# Toxicity Detection

Group 5: Marjaana Lahti (100842757), Mikael Hokkanen (101879693), Syed Ashraf (906586)

ELEC-E5550 Statistical Natural Language Processing D

April 26, 2024

# Contents

# 1 Introduction

With the advent of the internet, being just a name on a screen has made people feel significantly more comfortable in being toxic to other people when they are not face to face [6]. In response to the escalating toxicity observed in various digital platforms such as in games, livestream chats, and social media comment sections, developers increasingly turn to the field of natural language processing (NLP) for solutions.

Toxicity detection involves understanding the nature of toxic language, which can range from explicit hate speech to subtle microaggressions. Computational models that detect toxic language are trained on large datasets of labelled toxic and non-toxic language samples. Using the labels as a benchmark the models learn linguistic patterns indicative of toxicity. Integrating contextual information and user-specific characteristics can enhance the accuracy and effectiveness of these models. In an ideal setting such models would be integrated into content moderation to help foster kinder online interactions.

In our project we have developed a toxicity classification model that we used to participate in the Kaggle competition. We use a transformer based model architecture, most importantly an encoder, to transform input text into a fixed-size vector representation. Once the text is encoded, it's fed to a multilayer perceptron neural network, which we use as a classifier. The classifier is trained using labeled data, where examples of toxic and non-toxic language are annotated with corresponding labels. During training the aim is also to minimize a loss function, which we determined in accordance with the task specification. To assess the performance of our model, we use metrics such as accuracy and F1 score, evaluated on an independent validation dataset. These metrics serve as benchmarks which assess the model's performance on unseen data and the accuracy of its classification.

# 2 Methods

## 2.1 The Data

The toxicity dataset is divided into a training, validation, and test sets. The training data has 90 000 rows, validation 11 000, and test set 12 000. Each row contains an ID, the unprocessed text comment, and a binary label indicating whether the comment is classified as toxic or not. For the test set, the toxicity class labels are hidden.

**Example data row:**

| ID | Text | Label |
|---|---|---|
| 1 | Except that Desmond played first base last night. Tapia was in LF and Reynolds had a night off. | 0 |

We note that the training data has an imbalance of class labels. 36.7% of the sentences are labelled as toxic, and the rest as non-toxic. While we believe that the training data is enough for the encoder to learn the characteristics of both toxic and non-toxic sentences, the binary classifier MLP could benefit from balancing the class proportions. We return to this topic later on.

As preliminary analysis we examined word frequencies in the training set using the wordcloud library. This gave us an immediate visual idea of the contents of the dataset. Some simple cleaning was performed to remove special characters, as well as stopwords (including the custom stopword "like" as it was one of the top words and we found it uninformative). In Figure 1, from dominating words such as "trump", and "state" we can assume the contents of the dataset to be related to U.S. politics. Words such as "right" may hold multiple meanings as either referring to the political affiliation or people being correct. Either way it indicates that adversarial attitudes are present in the dataset.

## 2.2 Data Preprocessing

Data preprocessing starts with normalization, where the text is standardized by converting it

