ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

# Text Detoxification using Pre-Trained Language Models and Plug-and-Play Generation Methods

Master Thesis

Samuel Pullely

May 2, 2022

Advisors: Prof. Dr. Ryan Cotterell, Clara Meister, Béni Egressy
Department of Computer Science, ETH Zürich

# Acknowledgement

I would like to thank Prof. Dr. Ryan Cotterell for allowing me to write my thesis in his group and for giving his lecture on natural language processing which sparked my interest in the field and through which I learned the core concepts.

I would like to especially thank my two supervisors Clara Meister and Béni Egressy. I highly appreciate the time they took out of their busy schedules to have weekly meetings with me during which they provided guidance, helpful tips and interesting insights that helped me work on my thesis.

Finally, I would like to thank Prof. Dr. Peter Bühlmann for enabling me to write write my thesis outside of the Department of Mathematics.

**Abstract**

Many state-of-the-art natural language generation (NLG) systems use large pre-trained language models as their base model and then fine-tune on a downstream task. However, fine-tuning on a new task often requires altering the model architecture, additional data and involves the significant computational cost of re-training. Moreover, it is hard to control attributes of the generated text, such as topic or sentiment. Therefore, it is desirable to use large pre-trained language models out-of-the-box with methods for controlled text generation that do not require any changes to the pre-trained model or additional smaller models that need to be trained. Such methods are referred to as *plug-and-play* methods. However, the application of plug-and-play methods has been limited to a small subset of language generation tasks. This thesis explores plug-and-play methods compatible with large pre-trained language models for performing text detoxification, i.e., rewriting a toxic text into a non-toxic text while preserving its meaning. To that end, we focus on methods that modify the decoding process, i.e., the process of generating text from a probabilistic model, by adjusting the probability distribution over tokens in the language model's vocabulary. More precisely, we extend two existing methods. The first method uses word embeddings in order to compute cosine similarities and add a shift to the language model's probability distribution. The second method is based on zero-shot learning with task descriptions that prompts the language model to produce a probability distribution that can then be used to guide generation towards a desired attribute. Our results show that our proposed plug-and-play decoding methods are effective in performing text detoxification. However, they are still far behind the current state-of-the-art models for text detoxification that use learning.

# Contents

Chapter 1

# Introduction

With the latest improvements in natural language processing (NLP), natural language generation (NLG) systems are now able to generate coherent and fluent text. Many state-of-the-art NLG systems use large pre-trained language models based on the transformer architecture [53], such as GPT-2 [44] or T5 [45], as their base model and subsequently fine-tune on a downstream task. However, fine-tuning on a new task often requires altering the model architecture, additional data and involves the significant computational cost of re-training since the best performing pre-trained models contain millions or even billions of parameters. Moreover, it is hard to control attributes of the generated text, such as topic or sentiment, since these models are probabilistic, especially for undirected language generation tasks without a lot of constraints on the output [41], such as story generation [13]. Recently, it has become increasingly common to use large pre-trained language models out-of-the-box with methods for controlled text generation that do not require any changes to the pre-trained model [41]. While some of these methods [6, 27] still require training an additional albeit smaller model on attribute-specific data, others, referred to as *plug-and-play* methods [41], work without additional training. However, the application of plug-and-play methods has been limited to a small subset of language generation tasks, such as enforcing hard constraints on word appearance in keyword-to-phrase or story generation [41]. This thesis explores plug-and-play methods compatible with large pre-trained language models for performing text detoxification [5], i.e., rewriting a toxic text into a non-toxic text while preserving its meaning as much as possible. To that end, we focus on methods that modify the decoding process, i.e., the process of generating text from a probabilistic model, by adjusting the probability distribution over tokens in the language model's vocabulary. More concretely, we extend the methods in [41] and [47], making them applicable to text detoxification. Through these experiments, this thesis aims to present a set of plug-and-play methods that are

valuable for practical implementations of NLG systems while also gaining a better understanding of the set of tasks for which pre-trained language models can be employed straight out-of-the-box.

Chapter 2

# Background Information

## 2.1 Probabilistic Language Generators

State-of-the-art language models are trained on huge text corpora and are capable of approximating the distribution of natural language sufficiently well to generate coherent and fluent text. This ability, referred to as natural language generation (NLG), has many applications in natural language processing (NLP), such as prompt continuation and machine translation.

These language models model a probability distribution over strings $\mathbf{y}$ in an output space $\mathcal{Y}$. Additionally, this probability distribution can be conditioned on an input $\mathbf{x}$ in an input space $\mathcal{X}$ for tasks where the output $\mathbf{y}$ is dependent on the input $\mathbf{x}$. This probability distribution is usually modeled by an encoder-decoder architecture using attention mechanisms [33, 3], most commonly the transformer [53]. If there is no conditioning on an input $\mathbf{x}$, a decoder-only architecture is used. In order to generate text from a probability distribution over a set of strings $\mathcal{Y}$, we can choose from a variety of decoding methods [60].

### 2.1.1 Tokenization

Since language models do not understand text in its raw form, we first need to convert the raw text into numbers. This process is called *tokenization*. There are several different approaches but all of them work by splitting text into tokens according to a set of rules where the objective is to find a meaningful representation for downstream tasks. For example, a simple way of tokenizing text is to split it by spaces into words. These tokens are converted to ids (numbers) through a look-up table and can then be fed to a model which learns a representation for each token. The set of all unique tokens used by a model is called *vocabulary*. Most tokenizers fall into one of the following three categories of tokenization algorithms: word-based, character-

based or subword-based. There are different trade-offs for each of these categories. For example, word-based tokenizers can lead to a large vocabulary which forces the model to have a large embedding matrix as the input and output layer. Moreover, words that are not contained in the vocabulary are assigned to a special unknown token. On the other hand, character-based tokenization is very simple and has a small vocabulary but makes it much harder for the model to learn meaningful input representations. Subword-based tokenization lies inbetween word-based and character-based tokenization and relies on the principle that frequently used words should not be seperated into smaller subwords and rare words should be split into meaningful subwords [12]. Most models obtaining state-of-the-art results in English use some kind of subword tokenization algorithm, such as WordPiece [48] or Byte-Pair Encoding [49].

### 2.1.2 Auto-Regressive Models

From section 2.1.2 to 2.2, we follow the structure in [60] and introduce some equations where we consider the case of encoder-decoder models with conditioning on an input $\mathbf{x} \in \mathcal{X}$. For decoder-only models, we can simply leave out the conditioning on the input $\mathbf{x}$ while the rest of the equation stays the same. We start by defining the output space $\mathcal{Y}$ as follows:

$$\mathcal{Y} := \{\textsc{bos} \circ \mathbf{v} \circ \textsc{eos} \mid \mathbf{v} \in \mathcal{V}^*\} \tag{2.1}$$

where $\mathcal{V}$ is the vocabulary, $\mathcal{V}^*$ its Kleene closure [26] and BOS and EOS are special tokens that mark the beginning and end of a token sequence, respectively. Hence, the output space $\mathcal{Y}$ represents the set of all finite token sequences $\mathbf{y}$, i.e., strings of text, that are derived from the vocabulary $\mathcal{V}$. Language models using an encoder-decoder architecture model a probability distribution $p_{\boldsymbol{\theta}}(\cdot \mid \mathbf{x})$ on the output space $\mathcal{Y}$ conditioned on the input $\mathbf{x}$ which does not have to be text but can be in another form, for example, an image.

We only consider auto-regressive models that are locally normalized in this work. Therefore, the probability of a sequence $\mathbf{y} = (y_0, y_1, \ldots, y_n) \in \mathcal{Y}$ can be factorized into the following expression:

$$p_{\boldsymbol{\theta}}(\mathbf{y} \mid \mathbf{x}) = \prod_{t=1}^{n} p_{\boldsymbol{\theta}}(y_t \mid \mathbf{x}, \mathbf{y}_{<t}) \tag{2.2}$$

where $\mathbf{y}_{<t} := (y_0, ..., y_{t-1})$, $\mathbf{y}_{<1} = y_0 := \textsc{bos}$, $y_n := \textsc{eos}$ and for $t > 0$, $p_{\boldsymbol{\theta}}(\cdot \mid \mathbf{x}, \mathbf{y}_{<t})$ defines a probability distribution over the extended vocabulary $\bar{\mathcal{V}} := \mathcal{V} \cup \{\textsc{eos}\}$. At each step, we can choose the next token given the input $\mathbf{x}$ as well as any previously generated tokens $\mathbf{y}_{<t}$ according to this probability distribution which can be represented as

$$p_{\boldsymbol{\theta}}(y_t = v_i \mid \mathbf{x}, \mathbf{y}_{<t}) = \pi(\mathbf{z}_t)_i \tag{2.3}$$

where $v_i$ is the $i$-th token in $\bar{\mathcal{V}}$, $\mathbf{z}_t = f_{\boldsymbol{\theta}}(\mathbf{y}_{<t}, \mathbf{x}) \in \mathbb{R}^{|\bar{\mathcal{V}}|}$ is the output of the language model and the function $\pi \colon \mathbb{R}^{|\bar{\mathcal{V}}|} \to \Delta^{|\bar{\mathcal{V}}|-1}$ projects the output $\mathbf{z}_t$ onto the $|\bar{\mathcal{V}}| - 1$-dimensional probability simplex. The softmax function is most often chosen as the projection $\pi$ and deep neural networks are usually used in order to parameterize language models $f_{\boldsymbol{\theta}}$ with trainable weights $\boldsymbol{\theta}$.

### 2.1.3 Training Objective

An optimization algorithm is used to train the language model, i.e., to minimize a loss function $\ell(\boldsymbol{\theta})$ over the model's parameters $\boldsymbol{\theta}$. Given the language model $p_{\boldsymbol{\theta}}$ and a sample pair $(\mathbf{x}, \mathbf{y})$ consisting of the input $\mathbf{x} \in \mathcal{X}$ and the output $\mathbf{y} \in \mathcal{Y}$, the following negative log-likelihood loss (NLL) is usually used to train the model:

$$\ell_{(\mathbf{x},\mathbf{y})}(\boldsymbol{\theta}) = -\frac{1}{|\mathbf{y}|} \log p_{\boldsymbol{\theta}}(\mathbf{y}|\mathbf{x}) = -\frac{1}{|\mathbf{y}|} \sum_{t=1}^{|\mathbf{y}|} \log p_{\boldsymbol{\theta}}(y_t|\mathbf{x}, \mathbf{y}_{<t}). \tag{2.4}$$

The lower the value of the loss function, the higher is the probability conditioned on the input $\mathbf{x}$ that the language model assigns to the token sequence $\mathbf{y}$. During training, the model learns to simultaneously minimize the loss function for many samples $(\mathbf{x}, \mathbf{y})$ and thus also learns to assign a high probability to them.

### 2.1.4 Decoding Problem

Finding the token sequence $\mathbf{y}^* \in \mathcal{Y}$ with the highest probability under the language model $p_{\boldsymbol{\theta}}$ is known as the decoding problem. It requires solving the following discrete maximization problem:

$$\mathbf{y}^\star = \operatorname*{argmax}_{\mathbf{y} \in \mathcal{Y}} \; p_{\boldsymbol{\theta}}(\mathbf{y} \mid \mathbf{x}). \tag{2.5}$$

where the solution $\mathbf{y}^*$ is referred to as the maximum a posteriori (MAP) solution. However, this problem is rarely directly optimized because the output space is too large. More precisely, if we only allow for token sequences of a maximum length $n$, then the space $\mathcal{Y}$ is exponentially large in $n$, otherwise, $\mathcal{Y}$ is even infinite. Moreover, language models parameterized by deep neural networks usually do not possess the Markov property, i.e., the next token in a sequence does not only depend on the previous token but on all previous tokens, which makes direct optimization computationally expensive. In addition, recent research has found the MAP solution $\mathbf{y}^*$ in many cases not to be high quality text or considered natural by humans [11]. Therefore, $\mathbf{y}^*$ is almost always approximated with the use of heuristic decoding strategies, such as beam search [34] or nucleus sampling [18]. A framework containing two components can be used to describe these decoding strategies: a *score*

*function* and a *decoding algorithm* [41]. Decoding algorithms are defined as the class of algorithms that decode token sequences according to a score function in an auto-regressive manner. Note that decoding algorithms can be either deterministic or stochastic. For each $t > 0$, a score function

$$\text{score}(\cdot \mid \mathbf{x}, \mathbf{y}_{<t}) : \bar{\mathcal{V}} \to \mathbb{R} \tag{2.6}$$

is defined as a map from a token in the model's extended vocabulary to a real number. Clearly, the score function is dependent on the input $\mathbf{x}$ and on previously generated tokens $\mathbf{y}_{<t}$. The standard choice for the score function is $\text{score}(\cdot \mid \mathbf{x}, \mathbf{y}_{<t}) = \log p_{\theta}(\cdot|\mathbf{x}, \mathbf{y}_{<t})$. In this work, we focus on plug-and-play methods that modify the score function in order to control attributes of generated text.

