex-simple sentence pairs and are commonly used in the literature for training and evaluation. SARI is an evaluation metric for text simplification that compares the output sentence against the reference sentence and the original input sentence. We show that our approach outperforms all unsupervised sentence simplification models on the Newsela dataset. On the ASSET dataset, we are the second-best model among unsupervised models, but we bridge the gap between generative and revision-based methods. We also conducted a human evaluation study and showed that human evaluation is aligned with the automatic assessment on the ASSET dataset.
- We devised a novel unsupervised method to obtain complex components of a sentence automatically. We evaluate the proposed complex component detector and conduct an ablation study to show the effectiveness of this module in our system.
- We do a controllability study on GRS and show that we can control various factors of the simplification system. Also, we conduct an analysis to show how frequently the GRS system uses each edit operation on average.
- We release the open-source implementation of the GRS model at https://github.com/imohammad12/GRS.

- We also release all submodels that we have used in the GRS method. These submodels are neural models used in different parts of the GRS system and are uploaded at http://huggingface.co/imohammad12. The online demos of these models are also available at the provided link.

Chapter 2

Preliminaries

This chapter discusses some important NLP concepts and models, such as pretrained language models used throughout the GRS system. Pretrained language models are trained using unsupervised data; thus, we have used them in different components of the GRS design, such as the paraphrasing operation (Section 4.2.2) and the GRS score function (Section 4.3).

In Section 2.1, we define the language models. Section 2.2 discusses the sequence-to-sequence models and how the RNN and transformer architectures work. Understanding transformer models will give us the background knowledge needed to comprehend the pretrained language models (e.g., the Bert [8] model) that have enabled transfer learning in NLP. In Section 2.3, we discuss the benefits of pretrained language models and explicate the Bert pretrained model.

2.1 Language Models

Programming languages can be specified using formal rules, but natural languages are not designed like programming languages. There is no standard specification for natural languages. Instead of defining formal specifications for natural language, we use probabilistic models to learn the model of a language from examples.

Language models are probabilistic models that can predict the next word (or sequence of words) in the sequence, given words that come before it. Given the sequence s containing words $w_1, w_2, ..., w_m$, a language model assigns a probability $P_{\text{LM}}(w_1, \ldots, w_m)$ to the whole sentence. The following equation shows how a language model obtains the probability of a sentence:

$$P_{\text{LM}}(s) = P_{\text{LM}}(w_1, \ldots, w_m) = \prod_{i=1}^{m} P_{\text{LM}}(w_i \mid w_1, \ldots, w_{i-1}) \tag{2.1}$$

Language models may be created using statistical or neural methods. Today neural network-based models are preferred since they perform better than classical statistical models. The number of words the model takes into account when predicting the next word depends on the model's complexity. Neural language models can incorporate much more context than classical statistical models.

Language models are not just used to generate text. Language models are a core component for many real-world NLP tasks such as machine translation that need language understanding.

2.2 Sequence to Sequence (Seq2Seq) Models

What is a Seq2Seq model?

Seq2Seq [1] models take a sequence as input and generate another sequence as output. For example, a Seq2Seq model can take an English sentence and generate a French sentence. Today these generative models are used in various NLP tasks such as text simplification, question answering, and conversational agents.

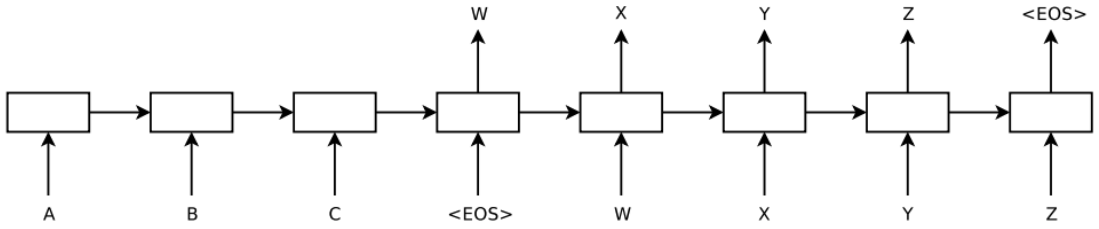

Figure 2.1 A Sequence to Sequence model that takes the "ABC" sequence as input and produces "WXYZ" sequence as output. Image takes from Sutskever et al. [1]

Figure 2.1 illustrates a Seq2Seq model. These models are composed of two parts; an encoder and a decoder. The encoder takes an input sentence $X = (x_1, x_2, \ldots, x_{|X|})$, and then the decoder generates an output sentence $Y = (y_1, y_2, \ldots, y_{|Y|})$. Generating a sentence in the decoder is done in an autoregressive way. This means that in each time step, the decoder outputs a single word. In each step, the decoder gets the incomplete sentence generated so far as input and produces the following word in the sentence. Essentially the decoder creates one probability distribution over all vocabulary in each decoding step. The training loss function is defined using these probability distributions. The model outputs the word with the highest probability during inference in each step. The probability of the output sentence Y given the input sentence X is:

$$P(Y \mid X) = \prod_{t=1}^{|Y|} P\left(y_t \mid y_{1:t-1}, X\right) \tag{2.2}$$

$$P\left(y_{t+1} \mid y_{1:t}, X\right) = \text{softmax}\left(\mathbf{W}_{lm}\mathbf{h}^t\right) \tag{2.3}$$

Where $\mathbf{W}_{lm} \in \mathbb{R}^{|V| \times d}$. $|V|$ is vocabulary size, and d is the hidden size of the model. $\mathbf{h}^t$ is the hidden state vector generated by the autoregressive decoder in step tth. $\mathbf{W}_{lm}$, the language model weights, converts a hidden state vector generated by the decoder to a vector of vocabulary size.

There are different architectures for Seq2Seq models. All of them use neural networks in their structures. Recurrent Neural Networks (RNNs) [35] were proposed earlier and are substituted by better architectures in recent years. Vaswani et al. [36] proposed Transformer models and today the Transformer model and its extensions are used for addressing various natural language understanding problems. In the following we will explain RNN (2.2.1), RNN with attention (2.2.2), and Transformer (2.2.3) models.

2.2.1 RNN

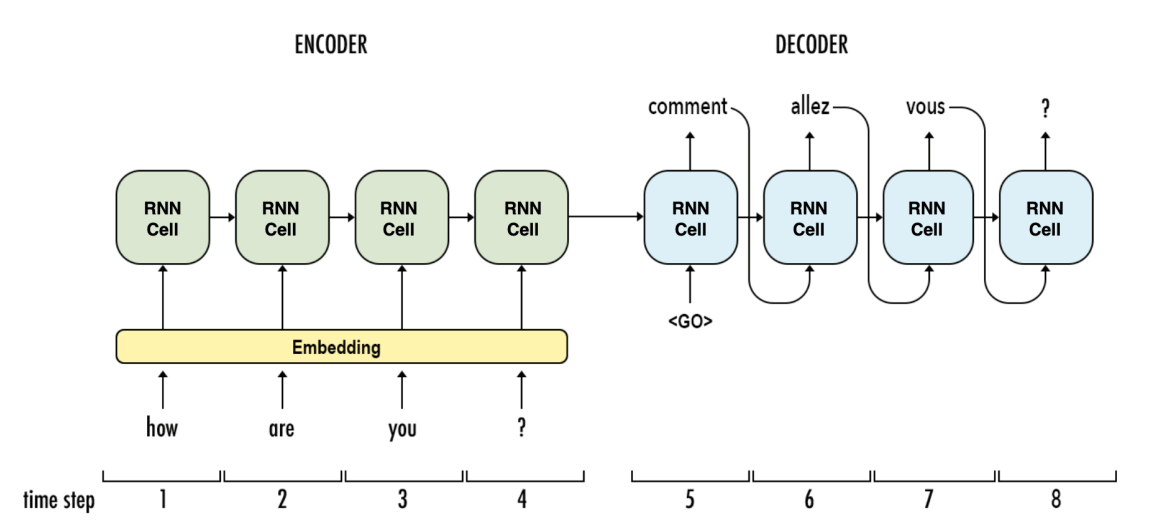

Figure 2.2 An RRN model that translates a given English sentence into a French sentence. The image is a modified version from Tracey [2].

An RNN cell takes two inputs in each time step:

1. The vector representation of a word ($\mathbf{x}^t$)

2. The hidden state generated in the previous time-step ($\mathbf{h}^{t-1}$)

Then the RNN cell generates a new hidden state for current time step ($\mathbf{h}^t$) and this hidden sate vector is given as input to the RNN cell in the next step. RRNs can process inputs of any length. The following equation shows how an RNN cell works:

$$\mathbf{h}^t = f_{act}\left(\mathbf{W}_{hh}\mathbf{h}^{t-1} + \mathbf{W}_{hx}\mathbf{x}^t + \mathbf{b}\right) \tag{2.4}$$

In this equation $\mathbf{h}^t, \mathbf{h}^{t-1}, \mathbf{x}^t$, and f_{act} refer to the hidden state vector in step tth, the hidden state vector in step $t-1$th, vector representation of the input word in step tth, and an activation function respectively. $\mathbf{W}_{hh}, \mathbf{W}_{hx}$, and $\mathbf{b}$ are weights and biases that are shared in all time steps. Figure 2.2 illustrates an RNN model used for machine translation task. The encoder of this model takes the words in the input sentence one by one in each time step. After taking the last word in the input sentence, the hidden state generated by the encoder in this time step (time step 4 in Figure 2.2) is given to the decoder. In the first decoding time step the decoder's RNN cell takes the last hidden state of the encoder and a start token (<GO> token in Figure 2.2) as input. In the next time steps the decoder's RNN cell takes the hidden state vector ($\mathbf{h}^{t-1}$) and the generated output word of the previous time step. In the decoder we use the hidden state vectors to generate output words. Equation 2.3 shows how we use a hidden state generated by the decoder of a Seq2Seq model to obtain a probability distribution over all vocabulary in each decoding step. We select the most probable word as the output word in each time step. Generally, during training instead of giving the generated word in previous time step, we give the ground truth word (target output) to the decoder's RNN.

2.2.2 RNN with Attention

As mentioned in the previous section, the decoder receives the last hidden state vector (called the "context" vector) of the encoder to generate the output sentence. The decoder should generate the output sentence using this context vector, and it becomes hard for the decoder to remember all parts of the input sentence while generating the output. Bahdanau et al. [4] and Luong et al. [3] proposed the attention method to help the model focus on relevant sections of the input sentence in each decoding step. The attention mechanism allows the decoder of an RNN model in each step to utilize all hidden states of the encoder instead of just using the last hidden state vector (context vector) generated by the encoder. In essence, the decoder can pay attention to different parts of the input sentence according to the current decoding step.

In the tth decoding step, the hidden state vector of the decoder (h_d^t) is obtained using Equation 2.4. Then for each hidden state of the encoder, we calculate an attention weight (a_{ti}). The following formula shows how an attention weight is calculated.

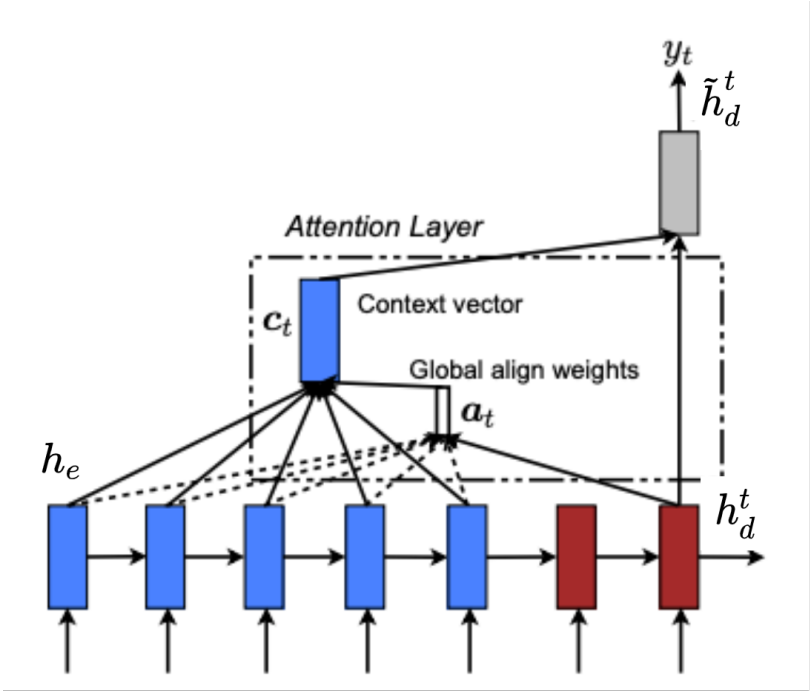

Figure 2.3 Attention mechanism in RNN models. The image is taken from Luong et al. [3]

$$a_{ti} = \text{softmax}\left(\text{score}\left(h_e^i, h_d^t\right)\right) \tag{2.5}$$

$$\text{score}\left(h_e^i, h_d^t\right) = \left(h_d^t\right)^T W_a h_e^i \tag{2.6}$$

a_{ti} indicates the attention weight associated with the ith input word in the tth decoding step.h_d^t and h_e^i refer to the hidden state of the decoder in time step t and the hidden state of the encoder in time step i, respectively. W_a is randomly initialized and updated during training. After calculating all attention weights in tth step of decoding, we calculate the context vector c_t as the weighted average over all encoder hidden state vectors:

$$c_t = \sum_{i=1}^{|\mathbf{X}|} a_{ti} h_e^i \tag{2.7}$$

In this equation |X| refer to the length of input sequence. We use a simple concatenation layer to aggregate the information from both the context (c_t) and the hidden state (h_d^t) vectors to construct an attentional hidden state vector ($\tilde{h}_d^t$):

$$\tilde{h}_d^t = \tanh\left(W_c\left[c_t; h_d^t\right]\right) \tag{2.8}$$

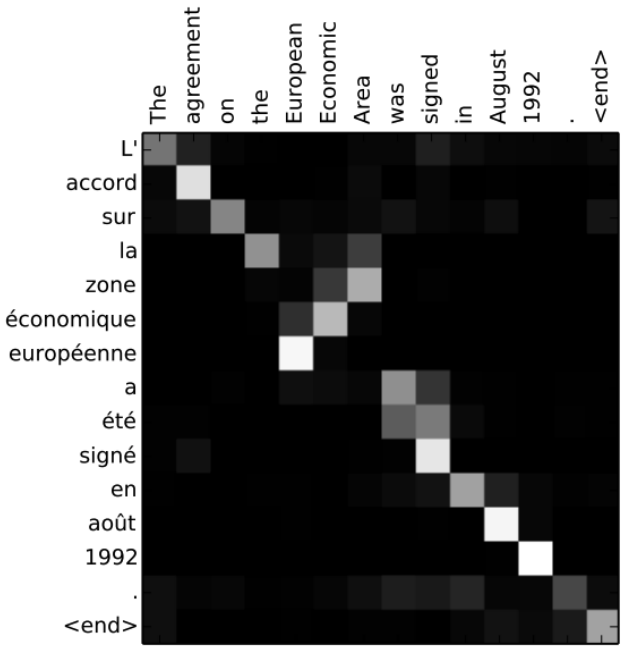

Figure 2.4 This image illustrates the attention weights of an RNN with an attention layer on the machine translation task. The words in the original sentence (English) and the produced translation (French) are represented in the x-axis and y-axis of the plot, respectively. Each pixel shows a_{ij}, which is the amount of attention that the model paid to the j-th word of the input sentence in the i-th decoding step. The image is taken from Bahdanau et al. [4].