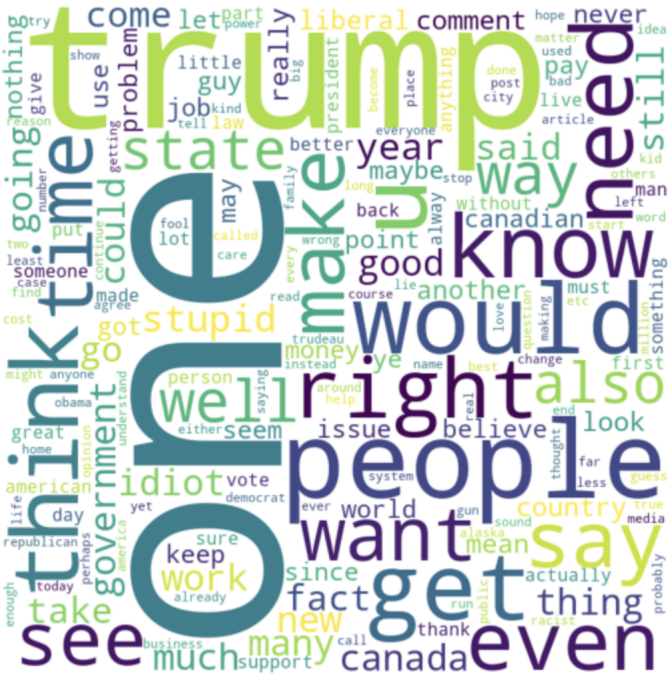


Figure 1: Wordcloud of word frequencies in training set

to lowercase and removing any leading or trailing whitespace. We also apply Unicode normalization for consistent encoding of characters, removing accents and other special characters that may introduce noise into the data. This ensures that variations in casing and character encoding do not affect the model's ability to learn meaningful patterns from the text.

After normalization the text is tokenized into individual words by splitting strings using whitespaces. During tokenization, we construct a vocabulary of unique words encountered in the dataset and map their to their index values to allow for decoding back to text form. We also track the frequency of each word in the dataset, which can be useful for identifying rare or out-of-vocabulary words that may require special handling during model training. Once the text has been tokenized and normalized, we use the vocabulary to map the individual words into their index values.

The encoder requires all sentences within a batch to have the same length. Our custom dataloader pads shorter sentences with zeros to match the length of the longest one in the batch. Once the lengths are equalized, the dataloader provides this adjusted batch along with a binary mask. The mask is responsible for helping the encoder differentiate the actual sentence from the padded values when passing through the encoder blocks.

### 2.3 Model Architecture

Our model contains two main components, a transformer encoder and a multi-layer perceptron neural network. The encoder is responsible for processing the input text and learning meaningful relationships between features, while the neural network learns to classify the encoded sentences to correct toxicity labels. The entire model is programmed from scratch with PyTorch.

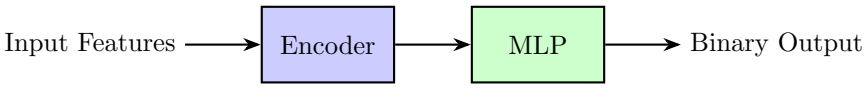


Figure 2: Model pipeline

### 2.4 Encoder

The encoder-architecture shown in Figure 3 is based on the transformer model introduced in the paper "Attention is All You Need" [9]. The encoder employs a series of encoder blocks, each consisting of self-attention and feed-forward layers. The self-attention mechanism allows the model to capture information from sentences from both left and right directions, capturing semantical meanings between each word in the sentence. Additionally, positional embeddings are incorporated into the encoder to encode the position of each token in the input sequence, enhancing the model's ability to differentiate between words based on their position. Both positional encoding and positional embedding were tested, but did not show a significant difference between the prediction score. The final output by the encoder is a latent representation of the original input sequences, which is passed to the neural network for predictions.

Transformers commonly learn word representations through pre-training the encoder on a large general dataset, which can then be adapted to a specific task. Our model will not be pre-trained on any general data, and it will learn the embeddings from scratch.

### 2.5 Multilayer Perceptron

On top of the encoder, a multi-layer perceptron (MLP) binary classifier is utilized to predict the toxicity level of the input text. The MLP classifier consists of multiple fully connected layers with ReLU activation functions and dropout regularization. During training, the model learns to minimize a binary cross-entropy loss, by adjusting its parameters using gradient-based optimization techniques such as the Adam optimizer.

The input to the MLP is a "mean representation" of the encoded input. This mean representation is computed either as the mean of the entire batch or specifically from the '[CLS]' classification-token, which essentially captures the mean of the data. The final output is a scalar logit, which can be transformed to a probability by taking its sigmoid. The binary class is attained by rounding the probability to the nearest integer.