## 2.2 Transformer Architecture

Prior to the invention of the transformer architecture [53], the best performing models for the sequence modeling task used recurrent neural networks (RNNs). These models process a sequence by encoding its elements sequentially into hidden states that are dependent on the current element of the sequence and the previous hidden state, hence the term recurrent. However, processing elements in a sequential manner is sub-optimal and leads to problems, such as not being able to remember long-term dependencies due to vanishing gradients [40]. Some of these problems can be overcome by using attention mechanisms [4, 33], which allow RNNs to selectively attend to the hidden states computed from the sequence elements instead of only learning to encode what is relevant for the future. However, by removing the need for sequential processing and focusing completely on attention mechanisms, the transformer can be trained much more efficiently and on larger datasets. The original transformer [53] uses an encoder-decoder architecture where both the encoder and the decoder consist of stacked layers that process an entire sequence iteratively, i.e., one layer after another. Each layer contains an attention mechanism and a feed-forward neural network. With this architecture, the transformer is able to learn rich representations and has enabled state-of-art results for many tasks in NLP. In the next few sections, we present some transformer-based models that we use for our experiments in chapter 3.

### 2.2.1 GPT-2

GPT-2 [44] stands for generative pre-trained transformer 2 and is an auto-regressive decoder based on the transformer architecture with 1.5 billion parameters. The model is trained on a dataset that is constructed from text from 8 million websites that were scraped for content according to some

rules that emphasize diversity [37]. The simple language modeling objective of predicting the next word, given the previous words in a text, leads to GPT-2 being able to generate coherent and fluent text continuations given an input prompt. Because the dataset is very diverse, GPT-2 encounters examples of tasks other than language modeling during training, e.g., some websites contain translations from English to French. As a consequence, GPT-2 is also able to perform other tasks, such as summarization, translation and question answering, without being explicitly trained on them with domain-specific data, i.e., by simply providing a task description as an input prompt, the answer is generated as the continuation of the input prompt. However, this ability, referred to as *zero-shot learning* [44], does not yield state-of-the-art results for tasks that the model was not trained on but it suggests that it is beneficial to first use an unsupervised training objective before applying the model to these tasks [37].

### 2.2.2 BART

BART [30] stands for bi-directional and auto-regressive transformers and is a denoising autoencoder. It is implemented as a sequence-to-sequence model with a bi-directional encoder like BERT [8] and a left-to-right auto-regressive decoder like GPT-2 [44]. Text from books and Wikipedia is used for the training dataset. During pre-training, the input text is first corrupted with various noising methods, such as sentence permutation, document rotation, token or span masking and token deletion. This is in contrast with previous autoencoders that typically only use one specific type of noising method. The encoder learns a representation for the corrupted text and the decoder then tries to reconstruct the original input text, i.e., the training objective is to optimize the negative log-likelihood of the original input text. Given its auto-regressive decoder, BART can be fine-tuned for generation tasks. At the time of the release, BART held state-of-the-art results for several text generation tasks, such as question answering and summarization.

### 2.2.3 T5

T5 [45] stands for text-to-text transfer transformer and is an encoder-decoder model. Unlike BART and GPT-2, T5 uses a mixture of supervised and unsupervised tasks during pre-training, where each task uses a text-to-text format. For example, T5 can be used for summarization by prepending the prefix ”summarize:” to the text we want to summarize and similarly for translation, we can prepend the prefix ”translate English to German:” to the text we want to translate. The unsupervised objective is to generate replacements for masked spans in the input text, where each masked span is replaced by a sentinel token that is unique to the sequence (see figure 2.1 for an example). In contrast to BART’s reconstruction objective, the decoder

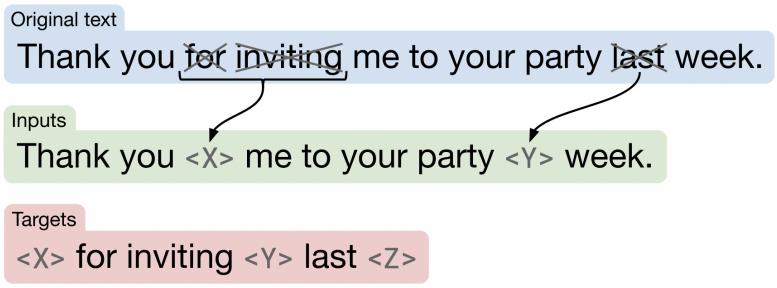



**Figure 2.1:** This figure is taken from [45]. Schematic of the objective used for the T5 baseline model. In this example, the sentence "Thank you for inviting me to your party last week." is processed. The words "for", "inviting" and "last" (marked with an $\times$) are randomly chosen for corruption. Each consecutive span of corrupted tokens is replaced by a sentinel token (shown as <X> and <Y>) that is unique over the example. Since "for" and "inviting" occur consecutively, they are replaced by a single sentinel <X>. The output sequence then consists of the dropped-out spans, delimited by the sentinel tokens used to replace them in the input plus a final sentinel token <Z>.

only generates the replacements and not the full input text. T5 is trained on the *Colossal Clean Crawled Corpus* (C4) [45] dataset which is derived from the *Common Crawl* dataset that consists of text scraped from websites obtained by removing non-text content from HTML files. However, a large proportion of the resulting text is not of high quality and hence many heuristics are applied to clean up the *Common Crawl* dataset in order to get the C4 dataset. For example, websites containing code snippets or words from a list of offensive words are discarded.

## 2.3 Controlled Text Generation

Language generation tasks are usually categorized into *directed* or *open-ended* generation. For directed generation tasks, such as abstractive summarization or machine translation, input and output are closely connected, but for open-ended generation tasks, the dependence of the output on the input is much weaker. For example, there are many possible continuations for a given input prompt, but for the translation task there are not as many correct solutions for a given text [60].

The goal of *controlled text generation* [19, 46, 20] is to control attributes, such as topic or politeness, while generating text. For example, text detoxification and news article writing are two out of many NLP tasks where controlled text generation is required. Most controlled text generation tasks can be categorized into two types [19]: *attribute-conditional generation* [14, 25] and *text attribute transfer* [50, 24]. Attribute-conditional generation is used to gen-

erate text that possesses a specific attribute, whereas text attribute transfer rewrites a text into another text with a given attribute while preserving all other original characteristics [19]. Conditional language models [25, 14] and guiding text generation with attribute classifiers [6, 27] are examples for attribute-conditional generation. For text attribute transfer, mainly unsupervised methods, such as STRAP [28] and ParaGeDi [5], are used, where the goal is to disentangle the desired attribute with other characteristics of the text [19].

Fine-tuning large pre-trained models for controlled text generation is the most straightforward and still the popular method but it is computationally expensive and there is not always enough attribute-specific data available. In this work, we focus on controlled text generation methods that can be used out-of-the-box with large pre-trained language models without using the paradigm of learning. We refer to these methods as *plug-and-play* [41] methods. For example, given a pre-trained language model, we want the generated text to be polite, even though the model was not trained in such a setting. We address this problem by focusing on methods that modify the decoding process by adjusting the probability distribution over tokens in the language model's vocabulary.

### 2.3.1 Text Detoxification

The detection of toxic language, especially for online content moderation, has already been researched for a long time [55, 29, 7, 62]. Studying toxic text generated by large pre-trained language models however, is a more recent trend [57]. Most large-scale text datasets used for training language models are scraped from the web and only filtered with some basic rules [45, 44]. As a result, these datasets also contain text with unwanted attributes. Language models that are trained on these datasets are then able to generate harmful text, such as threats, insults and hate speech [15, 47]. The term *toxicity* is often used to describe these harms [57]. In order to make large pre-trained language models safer and less toxic, we therefore need to use controlled text generation methods or filter their outputs. To that end, multiple methods have been proposed in previous work [57], such as fine-tuning pre-trained language models on non-toxic datasets [15, 16], guiding generation to be less toxic [6, 27, 47] or filtering generated texts at test-time with a toxicity classifier [61]. *Perspective API* is a commercially deployed model that is commonly used for evaluating toxicity in text by providing automatic metrics. It uses the following definition [23] of toxicity: "A comment is considered *toxic* if it is rude, disrespectful, or unreasonable language that is likely to make someone leave a discussion." It is important to note that this definition is subjective.

#### Definition of Text Detoxification Task

In contrast to methods that detoxify the continuation of an input prompt in the setting of open-ended generation [27, 47, 6], i.e., prohibit toxic text to be generated after a prompt, we consider text detoxification in a directed generation setting. More precisely, we treat text detoxification as a special case of text style transfer [28] where the goal is to rewrite a text while preserving its content and changing its style from toxic to non-toxic [5]. However, both style and content are ambiguous concepts and can be interconnected. In the context of text detoxification, we interpret content as the meaning and style as the choice of expressions that are distinct from the content of the text. As pointed out in [28], we acknowledge that this distinction is not universally accepted as style and semantics can also be considered inseparable [10] but this distinction is critical for practical applications of style transfer.

In the following, we adopt the formal definition of text detoxification from [5]. We treat the **toxicity** of a text as a binary variable in the set of styles $S = \{s^{\text{src}}, s^{\text{tg}}\} = \{\texttt{toxic}, \texttt{neutral}\}$ where the source style is `toxic` and the target style is `neutral`. For a given text $d$, toxicity can be measured by a function $\sigma(d) \mapsto s \in S$. Moreover, let $\delta : D \times D \to [0, 1]$ be a function that measures the **semantic similarity** of two texts and let $\psi : D \to [0, 1]$ be a function that measures the degree of **fluency** of a text, where $D$ can be any text dataset. Then, we can define a text detoxification model to be a function

$$\alpha : S \times S \times D \to D, (s^{\text{src}}, s^{\text{tg}}, d^{\text{src}}) \mapsto d^{\text{tg}} \tag{2.7}$$

that generates an output text $d^{\text{tg}}$, given an input text $d^{\text{src}}$, a target style $s^{\text{tg}}$ and a source style $s^{\text{src}}$ such that the following conditions hold:

- The text style transfers from the source style $s^{\text{src}}$ to the target style $s^{\text{tg}}$: $\sigma(d^{\text{src}}) \neq \sigma(d^{\text{tg}}),\ \sigma(d^{\text{tg}}) = s^{\text{tg}}$
- The source text's meaning is preserved in the target text for a given threshold $t^{\delta}$: $\delta(d^{\text{src}}, d^{\text{tg}}) \geq t^{\delta}$
- The degree of fluency is above a given threshold $t^{\psi}$: $\psi(d^{\text{tg}}) \geq t^{\psi}$

We note that it is not possible to completely preserve the meaning of a text when removing toxicity because inevitably, part of the content gets altered. However, the goal is to preserve as much of the meaning of the original text as possible and therefore achieve a high content preservation threshold $t^{\delta}$.

### 2.3.2 Keyword2Text

We now present *Keyword2Text* (K2T) [41], a plug-and-play method for controlled text generation that allows to enforce the appearance of given keywords when using large pre-trained language models for open-ended generation tasks. This is achieved by shifting the distribution of words in the language model's vocabulary towards words that are semantically similar to the

given keywords at each decoding step. Prior research [17] has shown that the cosine similarity between word embeddings obtained by algorithms such as *word2vec* [35] or *GloVe* [42] can be used to measure the semantic similarity between two words. Given this finding, we use the notation $\gamma(w)$ for the embedding of a word $w \in \mathcal{V}$ and use the cosine similarity $\cos(\gamma(w), \gamma(w'))$ between words $w, w' \in \mathcal{V}$ as our measure of semantic similarity. In addition to enforcing the given keywords to be generated, K2T also leads the model to generate a suitable context for them [41].

#### Controlled Generation Objective

Let $p_{\boldsymbol{\theta}}$ be a language model and $w \in \mathcal{V}$ a word in the vocabulary, where $w$ can either be a topic towards which we guide the text or a word that should appear in the generated text. By applying the following modification to the score function $\text{score}(\cdot \mid \mathbf{y}_{<t}) = \log(p_{\boldsymbol{\theta}}(\cdot \mid \mathbf{y}_{<t}))$, we can guide text generation towards $w$ [41]:

$$\text{score}'(y_t, w \mid \mathbf{y}_{<t}) = \text{score}(y_t \mid \mathbf{y}_{<t}) + \lambda \cdot \max(0, \cos(\gamma(y_t), \gamma(w))) \tag{2.8}$$

where the hyperparameter $\lambda > 0$ determines the shift strength. For $\lambda \to 0$, the original score function is recovered; for $\lambda \to \infty$, the distribution gets shifted more towards the word $w$ and if $\lambda$ is large enough, then the word $w$ has the highest score out of all words in the vocabulary. Instead of only guiding towards a single word $w$, we can control generation towards a list of keywords $W = [w_1, \ldots, w_n]$. If the keywords should appear in a fixed order, we can use the score function from Equation 2.8 and start with the first word $w_1$ until it appears in the text and then use the score function with the next word in the list. If the ordering is not important, $W$ is treated as a set and the shift to the score function is applied by taking the highest cosine similarity over the words $w \in W_t$ at each decoding step $t$

$$\text{score}'(y_t, W_t \mid \mathbf{y}_{<t}) = \text{score}(y_t \mid \mathbf{y}_{<t}) + \lambda \cdot \max(0, \max_{w \in W_t} \cos(\gamma(y_t), \gamma(w))) \tag{2.9}$$

where $W_t \subseteq W$ is the set of keywords that did not appear in the generated text before step $t$. In order to guarantee the appearance of the keywords, the shift parameter $\lambda$ is additionally controlled at each step $t$. More precisely, $\lambda$ can be increased exponentially until the given guide word is chosen deterministically.