The weight W_c is randomly initialized and updated during training. Then using the W_s weight and the softmax layer, we obtain a probability distribution over all vocabulary to predict the output word in the tth decoding step:

$$p\left(y_t \mid y_{1:t-1}, X\right) = \text{softmax}\left(W_s \tilde{h}_d^t\right) \tag{2.9}$$

Figure 2.4 illustrates how the attention mechanism works in RNN models.

2.2.3 Transformers (Self-Attention)

In 2017 Vaswani et al. [36] proposed the Transformer model, which is now widely used in the NLP field and in other AI areas such as computer vision and time series analysis. Vaswani et al. [36] proposed a new attention method called self-attention. When the model is processing a word, the self-attention mechanism allows the model to concentrate on other words in the input sentence that correlates to that word. Using this method, the Transformer model can obtain a better encoding for the current processing word by paying attention to other relevant words in the input sentence. Figure 2.5 shows the self-attention weights of the Transformer model for two given sentences. The two sentences are:

Figure 2.5 This image illustrates the self-attention weights of a transformer model. Attention weights are reflected by color intensity. The image is taken from Doshi [5]

- The cat drank the milk because it was hungry.
- The cat drank the milk because it was sweet.

In the first example the word "it" refers to "cat", but in the second example "it" refers to "milk". Figure 2.5 shows that the self-attention mechanism helps the model attend to related words when processing the word "it". This example is taken from Doshi [5].

Like previous models, the Transformer model has an encoding module and a decoding module. The encoding module is a stack of six encoders and the decoding module is a stack of six decoders. Figure 2.6 illustrates the structure of the transformer model. All encoder layers have the same architecture but they do not share the learning weights. Like encoders, all decoder layers also have the same structure but different weights. As shown in Figure 2.7, each encoder layer is composed of a self-attention and a feed forward sublayer. In addition to the feed forward and self-attention sublayers, the decoder has an encoder-decoder attention sublayer. In the following we will explain all these sublayers.

Self-Attention

Consider that the matrix $X \in \mathbb{R}^{n \times d_{\text{model}}}$ contains the embeddings of n input tokens (a token can be either a word or a smaller chunk of a word) and d_{model} is the dimension of the each embedding vector. In each self-attention sublayer we have three weight matrices; $W^Q \in \mathbb{R}^{d_{\text{model}} \times d_k}$, $W^K \in \mathbb{R}^{d_{\text{model}} \times d_k}$, and $W^V \in \mathbb{R}^{d_{\text{model}} \times d_v}$. By multiplying the matrix X by each of these weights we obtain queries (Q), keys (K), and values (V):

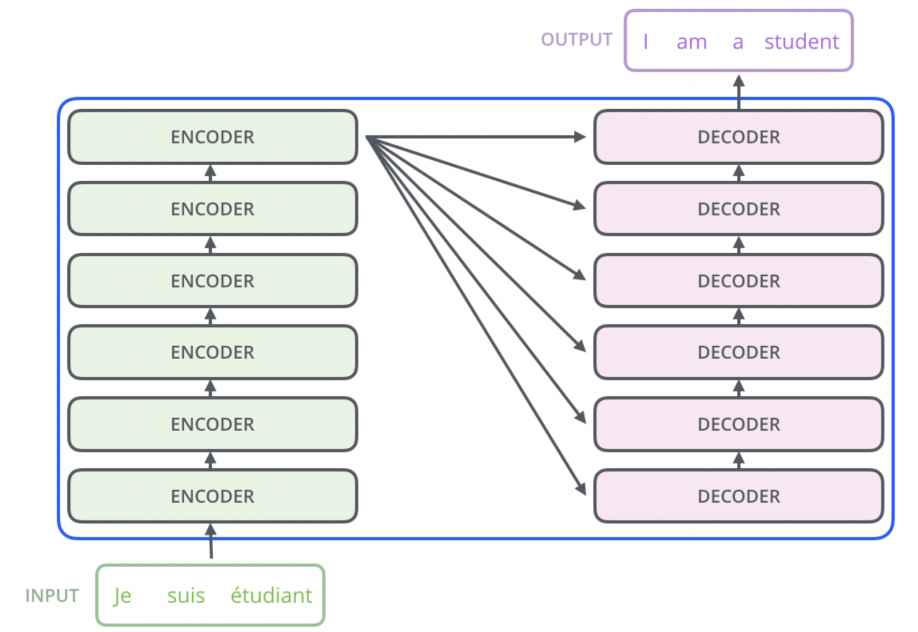

Figure 2.6 This image illustrates the structure of the Transformer model. The image is taken from Alammar [6].

$$\begin{aligned} Q &= XW^Q \\ K &= XW^K \\ V &= XW^V \end{aligned} \tag{2.10}$$

Queries, keys and values are given as input to an Attention function:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V \tag{2.11}$$

We calculate the attention for a set of queries simultaneously by using matrix multiplication. In the attention function, the dot products of each query with all keys are calculated and divided by $\sqrt{d_k}$. Then the results are given to a softmax function. The softmax scores are used as attention weights on values (V). In fact, by multiplying the softmax score matrix by the values matrix (V), we sum up weighted value vectors for each positions in the input sequence. The output of the attention function would be a matrix of shape n (number of input tokens) by d_v. The resulting matrix will then be given to a regular feed-forward neural network.

Each encoder layer has a self-attention and a feed-forward neural network with different weights. The input to the first encoder is the embedding vectors of input tokens (X), but the other encoders receive the output of the previous encoder (the one below) as input.

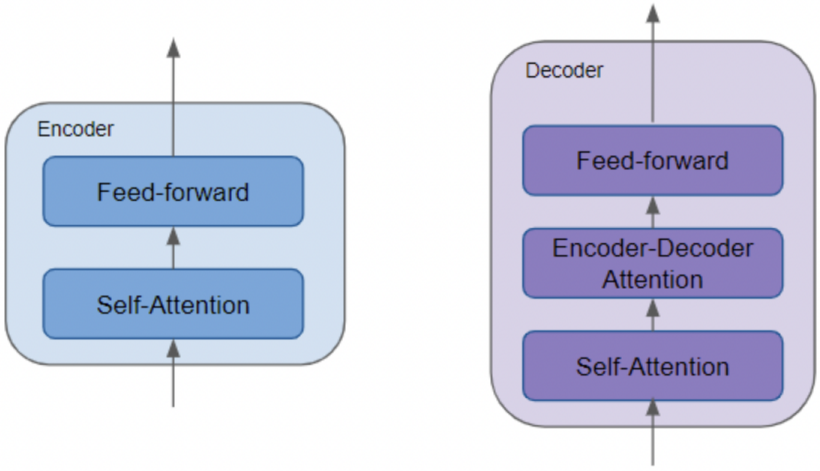

Figure 2.7 This image illustrates the structure of the encoders and decoders layers inside the encoding and decoding modules of the Transformer model. The image is taken from Doshi [5]

There are some further details regarding the self-attention sublayer. For instance, the paper introduces the "multi-headed" attention mechanism. Also, in each encoder, there are residual connections and layer-normalization steps. We do not mention these details and refer the interested readers to the original paper ([36]).

Encoder-Decoder-Attention

The structure of a decoder layer is very similar to an encoder layer. As illustrated in Figure 2.7, a decoder layer has one additional sublayer called the “Encoder-Decoder-Attention” sublayer. The structure of the self-attention and feed-forward sublayers inside encoders and decoders are the same. The attention function used in the encoder-decoder-attention sublayer is the same as the attention function (Equation 2.11) of a self-attention sublayer, but their inputs are different. In fact, for the encoder-decoder-attention sublayer, the keys and values are from the output of the last encoder layer of the encoding module (the stack of encoders), but the queries are from the previous decoding layer (the decoder below). A decoder can pay attention to all positions in the input sequence using the encoder-decoder-attention sublayer. The intuition behind the encoder-decoder-attention sublayer is similar to the regular attention mechanism used in RNNs; the decoder should be able to attend to different positions of the input sequence in the encoder while generating the output sequence. The self-attention sublayer inside the decoder layer allows the decoder to attend to the generated tokens so far.

In the end, the decoding module (the stack of decoders) will generate a hidden state vector for each position. The hidden state vectors are given to a linear layer followed by

a softmax layer like other autoregressive models. Equation 2.3 shows how a hidden state vector is converted to a vector of vocabulary size.

2.3 Pretrained Language Models

Transformer models are relatively large models with multiple hyperparameters that need a considerable amount of supervised data to be trained. For example, in the original setting [37], the transformer model was trained on a translation dataset (WMT 2014 [38]) with almost 1 million sentence pairs. Obtaining this amount of supervised data needs considerable human effort.

In many machine learning areas, Transfer Learning methods have helped researchers train accurate models with limited supervised data. Using transfer learning techniques, we can store the knowledge obtained from solving one task and apply it to another similar task. For example, consider that we train a model for classifying dogs and cats. A trained image classification model has learned how to find the generic features in the images. This model can be used as a pretrained model and then fine-tuned on another dataset to classify ants and wasps. Transfer learning techniques can reduce supervised data needed for training a model, the training time, and the generalization error.

2.3.1 BERT

In 2018, the BERT model was proposed by Devlin et al. [8]. Bert is a language representational model used for many NLP tasks such as text classification, question answering, and named-entity recognition. Training the Bert model has two steps: pretraining and fine-tuning steps. First, the model is pretrained on unlabeled text, and then the model is fine-tuned for a specific downstream task using supervised data. The pretraining step is the same for all downstream tasks, and there is no need to modify the model's architecture for each downstream task. We can transfer the language knowledge the model learned in the pretraining step to the various downstream tasks. The Bert model helped researchers obtain a very effective transfer learning method in NLP.

Model Architecture

The architecture of the Bert model is basically the encoding module (the stack of encoders) of the transformer model. In the Bert Base model 12 encoder layers and in the Bert Large model 24 encoder layers are stacked on top of each other (Figure 2.8). The structure of

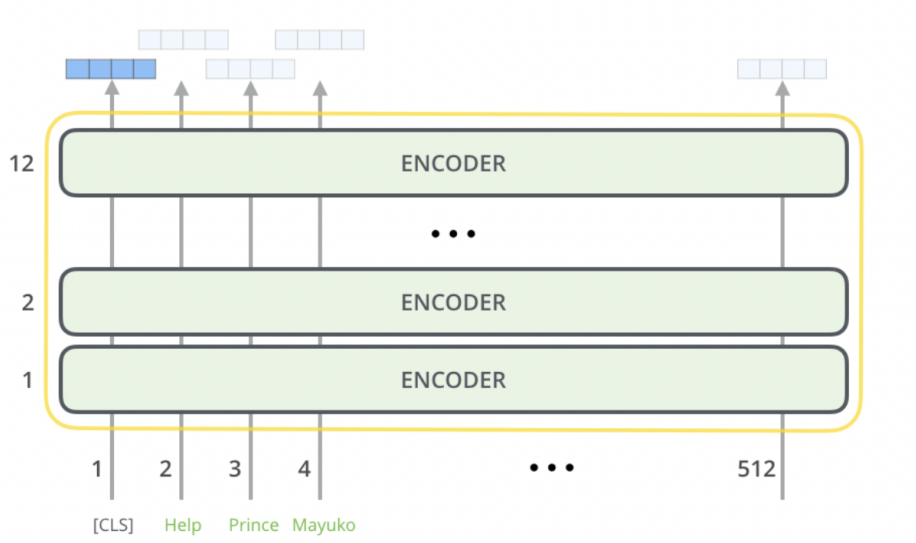

Figure 2.8 This image illustrates the structure of the Bert model. The image is taken from Alammar [7].