### 2.6 Training and Evaluation

To train our toxicity detection model, we utilize labeled datasets containing examples of toxic and non-toxic language, each annotated with corresponding labels. Following model training, we evaluate its performance using a separate validation dataset. We compute key metrics such as accuracy, precision, recall, and F1 score to assess the model's ability to generalize to unseen data and accurately classify instances of toxic and non-toxic language.

## 3 Experiments

### 3.1 Baseline and Variations

We iteratively tuned the hyperparameters of our models based on the previous results in a rather arbitrary manner. The first baseline model was a simplified version of the transformer model, with significantly less encoder blocks, attention layers, and embedding size. After each result on the test data, we changed the parameters according to the accuracy and F1 score. The tables below show all the variable parameters, and the models we tested are combinations of those.

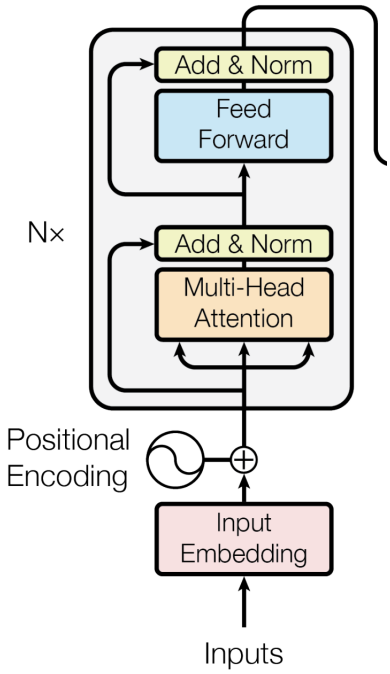


Figure 3: Encoder architecture, [9]

### 3.2 Encoder tuning

| Encoder Parameters | |
|---|---|
| **Variable** | **Values** |
| Embedding Size | 128, 256, 512, 1028 |
| Encoder Blocks | 2, 4, 6 |
| Attention Heads | 4, 8, 16 |
| Hidden Size | same as embedding size |
| Dropout | 0.1, 0.2 |

To tune the encoder, we first experimented with different embedding sizes. Since the input for the classifier will be the embedding size, we decided to keep it relatively low as either 256 or 512, to avoid overfitting. The number of encoder blocks had the largest impact on training time, but it also accounts on how much the encoder has time to learn token features. Four blocks seemed to be a good middle point. Rest of the parameters were tested by keeping the embedding size and number of blocks fixed.

We tested making the classification predictions based on either the output embedding of the "CLS"-classification token at the beginning of each sentence, versus taking the mean of the sentence. There was no significant difference between these two, and we stuck with taking the mean of the sentence.

### 3.3 MLP tuning

| Classifier Parameters | |
|---|---|
| **Variable** | **Values** |
| Hidden Layers | 2, 3, 4 |
| Hidden Size | $4*$, $8*$ embedding size |
| Dropout | 0.1, 0.2, 0.3 |

The tuning of the classifier followed a similar workflow. We tested different hidden sizes and the total number of hidden layers. The hidden

layer size was proportional to the embedding size. Also, different dropout values were tested to evaluate the chance of overfitting.

## 3.4 Training loop

We opted for BCE (Binary Cross Entropy) as our loss function. PyTorch offers a more stable alternative called BCEWithLogitsLoss, which integrates a sigmoid layer and BCE within a single class. To utilize this function, we adjusted our classifier to produce a scalar output instead of a probability. Subsequently, we manually applied the sigmoid function and rounded the result to the nearest integer to determine the binary class.

To combat the class imbalance, we added a weighing parameter for the loss function, that increases the weight for the underrepresented class i.e., the toxic class.

Finally, we used the Adam optimizer with same beta and epsilon parameters as the original transformer, and varied the learning rate between {0.1, 0.01, 0.001, 0.0001}. Higher learning rates were not able to converge, and only predicted the most common class in the batch. Lower learning rates were more stable, and possibly avoided the "exploding gradient" problem.