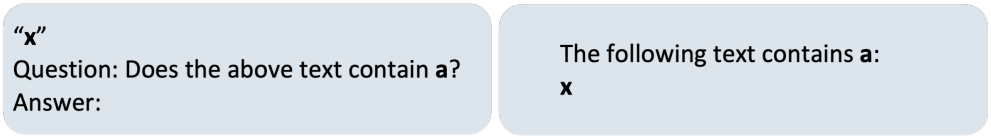
The results in [41] show that using K2T to generate text from GPT-2 works very well in practice and leads to fluent and diverse text that contains the provided keywords. Because K2T can be applied to any auto-regressive language model and decoding algorithm without changing the language model's parameters or requiring additional training, it is a plug-and-play decoding method. However, K2T requires word embeddings provided by

an embedding algorithm, such as GloVe, in addition to the pre-trained language model and depending on the difference in tokenization of the GloVe vocabulary for example and the vocabulary of the language model, there can be issues with the effectiveness of this method as we will discuss in section 3.3.3, where we modify K2T such that it can be applied to the text detoxification task.

### 2.3.3 Self-Diagnosis and Self-Debiasing

In contrast to K2T [41], self-debiasing [47] is a plug-and-play method that does not require any additional external resources, such as word embeddings, and is able to control text generation using only a large pre-trained language model itself. The results in [47] show that large pre-trained language models have the ability to recognize when the text they generate contains unwanted attributes without having to use external resources. This ability is referred to as *self-diagnosis* and can be used to perform *self-debiasing*, which is the ability to use self-diagnosis in order to avoid generating text with unwanted attributes in a completely unsupervised way. More precisely, self-debiasing is achieved by using a plug-and-play decoding algorithm that given an input prompt, reduces the probability of a model producing biased text for the continuation of the prompt using only a textual description of the unwanted attribute [47]. In particular, self-debiasing does not require lists containing unwanted words that have to be chosen carefully, nor does it make any changes to the language model's parameters or require additional training data [47].

#### Self-Diagnosis




**(a)** Self-diagnosis input $\text{sdg}(\mathbf{x}, \mathbf{a})$ **(b)** Self-debiasing input $\text{sdb}(\mathbf{x}, \mathbf{a})$

**Figure 2.2:** This figure is taken from [47] and shows the templates used for self-diagnosis (a) and self-debiasing (b) given an input text **x** and an attribute description **a**. See figure 2.3 for attribute descriptions.

Let $p_\theta$ be a language model, $w_1, ..., w_k \in \mathcal{V}$ a sequence of tokens and let $p_\theta(w \mid w_1, \ldots, w_k)$ be the probability that $w$ is generated as the next token. Then, we can build a self-diagnosis input $\text{sdg}(\mathbf{x}, \mathbf{a})$ using the template shown in figure 2.2 (a), where **x** is a text generated by the language model and **a** is the description for an unwanted attribute (see figure 2.3). The self-diagnosis input $\text{sdg}(\mathbf{x}, \mathbf{a})$ consists of the generated text **x** enclosed in quotation marks,

followed by a question that asks if the attribute **a** is contained in the text **x** and finally a prefix that prompts the language model to generate an answer to the question. For example, we can let the language model self-diagnose if the text **x** = `I'm going to kill you!` contains a threat (**a** = `a threat`) by letting it generate the next token after the following input prompt:

```
"I'm going to kill you!"
Question: Does the above text contain a threat?
Answer:
```

The probability of the text **x** containing attribute **a** can be computed based on the probabilities assigned to the words "Yes" and "No" by the language model after the self-diagnosis input is given as a prompt:

$$p(\mathbf{a} \mid \mathbf{x}) = \frac{p_\theta(\texttt{Yes} \mid \text{sdg}(\mathbf{x}, \mathbf{a}))}{\sum_{w \in \{\texttt{Yes, No}\}} p_\theta(w \mid \text{sdg}(\mathbf{x}, \mathbf{a}))} \tag{2.10}$$

The results in [47] clearly show that the self-diagnosis ability of a model strongly correlates with its size, i.e., the larger the model, the better its self-diagnosis ability. In particular, large pre-trained language models like GPT-2 and T5 have the ability to self-diagnose. However, a language model's self-diagnosis ability is only a necessary first step and not a direct solution for preventing biased text to be generated because self-diagnosis is used only after the text has already been generated.

| **Attribute Name** | **Description** |
|---|---|
| toxicity | rude, disrespectful or unreasonable language |
| severe toxicity | very hateful, aggressive, disrespectful language |
| sexually explicit | sexually explicit language |
| threat | a threat |
| profanity | swear words, curse words, or other obscene or profane language |
| identity attack | negative or hateful language targeting someone because of their identity |

**Figure 2.3:** This figure is taken from [47] and shows the attributes descriptions used for self-diagnosis and self-debiasing inputs.

### Self-Debiasing

For self-debiasing [47], let $\mathbf{x}$ be an input prompt for which we want to generate a continuation using our language model. As for self-diagnosis, we construct a self-debiasing input $\text{sdb}(\mathbf{x}, \mathbf{a})$ using the input prompt $\mathbf{x}$ and the attribute $\mathbf{a}$ and get the template shown in figure 2.2 (b). Then, the distribution of the next tokens $p_\theta(w \mid \mathbf{x})$ given the input prompt $\mathbf{x}$ and the distribution $p_\theta(w \mid \text{sdb}(\mathbf{x}, \mathbf{a}))$ given the self-debiasing input $\text{sdb}(\mathbf{x}, \mathbf{a})$ can be computed. It is important to note that the self-debiasing input leads to language model to generate text that contains unwanted attributes and hence these unwanted words are assigned a greater probability by the distribution $p_\theta(w \mid \text{sdb}(\mathbf{x}, \mathbf{a}))$ than by $p_\theta(w \mid \mathbf{x})$. In other words, the difference between both distributions

$$\Delta(w, \mathbf{x}, \mathbf{a}) = p_\theta(w|\mathbf{x}) - p_\theta(w|\text{sdb}(\mathbf{x}, \mathbf{a})) \tag{2.11}$$

will be less than zero for words that exhibit attribute $\mathbf{a}$. This observation can be used to get the following new distribution

$$\tilde{p}_\theta(w|\mathbf{x}) \propto \alpha(\Delta(w, \mathbf{x}, \mathbf{a})) \cdot p_\theta(w|\mathbf{x}) \tag{2.12}$$

where $\alpha : \mathbb{R} \to [0, 1]$ is a function that scales the probability of biased words according to the difference $\Delta(w, \mathbf{x}, \mathbf{a})$. We choose

$$\alpha(z) = \begin{cases} 1 & \text{if } z \geq 0 \\ e^{\lambda \cdot z} & \text{otherwise} \end{cases} \tag{2.13}$$

for the scaling function where the decay constant $\lambda \geq 0$ is introduced as a hyperparameter of the algorithm.

Applying a slight modification to the algorithm allows to perform self-debiasing with more than one attribute. Given attribute descriptions $A = \{\mathbf{a}_1, ..., \mathbf{a}_n\}$, we can replace $\Delta(w, \mathbf{x}, \mathbf{a})$ in equation 2.12 with:

$$\Delta(w, \mathbf{x}, A) = \min_{\mathbf{a} \in A} \Delta(w, \mathbf{x}, \mathbf{a}). \tag{2.14}$$

As a consequence, the probability of the word $w$ as a continuation of the input prompt $\mathbf{x}$ is reduced by our scaling function $\alpha$ if the difference $\Delta(w, \mathbf{x}, \mathbf{a})$ is negative for at least one of the attributes $\mathbf{a} \in A$.

The results in [47] show that the plug-and-play decoding algorithm used for self-debiasing achieves to reduce bias when generating continuations for input prompts from the *RealToxicityPrompts* dataset [15] for each of the six attributes shown in figure 2.3. Self-debiasing is especially effective for large values of $\lambda$ with the trade-off being a small increase in perplexity.

Chapter 3

# Experiments

In this chapter, we explain our experimental setup and propose several plug-and-play methods compatible with the large pre-trained models[1] presented in the previous chapter for solving the text detoxification task, formally defined in section 2.3.1. To reiterate, our goal is to rewrite a toxic sentence into a non-toxic sentence while preserving its meaning as much as possible using a large pre-trained language model without changing any of its parameters and without using any additional smaller models that need to be trained and require labelled data.

## 3.1 Dataset

Since our models do not require training, we only need a test dataset to evaluate the performance of our models. We use the same test dataset as in [5]. This dataset of 10,000 sentences is constructed from the English data of the first Jigsaw competition [22] called *Toxic Comment Classification Challenge*. The Jigsaw dataset consists of Wikipedia comments which have been labelled by human raters for toxic behavior, such as `toxic`, `threat`, `insult` or `obscene`. The comments labelled as `toxic` are divided into sentences, since the original comments are often too long, and then classified with a toxicity classifier based on a RoBERTa model [32] that has been fine-tuned on a separate part of the Jigsaw dataset. The 10,000 sentences with the highest toxicity score are used for our test dataset.

[1] We use models from the *Hugging Face* framework for all our experiments.

## 3.2 Evaluation Metrics

In order to evaluate our models, we use the same metrics[2] as in [5]. Because we do not have access to a parallel test set of toxic and non-toxic texts for our text detoxification task, we need to use referenceless evaluation and cannot use commonly used metrics for directed generation tasks, such as BLEU [39] or ROUGE [31], that are based on statistics of $n$-gram overlaps between output and reference text. Text detoxification models need to change the style of a text from toxic to non-toxic, preserve its content and produce fluent text. These criteria are usually inversely correlated and hence a combined metric is required to balance them [5].

In [28] a survey of 23 style transfer papers was conducted which discovered that poorly-defined automatic metrics fail to properly evaluate style transfer models. To illustrate this issue, a naive baseline is presented in [28] that achieves to outperform prior work on these metrics by randomly selecting to either output a sentence in the target style or simply copy its input. The three criteria by which style transfer models are usually evaluated are *style transfer accuracy* (ACC), *semantic similarity* (SIM) and *fluency* (FL). To fix the aforementioned issue, the combined metric J, that jointly optimizes the three relevant criteria, is proposed in [28].

> **Style transfer accuracy** (ACC): Given an input text **x** and its style transferred version **y**, a classifier is used to identify the style of **y** and check if it has the wanted target style. The style transfer accuracy (ACC) can then be measured over all outputs. For text detoxification, we check if the style transferred version **y** is classifed as non-toxic. We use the toxicity classifier mentioned in section 3.1.
>
> **Semantic similarity** (SIM): It is possible for a style transfer model to achieve high ACC scores without actually preserving the content of the input text. Hence, it is necessary to to measure how much of the semantics of the input text **x** is preserved in the style transferred text **y**. We use the subword embedding-based model from [59] that computes the cosine similarity between sentence embeddings in order to measure semantic similarity. This model performs well on semantic textual similarity (STS) benchmarks in *SemEval* workshops [2].
>
> **Fluency** (FL): Even if a model achieves to get high scores and both SIM and ACC, the outputs it produces may be ungrammatical. For this reason, a separate metric is required to measure fluency. We measure fluency with a classifier of linguistic acceptability trained on the *CoLA* dataset [56]. Language model perplexity is not used because "(1) it is

[2]The models used for evaluation are available in the following repository: github.com/skoltech-nlp/detox

unbounded and (2) unnatural sentences with common words tend to have low perplexity [36, 38]" and therefore does not serve well as a fluency measure as argued in [28]. Given our classifier, we can then measure fluency as the accuracy over all outputs, i.e., the proportion of outputs that are classified as fluent.