Bert's encoder layers is the same as encoder layers in the transformer model. As show in Figure 2.8, the Bert model receives a sequence of tokens and generates a representation (hidden state vector) for each input token. In the following we will discuss how these hidden vectors are used in each downstream task.

Pretraining

All decoders in seq2seq models, including the transformer decoder, generate text auto-regressively. The auto-regressive models generate tokens in one direction (left to right or right to left), but the Bert model is a bidirectional model that jointly considers both left and right contexts. The left-to-right models are trained using traditional language modeling objectives. However, the Bert model introduced the "*masked language model*" objective, a new method for training a deep bidirectional model. They randomly mask some input tokens in a given sentence and ask the model to predict the masked tokens. For instance, assume that the second position in the input sequence is randomly selected to be masked. Instead of giving the original token to the model, [MASK] token is given to the model in this position. Then the last hidden vector corresponding to the masked position is used to predict the actual correct word in this position. In order to create a probability distribution over all vocabulary, the hidden vector is multiplied by a language model weight followed by a softmax layer like other language models (Equation 2.3). No supervised data is needed to train a model with a masked language model objective.

Capturing the relationship between sentences in a corpus is crucial for downstream tasks

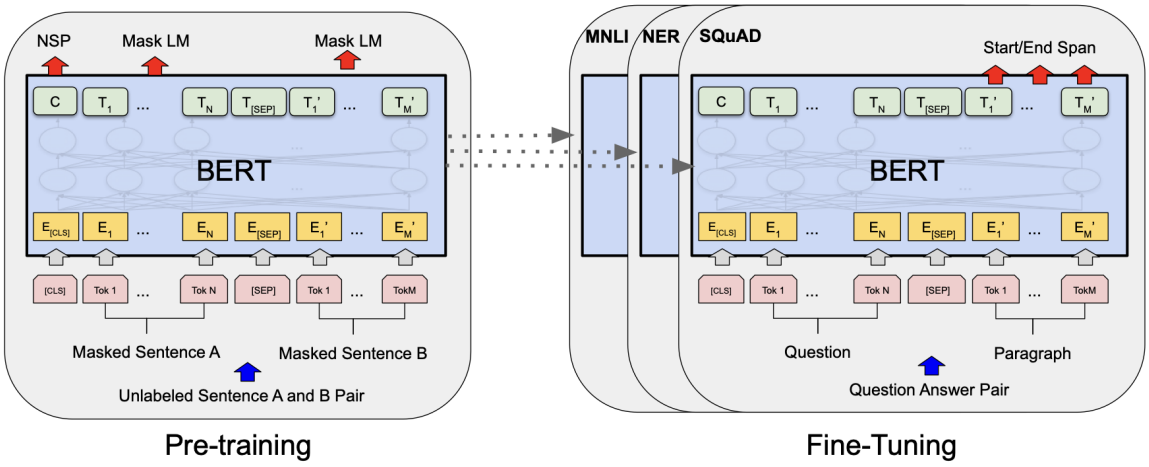

Figure 2.9 This image shows the pretraining and fine-tuning steps for the Bert model. Bert model architecture is not modified for each downstream task, and the pretraining process is the same for all tasks. [CLS] is a special token always given as the first input token, and [SEP] is a special token that separates two pieces of text (e.g., separating two sentences in the *next sentence prediction* pretraining task or the question and answer in the *question answering* downstream task). The image is taken from Devlin et al. [8].

such as question answering or natural language inference (NLI). Natural language inference is an NLP task in which we have a “hypothesis" and a “premise," and we want to find if the “hypothesis" is true, false, or undetermined given the “premise." The model can not learn the relationship between sentences by the masked language model objective. To address this problem, Devlin et al. [8] used the “*next sentence prediction*" task for training the Bert model, which is also an unsupervised task. In this task, two sentences separated by a [SEP] token are given to the model, and the model should predict if these two sentences are consecutive or not. 50% of the time, the second sentence is the following sentence coming after the first sentence in the original corpus, and 50% of the time, it is not. In the Bert model, a special [CLS] token is always given as the first input token (Figure 2.8). The final hidden vector state corresponding to the [CLS] token is used to predict if the two given sentences are consecutive or not.

Fine-Tuning

Unlike pretraining, the fine-tuning step is inexpensive and fast but needs supervised data. During fine-tuning, we should give task-specific input to the model and pass the output of the Bert model (the hidden vectors of the last encoder layer) to an output layer to train the whole Bert and the output layer end-to-end. For tasks like natural language inference (NLI) or question answering (QA) that contain two pieces of text, the [SEP] token separates the

two text fragments in the input. Also, the [CLS] token is always added in front of the input sequence. Depending on the downstream task, the final hidden vectors of different positions are given to an output layer. For text classification tasks such as sentiment analysis or NLI, the output of the [CLS] (which stands for classification) is given to an output layer. The output layer can be a neural network with a softmax activation layer to obtain probability distribution over classes. For token-level tasks such as sequence tagging, the final hidden vectors of all positions are given to an output layer. We refer the interested readers to the original paper [8] for more detail.

Chapter 3

Related Work

In this chapter, we review the related work in text simplification. First, we have a short review of early non-neural methods (Section 3.1). Then we classify recent neural methods into five categories (Section 3.2 to Section 3.6) and discuss each separately. Recent previous work can be categorized into generative or revision-based methods. Also, each method is either supervised or unsupervised. As previously explained, generative models provide less controllability; however, a few new techniques have added controllability to the generative methods. We discuss controllable generative methods in Section 3.4.

3.1 Non-neural Methods

Early work on simplification relied on rules, e.g., to split or shorten long sentences [39–41]. Later work treated simplification as a monolingual phrase-based machine translation (MT) task [42, 43], in which the source language is the complex text and the target language is the simple text. Probabilistic syntax-based machine translation [44] and phrase-based machine translation [43, 45] approaches use statistical methods to learn simplification operations from complex-simple sentence pairs.

Recent work, reviewed below, leverages neural models in a *generative* and *revision-based* manner.

3.2 Supervised Generative Methods

Supervised generative methods employ Seq2Seq [46] models to learn simplification operations from aligned complex-simple sentence pairs. Inspired by the success of Seq2Seq neural machine translation models, Nisioi et al. [47] applied a Seq2Seq model for text simplification. Building on a Seq2Seq model, Zhang and Lapata [17] used reinforcement

learning to optimize a reward based on simplicity, fluency and relevance. Recent methods build on transformer [36] models, by integrating external databases containing simplification rules [48], using an additional loss function to generate diverse outputs [19], combining syntactic rules [26], and conditioning on length and syntactic and lexical complexity features [21]. Generally, supervised models have reported better accuracy and generalized better in the text simplification domain. However, they require a large number of complex-simple pair sentences to be trained, which is not easy to obtain.

3.3 Unsupervised Generative Methods

Unsupervised generative methods, rely on non-aligned complex and simple corpora. Zhao et al. [27] leverage a back-translation framework in unsupervised machine translation and used non-aligned corpora instead of aligned complex-simple sentence pairs. Surya et al. [20] used an unsupervised style transfer paradigm, and Martin et al. [25] used a pre-trained BART model fine-tuned on paraphrased sentence pairs for addressing the text simplification problem.

3.4 Controllable Generative Methods

Controllable generative methods produce outputs at specified grade levels [49, 50], or apply syntactic or lexical constraints on the generated sentences [21, 25]. Controllable generative methods generate the output text with some constraints and specifications, but the generation process is still done in one step; thus, these models do not provide any insights into the simplification process. In contrast, revision-based methods simplify a sentence in multiple steps with explicit edit operations and provide a simplification path leading to the final sentence.

3.5 Supervised Revision-Based Methods

Supervised revision-based methods use complex-simple sentence pairs to learn where to apply edit operations. Alva-Manchego et al. [22] use keep, replace, and delete edit operations. Agrawal et al. [24] proposed a revision-based non-autoregressive model that refines a sentence in multiple steps using a fixed pipeline of edit operations to generate sentences appropriate for specific grade levels. Omelianchuk et al. [51] proposed an iterative non-autoregressive model that predicts token-level edit-operation using a tagging system. Dong et al. [23] proposed a hybrid method with explicit edit operations in an end-to-end genera-

tive model. Though these methods are more interpretable and controllable than generative methods, they still require complex-simple sentence pairs for training.

3.6 Unsupervised Revision-Based Methods

Unsupervised revision-based methods such as Narayan and Gardent [52] apply a pipeline of edit operations (e.g. lexical simplification, sentence splitting, and phrase deletion) in a fixed order. Kumar et al. [11] presented an unsupervised revision-based approach by modelling text simplification as an unsupervised search problem. Unsupervised methods usually perform worse than supervised methods.

While our method also uses a revision-based framework and an unsupervised search strategy, we integrate a generative paraphrasing model into the framework to leverage the strengths of both text generation and text revision approaches.

Chapter 4

Combining Generation and Revision in Unsupervised Sentence Simplification

4.1 Overview

Our solution, GRS, iteratively revises a given complex sentence by applying edit operations on sentence fragments. Multiple candidate simplifications are produced and evaluated using a scoring function in each iteration, and the best candidate is selected. The selected sentence acts as the input to the next iteration. This process continues until none of the candidate sentences are simpler than the input sentence.

Figure 4.1 gives an overview of GRS. GRS uses two edit operations: *paraphrasing* (Section 4.2.2; guided by the complex component detector described in Section 4.2.3) and *deletion* (Section 4.2.1). We apply a search algorithm to navigate through the search space of candidate sentences. In each iteration, edit operations generate candidate sentences, and the best sentence is selected based on the score function. The scoring function (Section 4.3) guides our search for the best simplification, using soft and hard constraints on simplicity, linguistic acceptability, and meaning preservation. Given a score for each candidate sentence, we filter out those candidates that do not improve the score of the input sentence by some threshold. The threshold depends on the edit operation from which the candidate sentence has been created (Section 4.4).

We have used multiple supervised components in our model. However, our model is unsupervised in the context of simplification since we have not used any complex-simple sentence pair in our approach. These supervised components compose the score function and the paraphrasing operation.

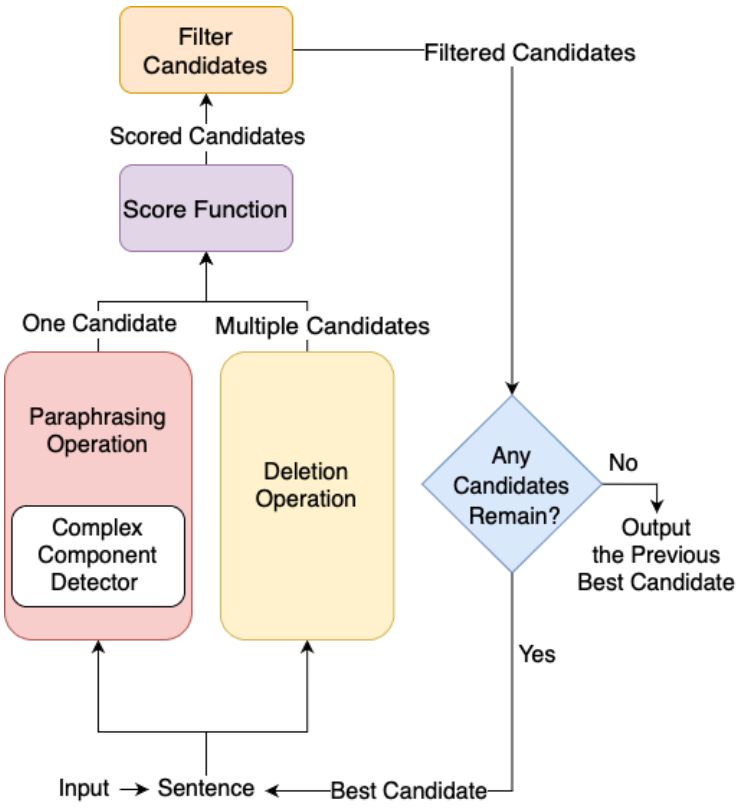

Figure 4.1 An overview of GRS. Given a complex input sentence, simplifications are iteratively produced via paraphrasing and deletion, with paraphrasing guided by the complex component detector. Sentences passing a filter (Equation 4.4) are candidates for input to the next iteration.

4.2 Edit Operations

Given an input sentence in each iteration, the GRS edit operation (i.e., paraphrasing and deletion) generates multiple candidate sentences in parallel. All the candidate sentences are evaluated using the score function. Finally, only one of the candidate sentences is selected at the end of each iteration and given as input in the next iteration. Hence, each iteration will apply just one edit operation to the sentence (deletion or paraphrasing). In the next iteration, either of these two operations can be applied to the sentence again. So the order of edit operations is not fixed, and they can be applied to a sentence many times in any order. Figure 4.2 illustrates two iterations of the GRS model and performed edit operations. In the following, we will discuss each edit operation in more detail.