**Combined Metric** (J): After computing ACC, SIM and FL, it is necessary to combine these three metrics into a compound metric that finds a balance between them. This allows us to compare the overall performance of style transfer models using a single score [38]. We use the metric J proposed in [28], more precisely:

$$\text{J}(\mathbf{x}, \mathbf{y}) = \text{ACC}(\mathbf{y}) \cdot \text{SIM}(\mathbf{x}, \mathbf{y}) \cdot \text{FL}(\mathbf{y}) \tag{3.1}$$

where $\mathbf{x}$ is an input text and $\mathbf{y}$ its style-transferred version. It is important to note that we treat ACC and FL as a binary value in $\{0, 1\}$, making sure that disfluent or incorrectly classified sentences are always assigned a J-score of 0. After computing the J-score for each text in the dataset, we take the mean over all texts to get an aggregated J-score for the overall performance of the style transfer model.

The metrics above are proposed in [28] can be used for style transfer tasks in general. In addition, we compute another metric specifically for the text detoxification task, which we refer to as the *detox rate* (DR).

**Detox Rate** (DR): First, we construct a list of 1,844 toxic words from various sources[3,4,5] that have been used in prior research related to text detoxification [45, 51]. Then, we set $\text{DR}(\mathbf{y}) = 0$ if the detoxified text $\mathbf{y}$ contains a word from this list, else $\text{DR}(\mathbf{y}) = 1$. We define the aggregated DR-score to be the proportion of output texts $\mathbf{y}$ that do not contain a word from the toxic word list. We note that some words in this list may not be considered toxic by many people but we choose to use a rather large list even if not all words are considered stricly toxic.[6] Often, these words by themselves are unproblematic but can become toxic depending on the context.

## 3.3 Methods

Before presenting our plug-and-play methods, we present two models that currently yield state-of-the-art results for text detoxification and both use an unsupervised learning approach. We use these models as baselines to which we can compare our methods to.

[3] https://www.noswearing.com/dictionary

[4] https://www.cs.cmu.edu/ biglou/resources/bad-words.txt

[5] https://github.com/LDNOOBW/List-of-Dirty-Naughty-Obscene-and-Otherwise-Bad-Words

[6] For example, the words 'fat', 'nudity' and 'semen' are included in our list.

### 3.3.1 ParaGeDi

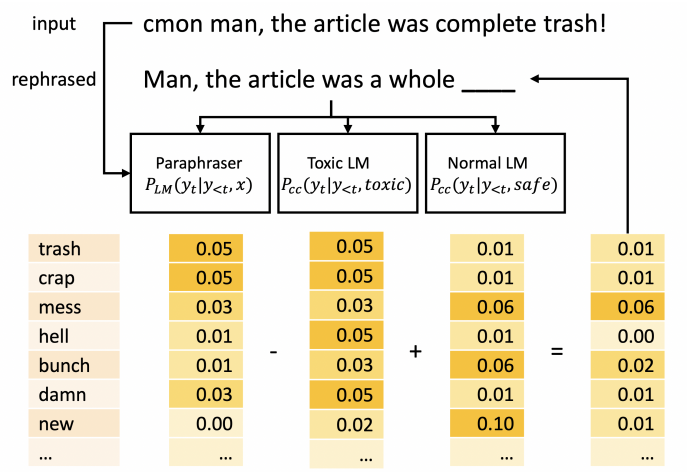



**Figure 3.1:** This figure is taken from [5] and shows an overview of the ParaGeDi model.

*ParaGeDi* [5] is a model that consists of two components, a paraphraser based on a fine-tuned T5 model and a class-conditional language model with classes `toxic` and `non-toxic` based on a fine-tuned GPT-2 model. An overview of the model is depicted in figure 3.1. ParaGeDi works by rewriting a toxic input sentence $\mathbf{x}$ into a paraphrased version $\mathbf{y}$ such that both sentences remain semantically similar. At each decoding step $t$, the distribution of the paraphraser is guided towards non-toxic words using the class-conditional language model as a generative discriminator [27]. ParaGeDi is unsupervised since there is no parallel dataset of toxic and non-toxic sentences required to train the model but the class-conditional language model requires labelled data to train. We do not re-run the experiment since we are using the same test dataset as in [5] and the output of ParaGeDi is available here.

### 3.3.2 CondBert

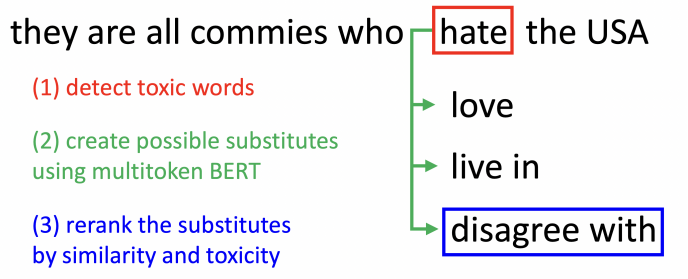



**Figure 3.2:** This figure is taken from [5] and shows an overview of the CondBERT model.

*CondBERT* is class-conditional masked language model based on a fine-tuned BERT [9] model with classes `toxic` and `non-toxic`. An overview of the model is depicted in figure 3.2. First, toxic words in a toxic input sentence $\mathbf{x}$ are identified and masked using a logistic bag-of-words classifier. Next, the masked language model conditioned on the non-toxic class generates possible replacements for the masked toxic word. Finally, these replacements are re-ranked according to their similarity to the original word and their level of toxicity. The semantic similarity is computed using the cosine similarity of word embeddings and toxicity is measured with the same toxicity classifier that is used to identify toxic words. As ParaGedi, CondBERT is an unsupervised method but requires labelled data in order to train the class-conditional masked language model. The outputs of the CondBERT model for our test dataset are available here.

### 3.3.3 Keyword2Text with Encoder-Decoder Model

We take the Keyword2Text method [41] described in section 2.3.2 and modify it such that we can apply it to our text detoxification task. Since we need to transform a toxic input sentence into a non-toxic sentence, we use an encoder-decoder model, instead of a decoder-only model that is used for the original K2T method. We choose the large-sized version of BART [30] (see table 3.1 for architecture details) as our encoder-decoder model because it is trained by corrupting the input text and then optimizing a reconstruction loss, i.e., the cross-entropy between the decoder's output text and the original input text. Intuitively, the decoder tries to generate text that is similar to the corrupted input text which is useful for text detoxification because we want the output text to be semantically similar to the input text but non-toxic. However, in contrast to the inputs during BART's pre-training, we do not corrupt the toxic input text but rather modify the score function of the decoder such that the generated output will be non-toxic. In order to get rid of the toxic elements in the input sentence, we use the same list of 1,844 toxic words that is used to compute the DR-score. Again, we note that this list contains some words that are by themselves not very toxic but our results show that using a large list is beneficial. However, we acknowledge that using a list of toxic words is not optimal in order to detect toxic elements in a text, but given that we constrain ourselves not to use an external classifier that requires learning, it is a good alternative.

Instead of using a list of keywords to guide generation towards them as for K2T, we now use the list of toxic words $W = [w_1, \ldots, w_n]$ to shift the score function conditioned on our (uncorrupted) toxic input text $\mathbf{x}$ away from these toxic words and semantically similar ones at each step $t$, i.e., we simply modify the sign in equation 2.9 from a plus to a minus. as for K2T, we use GloVe embeddings [42] to compute cosine similarities between

**Table 3.1:** Architecture of large-sized BART model

| Model Architecture | |
|---|---|
| encoder layers | 12 |
| decoder layers | 12 |
| attention heads | 16 |
| embedding dim | 1024 |
| vocabulary size | 50265 |
| parameters | 406M |

words. In addition, we introduce a threshold parameter $\alpha \geq 0$ so that we can vary how semantically similar the words should be that will be shifted:

$$\text{score}'(y_t, W \mid \mathbf{y}_{<t}, \mathbf{x}) = \text{score}(y_t \mid \mathbf{y}_{<t}, \mathbf{x}) - \lambda \cdot \max(\alpha, \max_{w \in W} \cos(\gamma(y_t), \gamma(w))) \tag{3.2}$$

where $y_t$ is a token in BART's vocabulary. In contrast to the original formulation, we always use the full list of toxic words $W$ and keep the shift parameter $\lambda$ constant at each decoding step $t$ and do not increase it, as is necessary if the appearance of a guide word has to be guaranteed.

The intuition for this method is that the toxic words in the input sentence do not get generated since they are being shifted away from using our list of toxic words and given BART's objective of reconstructing the original input, we still expect the output sentence to be semantically similar to the input sentence.

Given that GloVe and BART use different tokenization algorithms and hence also have different vocabularies, there are various approaches for computing cosine similarities using GloVe embeddings and then assigning these values to the tokens in BART's vocabulary as is required in equation 3.2. GloVe uses a word-based tokenization algorithm and has a vocabulary $\mathcal{V}_{\text{GloVe}}$ of size $400,000$ with only lowercase words. BART uses a subword-based tokenization algorithm and has a vocabulary $\mathcal{V}_{\text{BART}}$ of size $50,265$. In the following, we present the original approach from [41] and a modified approach which is slightly more involved.

> **Original approach:** In the original approach from [41], each token $v \in \mathcal{V}_{\text{BART}}$ is transformed into $v'$ by lower casing it and removing spaces. If $v' \in \mathcal{V}_{\text{GloVe}}$, then $v$ is assigned the GloVe embedding of $v'$, else a vector of zeros is assigned and consequently cosine similarities with that token are equal to zero.[7] Hence, the cosine similarity between a toxic word $w \in \mathcal{V}_{\text{GloVe}}$ and a token $v \in \mathcal{V}_{\text{BART}}$ is $\cos(\gamma(v), \gamma(w))$ where $\gamma(w)$ is the GloVe embedding of $w$ and $\gamma(v)$ is the embedding assigned to $v$ by the procedure described above. Note

[7] 86% of the tokens in $\mathcal{V}_{\text{BART}}$ are matched with a token in $\mathcal{V}_{\text{GloVe}}$.

that the toxic word $w$ does not have to be a token in $\mathcal{V}_{\text{BART}}$ but needs to be in $\mathcal{V}_{\text{GloVe}}$. A problem with this approach is that longer words consisting of subwords get split into multiple tokens with BART's tokenizer. For example, the toxic word 'motherfucker' gets tokenized into the tokens ['mother', 'f', 'ucker']. As a result, the toxic word 'motherfucker' may still get generated even for large values of the shift strength parameter $\lambda$ because there is no guarantee that the score of its first token 'mother' is decreased. More precisely, the cosine similarity $\cos(\gamma(v), \gamma(w))$ between $w =$ 'motherfucker' and $v =$ 'mother' may be very small or even negative. However, other tokens $v \in \mathcal{V}_{\text{BART}}$ that are semantically similar to 'motherfucker' still get shifted away from.

**Modified approach:** In order to solve the problem of not being able to avoid generating toxic words when they are tokenized into multiple tokens, we try another approach. The idea is to assign the cosine similarity of a word to their first token if that word is tokenized into multiple tokens. Then, we can decrease the score of its first token and consequently the word itself does not get generated. Let $\tau$ be the tokenizer used by BART and $W$ our list of toxic words. For each toxic word $w \in W$, we proceed as follows:

1. If our toxic word $w$ is in $\mathcal{V}_{\text{GloVe}}$, we continue with the next step, otherwise we cannot use this toxic word in equation 3.2 since there is no GloVe embedding to compute cosine similarities with.
2. Compute the cosine similarity of $w$ with all other tokens $v \in \mathcal{V}_{\text{GloVe}}$: $\cos(\gamma(w), \gamma(v))$.
3. We tokenize all tokens $v \in \mathcal{V}_{\text{GloVe}}$ using the BART tokenizer $\tau$ to get $\bar{v} = [\bar{v}_1, \ldots, \bar{v}_{|\bar{v}|}] = \tau(v)$ and assign the cosine similarity $\cos(\gamma(w), \gamma(v))$ to the first token $\bar{v}_1$ of the BART-tokenization of $v$. For example, take $w =$ 'idiot' and $v =$ 'moron'. Then we have $\tau(v) =$ ['mor', 'on'] and hence the cosine similarity of $w$ and $v$ gets assigned to the token 'mor'.
4. However, BART's tokenizer distinguishes between upper and lower case characters and treats spaces as parts of tokens. This means that the same word is encoded differently if it is written with upper or lower case or it contains a prefix space. Concretely, this means that we also need to assign the cosine similarity $\cos(\gamma(w), \gamma(v))$ to the first token of $\tau(v')$ where $v'$ is a possible transformation of $v$. We transform each $v \in \mathcal{V}_{\text{GloVe}}$ by adding a prefix space, uppercasing and capitalizing.
5. After completing the steps above, some tokens in $\mathcal{V}_{\text{BART}}$ are assigned multiple values of cosine similarities. For example, taking $w =$ 'idiot' as our toxic word, the token 'id' gets assigned

the cosine similarity $\cos(\gamma(w), \gamma(v)) = 1$ for $v =$ 'idiot' and $\cos(\gamma(w), \gamma(v)) = 0.17$ for $v =$ 'idol' because $\tau(v)_1 =$ 'id' in both cases. We deal with this issue of multiple assignment by taking the maximum cosine similarity. As a result, some words that are not toxic may not get generated if their first token is equal to the first token of a word in $W$ because the score of that first token is decreased via equation 3.2. If a token in $\mathcal{V}_{\text{BART}}$ is not assigned a cosine similarity, we set the its value to zero.

As a baseline, we compare the two approaches described above with the model that also uses the large-sized version of BART and takes a toxic sentence as encoder input but during decoding, we do not allow words from the toxic word list to be generated. We achieve this by simply blocking their exact token sequence using the parameter `bad_words_ids` in the *Hugging Face* framework. As in the modified approach, we additionally transform the toxic words and block their transformed token sequences as well.

We run experiments for the baseline model and models using the original and modified approach using beam search with beam size 4. For the original and modified approach we perform grid search over the threshold parameter $\alpha \in [0, 1]$ and shift strength parameter $\lambda \in [2, 40]$ to find the optimal hyperparameter combination.

### 3.3.4 Self-Debiasing with Encoder-Decoder Model

As for K2T, we need to modify the self-debiasing method described in section 2.3.3 such that we can apply it to our text detoxification task. Instead of self-debiasing the continuation of an input prompt $\mathbf{x}$ with a decoder-only model, we would like to use self-debiasing to detoxify a toxic input sentence $\mathbf{x}$ into a non-toxic sentence $\mathbf{y}$ and hence need an encoder-decoder model. As encoder-decoder model, we use the large-sized BART model and the large-sized T5 model[8] (see table 3.2 for architecture details). Both models can be used for mask filling. Therefore, if we replace the toxic words in the toxic input sentence, both BART and T5 will find a suitable replacement for the masked span depending on its context. In order to find the toxic words in the input sentence, we again use the list of toxic words mentioned in section 3.2. More precisely, the toxic input sentence $\mathbf{x}$ is tokenized using a word-based tokenizer[9] $\tau$ into $\tau(\mathbf{x})$. Next, we replace tokens in $\tau(\mathbf{x})$ that contain words from the toxic word list (using case-insensitive matching) with mask tokens in order to obtain the masked sentence $\mathbf{x}'$. Consecutive mask tokens are replaced by a single mask token. For example, the sentence $\mathbf{x} =$ 'These guys are idiots' gets tokenized

[8] We use T5 version 1.1. from *Hugging Face* which was only pre-trained unsupervisedly, excluding any supervised training with task prefixes, which is used for the original T5 model.

[9] We use the *TreebankWordTokenizer* from the *nltk* framework.

**Table 3.2:** Architecture of large-sized T5 model

| Model Architecture | |
|---|---|
| encoder layers | 24 |
| decoder layers | 24 |
| attention heads | 16 |
| embedding dim | 1024 |
| vocabulary size | 32128 |
| parameters | 783M |

into $\tau(\mathbf{x})$ = ['These', 'guys', 'are', 'idiots'] and because the word 'idiot' is in our list, the last token gets masked and the resulting masked sentence is $\mathbf{x}'$ = 'These guys are <mask>' where '<mask>' is a special mask token. However, we note that using a list of toxic words is imperfect since it can lead to masking of wrong words, e.g., the word 'class' gets masked because it contains the word 'ass' which is in our list of toxic words.

Given that we use an encoder-decoder setting, we need to adjust the self-debiasing input shown in figure 2.2 (b). We use two different approaches for BART and T5. BART reconstructs the full input, but T5 only generates replacements for the masked tokens and leaves the rest of the sentence intact. For both approaches, let $\mathbf{x}$ be a toxic input sentence and $\mathbf{x}'$ the version where toxic words are masked using our list of toxic words and the procedure described above.

**BART:** We use a self-debiasing input for the encoder as well as the decoder. The self-debiasing encoder input is defined as follows:

$$\text{sdb}_{\text{ENC}}(\mathbf{x}', \mathbf{a}) = \begin{cases} \texttt{The following text contains } \mathbf{a}\texttt{:} \\ \mathbf{x}' \end{cases} \tag{3.3}$$

and for the decoder, we use the following self-debiasing decoder input:

$$\text{sdb}_{\text{DEC}}(\mathbf{a}) = \texttt{The following text contains } \mathbf{a}\texttt{:} \tag{3.4}$$

In words, we simply provide the self-debiasing encoder input with the attribute $\mathbf{a}$ but without the masked toxic sentence $\mathbf{x}'$ as input to the decoder. During the next decoding steps, BART will try to reconstruct the encoder input while filling masked spans in $\mathbf{x}'$. In order to self-debias the text generated during the next decoding steps after providing $\text{sdb}_{\text{DEC}}(\mathbf{a})$ to the decoder as a prompt, we need to adjust equation 2.11 as follows for each $t > 0$:

$$\Delta(y_t, \mathbf{x}', \mathbf{a}, \mathbf{y}_{<t}) = p_{\theta}(y_t|\mathbf{x}', \mathbf{y}_{<t}) - p_{\theta}(y_t|\text{sdb}_{\text{ENC}}(\mathbf{x}', \mathbf{a}), \mathbf{y}_{<t}, \text{sdb}_{\text{DEC}}(\mathbf{a})) \tag{3.5}$$

where $p_\theta$ is the probability distribution given by our encoder-decoder model, $\mathbf{y}_{<t}$ are the tokens generated by the decoder up to step $t$ and $y_t$ is the token considered at step $t$. The idea is that the probability distribution $p_\theta(y_t|\mathbf{x}', \mathbf{y}_{<t})$ conditioned on the masked sentence $\mathbf{x}'$ without the self-debiasing inputs assigns less probability to toxic tokens $y_t$ than with conditioning on the self-debiasing inputs. Moreover, we expect the difference in equation 3.5 to be large especially for the masked tokens that were replaced because they are toxic. Next, we reformulate equation 2.12 for each $t > 0$:

$$\tilde{p}_\theta(y_t|\mathbf{x}', \mathbf{y}_{<t}) \propto \alpha(\Delta(y_t, \mathbf{x}', \mathbf{a}, \mathbf{y}_{<t})) \cdot p_\theta(y_t|\mathbf{x}', \mathbf{y}_{<t}) \tag{3.6}$$

where $\alpha$ is the scaling function defined in equation 2.13 containing the decay constant hyperparameter $\lambda \geq 0$. At each decoding step $t$, the next token $y_t$ is generated according to the self-debiased probability distribution $\tilde{p}_\theta(y_t|\mathbf{x}', \mathbf{y}_{<t})$. In the end, we get the self-debiased version $\mathbf{y}$ of our toxic input sentence $\mathbf{x}$ that is generated by self-debiasing and mask filling the masked sentence $\mathbf{x}'$.

**T5:** As explained in section 2.2.3 and illustrated in figure 2.1, T5 only generates replacements for masked tokens. Hence, we do not need to provide a self-debiasing decoder input $\text{sdb}_{\text{DEC}}(\mathbf{a})$ while the the rest stays the same as in equation 3.5. We then get the self-debiased version $\mathbf{y}$ of our toxic input sentence $\mathbf{x}$ by taking the replacements generated by the decoder and filling them into the masked sentence $\mathbf{x}'$.

The intuition for this method is that it tries to reconstruct the input sentence by filling the masked spans where toxic words were replaced with our toxic word list and at the same time performs self-debiasing. However, it is important to note that in contrast to the K2T method applied to text detoxification described in section 3.3.3, we do not provide the original toxic input sentence $\mathbf{x}$ to the encoder and only provide the masked sentence $\mathbf{x}'$. This means that the model never actually sees the original input sentence. For BART, we can try to provide the unmasked toxic input sentence $\mathbf{x}$ by modifying equation 3.5 into

$$\Delta(y_t, \mathbf{x}, \mathbf{x}', \mathbf{a}, \mathbf{y}_{<t}) = p_\theta(y_t|\mathbf{x}, \mathbf{y}_{<t}) - p_\theta(y_t|\text{sdb}_{\text{ENC}}(\mathbf{x}', \mathbf{a}), \mathbf{y}_{<t}, \text{sdb}_{\text{DEC}}(\mathbf{a})) \tag{3.7}$$

where $p_\theta(y_t|\mathbf{x}, \mathbf{y}_{<t})$ is the probability distribution conditioned on the original input $\mathbf{x}$ and we subtract from it the probability distribution conditioned on the masked input $\mathbf{x}'$ and the self-debiasing encoder input $\text{sdb}_{\text{ENC}}(\mathbf{x}', \mathbf{a})$ and the self-debiasing decoder input $\text{sdb}_{\text{DEC}}(\mathbf{a})$. Moreover, equation 3.6 is modified into

$$\tilde{p}_\theta(y_t|\mathbf{x}, \mathbf{x}', \mathbf{y}_{<t}) \propto \alpha(\Delta(y_t, \mathbf{x}, \mathbf{x}', \mathbf{a}, \mathbf{y}_{<t})) \cdot p_\theta(y_t|\mathbf{x}, \mathbf{y}_{<t}). \tag{3.8}$$

The problem is that the probability of toxic tokens $y_t$ is reduced via self-debiasing only if the difference $\Delta(y_t, \mathbf{x}, \mathbf{x}', \mathbf{a}, \mathbf{y}_{<t})$ is negative. This is the

**Table 3.3:** Self-diagnosis results for the six attributes covered in figure 2.3 for the large-sized BART and large-sized T5 model. The performance is measured using classification accuracy (ACC) and Pearson's correlation coefficient (PCC).

| | BART | | T5 | |
|---|---|---|---|---|
| | ACC | PCC | ACC | PCC |
| toxicity | 67.05 | 43.94 | 62.99 | 31.49 |
| severe toxicity | 71.83 | 46.94 | 70.13 | 48.28 |
| sexually explicit | 59.69 | 16.66 | 72.61 | 49.09 |
| threat | 60.62 | 25.10 | 74.23 | 55.06 |
| profanity | 73.42 | 56.72 | 80.90 | 63.61 |
| identity attack | 69.58 | 49.23 | 71.24 | 49.99 |

case for some toxic tokens in the vocabulary but almost never for the toxic word in $\mathbf{x}$ that is masked in $\mathbf{x}'$. In other words, the probability distribution $p_{\theta}(y_t|\mathbf{x}, \mathbf{y}_{<t})$ conditioned on the unmasked toxic input sentence $\mathbf{x}$ is very peaked at each decoding step and exactly generates $\mathbf{x}$ with high probability even with self-debiasing.

For both BART and T5, we run experiments using all attributes in figure 2.3 when performing self-debiasing and search over multiple values of the decay constant hyperparameter $\lambda \in [0, 500]$. As a baseline, we use the models that simply perform mask filling without self-debiasing, i.e., $\lambda = 0$.

Because the large-sized BART model and the large-sized T5 have not been tested for their self-diagnosis abilities in [47], we run the same self-diagnosis experiment as in [47]. The experiment measures the classification accuracy and the Pearson correlation coefficient of our models using self-diagnosis via equation 2.10 with the scores from the commercially deployed *Perspective API* model for each of the attributes in figure 2.3. The results are presented in table 3.3.

### 3.3.5 Masking Toxic Words with Attention Weights

Both K2T and self-debiasing rely on an external list of toxic words in order to identify toxic elements in a text when performing text detoxification. In addition, K2T also requires external word embeddings such as GloVe [42]. In an attempt to remove the need for any external resources except for the large pre-trained language model itself, we try to use attention weights in combination with self-diagnosis in order to identify toxic words.

To that end, we use *attention rollout*, a method proposed in [1] that tries to interpret self-attention weights in transformer models. Prior research on attention weights has shown that they do not always provide an explanation for a model's output [43, 21] but can sometimes still be useful to gain in-

**Table 3.4:** Architecture of xl-sized GPT-2 model

| Model Architecture | |
|---|---|
| decoder layers | 48 |
| attention heads | 25 |
| embedding dim | 1600 |
| vocabulary size | 50257 |
| parameters | 1.5B |

sights into a transformer model's internals [58, 52, 54]. Attention rollout is a method that tries to interpret attention weights across multiple layers in a transformer model with respect to the input tokens [1]. In each layer, the embeddings from the previous layer are combined via self-attention in order to compute the new embedding for a specific token. As a result, the information from multiple tokens is mixed as we go into deeper layers, making it difficult to attribute attention weights in a specific layer to the input tokens, except for the first layer where the original embeddings belong to the input tokens. Attention rollout takes the flow of information across layers into account by approximating the transformer architecture and computes attention scores for each layer that can then be related back to the input tokens. Finally, the attention rollout scores in the last layer are used to provide an explanation to the following question: "Which part of the input is the most important when generating the output?" [1].