4.2.1 Deletion

Deletion aims to remove peripheral information to make sentences simpler. The Deletion operation is composed of two sub-operations: removal and extraction. Inspired by Kumar

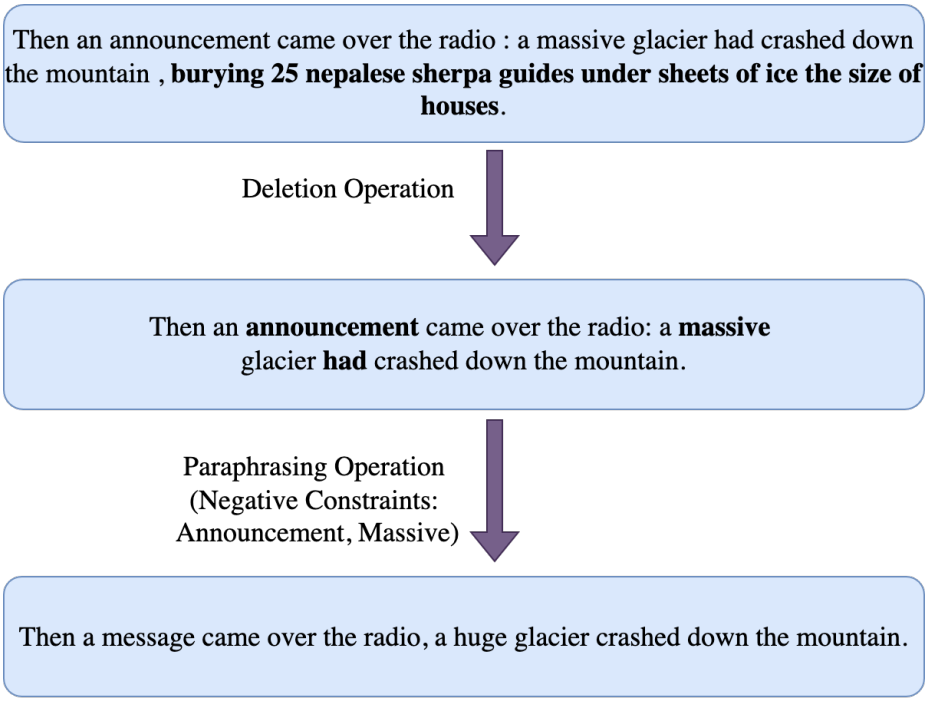

Figure 4.2 Two iterations of GRS are applied to the given complex sentence. In the first iteration, the deletion operation revised the sentence, and then the paraphrasing operation simplified the sentence in the next iteration. In the second iteration, the complex component detector identified the complex fragments ("announcement" and "massive") and passed them as negative constraints to the paraphrasing model. This example demonstrates the interpretability of GRS through building a simplification path leading to the final sentence.

et al. [11], we use the *constituency tree* of the input sentence to obtain all constituents from different depths of the parse tree. These constituents can be deleted (removal) or selected as a simplified candidate sentence (extraction). The removal sub-operation creates new candidate sentences by removing each of these phrases from the input sentence. The extraction sub-operation selects phrases as candidate sentences, which helps the model extract the main clause and drop peripheral information. Figure 4.3 demonstrates a constituency parse tree. All clause-level and phrase-level constituents (i.e., phrasal categories) such as S (sentence), VP (verb phrase), and NP (noun phrase) can be deleted or selected as a simplified sentence. For example in Figure 4.3 the constituent *"with exceptions for the infirm"* can be deleted from the sentence and the revised sentence will be *"All adult Muslims are required to offer Salat prayers five times daily. "*. Also, in the example in Figure 4.2 the deletion operation dropped the phrase *"burying 25 nepalese sherpa guides under sheets of ice the size of houses"* from the complex input sentence.

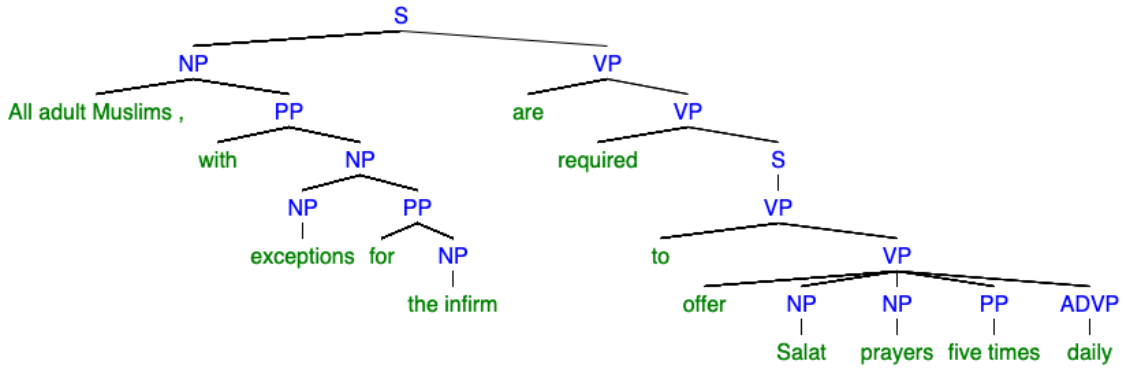

Figure 4.3 Constituency parse tree is used for deletion operation.

Input	Constraints	Paraphrase
This myth involves three **misconceptions**.	⊖ misconceptions	This myth has three false ideas.
This myth **involves** three misconceptions.	⊖ involves	The myth has three misconceptions.
How often do earthquakes **occur**?	⊖ occur	How often are earthquakes happening?
How **often** do earthquakes **occur**?	⊖ occur,often	What frequency do earthquakes happen?
How often do earthquakes occur?	⊖ How	What frequency do earthquakes occur?

Table 4.1 Outputs from a constrained decoding model trained on PARABANK paraphrasing dataset [10]. Negative constraints are denoted by ⊖. With constrained decoding techniques, the seq2seq paraphrasing model can be forced not to generate some specific words while preserving the meaning of the input sentence. Examples are taken from [10].

4.2.2 Paraphrasing

By adding paraphrasing to an iterative simplification search framework, we benefit from context-aware modifications of generative approaches in a revision-based method. We use a pre-trained BART model [28], fine-tuned on a small subset of ParaBank 2 paraphrasing dataset [53]; however, any paraphrasing auto-regressive model can be used instead.

During inference, we use lexical-constrained decoding [29–31] to place negative constraints on complex words and phrases in the input sentence. Negative constraints are words that the paraphrasing model is forced not to generate during decoding. Table 4.1 demonstrates how the constrained decoding method works when applied in the decoding phase of a Seq2Seq paraphrasing model. Consider that any autoregressive model can use this technique during its decoding stage to prevent generating some specific words or tokens. Figure 4.2 shows an example in which an input sentence was paraphrased to exclude two complex words (negative constraints): “massive” and “announcement”. We explain how to choose negative constraints below, with the help of the complex component detector.

Figure 4.4 One of the attention matrices of the DeBERTa complex-simple classifier (head 11 of the second layer). Attention weights are reflected by color intensity. The input sentence in this example is “below are some useful links to facilitate your involvement." We used BertViz [9] to visualize attention weights. The demonstration is for the attention matrix of head 11 of the second layer.

4.2.3 Complex Component Detector (CCD)

4.2.3.1 Why do we need a complex component detector?

In a brute-force search strategy for simple sentences, a complex sentence may be paraphrased using many combinations of negative constraints to find a simplified version of the given complex sentence among many paraphrased sentences. To reduce the running time of the simplification process, GRS includes a complex component detector, which reduces the number of times calling the paraphrasing model. In GRS, before paraphrasing a sentence, the complex component detector predicts the best negative constraints; then, the sentence and the predicted negative constraints are given to the paraphrasing model to generate a new candidate sentence. As a result, the paraphrasing operation is called only once per iteration of GRS, using the predicted negative constraints, avoiding the expensive process of repeatedly paraphrasing the input using different combinations of negative constraints.

4.2.3.2 How to obtain an unsupervised complex component detector?

First, we implemented a complex-simple classifier that gives a simplicity probability to a given sentence. We only require two corpora (non-aligned) with different complexity levels to train this classifier. Reid and Zhong [54] showed that it is possible to extract style-specific sections of a sentence using the attention layers of a style classifier. Similarly, we use the attention layers of our complex-simple classifier to extract the complex components from a given input sentence. We fine-tune the pre-trained DeBERTa model [55] as our complex-

simple classifier. In section 5.2 we have provided more detail about the classifier model and the used dataset. Figure 4.4 illustrates one of the attention heads of the second layer of DeBERTa. This visualization shows that the word “faciliate” was attended to more than the other words in the given sentence. We use this intuition and devise a formula (Equations 4.1 and 4.2 below) to detect complex words by analyzing attention weights.

BERT [56] and its extensions (e.g. DeBERTa) add a [CLS] token to the beginning and a [SEP] token to the end of each sentence (as shown in Figure 4.4). In these models, the hidden states of the [CLS] token in the last layer are used for classification tasks. In our complex-simple classifier, we found that the attention paid by the [CLS] token in the second layer to other words in the sentence can help us detect complex components. Equations 4.1 and 4.2 demonstrate how we extract complex components from attention head matrices of the second layer of the classifier. Here, $\mathbf{A}_{h,i}^{[CLS]}$ refers to the amount of attention the [CLS] token in the hth attention head of the second layer pays to the ith token of the input sentence. N and H refer to the length of the input sentence and the number of attention heads, respectively. c_i defines whether the ith token is complex or not. If $c_i = 1$, then this token will be set as a negative constraint. $\bar{T}$ is a threshold used for finding complex tokens. In the example demonstrated in Figure 4.4, only c_8, which refers to the word “facilitate", is a complex token.

$$\bar{T} = \frac{\sum_{h=0}^{H-1} \sum_{i=0}^{N-1} \mathbf{A}_{h,i}^{[CLS]}}{N} \tag{4.1}$$

$$c_i = \begin{cases} 1, & \text{if } \sum_{h=0}^{H-1} \mathbf{A}_{h,i}^{[CLS]} \geq \bar{T} \\ 0, & \text{otherwise} \end{cases} \tag{4.2}$$

On the whole, the paraphrasing operation receives a sentence as input, and before calling the paraphrasing model, the input sentence is given to the complex component detector. After specifying the negative constraints (complex words), the paraphrasing model rewrites the input sentence without generating the negative constraints using the lexically-constrained decoding technique. The output of the paraphrasing operation is just one candidate sentence.

4.3 Scoring Function

Candidates generated by our two edit operations may not be correct regarding linguistic acceptability. Furthermore, important information from the original sentence may have been removed. We use a score function to evaluate the simplicity of the candidate sentence and filter out non-grammatical candidates or sentences that are not conceptually similar to the original sentence. The score function is composed of three components.

4.3.1 Meaning Preservation (H_{mp})

A good text simplification model does not sacrifice the meaning of the original sentence to obtain a simple sentence. The meaning preservation module of the score function calculates the semantic similarity between the original sentence and a given candidate sentence. First, we use the method proposed in Reimers and Gurevych [57] to obtain the semantic representations of the sentences. We then use the cosine similarity measure between the original and the candidate sentence representations. Our meaning preservation measure acts as a hard filter. A hard filter assigns a zero score to candidate sentences that do not pass a certain threshold. Section 5.7 discusses the effects of different values for this threshold on the generated outputs. By changing this threshold, we can control how much the model is conservative.

4.3.2 Linguistic Acceptability (H_{la})

By removing some components of a complex input sentence, the output sentence may become nonsensical. To check the linguistic acceptability of the generated sentences, we train a classifier on the CoLA (the corpus of linguistic acceptability) [58] dataset. This classifier measures the probability that a given sentence is grammatical. This module, like the meaning preservation module, is used as a hard filter in the score function.

4.3.3 Simplicity (S_{simp})

This module evaluates the simplicity of a given candidate and is a soft constraint. It does not consider the original sentence while evaluating the simplicity of the candidate, so an oversimplified sentence may also receive a high simplicity score. For obtaining the simplicity score of a given text, we use the complex-simple classifier mentioned in Section 4.2.3, which computes the simplicity probability of a sentence.

These three measures together evaluate the quality of each candidate sentence, as shown in Equation 4.3. In this equation, S, S_{simp}, H_{la}, H_{mp}, c, and o refer to the score function, simplicity module, linguistic acceptability hard filter, meaning preservation hard filter, candidate sentence, and the original sentence, respectively. Outputs of the hard filters are zero or one ($Range : \{0, 1\}$) and the output of the simplicity module is between zero and one ($Range : (0, 1)$).

$$S(c) = S_{simp}(c) * H_{mp}(c, o) * H_{la}(c) \tag{4.3}$$

4.4 Simplification Search and Stopping Criteria

The GRS simplification system is an iterative search in the space of candidate sentences (Figure 4.1). The unsupervised search method is inspired by Kumar et al. [11], but with different simplification operations and a different score function. Given a complex input sentence, paraphrasing and deletion operations generate candidate sentences separately. In each iteration, the paraphrasing operation creates only one candidate sentence, as described in Section 4.2.2, whereas the deletion operation generates multiple candidate sentences (Section 4.2.1). Candidates sentences are then evaluated according to the scoring function. Those candidates that do not improve the score of the input sentence by some threshold are filtered out. The threshold depends on the edit operation that the candidate sentence has been created from. In Equation 4.4, t_{op} is the threshold associated with operator op. S, c, and c' refer to the score function, the candidate sentence, and the input sentence in the current iteration, respectively. Candidate sentences having scores lower than $S(c') * t_{op}$ will be removed. Among sentences that have passed the operation filters, the sentence having the highest score will be given to the system again in the next iteration. This iterative search algorithm continues until no candidate sentence can pass the filters and improve the input sentence in the current iteration, which is the best candidate sentence in the previous iteration.

$$S(c) > S(c') * t_{op} \tag{4.4}$$

Chapter 5

Experiments

This chapter discusses various experiments, analysis, and ablation studies, as well as the experimental setup. Sections 5.1, 5.2, and 5.3, discuss the used datasets, the training details of various models used in the system, and the evaluation metrics. Section 5.4 explains the existing simplification models with which we have compared GRS. In Sections 5.5 and 5.6 we evaluate GRS and compare it with existing approaches using automatic and human evaluation, respectively. Section 5.7 illustrates a controllability study on the GRS system. In Section 5.8, we evaluate the complex component detector and discuss an ablation study by replacing the complex component detector with a simpler module in the GRS system. Section 5.9 provides an analysis of the simplification search.