For our use case, we compute attention rollout scores when performing self-diagnosis using a xl-sized GPT-2 model (see table 3.4 for architecture details). More precisely, the self-diagnosis input in figure 2.2 (a) is used together with a toxic input sentence **x** and the attribute `toxicity` from figure 2.3. Then, we visualize the attention rollout scores with respect to the output token. This token corresponds to the generated self-diagnosis answer which indicates if the attribute `toxicity` is contained in the sentence **x**. Intuitively, we expect words that are toxic in the input sentence **x** to be important when generating the self-diagnosis answer and thus we expect the corresponding tokens to have high attention rollout scores. In contrast to the experiments in [1] however, we use an auto-regressive decoder-only model instead of an encoder-only masked language model. Raw attention weights and rollout attentions scores across all layers with respect to the generated self-diagnosis answer are visualized for three examples of toxic sentences **x** in figure 3.4. We observe that the rollout attention scores for all three examples tend to zero as we go into deeper layers but we also notice that the raw attention weights in the first layer assign large weights to toxic words. Visualizing more examples reveals that this behaviour is true for them as well. Therefore, we do not use attention rollout scores and instead simply use the attention weights in the first layer in order to identify and

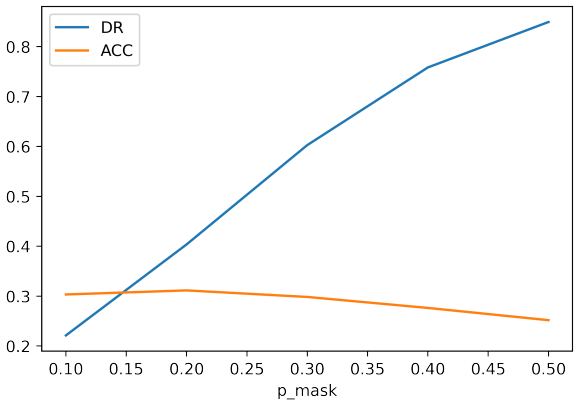



**Figure 3.3:** Detox rate (DR) and masking accuracy (ACC) for different values of the masking percentage $p$ on the $x$-axis.

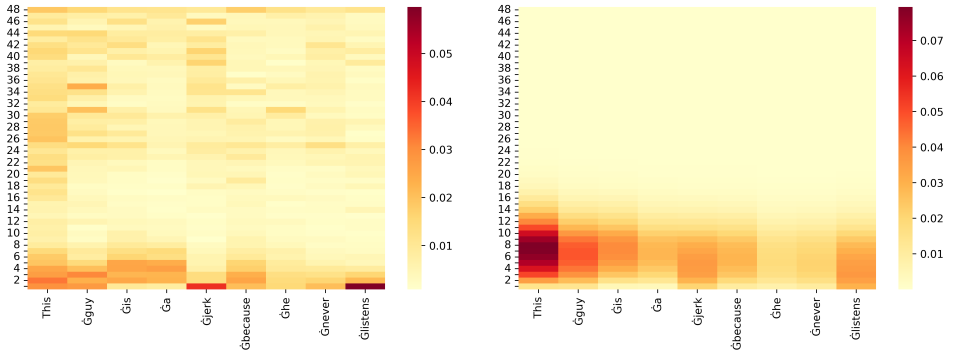
mask toxic words. In a second step, we can then apply the self-debiasing method described in section 3.3.4. Our idea is to mask the words corresponding to the tokens that are assigned the largest raw attentions weights in the first layer. The number of tokens to mask per toxic sentence $\mathbf{x}$ should be dependent on the length of its tokenization $\tau(\mathbf{x})$ where $\tau$ is the tokenizer. The number of tokens $n$ to mask per toxic sentence $\mathbf{x}$ is determined by the following equation:

$$n = \lceil p \cdot |\tau(\mathbf{x})| \rceil \tag{3.9}$$

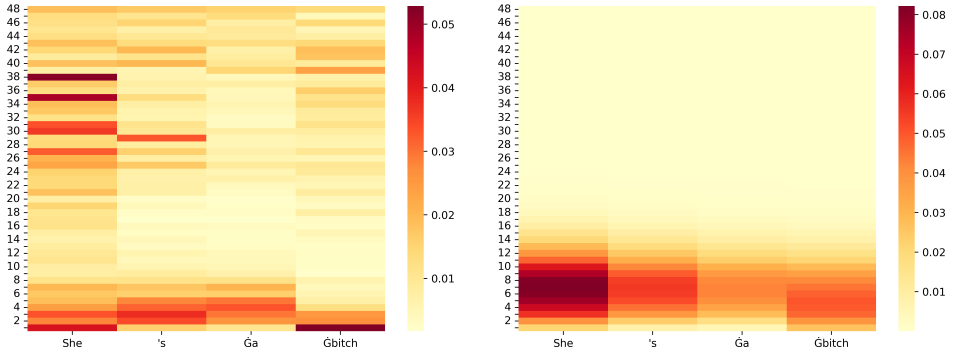
where we introduce masking percentage hyperparameter $p$ and $\lceil \cdot \rceil$ denotes the ceiling function that rounds a number up to its next integer.

We mask all sentences in our test dataset using the aforementioned procedure with masking percentage values $p \in [0.1, 0.5]$. Before performing self-debiasing on this masked dataset, we can already check how effective this method is by computing the detox rate DR mentioned in section 3.2 for the masked sentences. Moreover, we check if the masked words are in our external list of toxic words that is used to compute the DR-score. We refer to this score as the masking accuracy. The results are presented in figure 3.3. We observe that the DR-score monotonically increases from 0.2 to over 0.8 as we increase the masking percentage $p$ from 0.1 to 0.5. This is expected since the more tokens we mask, the more toxic tokens also get masked. The masking accuracy (ACC) is relatively low and stays around 0.3 up to $p = 0.3$ and then decreases down to 0.25 as the masking percentage is increased to $p = 0.5$. This means that often also non-toxic words are masked. For the three examples presented in figure 3.4 and taking $p = 0.3$ as our masking percentage, the following words are masked: (a) 'This', 'jerk', 'listens' (b) 'She', 'bitch' (c) 'fucking', 'idiot'. By inspecting more examples, we notice that our method often correctly masks the toxic words but then also masks adjectives, pronouns or verbs related to that toxic word.

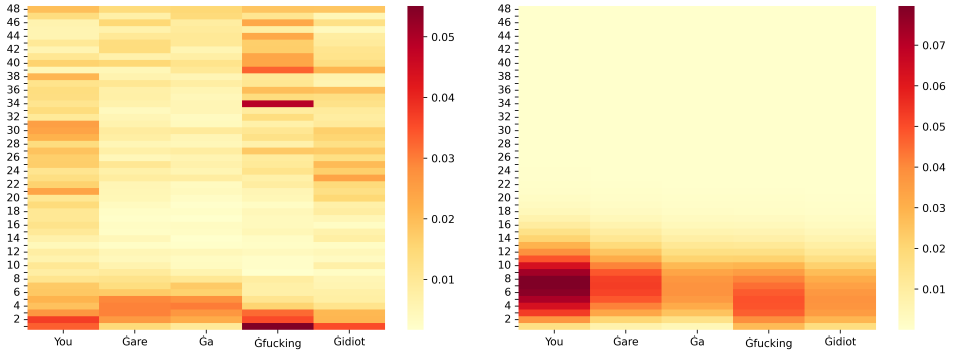
After masking the sentences in the test dataset with masking percentage values $p \in [0.1, 0.5]$, we perform self-debiasing using both BART and T5 with decay constant values $\lambda \in [0, 500]$.




**(a)** $\mathbf{x}$ = This guy is a jerk because he never listens




**(b)** $\mathbf{x}$ = She's a bitch




**(c)** $\mathbf{x}$ = You are a fucking idiot

**Figure 3.4:** Raw self-attention weights (left) and rollout attention scores (right) of the xl-sized GPT-2 model when performing self-diagnosis with the attribute `toxicity` for several toxic sentences. Note that self-attention weights and rollout attention scores are averaged over the 25 attention heads. Moreover, we only visualize tokens belonging to the toxic input sentence $\mathbf{x}$ without showing the rest of the self-diagnosis input $\text{sdg}(\mathbf{x}, \mathbf{a})$ from 2.2 (a).

**Table 3.5:** Evaluation results for the baselines and the best hyperparameter combinations of the plug-and-play methods. SD stands for self-debiasing and TM stands for token masking using the approach described in section 3.3.5.

| Model | DR | ACC | SIM | FL | J |
|---|---|---|---|---|---|
| CondBERT | 57.68 | 91.65 | 73.22 | 76.97 | 51.40 |
| ParaGeDi | 60.41 | 93.77 | 66.33 | 80.75 | 50.72 |
| SD + BART ($\lambda = 500$) | 80.98 | 65.32 | 78.49 | 79.63 | 36.41 |
| SD + T5 ($\lambda = 75$) | 81.99 | 63.77 | 81.11 | 78.43 | 36.11 |
| SD + T5 ($\lambda = 0$) | 79.29 | 61.78 | 82.01 | 79.88 | 35.92 |
| SD + BART ($\lambda = 0$) | 79.85 | 62.11 | 82.80 | 76.24 | 34.49 |
| TM ($p = 0.3$) + SD + BART ($\lambda = 100$) | 53.58 | 65.49 | 58.75 | 70.53 | 24.63 |
| TM ($p = 0.3$) + SD + T5 ($\lambda = 0$) | 51.04 | 62.69 | 61.43 | 66.65 | 24.07 |
| K2T Orig. + BART ($\alpha = 0.55$, $\lambda = 22$) | 52.03 | 45.28 | 73.95 | 84.28 | 23.47 |
| BART + block toxic words | 53.70 | 39.17 | 81.71 | 85.12 | 23.32 |
| K2T Mod. + BART ($\alpha = 0.70$, $\lambda = 28$) | 70.16 | 75.64 | 45.05 | 85.69 | 23.29 |
| Test Dataset | 12.29 | 0.00 | 100.00 | 85.22 | 0.00 |

## 3.4 Results

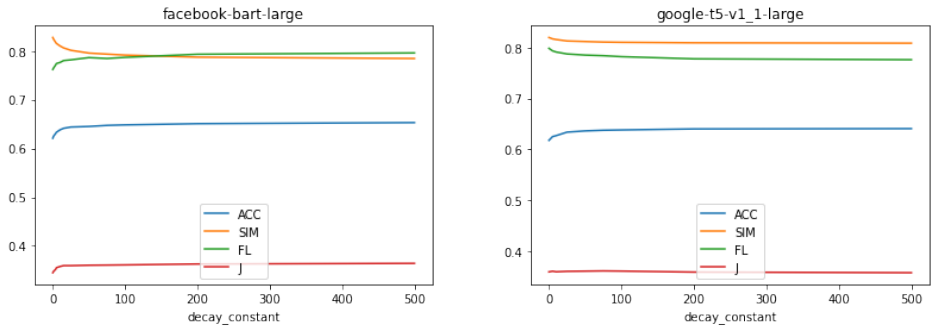
The evaluation results for our baselines and the best hyperparameter combinations of our plug-and-play methods are presented in table 3.5 sorted by decreasing J-scores. In addition, we also evaluate the test dataset as an additional baseline.

Clearly, all our plug-and-play decoding methods for text detoxification underperform the current state-of-the-art models CondBERT and ParaGeDi. CondBERT achieves the highest J-score of 51.40 and ParaGeDi achieves a J-score of 50.72, but the best performing plug-and-play method SD + BART ($\lambda = 500$) only has a J-score of 36.41. While the SIM-score and the FL-score are not too far apart, the ACC-scores of 91.65 for CondBERT and 93.77 for ParaGeDi, are much higher than for SD + BART ($\lambda = 500$) with ACC-score of 65.32. This shows that CondBERT and ParaGeDi are much better in making sure that the outputs they generate are non-toxic. This can be explained by their use of class-conditonal models that are trained in a supervised way with labelled data and are able to better control generation towards non-toxic text. Interestingly, the DR-scores for CondBERT and ParaGeDi are much lower than the DR-score for SD + BART ($\lambda = 500$) despite the ACC-score being higher. This indicates that the list of toxic words that is used to compute the DR-score is not optimal to identify toxic elements in a sentence. We note however, that our best plug-and-play method outperforms all the models that are compared to CondBERT and ParaGeDi in table 1 in

[5] according to the J-score.[10]

Comparing results within our plug-and-play models, we notice that models using K2T perform worse than models using self-debiasing according to the J-score. Even the self-debiasing models, which do not require the toxic word list and perform token masking (TM) with attention weights, outperform the K2T models that require GloVe embeddings and a list of toxic words. We observe that the FL-score is higher for the K2T models than for the SD models but they have a lower J-score because they are not able to balance SIM and ACC as well as the SD models, in the sense that there is a stronger trade-off between SIM and ACC. Furthermore, the SD models masking toxic words using the list of toxic words have higher SIM-scores than the models using K2T. Even though the SD models only receive the masked toxic sentence $\mathbf{x}'$ and not the original toxic input sentence $\mathbf{x}$ as input, they often leave the rest of the intact leading to a high SIM-score.