5.1 Datasets

We use the Newsela [16] and ASSET datasets [33] to evaluate GRS against existing simplification methods. Newsela contains 1840 news articles for children at five reading levels. We use the split from Zhang and Lapata [17], containing 1129 validation and 1077 test sentence pairs. ASSET includes 2000 validation and 359 test sentences pairs. Each sentence has ten human-written references.

5.2 Training Details

5.2.1 Paraphrasing Model

We fine-tune a pre-trained BART model [28] implemented by Wolf et al. [59]. To do this, we use a subset of the ParaBank 2 paraphrasing dataset [53] containing 47,000 pairs. The data used for fine-tuning the model influences how the model rewrites the sentence. We observe

that a conservative paraphrasing model helps us to control the generated output. This is because such a model is better at specifically changing only words provided as negative constraints. Thus, for fine-tuning the BART model, we select paraphrasing sentence pairs that are semantically similar to each other. The Parabank v2.0 dataset has up to 5 paraphrases per reference. From all possible pairs for each reference, we select the pair that are more semantically similar to each other since they have few differences. For calculating the semantic similarity of sentence pairs, we use the model from Reimers and Gurevych [57] to obtain sentence embeddings, and then we use cosine-similarity to find the most similar sentence pairs in each reference. Consider that the lexically-constrained decoding method used in the simplification search is independent of the paraphrasing model.

The BART model is composed of a 12-layer encoder and a 12-layer decoder, each layer containing 16 attention heads. The model's hidden size is 1024, and the tokenizer vocabulary size is 50265. We use a batch size of 8 (per device). It took approximately 1.5 hours to fine-tune the model using three NVIDIA 2080 Ti GPUs.

5.2.2 Complex-Simple Classifier

We use a pre-trained DeBERTa [55] model implemented by Wolf et al. [59]. This model is composed of a 12-layer self-attentional encoder, each layer containing 12 attention heads. The model's hidden size is 768, and the tokenizer vocabulary size is 30522. To fine-tune the DeBERTa model for the binary classification task, we use the Newsela-Auto dataset [60]. To train the classifier, we use the AdamW [61] optimizer with a learning rate of 5×10^{-5} and a batch size of 16. Note that we do not use the alignment between complex-simple sentence pairs in the Newsela-Auto dataset. Thus, our complex-simple classifier can be trained on any text corpora with different complexity levels. It took approximately one hour to fine-tune the classifier using a single NVIDIA 2080 Ti GPU. The accuracy of this classifier is 78.46.

5.2.3 Meaning Preservation Module of the Scoring Function

To obtain contextual embeddings of sentences, we use the SentenceTransformers [57] framework, specifically, the paraphrase-mpnet-base-v2 pre-trained model.

5.2.4 Linguistic Acceptability Module of the Scoring Function

To score the linguistic acceptability of a sentence, we fine-tune a pre-trained DeBERTa model [55] for a binary classification task on the CoLA (the corpus of linguistic acceptability) [58] dataset. It contains 10,657 sentences, each labelled either as grammatical or ungrammatical. The configuration and training hyperparameters of this classifier are the

same as the complex-simple classifier explained above. It took approximately 30 minutes to fine-tune the model using a single NVIDIA 2080 Ti GPU. On the validation set, the accuracy of the model is 79.33.

5.2.5 Simplification Search and Score Function

The threshold associated with paraphrasing (t_{par}) is 0.8, and the thresholds related to the removal (t_{dl-rm}) and extraction (t_{dl-ex}) sub-operations of the deletion operation are 1.1, and 1.25, respectively. The score function's meaning preservation (H_{mp}) and linguistic acceptability (H_{la}) thresholds are 0.7 and 0.3, respectively. We obtained these values using the validation set. These values are used for both ASSET and Newsela datasets.

5.3 Evaluation Metrics

To evaluate GRS and other models, we use SARI [34] as our primary metric. SARI (System output Against References and against the Input sentence) evaluates the quality of the output text by calculating how often the output text correctly keeps, inserts, and deletes n-grams from the complex sentence, compared to the reference text, where $1 \leq n \leq 4$. We report the overall SARI score, as well as individual SARI scores corresponding to n-grams correctly added (ADD), deleted (DELETE) and kept (KEEP); the overall SARI score is the mean of these three scores. We also report the FKGL score, which only considers the output sentence, not the source and reference sentences. It is computed based on sentence length and the number of syllables for each word in the sentence. We use the EASSE package [62] to calculate SARI and FKGL. We do not use the BLEU metric [63] since Sulem et al. [64] showed that BLEU does not correlate well with simplicity.

5.4 Competing Models

We evaluate GRS with different configurations: only deletion - GRS: DL(RM+EX), only paraphrasing - GRS: PA, and both deletion and paraphrasing - GRS: PA+DL(RM+EX). We also consider the complex sentence itself as a trivial baseline, denoted by 'Identity Baseline'. The ASSET dataset contains multiple references for a sentence, so we also calculate an upper bound for a given evaluation metric, which we denote as 'Gold Reference'. To calculate the 'Gold Reference' score, each reference is selected once, and the scores are calculated against others. Finally, we average across all the reference scores to obtain the final 'Gold Reference' score. 'Gold Reference' scores for the Newsela dataset cannot be calculated since only one reference sentence is available.

Model	SARI ↑	Add ↑	Delete ↑	Keep ↑	FKGL ↓	Len
Identity Baseline	12.24	0.00	0.00	36.72	8.82	23.04
Unsupervised Models						
Zhao et al. [27]	37.20	1.51	73.53	36.54	3.80	11.78
Kumar et al. [11]	38.36	1.01	77.58	36.51	**2.81**	9.61
Martin et al. [25]	38.29	**4.44**	76.02	34.42	4.65	12.49
GRS: DL (RM + EX)	37.52	0.66	69.45	**42.44**	3.93	12.64
GRS: PA	36.42	3.44	69.55	36.28	5.79	19.08
GRS: PA + DL (RM + EX)	**40.01**	3.06	**80.43**	36.53	3.20	11.72
Supervised Models						
Narayan and Gardent [45]	34.73	0.77	73.22	30.21	4.52	12.40
Zhang and Lapata [17]	38.03	2.43	69.47	**42.20**	4.78	14.36
Dong et al. [23]	39.28	2.13	77.17	38.53	3.80	10.92
Zhao et al. [27]	39.14	2.80	74.28	40.34	4.11	11.63
Martin et al. [25]	**41.20**	**6.02**	**81.70**	35.88	**2.35**	9.22

Table 5.1 Comparison of supervised and unsupervised simplification models on the Newsela test set. PA and DL refer to paraphrasing and deletion, respectively. RM and EX refer to the removal and extraction sub-operations of the deletion operation. ↑ denotes the higher the value, the better. ↓ denotes the lower the value, the better.

We also compare GRS with existing approaches. From unsupervised methods, we select unsupervised generative models that use Seq2Seq models Surya et al. [20], Zhao et al. [27]. We also compare with Martin et al. [25], which leverages pretrained language models and a large paraphrase pair dataset, and Kumar et al. [11], an iterative revision-based method with several explicit edit operations (deletion, lexical substitution and reordering).

From supervised methods, we start with Narayan and Gardent [45] and Xu et al. [34], which use phrase-based MT models. We also consider Seq2Seq generative methods: Zhang and Lapata [17], which uses reinforcement learning, and Martin et al. [21, 25], Zhao et al. [27], which use Seq2Seq transformer models. Next, we select Omelianchuk et al. [51], a recent supervised revision-based method. Finally, we consider Dong et al. [23], a hybrid approach using explicit edit operations in a generative framework.

5.5 Automatic Evaluation

Tables 5.1 and 5.2 illustrate the results on Newsela and ASSET, respectively. We report the overall SARI score, the individual scores of three operations used in SARI score, the FKGL score, and the average length of the output sentences. To evaluate previous methods, we obtained their output sentences on ASSET and Newsela from the respective project Github

pages or by contacting the respective authors, followed by calculating the SARI and FKGL scores using the EASSE package (described in Section 5.3). One exception is Omelianchuk et al. [51]: since they also used the EASSE package, we copied their reported ASSET scores in Table 5.2, but they did not report the average sentence length.

For Newsela, using paraphrasing and deletion together (GRS: PA + DL(RM+EX)) gives the best performance on the SARI metric. On the Newsela dataset, our best model outperforms previous unsupervised methods and achieves +1.6 SARI improvement. It also outperforms all supervised methods except Martin et al. [25].

For ASSET, even though Martin et al. [25] perform better than our best model, we improve the performance over Kumar et al. [11] by +3.6 SARI points and close the gap between revision-based and generative approaches. Compared to supervised models, our unsupervised model again outperforms others except Martin et al. [25] and Omelianchuk et al. [51]. For the ASSET dataset, we observe that our model with only paraphrasing (GRS (PA)) has the best SARI score.

Analyzing the results, we observe that simplification is done differently by human annotators in Newsela than in ASSET. In Newsela, removal of peripheral information through content deletion happens more aggressively. The average reference sentence length is 12.75 compared to 23.04 for the source sentences. Figure 4.2 demonstrates one such example from the Newsela dataset. In this example, two long phrases have been removed from the beginning and the end of the source sentence (compare the source sentence with reference). However, in ASSET, content removal is conservative and can be handled by paraphrasing alone. The average reference sentence length is 16.54 compared to 19.72 for the source sentences. Simplifications in ASSET focus on lexical simplification, sentence splitting and word reordering.

Martin et al. [25] leverage a pretrained BART model [28] and fine-tune it on a paraphrasing dataset containing 1.1 million sequence pairs. Unlike traditional paraphrasing datasets that are structured at the sentence level, their paraphrasing dataset contains multiple sentences in a sequence, thus allowing the model to learn a sentence splitting operation as well. Thus, they outperform the previous best unsupervised models on ASSET. On Newsela, both GRS and the model from Kumar et al. [11] perform better than Martin et al. [25] since they include an explicit removal edit operation that focuses only on content deletion. Martin et al. [25] instead do not explicitly perform content removal and only do content deletion by way of paraphrasing. Finally, Kumar et al. [11] does not perform well on ASSET since they do not perform paraphrasing. Our new design thus combines the advantages of both revision-based and generative approaches.

Model	SARI ↑	Add ↑	Delete ↑	Keep ↑	FKGL ↓	Len
Identity Baseline	20.73	0.00	0.00	62.20	10.02	19.72
Gold Reference	44.89	10.17	58.76	65.73	6.49	16.54
Unsupervised Models						
Surya et al. [20]	35.19	0.83	45.98	58.75	7.60	16.81
Zhao et al. [27]	33.95	1.99	42.09	57.77	7.51	18.80
Kumar et al. [11]	36.67	1.29	51.33	57.40	7.33	16.56
Martin et al. [25]	**42.42**	**7.15**	61.32	**58.77**	7.49	16.36
GRS: DL (RM + EX)	37.90	0.89	62.32	50.50	4.17	11.18
GRS: PA	40.41	7.00	62.37	51.88	6.70	17.94
GRS: PA + DL (RM + EX)	37.40	3.89	**67.46**	40.85	**3.45**	10.69
Supervised Models						
Narayan and Gardent [45]	34.65	1.3	59.24	43.41	**5.18**	10.95
Xu et al. [34]	37.11	5.07	45.21	61.06	7.95	20.50
Zhang and Lapata [17]	36.59	2.38	50.10	57.30	7.66	14.37
Zhao et al. [48]	38.67	4.36	51.37	60.29	7.73	18.36
Dong et al. [23]	34.95	2.40	42.69	59.73	8.38	16.49
Martin et al. [21]	40.13	6.53	50.84	**62.99**	7.29	19.49
Zhao et al. [27]	35.15	2.22	45.32	57.91	7.83	16.14
Martin et al. [25]	**44.05**	**10.93**	61.91	59.30	6.13	18.49
Omelianchuk et al. [51]	43.21	8.04	**64.25**	57.35	6.87	——-

Table 5.2 Comparison of supervised and unsupervised simplification models on the ASSET test set. PA and DL refer to paraphrasing and deletion, respectively. RM and EX refer to the removal and extraction sub-operations, the sub-operations of the deletion operation. ↑ denotes the higher value, the better. ↓ denotes the lower value, the better.

5.6 Human Evaluation

We randomly selected 30 sentences from the ASSET test set for human evaluation. Following [19], we measure Fluency (whether the sentence is grammatical and well-formed), Simplicity (whether it is simpler than the complex sentence), and Adequacy (whether it keeps the meaning of the complex sentence).

We recruited four volunteers to assess the sentences based on the defined metrics and evaluate the performance of various models, including GRS. We provided thorough instruction to the volunteers. In the instruction, we gave multiple examples and discussed how they should mark sentences based on the defined criteria. They were given enough time to evaluate all the sentences. Also, we answered their questions during the evaluation process and randomly checked some of their evaluations to ensure they had done their

CCD-module	Adequacy ↑	Simplicity ↑	Fluency ↑	Average ↑
Reference	4.29	4.08	4.76	4.37
GRS(PA)	**3.98**	4.04	4.47	4.17
Surya et al. [20]	3.57	3.48	4.32	3.79
Zhao et al. [27]	3.89	3.27	4.54	3.89
Martin et al. [25]	3.95	**4.17**	**4.78**	**4.30**
Kumar et al. [11]	3.15	3.56	4.15	3.62
Zhang and Lapata [17]	3.67	3.64	4.69	4.00

Table 5.3 Human evaluation on the ASSET dataset. Adequacy, simplicity, and fluency are human evaluation metrics, and in the fourth column, the average of these metrics is shown. Each row represents a simplification model. Human evaluation scores are based on a 1–5 Likert scale. ↑ denotes the higher value, the better.

duties correctly. We estimated the time needed to evaluate all given sentences and paid the volunteers accordingly. Results are shown in Table 5.3. All models are unsupervised except Zhang and Lapata [17].