Within the self-debiasing models, we see that the models using attention weights in order to mask toxic words underperform the SD models using the external list of toxic words. The best model using attention weights is TM ($p = 0.3$) + SD + BART ($\lambda = 100$) with a J-score of 24.63 which is much lower than the next best model SD + BART ($\lambda = 0$) with a J-score of 34.49. While the ACC-scores of TM models are in similar range to the SD models, the FL-scores are lower and the SIM-scores are much lower. This is the case because the toxic word masking accuracy when using attention weights is relatively low as previously mentioned in section 3.3.5 and is illustrated in figure 3.3.



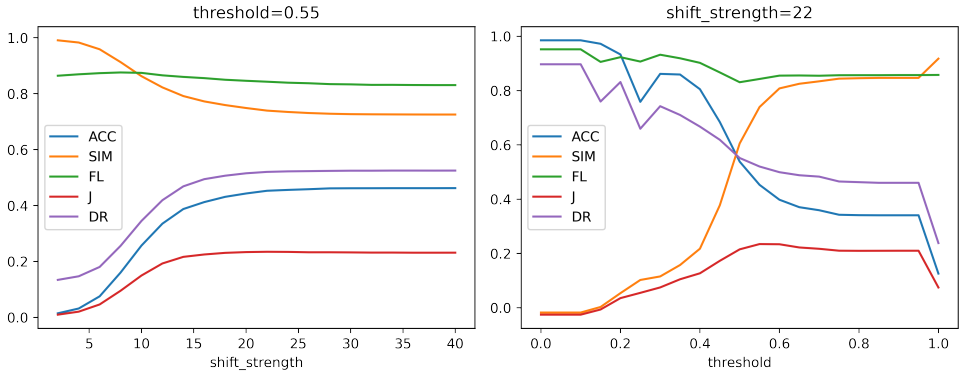

**(a)** BART **(b)** T5

**Figure 3.5:** Evaluation metrics the self-debasing models using BART and T5 with decay constant values on the $x$-axis.

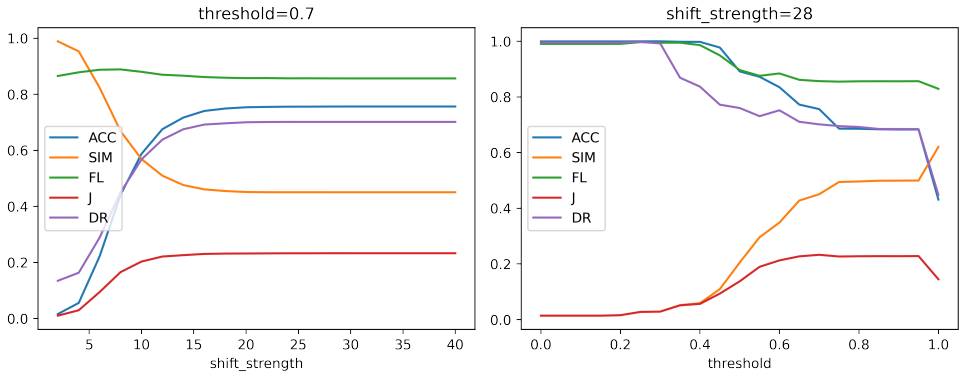
In figure 3.5, we plot the evaluation metrics for the self-debiasing models

[10] We did not include these models in this work since our focus is on plug-and-play methods

using BART and T5 over decay constant values. For both BART and T5, we observe that the decay constant $\lambda$ has its main effect on the evaluation metrics in the beginning as its increased from zero. After that, all metrics stay relatively flat as $\lambda$ increases. However, we notice that this initial effect is stronger for BART than for T5. In particular, the absolute gain in J-score is higher for BART than for T5 when comparing their self-debiasing models with the models that simply perform mask filling, i.e., $\lambda = 0$. Moreover, the best performing SD model with BART uses $\lambda = 500$, whereas the best performing SD model with T5 uses $\lambda = 75$. We can argue that T5 requires less debiasing since it is trained on the C4 [45] dataset, which does not include text containing words from a list of toxic words. Most importantly, we note that ACC increases as $\lambda$ increases, validating that self-debiasing can be used for detoxification.




**(a)** K2T original approach: (left) Fixed similarity threshold $\alpha = 0.55$ over shift strength values on the $x$-axis. (right) Fixed shift strength $\lambda = 22$ over threshold values on the $x$-axis.




**(b)** K2T modified approach: (left) Fixed similarity threshold $\alpha = 0.7$ over shift strength values on the $x$-axis. (right) Fixed shift strength $\lambda = 28$ over threshold values on the $x$-axis.

**Figure 3.6:** Evaluation metrics for the K2T models using the original and modified approach for different hyperparameter values.

Looking at the K2T models, we see that the original approach with $\alpha = 0.55$ and $\lambda = 22$ achieves the best J-score of 23.47 and the modified approach with $\alpha = 0.7$ and $\lambda = 28$ achieves J-score of 23.29, performing slightly worse than the baseline with a J-score of 23.32, which is the model that simply prevents words from the toxic word list to be generated. Even though the J-scores of the three models are not very far apart, the original approach has a much lower ACC-score than the modified approach. This is expected because it may not be able to reduce the probability of toxic words that are tokenized into multiple tokens as previously mentioned in section 3.3.3. On the other hand, the modified approach has much lower SIM-score than the original approach which is also expected since it may prohibit words to be generated even if they are not toxic. The baseline that simply prohibits token sequences that correspond to words in the toxic word list has the highest SIM-score as most of the times, it only changes the toxic word and leaves the rest of of the sentence as it is.

In figure 3.6 (a) and 3.6 (b) we plot the evaluation metrics for the original and modified approach where we fix either the threshold $\alpha$ or the shift strength $\lambda$. Given a fixed threshold $\alpha$ and for both the original and modified approach, we can see that there is a clear trade off between ACC and SIM where ACC increases and SIM decreases as $\lambda$ increases up to a certain value and after that both ACC and SIM stay relatively flat. This trade off is stronger for the modified approach than for the original approach. Given a fixed shift strength, we see that it is crucial to be able to increase the similarity threshold and not leave it at zero. For low values of the similarity threshold $\alpha$, we see that SIM is very low, which means that we shift too many tokens and the model is not able to preserve the meaning of the original sentence. However, if $\alpha$ is too large, ACC is low since the model cannot prevent toxic words to be generated. Interestingly, the trade-off between ACC and SIM, when the shift strength is fixed, is stronger for the original approach than for the modified approach.

In this work, we do not perform human evaluations. However, it is noted in [5] that while ACC and FL scores often correspond with human judgements, there is only weak correlation between human scores for semantic similarity and our SIM-score. We therefore manually inspect some outputs generated by our models to qualitatively assess them. Some examples are shown in table 3.6. From looking at the examples, we observe that most generated outputs do not actually preserve the content. Even if SIM-scores for our best self-debiasing models are high, we notice that often they replace toxic words and change the meaning of the original sentence. Moreover, the K2T original model often produces repetitive or even degenerate outputs. Only in some rare cases do our models achieve to successfully detoxify the input sentence while preserving the content. However, it is important to note that even for the current state-of-the-art models ParaGeDi and CondBERT, only around

20% and 14% respectively of the generated outputs are considered to be valid detoxified outputs that preserve the meaning when judged by human evaluators. Therefore, given that our best performing models underperform these state-of-the-art models, we can expect the percentage of valid outputs to be even less.

| | |
|---|---|
| M0 | quit your bullshit before you get blocked , moron . |
| M1 | If you get blocked,.quit your account! |
| M2 | Get your act together before you get blocked,. |
| M3 | quit your bullsh*t and shut up, moron. |
| M0 | i am so incredibly pissed off with you fucks right now its unreal . |
| M1 | i am so incredibly lucky to live off with you guys right now its unreal. |
| M2 | i am so ticked off with you right now. |
| M3 | i am so incredibly screwed off with you fucks right now its unreal. |
| M0 | i dont want to read about how men are assholes in mexico . |
| M1 | i dont want to read more stories on how men are killed in mexico. |
| M2 | i want to write a book about how men are assholes. |
| M3 | i dont have time for this.i dont want this. |
| M0 | read the fucking page before you declare it vandalism , asshole . |
| M1 | read the front page before you declare it vandalism, |
| M2 | Turn the fucking page before you turn it, asshole. Period. |
| M3 | read the freaking page before you declare it vandalism, you are a moron. |
| M0 | she is is also a whore so she would take in at anytime . |
| M1 | she is is also a foster parent as well so she would take in at anytime. |
| M2 | She is also a size 10, so she would weigh in at. 5.5 |
| M3 | she said that she has no money.she said she's an alcoholic. |
| M0 | cowards lie , deny and blame everyone but themselves and trying to save their own ass . |
| M1 | cowards lie, deny and blame everyone but themselves and trying to save their own. |
| M2 | , denying everything to everyone but themselves and trying to save their own. |
| M3 | cowards lie, deny and blame everyone except themselves and they have no shame in that. |
| M0 | all wikipedia editors are virgins and cunts and they deserve to die of cancer . |
| M1 | All and they deserve to be proud of. |
| M2 | all men are jerks and cunts and they deserve to die of cancer. |
| M3 | all wikipedia editors who don't like wikipedia... |
| M0 | i hate killing and fighting , but when other people have no fucking goal in life but to die and kill as many nonmuslims as possible , then im all fucking them up . |
| M1 | i hate change and, but when other people have no goal in life but to be happy and to make as many friends as possible, then im all for shaking them up. |
| M2 | i and, but when other people have no real purpose in life but to hurt and kill as many people as possible, then we all need to shut them up. |
| M3 | i hate war, i hate hate, and im not proud of my religion. i hate war. |
| M0 | dont edit shit that you have zero knowledge of . |
| M1 | dont edit things that you have absolutely no knowledge of. |
| M2 | 4. Learn something that you have zero knowledge of. |
| M3 | i don't care what you say.do.what.you.say.i.don't care. |
| M0 | ugly butt faced boy who didnt know anything |
| M1 | ugly little red faced boy who didnt know anything |
| M2 | 5. A butt faced idiot who doesn't know anything |
| M3 | ugly bum faced boy who didnt know anything |

**Table 3.6:** Original inputs (M0) and detoxified outputs generated by the following models: (M1) SD + BART ($\lambda = 500$) (M2) TM ($p = 0.3$ + SD + BART ($\lambda = 100$) (M3) K2T Original + BART ($\alpha = 0.55, \lambda = 22$)

Chapter 4

# Conclusion

This thesis proposes several plug-and-play decoding methods that can be used with large pre-trained language models to perform text detoxification. Some of these methods still require external resources, such as word embeddings in order to compute scores for semantic similarity and a list of toxic words to identify toxic elements in a text. But we also propose a method that uses self-attention weights in order to completely remove the need for any external resources to perform text detoxification and hence is a truly plug-and-play method. All of our methods focus on modifying the decoding process by adjusting the probability distribution produced by our language model at each decoding step. While our results show that our proposed methods are effective in performing text detoxification according to automatic metrics, they are still far behind models like ParaGeDi and CondBERT [5] that use learning to solve this task. However, we note that the text detoxification task as formally defined in section 2.3.1 has not been an active area of research for a long time and even the current state-of-the-art models ParaGeDi and CondBERT are far from being able to solve this task in a way that these models could be deployed commercially. Furthermore, we point out that we made the assumption that the toxic style and the content of a sentence can be separated. This assumption is made such that we can come up with practical methods to solve this task. In practice, our models show that this assumption is unrealistic since one of the main weaknesses of our models is the ability to preserve the content of the original sentence. Finally, we believe that even if current plug-and-play methods are not very effective for solving the specific task of text detoxification, there may be potential in applying such methods for other tasks, especially in combination with zero-shot learning techniques like self-diagnosis and self-debiasing [47] that use the internal knowledge of a large pre-trained language model.

# Bibliography


[1] Samira Abnar and Willem Zuidema. Quantifying attention flow in transformers. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 4190–4197, Online, July 2020. Association for Computational Linguistics.

[2] Eneko Agirre, Carmen Banea, Daniel Cer, Mona Diab, Aitor Gonzalez-Agirre, Rada Mihalcea, German Rigau, and Janyce Wiebe. SemEval-2016 task 1: Semantic textual similarity, monolingual and cross-lingual evaluation. In *Proceedings of the 10th International Workshop on Semantic Evaluation (SemEval-2016)*, pages 497–511, San Diego, California, June 2016. Association for Computational Linguistics.