The fourth column in Table 5.3 shows the average score of all three metrics used in the human evaluation. According to the average scores, MUSS [25] has the best performance, followed by GRS. The human evaluation demonstrates that the automatic evaluation (SARI scores shown in Table 5.2) is aligned with human evaluation scores. GRS has the best performance in meaning preservation (Adequacy). This may be because we have a relatively conservative paraphrasing model. Also, GRS evaluated in this study is only leveraging paraphrasing, and this version is the most conservative.

5.7 Controllability

By manipulating the thresholds for the components of the score function and the edit operations, we can control the amount of deletion, paraphrasing, and the trade-off between simplicity and meaning preservation. We show the results in Table 5.4 using the GRS (PA + DL) model and the Newsela test set. The column labels have the same meaning as in Tables 5.1 and 5.2.

5.7.1 Meaning Preservation Threshold

As mentioned in Section 4.3, meaning preservation is a hard filter in our score function. We assign a zero score to candidate sentences having a similarity score lower than the meaning preservation threshold when compared to the original sentence. As the meaning

preservation threshold increases, candidate sentences less similar to the original sentence are pruned. Sentences more similar to the original sentence have higher Keep and lower Delete SARI scores. The SARI Add score increases since paraphrasing is prioritized over deletion. Finally, the length of the output sentences increases since the model becomes more conservative.

5.7.2 Removal Threshold of Deletion Operation

This threshold is described in Equation 4.4. By increasing this threshold, the SARI Keep score increases and the SARI Delete score decreases, which also results in increased average length. This is consistent with our intuition about the deletion operation. The SARI Add score increases as well since a higher deletion threshold makes the model conservative on deletions and thus candidates from the paraphrasing operation are more likely to be selected.

5.7.3 Paraphrasing Threshold

This threshold is described in Equation 4.4. Reducing this threshold results in more aggressive paraphrasing. Thus, we observe an increase in the SARI Delete and Add scores since paraphrasing replaces complex words and phrases with simpler ones. In fact, the complex component detector used in the paraphrasing operation detects the complex words and then these complex words are removed by the constrained paraphrasing model. This is the reason why reducing the paraphrasing threshold helps the deletion score to be increased. However, by reducing the paraphrasing threshold the keeping score reduces, and we should find the best value that maximizes the overall SARI score.

5.7.4 Linguistic Acceptability Threshold

Like meaning preservation, linguistic acceptability is a hard filter in our score function (Section 4.3). We assign a zero score to candidate sentences having a linguistic acceptability score lower than the linguistic acceptability threshold. As the linguistic acceptability threshold increases, more candidate sentences receive a zero score. This results in a more conservative model that makes fewer changes to the input sentences because the original sentences are already linguistically acceptable. By increasing the linguistic acceptability threshold, the SARI Deletion score drops and the SARI Keep score increases. Also, this results in longer sentences.

Value	SARI ↑	Add ↑	Delete ↑	Keep ↑	FKGL ↓	Len
Effect of Meaning Preservation Threshold (H_{mp})						
0.25	38.18	2.15	84.64	27.76	0.42	7.68
0.5	39.49	2.30	83.58	32.59	1.63	8.99
0.6	39.78	2.59	81.94	34.80	2.46	10.29
0.7	39.99	3.16	79.81	36.99	3.27	12.16
Effect of the Removal Threshold of Deletion Operation (t_{dl-rm})						
0.9	37.33	1.93	82.72	27.34	2.19	8.58
1.0	38.03	2.20	81.63	30.25	2.53	9.80
1.1	40.01	3.06	80.43	36.53	3.20	11.72
1.2	39.98	3.15	79.96	36.85	3.26	12.07
Effect of the Paraphrasing Threshold (t_{par})						
0.8	39.99	3.16	79.81	36.99	3.27	12.16
0.9	40.01	3.15	79.55	37.31	3.32	12.24
1.0	39.42	2.69	75.03	40.54	3.84	12.99
1.1	38.55	1.97	70.47	43.23	4.11	13.50
Effect of the Linguistic Acceptability Threshold (H_{la})						
0.6	39.42	3.06	78.19	37.00	3.65	12.54
0.7	39.52	3.00	77.98	37.57	3.68	12.67
0.8	39.69	3.08	77.60	38.40	3.79	12.85
0.9	38.41	2.93	76.87	38.42	4.04	13.04

Table 5.4 Impact of paraphrasing, deletion, meaning preservation, and linguistic acceptability thresholds on the Newsela dataset.

5.8 Complex Component Detector Evaluation

To show the effectiveness of the proposed Complex Component Detector (CCD) mentioned in Section 4.2.3, we evaluate the CCD module on the Complex Word Identification (CWI) task. The task is defined as a sequence tagging problem in which each word in a sentence is tagged as complex or not complex. We use the test set of CWIG3G2 [65], a professionally written news dataset. As explained in Section 4.2.3, the CCD module used in GRS (denoted by Att-Cls) operates by interpreting the attention matrix of the second layer of the complex-simple classifier. We also compare with the lexical simplification operation of Kumar et al. [11], denoted by LS-CCD. It uses the IDF scores to find complex words in a sentence.

Model	SARI ↑	Add ↑	Delete ↑	Keep ↑	FKGL ↓	Len
			ASSET			
Identity Baseline	20.73	0.00	0.00	62.20	10.02	19.72
Gold Reference	44.89	10.17	58.76	65.73	6.49	16.54
GRS(PA, CCD:LS-CCD)	37.80	5.59	57.39	50.44	7.17	18.75
GRS(PA, CCD:Att-Cls)	**40.41**	**7.00**	**62.37**	**51.88**	**6.70**	17.94
			Newsela			
Identity Baseline	12.24	0.00	0.00	36.72	8.82	23.04
GRS(PA+DL(RM), CCD:LS-CCD)	39.30	2.87	78.18	**36.85**	**3.39**	13.54
GRS(PA+DL(RM), CCD:Att-Cls)	**39.61**	**3.18**	**79.13**	36.52	3.45	13.43

Table 5.5 Comparison of GRS versions that use different Complex Component Detectors (CCD) on ASSET and Newsela. PA and DL refer to paraphrasing and deletion, respectively. RM refers to removal, which is the sub-operation used in deletion operation. ↑ denotes the higher value, the better. ↓ denotes the lower value, the better.

CCD-module	Acc ↑	Rec ↑	Prec ↑	F1 ↑
LS-CCD	84.67	37.36	70.51	48.84
Att-Cls	84.58	47.95	72.59	57.75

Table 5.6 Performance of different Complex Component Detectors (CCD) on the Complex Word Identification (CWI) task. CWIG3G2 dataset has been used for this evaluation. LS-CCD and Att-Cls refer to the CCD module obtained from Lexical Simplification edit operation of Kumar et al. [11] and the original CCD module used in GRS design explained in Section 4.2.3, respectively. ↑ denotes the higher value, the better.

Table 5.6 shows the complex word identification performance of the two CCD modules on CWIG3G2. Att-Cls outperforms LS-CCD in recall, precision, and the F1 score. Tables 5.5 demonstrates the performance of GRS with different CCD modules on ASSET [33] and Newsela [16] test sets. On both datasets, the GRS model using Att-Cls has higher deletion and addition scores compared to the GRS model using LS-CCD. The overall SARI score is considerably higher when using Att-Cls on ASSET.

5.9 Simplification Search Analysis

GRS is an interpretable unsupervised simplification method in which we can trace the simplification process. That is, we know which edit operation is applied on a given complex sentence in each iteration. Table 5.7 demonstrates how many simplification iterations were

Model	Iterations/Sent (all-Operations)	PA-iterations	DL-iterations	
			RM	EX
Newsela				
GRS: PA	4.72	4.72	–	–
GRS: DL	0.79	–	0.46	0.33
GRS: PA + DL	4.40	3.72	0.37	0.31
ASSET				
GRS: PA	4.18	4.18	–	–
GRS: DL	1.05	–	0.54	0.51
GRS: PA + DL	3.79	2.94	0.39	0.45

Table 5.7 Analysis of edit operation used during simplification search, showing the average number of simplification iterations of GRS and the average share of each edit operation. PA and DL refer to paraphrasing and deletion, respectively. RM and EX refer to the removal and extraction sub-operations of the deletion operation.

needed to simplify a complex sentence, on average, in the Newsela and ASSET datasets. We also show the average frequency of each operation to simplify a given sentence.

Table 5.7 illustrates that when both edit operations are allowed (GRS:PA+DL), almost four simplification iterations are applied to a sentence, and paraphrasing is generally more common than deletion.

Chapter 6

Conclusions and Future Work

In this work, we propose GRS, an unsupervised, interpretable, and controllable approach for sentence simplification. We observe that sentence simplification approaches can be categorized into two generative and revision-based approaches. Each of these methods has its advantages. Edit-based methods provide more interpretability and controllability but are restricted by some explicit edit operations. These explicit edit operations are pre-defined; however, the generative models learn implicit edit operations from data. Generative are less interpretable and controllable but can perform more complex substitutions using neural components.

Text simplification approaches can also be categorized into two supervised and unsupervised methods. Supervised methods use complex-simple sentence pairs; however, unsupervised methods do not use such data.

We propose GRS, an unsupervised method that combines generative and revision-based methods and has the advantage of both methods.

We provide the following contributions and insights:

- We propose a method that simplifies a complex sentence using two pre-defined operations: paraphrasing and deletion.
- We design a hybrid text simplification system that lies between generative and revision-based methods. We integrate a generative paraphrasing model with a lexically-constrained decoding technique into a revisions-based framework.
- GRS system is interpretable since it provides a simplification path leading to the final sentence (Figure 4.2). We find that the deletion operation is more effective on the Newsela dataset; however, on the ASSET dataset, the paraphrasing operation is more effective. These insights can be obtained from our interpretable model without analyzing manually.

- The GRS system gives control over various factors of the simplification process. Table 5.4 illustrates the controllability of our model. Having controllability can be beneficial for generating simplified sentences for different sets of target audiences with different needs.
- We empirically show that our model outperforms all unsupervised models on the Newsela dataset. Also, GRS reduced the gap between generative and revision-based unsupervised models on the ASSET dataset.
- We propose an unsupervised method for finding complex words in a sentence. The proposed complex component detector applies a novel technique to interpret the attention layers of a transformer model. The ablation study on the proposed complex component detector (Table 5.5) demonstrates that this module is essential for obtaining decent simplification results. Also, we have evaluated the complex component detector on the complex word identification (CWI) task (Table 5.6).
- We release the open-source implementation of the GRS model at https://github.com/imohammad12/GRS.
- We also release all the fine-tuned models that we have used in the GRS. The paraphrasing model, the complex-simple classifier, and the linguistic acceptability classifier are uploaded at http://huggingface.co/imohammad12. The online demo of these models is also available at the provided link.

6.1 Future Work: Unsupervised Generative Text Simplification with Deep Reinforcement Learning

Moving forward, we think there are multiple ways to expand on our work. One of the most promising directions is to apply reinforcement learning (RL) to the unsupervised text simplification problem using the unsupervised components proposed in this thesis. In fact, the GRS unsupervised score function can be used as an unsupervised reward in a reinforcement learning framework. In the following, we will discuss this idea in more detail.

6.1.1 The General Idea

Reinforcement Learning (RL) is about making decisions optimally. When a machine learns to play a game such as chess using RL, it learns the optimal behavior in an environment to gain maximum reward. In the chess example, the reward can be winning or losing, and

the environment can be the chess game playground and rules. Text generation is also about decisions the model makes while generating an output sentence. Consider a generative text simplification model. When the model generates a simple sentence given a complex sentence, in each decoding step, it makes a decision: the model selects a word between all possible words. The model may make good or bad decisions (i.e., generating appropriate or not appropriate words). Like the chess game, the text simplification model should receive a reward (feedback) after generating the sentence. We know there are multiple ways to evaluate the simplicity of a generated text. For instance, we can use the SARI evaluation metric or the unsupervised GRS score function (Section 4.3). The evaluation metric can be considered the reward function we need to optimize in the text simplification problem using RL. A generative text simplification model can use RL algorithms to learn the optimal behavior that leads to the maximum simplicity reward. We can train the generative text simplification model without supervised data using an unsupervised reward function, such as the GRS score function. The general idea would be to obtain an unsupervised generative text simplification method by training a model with RL algorithms and using the unsupervised GRS score function; however, there are some obstacles to obtaining this goal. In the following, we will discuss the idea and the obstacles in more detail and provide some possible solutions.

How an Reinforcement Learning problem is defined?

In reinforcement learning problems, we have an agent that interacts with an environment. In each step, the agent takes action according to the current state and then goes to another state. Based on the final state that the agent ends up in, it receives a reward. If the final state is a decent state, the agent receives positive feedback (high rewards); otherwise, it gets negative feedback (low rewards). At first, the agent selects actions based on a random policy; gradually, it improves the policy to maximize the reward. The final goal is to find the optimal policy that maximizes the final reward.

How to formalize text generation as a Reinforcement Learning problem?