[3] Dzmitry Bahdanau, Kyung Hyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. January 2015. 3rd International Conference on Learning Representations, ICLR 2015 ; Conference date: 07-05-2015 Through 09-05-2015.

[4] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. In *3rd International Conference on Learning Representations*, 2015.

[5] David Dale, Anton Voronov, Daryna Dementieva, Varvara Logacheva, Olga Kozlova, Nikita Semenov, and Alexander Panchenko. Text detoxification using large pre-trained neural models. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 7979–7996, Online and Punta Cana, Dominican Republic, November 2021. Association for Computational Linguistics.

[6] Sumanth Dathathri, Andrea Madotto, Janice Lan, Jane Hung, Eric Frank, Piero Molino, Jason Yosinski, and Rosanne Liu. Plug and play language models: A simple approach to controlled text generation. In *International Conference on Learning Representations*, 2020.

[7] Thomas Davidson, Dana Warmsley, Michael Macy, and Ingmar Weber. Automated hate speech detection and the problem of offensive language. *Proceedings of the International AAAI Conference on Web and Social Media*, 11(1):512–515, May 2017.

[8] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina 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)*, pages 4171–4186, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics.

[9] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina 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)*, pages 4171–4186, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics.

[10] Penelope Eckert. Variation and the indexical field1. *Journal of Sociolinguistics*, 12(4):453–476, 2008.

[11] Bryan Eikema and Wilker Aziz. Is MAP decoding all you need? the inadequacy of the mode in neural machine translation. In Donia Scott, Núria Bel, and Chengqing Zong, editors, *Proceedings of the 28th International Conference on Computational Linguistics, COLING 2020, Barcelona, Spain (Online), December 8-13, 2020*, pages 4506–4520. International Committee on Computational Linguistics, 2020.

[12] Hugging Face. Summary of the tokenizers, 2022. Accessed on 24 April 2022, https://huggingface.co/docs/transformers/v4.16.2/en/tokenizer_summary.

[13] Angela Fan, Mike Lewis, and Yann Dauphin. Hierarchical neural story generation. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 889–898, Melbourne, Australia, July 2018. Association for Computational Linguistics.

[14] Jessica Ficler and Yoav Goldberg. Controlling linguistic style aspects in neural language generation. In *Proceedings of the Workshop on Stylistic Variation*, pages 94–104, Copenhagen, Denmark, September 2017. Association for Computational Linguistics.

[15] Samuel Gehman, Suchin Gururangan, Maarten Sap, Yejin Choi, and Noah A. Smith. RealToxicityPrompts: Evaluating neural toxic degeneration in language models. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 3356–3369, Online, November 2020. Association for Computational Linguistics.

[16] Suchin Gururangan, Ana Marasović, Swabha Swayamdipta, Kyle Lo, Iz Beltagy, Doug Downey, and Noah A. Smith. Don't stop pretraining: Adapt language models to domains and tasks. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 8342–8360, Online, July 2020. Association for Computational Linguistics.

[17] Tatsunori B. Hashimoto, David Alvarez-Melis, and Tommi S. Jaakkola. Word Embeddings as Metric Recovery in Semantic Spaces. *Transactions of the Association for Computational Linguistics*, 4:273–286, 07 2016.

[18] Ari Holtzman, Jan Buys, Maxwell Forbes, and Yejin Choi. The curious case of neural text degeneration. *International Conference on Learning Representations*, 2020.

[19] Zhiting Hu and Li Erran Li. A causal lens for controllable text generation, 2022.

[20] Zhiting Hu, Zichao Yang, Xiaodan Liang, Ruslan Salakhutdinov, and Eric P. Xing. Toward controlled generation of text. In Doina Precup and Yee Whye Teh, editors, *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 1587–1596. PMLR, 06–11 Aug 2017.

[21] Sarthak Jain and Byron C. Wallace. Attention is not Explanation. 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)*, pages 3543–3556, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics.

[22] Jigsaw. Toxic comment classification challenge, 2018. Accessed on 1 March 2021, https://www.kaggle.com/c/jigsaw-toxic-comment-classification-challenge.

[23] Jigsaw. Perspective API, 2021. Accessed on 26 April 2022, https://perspectiveapi.com/.

[24] Di Jin, Zhijing Jin, Zhiting Hu, Olga Vechtomova, and Rada Mihalcea. Deep learning for text style transfer: A survey, 2020.

[25] Nitish Shirish Keskar, Bryan McCann, Lav Varshney, Caiming Xiong, and Richard Socher. CTRL - A Conditional Transformer Language Model for Controllable Generation. *arXiv preprint arXiv:1909.05858*, 2019.

[26] S. C. Kleene. *Representation of Events in Nerve Nets and Finite Automata*. RAND Corporation, Santa Monica, CA, 1951.

[27] Ben Krause, Akhilesh Deepak Gotmare, Bryan McCann, Nitish Shirish Keskar, Shafiq Joty, Richard Socher, and Nazneen Fatema Rajani. GeDi: Generative discriminator guided sequence generation. In *Findings of the Association for Computational Linguistics: EMNLP 2021*, pages 4929–4952, Punta Cana, Dominican Republic, November 2021. Association for Computational Linguistics.

[28] Kalpesh Krishna, John Wieting, and Mohit Iyyer. Reformulating unsupervised style transfer as paraphrase generation. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 737–762, Online, November 2020. Association for Computational Linguistics.

[29] Irene Kwok and Yuzhou Wang. Locate the hate: Detecting tweets against blacks. *Proceedings of the AAAI Conference on Artificial Intelligence*, 27(1):1621–1622, Jun. 2013.

[30] Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Veselin Stoyanov, and Luke 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*, pages 7871–7880, Online, July 2020. Association for Computational Linguistics.

[31] Chin-Yew Lin. ROUGE: A package for automatic evaluation of summaries. In *Text Summarization Branches Out*, pages 74–81, Barcelona, Spain, July 2004. Association for Computational Linguistics.

[32] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. Roberta: A robustly optimized bert pretraining approach, 2019.

[33] Thang Luong, Hieu Pham, and Christopher D. Manning. Effective approaches to attention-based neural machine translation. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 1412–1421, Lisbon, Portugal, September 2015. Association for Computational Linguistics.

[34] Clara Meister, Tim Vieira, and Ryan Cotterell. If beam search is the answer, what was the question? In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing*, Online, November 2020. Association for Computational Linguistics.

[35] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space, 2013.

[36] Remi Mir, Bjarke Felbo, Nick Obradovich, and Iyad Rahwan. Evaluating style transfer for text. 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)*, pages 495–504, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics.

[37] OpenAI. Better language models and their implications, 2019. Accessed on 26 April 2022, https://openai.com/blog/better-language-models/.

[38] Richard Yuanzhe Pang. Towards actual (not operational) textual style transfer auto-evaluation. In *Proceedings of the 5th Workshop on Noisy User-generated Text (W-NUT 2019)*, pages 444–445, Hong Kong, China, November 2019. Association for Computational Linguistics.

[39] Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*, pages 311–318, Philadelphia, Pennsylvania, USA, July 2002. Association for Computational Linguistics.

[40] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. On the difficulty of training recurrent neural networks. In Sanjoy Dasgupta and David McAllester, editors, *Proceedings of the 30th International Conference on Machine Learning*, volume 28 of *Proceedings of Machine Learning Research*, pages 1310–1318, Atlanta, Georgia, USA, 17–19 Jun 2013. PMLR.

[41] Damian Pascual, Beni Egressy, Clara Meister, Ryan Cotterell, and Roger Wattenhofer. A plug-and-play method for controlled text generation. In *Findings of the Association for Computational Linguistics: EMNLP 2021*, pages 3973–3997, Punta Cana, Dominican Republic, November 2021. Association for Computational Linguistics.

[42] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. Glove: Global vectors for word representation. In *Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, 2014.

[43] Danish Pruthi, Mansi Gupta, Bhuwan Dhingra, Graham Neubig, and Zachary C. Lipton. Learning to deceive with attention-based explanations. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 4782–4793, Online, July 2020. Association for Computational Linguistics.

[44] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language Models are Unsupervised Multitask Learners. 2019.

[45] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of Machine Learning Research*, 21(140):1–67, 2020.

[46] EHUD REITER and ROBERT DALE. Building applied natural language generation systems. *Natural Language Engineering*, 3(1):57–87, 1997.

[47] Timo Schick, Sahana Udupa, and Hinrich Schütze. Self-Diagnosis and Self-Debiasing: A Proposal for Reducing Corpus-Based Bias in NLP. *Transactions of the Association for Computational Linguistics*, 9:1408–1424, 12 2021.

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

[49] Rico Sennrich, Barry Haddow, and Alexandra Birch. Neural machine translation of rare words with subword units. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1715–1725, Berlin, Germany, August 2016. Association for Computational Linguistics.

[50] Tianxiao Shen, Tao Lei, Regina Barzilay, and Tommi Jaakkola. Style transfer from non-parallel text by cross-alignment. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, page 6833–6844, Red Hook, NY, USA, 2017. Curran Associates Inc.

[51] Minh Tran, Yipeng Zhang, and Mohammad Soleymani. Towards a friendly online community: An unsupervised style transfer framework for profanity redaction. In *Proceedings of the 28th International Conference on Computational Linguistics*, pages 2107–2114, Barcelona, Spain (Online), December 2020. International Committee on Computational Linguistics.

[52] Shikhar Vashishth, Shyam Upadhyay, Gaurav Singh Tomar, and Manaal Faruqui. Attention interpretability across nlp tasks, 2019.

[53] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. Attention is all you need. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.

[54] Jesse Vig. Visualizing attention in transformer-based language representation models, 2019.

[55] William Warner and Julia Hirschberg. Detecting hate speech on the world wide web. In *Proceedings of the Second Workshop on Language in Social Media*, pages 19–26, Montréal, Canada, June 2012. Association for Computational Linguistics.

[56] Alex Warstadt, Amanpreet Singh, and Samuel R. Bowman. Neural network acceptability judgments. *Transactions of the Association for Computational Linguistics*, 7:625–641, 2019.

[57] Johannes Welbl, Amelia Glaese, Jonathan Uesato, Sumanth Dathathri, John Mellor, Lisa Anne Hendricks, Kirsty Anderson, Pushmeet Kohli, Ben Coppin, and Po-Sen Huang. Challenges in detoxifying language models. In *Findings of the Association for Computational Linguistics: EMNLP 2021*, pages 2447–2469, Punta Cana, Dominican Republic, November 2021. Association for Computational Linguistics.

[58] Sarah Wiegreffe and Yuval Pinter. Attention is not not explanation. 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)*, pages 11–20, Hong Kong, China, November 2019. Association for Computational Linguistics.

[59] John Wieting, Taylor Berg-Kirkpatrick, Kevin Gimpel, and Graham Neubig. Beyond BLEU:training neural machine translation with semantic similarity. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 4344–4355, Florence, Italy, July 2019. Association for Computational Linguistics.

[60] Gian Wiher. Decoding Schemes for Language Generation Models. Master's thesis, ETH Zurich, 2021.

[61] Albert Xu, Eshaan Pathak, Eric Wallace, Suchin Gururangan, Maarten Sap, and Dan Klein. Detoxifying language models risks marginalizing minority voices, 2021.

[62] Marcos Zampieri, Shervin Malmasi, Preslav Nakov, Sara Rosenthal, Noura Farra, and Ritesh Kumar. SemEval-2019 task 6: Identifying and categorizing offensive language in social media (OffensEval). In *Proceedings of the 13th International Workshop on Semantic Evaluation*, pages 75–86, Minneapolis, Minnesota, USA, June 2019. Association for Computational Linguistics.

Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

## Declaration of originality

The signed declaration of originality is a component of every semester paper, Bachelor's thesis, Master's thesis and any other degree paper undertaken during the course of studies, including the respective electronic versions.

Lecturers may also require a declaration of originality for other written papers compiled for their courses.

---

I hereby confirm that I am the sole author of the written work here enclosed and that I have compiled it in my own words. Parts excepted are corrections of form and content by the supervisor.

**Title of work** (in block letters):

Text Detoxification using Pre-Trained Language Models and Plug-and-Play Generation Methods

**Authored by** (in block letters):

*For papers written by groups the names of all authors are required.*

| **Name(s):** | **First name(s):** |
|---|---|
| Pullely | Samuel |
| | |
| | |
| | |
| | |

With my signature I confirm that

- I have committed none of the forms of plagiarism described in the 'Citation etiquette' information sheet.
- I have documented all methods, data and processes truthfully.
- I have not manipulated any data.
- I have mentioned all persons who were significant facilitators of the work.

I am aware that the work may be screened electronically for plagiarism.

| **Place, date** | **Signature(s)** |
|---|---|
| Zurich, 02.05.2022 |  |
| | |
| | |
| | |
| | |

*For papers written by groups the names of all authors are required. Their signatures collectively guarantee the entire content of the written paper.*