Text generation can be considered as a reinforcement learning problem in which the model is the agents and the selected words in each step are the actions that the agent takes. In step t, the state is defined based on the given input sentence (X) and the generated sentence so far ($y_{1:t-1}$) . The model (agent) selects a word $\hat{y}_t \in V$ (V is the vocabulary) based on policy $P_{RL}\left(\hat{y}_t \mid \hat{y}_{1:t-1}, X\right)$. When the output sentence is completed (final state), the score of the output sentence is calculated (reward). The model may also receive some rewards during generating a sentence.

In general, it is not practically possible to use reinforcement learning in text generation since the space of actions in each step is very large (i.e. the large vocabularies). Hence,

instead of using a random policy, with which the agent should explore a lot to learn how to maximize the reward, we should start with a trained policy. Previous work that used reinforcement learning for text simplification [17, 66] first trained the model on complex-simple sentence pairs in a supervised way (using cross entropy loss), and then improved the pre-trained model by using reinforcement learning. However, we do not want to use complex-simple sentence pairs in the pre-training step; therefore, we can use a pretrained paraphrasing model fine-tuned on a paraphrasing dataset (described in Section 4.2.2).

6.1.2 RL for Text Generation

Reinforcement learning (RL) has been applied to various NLP tasks in recent years. Ranzato et al. [67] used the REINFORCE algorithm [68] to address two main problems of Seq2Seq generative models, trained with regular supervised methods: 1) Exposure Bias which means that during training, the model is only exposed to training data distribution, as opposed to the inference phase in which the words are drawn from the model distribution. 2) Loss-Evaluation Mismatch which related to the problem that the training loss of supervised methods (like cross entropy loss) is word-level while inference evaluation metrics like BLEU [69] and SARI [70] are sequence-level metrics. Using reinforcement learning algorithms, BLEU is directly optimized for translation [67, 71], ROGUE is optimized for summarizing [67, 72, 73] , and SARI is optimized for text simplification [17].

6.1.3 Shortcoming of Previous Work on Text Simplification with RL: Supervised Data

Previous work on text generation and simplification that used RL in their systems utilized supervised data in different steps. Most methods trained a base model on supervised data (with cross-entropy loss function) and then used RL algorithms to improve the performance of the base models. The reward function used in their RL algorithms also utilized supervised data. Zhang and Lapata [17] and Nakamachi et al. [66] are the only two works that used RL for text simplification. They first trained a Seq2Seq model on a large amount of aligned simplification data and then fine-tuned the base model with RL algorithms to improve performance and text diversity. The reward function of their RL algorithm needs supervised data because the rewards were based on the SARI score, and calculating SARI needs reference sentences. For having an unsupervised method, we should:

1. have a base model that has not been trained on supervised data.
2. use an unsupervised reward function.

6.1.4 Possible Solution for Unsupervised Generative Text Simplification with RL

Ziegler et al. [74] used pretrained language models (e.g., Bert) as the base model (before the reinforcement learning step). As explained in Section 2.3, unsupervised large corpora are used for training these language models. Following Ziegler et al. [74], we can start with a pretrained language model, fine-tuned on paraphrasing data to obtain an unsupervised base model, and then use an RL algorithm (e.g., PPO [75] algorithm) to optimize a simplification reward. The pre-trained language model can be the paraphrasing model used in the GRS paraphrasing operation (Section 4.2.2). By optimizing a simplification reward with reinforcement learning, we will update (fine-tune) the paraphrasing model to simplify the sentences instead of just paraphrasing. Instead of using the SARI score as the simplification reward, which needs complex-simple sentence pairs, we can use the GRS unsupervised score function (Section 4.3) as the unsupervised simplification reward.

Although our method will utilize a powerful language model fine-tuned on paraphrase sentence pairs, it will not use any aligned simplification dataset. Gathering high-quality paraphrasing data is less complicated than collecting complex-simple sentence pairs.

Unsupervised Reward

The challenging part of the text simplification problem is to make a sentence simple while preserving the original meaning. In fact, there is a trade-off between simplicity and meaning preservation. The unsupervised simplification reward reflects how a generated sentence is suitable according to the trade-off between simplicity and meaning preservation. We want to learn a policy that can maximize this reward by fine-tuning the paraphrasing model using reinforcement learning. Following the structure used in the GRS score function (Section 4.3), the unsupervised reward can be comprised of meaning preservation, text simplicity, and linguistic acceptability models. No complex-simple sentence pairs will be used in the reward function, so the reward is unsupervised in the context of text simplification. Following [74], we can add a KL-divergence penalty to the calculated reward to prevent the learning policy from moving too far from the original paraphrasing model. This penalty term will help the learning policy to remain faithful to what it has learned in the pre-training phase. Using KL-divergence in reward, the proposed model will generate sentences similar to the original paraphrasing model while trying to use paraphrase fragments that are simpler to increase the simplicity score. The meaning preservation, simplicity, and linguistic acceptability modules are explained in Section 4.3.

References

[1] I. Sutskever, O. Vinyals, Q. V. Le, Sequence to sequence learning with neural networks, in: Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence, K. Weinberger (Eds.), Advances in Neural Information Processing Systems, volume 27, Curran Associates, Inc., 2014. URL: https://proceedings.neurips.cc/paper/2014/file/a14ac55a4f27472c5d894ec1c3c743d2-Paper.pdf.

[2] T. Tracey, Language translation with rnns, https://towardsdatascience.com/language-translation-with-rnns-d84d43b40571 (2018).

[3] T. Luong, H. Pham, C. D. Manning, Effective approaches to attention-based neural machine translation, in: Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing, Association for Computational Linguistics, Lisbon, Portugal, 2015, pp. 1412–1421. URL: https://aclanthology.org/D15-1166. doi:10.18653/v1/D15-1166.

[4] D. Bahdanau, K. Cho, Y. Bengio, Neural machine translation by jointly learning to align and translate, arXiv preprint arXiv:1409.0473 (2014).

[5] K. Doshi, Language translation with rnns, https://towardsdatascience.com/transformers-explained-visually-part-1-overview-of-functionality-95a6dd460452 (2020).

[6] J. Alammar, The illustrated transformer, http://jalammar.github.io/illustrated-transformer/ (2020).

[7] J. Alammar, The illustrated bert, elmo, and co. (how nlp cracked transfer learning), http://jalammar.github.io/illustrated-bert/ (2020).

[8] J. Devlin, M.-W. Chang, K. Lee, K. Toutanova, BERT: Pre-training of deep bidirectional transformers for language understanding, in: 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), Association for Computational Linguistics, Minneapolis, Minnesota, 2019, pp. 4171–4186. URL: https://aclanthology.org/N19-1423. doi:10.18653/v1/N19-1423.

[9] J. Vig, A multiscale visualization of attention in the transformer model, in: Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics: System Demonstrations, Association for Computational Linguistics, Florence, Italy, 2019, pp. 37–42. URL: https://aclanthology.org/P19-3007. doi:10.18653/v1/P19-3007.

[10] J. E. Hu, R. Rudinger, M. Post, B. Van Durme, Parabank: Monolingual bitext generation and sentential paraphrasing via lexically-constrained neural machine translation, in: Proceedings of the AAAI Conference on Artificial Intelligence, volume 33, 2019, pp. 6521–6528.

[11] D. Kumar, L. Mou, L. Golab, O. Vechtomova, Iterative edit-based unsupervised sentence simplification, in: Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, Association for Computational Linguistics, Online, 2020, pp. 7918–7928. URL: https://aclanthology.org/2020.acl-main.707. doi:10.18653/v1/2020.acl-main.707.

[12] R. Evans, C. Orăsan, I. Dornescu, An evaluation of syntactic simplification rules for people with autism, in: Proceedings of the 3rd Workshop on Predicting and Improving Text Readability for Target Reader Populations (PITR), Association for Computational Linguistics, Gothenburg, Sweden, 2014, pp. 131–140. URL: https://aclanthology.org/W14-1215. doi:10.3115/v1/W14-1215.

[13] B. B. Klebanov, K. Knight, D. Marcu, Text simplification for information-seeking applications, in: OTM Confederated International Conferences" On the Move to Meaningful Internet Systems", Springer, 2004, pp. 735–747.

[14] R. Chandrasekar, C. Doran, B. Srinivas, Motivations and methods for text simplification, in: COLING 1996 Volume 2: The 16th International Conference on Computational Linguistics, 1996. URL: https://aclanthology.org/C96-2183.

[15] S. Štajner, M. Popovic, Can text simplification help machine translation?, in: Proceedings of the 19th Annual Conference of the European Association for Machine Translation, 2016, pp. 230–242. URL: https://aclanthology.org/W16-3411.

[16] W. Xu, C. Callison-Burch, C. Napoles, Problems in current text simplification research: New data can help, Transactions of the Association for Computational Linguistics 3 (2015) 283–297. URL: https://aclanthology.org/Q15-1021. doi:10.1162/tacl_a_00139.

[17] X. Zhang, M. Lapata, Sentence simplification with deep reinforcement learning, in: Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing, Association for Computational Linguistics, Copenhagen, Denmark, 2017, pp. 584–594. URL: https://aclanthology.org/D17-1062. doi:10.18653/v1/D17-1062.

[18] H. Guo, R. Pasunuru, M. Bansal, Dynamic multi-level multi-task learning for sentence simplification, in: Proceedings of the 27th International Conference on Computational Linguistics, Association for Computational Linguistics, Santa Fe, New Mexico, USA, 2018, pp. 462–476. URL: https://aclanthology.org/C18-1039.

[19] R. Kriz, J. Sedoc, M. Apidianaki, C. Zheng, G. Kumar, E. Miltsakaki, C. Callison-Burch, Complexity-weighted loss and diverse reranking for sentence simplification, in: 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), 2019, pp. 3137–3147.

[20] S. Surya, A. Mishra, A. Laha, P. Jain, K. Sankaranarayanan, Unsupervised neural text simplification, in: Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics, Association for Computational Linguistics, Florence, Italy, 2019, pp. 2058–2068. URL: https://aclanthology.org/P19-1198. doi:10.18653/v1/P19-1198.

[21] L. Martin, É. de la Clergerie, B. Sagot, A. Bordes, Controllable sentence simplification, in: Proceedings of the 12th Language Resources and Evaluation Conference, Eu-

ropean Language Resources Association, Marseille, France, 2020, pp. 4689–4698. URL: https://aclanthology.org/2020.lrec-1.577.

[22] F. Alva-Manchego, J. Bingel, G. Paetzold, C. Scarton, L. Specia, Learning how to simplify from explicit labeling of complex-simplified text pairs, in: Proceedings of the Eighth International Joint Conference on Natural Language Processing (Volume 1: Long Papers), Asian Federation of Natural Language Processing, Taipei, Taiwan, 2017, pp. 295–305. URL: https://aclanthology.org/I17-1030.

[23] Y. Dong, Z. Li, M. Rezagholizadeh, J. C. K. Cheung, EditNTS: An neural programmer-interpreter model for sentence simplification through explicit editing, in: Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics, Association for Computational Linguistics, Florence, Italy, 2019, pp. 3393–3402. URL: https://aclanthology.org/P19-1331. doi:10.18653/v1/P19-1331.

[24] S. Agrawal, W. Xu, M. Carpuat, A non-autoregressive edit-based approach to controllable text simplification, in: Findings of the Association for Computational Linguistics: ACL-IJCNLP 2021, Association for Computational Linguistics, Online, 2021, pp. 3757–3769. URL: https://aclanthology.org/2021.findings-acl.330. doi:10.18653/v1/2021.findings-acl.330.

[25] L. Martin, A. Fan, É. de la Clergerie, A. Bordes, B. Sagot, Muss: Multilingual unsupervised sentence simplification by mining paraphrases, arXiv preprint arXiv:2005.00352 (2020).

[26] M. Maddela, F. Alva-Manchego, W. Xu, Controllable text simplification with explicit paraphrasing, in: Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Association for Computational Linguistics, Online, 2021, pp. 3536–3553. URL: https://aclanthology.org/2021.naacl-main.277. doi:10.18653/v1/2021.naacl-main.277.

[27] Y. Zhao, L. Chen, Z. Chen, K. Yu, Semi-supervised text simplification with back-translation and asymmetric denoising autoencoders, Proceedings of the AAAI Conference on Artificial Intelligence 34 (2020) 9668–9675. URL: https://ojs.aaai.org/index.php/AAAI/article/view/6515. doi:10.1609/aaai.v34i05.6515.

[28] M. Lewis, Y. Liu, N. Goyal, M. Ghazvininejad, A. Mohamed, O. Levy, V. Stoyanov, L. Zettlemoyer, BART: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension, in: Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, Association for Computational Linguistics, Online, 2020, pp. 7871–7880. URL: https://aclanthology.org/2020.acl-main.703. doi:10.18653/v1/2020.acl-main.703.

[29] C. Hokamp, Q. Liu, Lexically constrained decoding for sequence generation using grid beam search, in: Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), Association for Computational Linguistics, Vancouver, Canada, 2017, pp. 1535–1546. URL: https://aclanthology.org/P17-1141. doi:10.18653/v1/P17-1141.

[30] M. Post, D. Vilar, Fast lexically constrained decoding with dynamic beam allocation for neural machine translation, in: Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies,

Volume 1 (Long Papers), Association for Computational Linguistics, New Orleans, Louisiana, 2018, pp. 1314–1324. URL: https://aclanthology.org/N18-1119. doi:10.18653/v1/N18-1119.

[31] J. E. Hu, H. Khayrallah, R. Culkin, P. Xia, T. Chen, M. Post, B. Van Durme, Improved lexically constrained decoding for translation and monolingual rewriting, in: 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), Association for Computational Linguistics, Minneapolis, Minnesota, 2019, pp. 839–850. URL: https://aclanthology.org/N19-1090. doi:10.18653/v1/N19-1090.

[32] M. Dehghan, D. Kumar, L. Golab, GRS: Combining generation and revision in unsupervised sentence simplification, in: Findings of the Association for Computational Linguistics: ACL 2022, Association for Computational Linguistics, Dublin, Ireland, 2022, pp. 949–960. URL: https://aclanthology.org/2022.findings-acl.77. doi:10.18653/v1/2022.findings-acl.77.

[33] F. Alva-Manchego, L. Martin, A. Bordes, C. Scarton, B. Sagot, L. Specia, ASSET: A dataset for tuning and evaluation of sentence simplification models with multiple rewriting transformations, in: Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, Association for Computational Linguistics, Online, 2020, pp. 4668–4679. URL: https://aclanthology.org/2020.acl-main.424. doi:10.18653/v1/2020.acl-main.424.

[34] W. Xu, C. Napoles, E. Pavlick, Q. Chen, C. Callison-Burch, Optimizing statistical machine translation for text simplification, Transactions of the Association for Computational Linguistics 4 (2016) 401–415. URL: https://aclanthology.org/Q16-1029. doi:10.1162/tacl_a_00107.

[35] T. Mikolov, M. Karafiát, L. Burget, J. Cernockỳ, S. Khudanpur, Recurrent neural network based language model., in: Interspeech, volume 2, Makuhari, 2010, pp. 1045–1048.

[36] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. u. Kaiser, I. Polosukhin, Attention is all you need, in: I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, R. Garnett (Eds.), Advances in Neural Information Processing Systems, volume 30, Curran Associates, Inc., 2017. URL: https://proceedings.neurips.cc/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf.

[37] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, I. Polosukhin, Attention is all you need, in: Advances in neural information processing systems, 2017, pp. 5998–6008.

[38] O. Bojar, C. Buck, C. Federmann, B. Haddow, P. Koehn, J. Leveling, C. Monz, P. Pecina, M. Post, H. Saint-Amand, R. Soricut, L. Specia, A. s. Tamchyna, Findings of the 2014 workshop on statistical machine translation, in: Proceedings of the Ninth Workshop on Statistical Machine Translation, Association for Computational Linguistics, Baltimore, Maryland, USA, 2014, pp. 12–58. URL: http://www.aclweb.org/anthology/W/W14/W14-3302.

[39] R. Chandrasekar, B. Srinivas, Automatic induction of rules for text simplification1revised version of the article originally published in knowledge-based computer systems: Research and applications. (eds k.s.r. anjaneyulu, m. sasikumar and s. ramani) narosa publishing house,

new delhi, 1997.1, Knowledge-Based Systems 10 (1997) 183–190. URL: https://www.sciencedirect.com/science/article/pii/S0950705197000294. doi:https://doi.org/10.1016/S0950-7051(97)00029-4.

[40] J. Carroll, G. Minnen, Y. Canning, S. Devlin, J. Tait, Practical simplification of english newspaper text to assist aphasic readers, in: Proceedings of the AAAI-98 Workshop on Integrating Artificial Intelligence and Assistive Technology, Citeseer, 1998, pp. 7–10.

[41] D. Vickrey, D. Koller, Sentence simplification for semantic role labeling, in: Proceedings of ACL-08: HLT, Association for Computational Linguistics, Columbus, Ohio, 2008, pp. 344–352. URL: https://aclanthology.org/P08-1040.

[42] W. Coster, D. Kauchak, Learning to simplify sentences using Wikipedia, in: Proceedings of the Workshop on Monolingual Text-To-Text Generation, Association for Computational Linguistics, Portland, Oregon, 2011, pp. 1–9. URL: https://aclanthology.org/W11-1601.

[43] S. Wubben, A. van den Bosch, E. Krahmer, Sentence simplification by monolingual machine translation, in: Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), Association for Computational Linguistics, Jeju Island, Korea, 2012, pp. 1015–1024. URL: https://aclanthology.org/P12-1107.

[44] Z. Zhu, D. Bernhard, I. Gurevych, A monolingual tree-based translation model for sentence simplification, in: Proceedings of the 23rd International Conference on Computational Linguistics (Coling 2010), Coling 2010 Organizing Committee, Beijing, China, 2010, pp. 1353–1361. URL: https://aclanthology.org/C10-1152.

[45] S. Narayan, C. Gardent, Hybrid simplification using deep semantics and machine translation, in: Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), Association for Computational Linguistics, Baltimore, Maryland, 2014, pp. 435–445. URL: https://aclanthology.org/P14-1041. doi:10.3115/v1/P14-1041.

[46] I. Sutskever, O. Vinyals, Q. Le, Sequence to sequence learning with neural networks, Advances in NIPS (2014). URL: http://papers.nips.cc/paper/5346-sequence-to-sequence-learning-with-neural-networks.pdf.

[47] S. Nisioi, S. Štajner, S. P. Ponzetto, L. P. Dinu, Exploring neural text simplification models, in: Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers), Association for Computational Linguistics, Vancouver, Canada, 2017, pp. 85–91. URL: https://aclanthology.org/P17-2014. doi:10.18653/v1/P17-2014.

[48] S. Zhao, R. Meng, D. He, A. Saptono, B. Parmanto, Integrating transformer and paraphrase rules for sentence simplification, in: Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing, Association for Computational Linguistics, Brussels, Belgium, 2018, pp. 3164–3173. URL: https://aclanthology.org/D18-1355. doi:10.18653/v1/D18-1355.

[49] C. Scarton, L. Specia, Learning simplifications for specific target audiences, in: Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers), Association for Computational Linguistics, Melbourne, Australia, 2018, pp. 712–718. URL: https://aclanthology.org/P18-2113. doi:10.18653/v1/P18-2113.

[50] D. Nishihara, T. Kajiwara, Y. Arase, Controllable text simplification with lexical constraint loss, in: Proceedings of the 57th Annual Meeting of the Association for Computational

Linguistics: Student Research Workshop, Association for Computational Linguistics, Florence, Italy, 2019, pp. 260–266. URL: https://aclanthology.org/P19-2036. doi:10.18653/v1/P19-2036.

[51] K. Omelianchuk, V. Raheja, O. Skurzhanskyi, Text Simplification by Tagging, in: Proceedings of the 16th Workshop on Innovative Use of NLP for Building Educational Applications, Association for Computational Linguistics, Online, 2021, pp. 11–25. URL: https://aclanthology.org/2021.bea-1.2.

[52] S. Narayan, C. Gardent, Unsupervised sentence simplification using deep semantics, in: Proceedings of the 9th International Natural Language Generation conference (INLG), 2016, pp. 111–120. URL: https://www.aclweb.org/anthology/W16-6620. doi:10.18653/v1/W16-6620.

[53] J. E. Hu, A. Singh, N. Holzenberger, M. Post, B. Van Durme, Large-scale, diverse, paraphrastic bitexts via sampling and clustering, in: Proceedings of the 23rd Conference on Computational Natural Language Learning (CoNLL), Association for Computational Linguistics, Hong Kong, China, 2019, pp. 44–54. URL: https://aclanthology.org/K19-1005. doi:10.18653/v1/K19-1005.

[54] M. Reid, V. Zhong, LEWIS: Levenshtein editing for unsupervised text style transfer, in: Findings of the Association for Computational Linguistics: ACL-IJCNLP 2021, Association for Computational Linguistics, Online, 2021, pp. 3932–3944. URL: https://aclanthology.org/2021.findings-acl.344. doi:10.18653/v1/2021.findings-acl.344.

[55] P. He, X. Liu, J. Gao, W. Chen, Deberta: Decoding-enhanced bert with disentangled attention, in: International Conference on Learning Representations, 2020.

[56] J. Devlin, M.-W. Chang, K. Lee, K. Toutanova, BERT: Pre-training of deep bidirectional transformers for language understanding, in: 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), Association for Computational Linguistics, Minneapolis, Minnesota, 2019, pp. 4171–4186. URL: https://aclanthology.org/N19-1423. doi:10.18653/v1/N19-1423.

[57] N. Reimers, I. Gurevych, Sentence-bert: Sentence embeddings using siamese bert-networks, in: Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing, Association for Computational Linguistics, 2019. URL: https://arxiv.org/abs/1908.10084.

[58] A. Warstadt, A. Singh, S. R. Bowman, Neural network acceptability judgments, Transactions of the Association for Computational Linguistics 7 (2019) 625–641. URL: https://aclanthology.org/Q19-1040. doi:10.1162/tacl_a_00290.

[59] T. Wolf, L. Debut, V. Sanh, J. Chaumond, C. Delangue, A. Moi, P. Cistac, T. Rault, R. Louf, M. Funtowicz, J. Davison, S. Shleifer, P. von Platen, C. Ma, Y. Jernite, J. Plu, C. Xu, T. L. Scao, S. Gugger, M. Drame, Q. Lhoest, A. M. Rush, Transformers: State-of-the-art natural language processing, in: Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations, Association for Computational Linguistics, Online, 2020, pp. 38–45. URL: https://www.aclweb.org/anthology/2020.emnlp-demos.6.

[60] C. Jiang, M. Maddela, W. Lan, Y. Zhong, W. Xu, Neural CRF model for sentence align-

ment in text simplification, in: Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, Association for Computational Linguistics, Online, 2020, pp. 7943–7960. URL: https://aclanthology.org/2020.acl-main.709. doi:10.18653/v1/2020.acl-main.709.

[61] I. Loshchilov, F. Hutter, Decoupled weight decay regularization, in: International Conference on Learning Representations, 2019. URL: https://openreview.net/forum?id=Bkg6RiCqY7.

[62] F. Alva-Manchego, L. Martin, C. Scarton, L. Specia, EASSE: Easier automatic sentence simplification evaluation, in: Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP): System Demonstrations, Association for Computational Linguistics, Hong Kong, China, 2019, pp. 49–54. URL: https://aclanthology.org/D19-3009. doi:10.18653/v1/D19-3009.

[63] K. Papineni, S. Roukos, T. Ward, W.-J. Zhu, Bleu: a method for automatic evaluation of machine translation, in: Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics, Association for Computational Linguistics, Philadelphia, Pennsylvania, USA, 2002, pp. 311–318. URL: https://aclanthology.org/P02-1040. doi:10.3115/1073083.1073135.

[64] E. Sulem, O. Abend, A. Rappoport, Semantic structural evaluation for text simplification, in: Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers), Association for Computational Linguistics, New Orleans, Louisiana, 2018, pp. 685–696. URL: https://aclanthology.org/N18-1063. doi:10.18653/v1/N18-1063.

[65] S. M. Yimam, S. Štajner, M. Riedl, C. Biemann, CWIG3G2 - complex word identification task across three text genres and two user groups, in: Proceedings of the Eighth International Joint Conference on Natural Language Processing (Volume 2: Short Papers), Asian Federation of Natural Language Processing, Taipei, Taiwan, 2017, pp. 401–407. URL: https://aclanthology.org/I17-2068.

[66] A. Nakamachi, T. Kajiwara, Y. Arase, Text simplification with reinforcement learning using supervised rewards on grammaticality, meaning preservation, and simplicity, in: Proceedings of the 1st Conference of the Asia-Pacific Chapter of the Association for Computational Linguistics and the 10th International Joint Conference on Natural Language Processing: Student Research Workshop, Association for Computational Linguistics, Suzhou, China, 2020, pp. 153–159. URL: https://aclanthology.org/2020.aacl-srw.22.

[67] M. Ranzato, S. Chopra, M. Auli, W. Zaremba, Sequence level training with recurrent neural networks, arXiv preprint arXiv:1511.06732 (2015).

[68] R. J. Williams, Simple statistical gradient-following algorithms for connectionist reinforcement learning, Machine learning 8 (1992) 229–256.

[69] K. Papineni, S. Roukos, T. Ward, W.-J. Zhu, Bleu: a method for automatic evaluation of machine translation, in: Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics, Association for Computational Linguistics, Philadelphia,

Pennsylvania, USA, 2002, pp. 311–318. URL: https://aclanthology.org/P02-1040. doi:10.3115/1073083.1073135.

[70] W. Xu, C. Napoles, E. Pavlick, Q. Chen, C. Callison-Burch, Optimizing statistical machine translation for text simplification, Transactions of the Association for Computational Linguistics 4 (2016) 401–415. URL: https://aclanthology.org/Q16-1029. doi:10.1162/tacl_a_00107.

[71] Y. Wu, M. Schuster, Z. Chen, Q. V. Le, M. Norouzi, W. Macherey, M. Krikun, Y. Cao, Q. Gao, K. Macherey, et al., Google's neural machine translation system: Bridging the gap between human and machine translation, arXiv preprint arXiv:1609.08144 (2016).

[72] Y. Wu, B. Hu, Learning to extract coherent summary via deep reinforcement learning, in: Thirty-Second AAAI Conference on Artificial Intelligence, 2018.

[73] Y. Gao, C. M. Meyer, M. Mesgar, I. Gurevych, Reward learning for efficient reinforcement learning in extractive document summarisation, arXiv preprint arXiv:1907.12894 (2019).

[74] D. M. Ziegler, N. Stiennon, J. Wu, T. B. Brown, A. Radford, D. Amodei, P. Christiano, G. Irving, Fine-tuning language models from human preferences, arXiv preprint arXiv:1909.08593 (2019).

[75] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, O. Klimov, Proximal policy optimization algorithms, arXiv preprint arXiv:1707.06347 (2017).