## 3.5 Extra experiments

### 3.5.1 CNN

We also considered using a convolutional neural network instead of the perceptron for our classification. The aim was to include spatial information to better capture meaning from word order. However, through our research we found no concrete evidence that it would perform better as multi layer perceptrons also consider word order. Convolutional neural networks are also more prone to overfitting and thus we decided not to continue with its implementation.

### 3.5.2 Detoxification

One of our original goals with the project was to find a way to 'detoxify' toxic sentences, in other words transform them into a less offensive form. This idea could have potentially been valuable in a content moderation context, to prevent the use of overly toxic rhetoric in online discussions. However, after thorough research it became clear that it would be unfeasible lest we manually go through the dataset and come up with detoxified alternatives to every toxic sentence, which for a dataset of this size seemed unrealistic. The idea was to finetune a pretrained model to learn the difference between sentences labelled as toxic or non-toxic in our specific dataset, and produce an input-output pair where the input would be a toxic sentence, and the output would be a non-toxic sentence that would semantically match the input as much as possible. We envisioned that this might have entailed switching out as few characters as possible from the sentence to transform the 'toxic' part into something less toxic (e.g. "you suck" into "you rock"), using the vocabulary to ensure that the transformed sentence contains real words. It might have been possible to find some vocabulary list of insults and common words and use those to replace toxic words. In practice this approach may not have worked for all entries as for many of them the 'toxicity' was more ambiguous in nature, such as political discourse as referred to in Figure 1. The T5 (Text-to-Text Transfer Transformer) [8] seemed like the most interesting candidate for a pretrained model to be used for this task. It is a encoder-decoder model that is trained on a multitude of language tasks and works out-of-the-box by adding a prefix corresponding to the task in the input. Its text-to-text framework transforms any problem into a text input and output pair - meaning that the input acts as conditioning for the model. This seemed close to our modelling goal. Unfortunately it is a supervised model which as mentioned would have required samples of detoxified sentences.

### 3.5.3 Text Generation

As a workaround for the detoxification we wanted to finetune a text generation model to be able to produce toxic and non-toxic text according to the labels in our dataset. This model would take as input a toxicity label and use this to generate a corresponding sentence. By finetuning a pre-

trained model we would be able to use its large context and thus achieve accurate text generation that takes into account the specific type of toxicity in our dataset.

A similar project by [3] closely aligned with our goals for this task and thus we read their implementation as background research. They finetuned gpt-2 to learn the tweet style of a specific twitter user, and used it to generate new tweets. Gpt-2 [7] is an unsupervised transformer model that is capable of a wide range of language generation tasks thanks to its 1.5 billion parameters.

Inspired by the [3] project structure we added a start of text and toxicity label as the input. The expected output would then be a sentence that is either toxic or non-toxic, depending on which label was used as input.

Additional preprocessing on the dataset was needed, including adding start of text and end of text labels, and adding the toxicity label directly to the beginning of the text in order to feed it to the model for finetuning. Normalization and unicode normalization was applied like for the classifier. This task also required creating a custom dataset class which processed the text inputs into tensors line by line. Originally the transformers TextDataset class was used, however that did not correctly tokenize the data as samples contained multiple sentences that were cut off according to the block size.

The collate function used was DataCollatorForLanguageModeling from transformers and we also used the transformers Trainer class instead of implementing our own training loop.

The number of training epochs was set to 4, and the batch size to 8. The main reason for having such a large batch size was to speed up the training time. For a similar reason the training set was truncated to only the first 10k samples. It is possible that reducing the training set so drastically has negatively impacted the quality of the results. As we also did not define an attention mask for this task, it may have also contributed to unexpected results.

For the predictions, temperature was set to 1. Gpt's temperature parameter controls randomness, or in other words how "creative" the model gets with the outputs. A temperature of 1 is the maximal value. For this task it was chosen as we wanted there to be as much variation in the outputs as possible. The top_p value of 0.95 means that tokens comprising the top 95% probability mass are considered. Some articles suggested that temperature and top_p should not be used together, however we decided on keeping both as 0.95 still includes most of the tokens and thus would not hinder the temperature predictions too much.

# 4 Results

## 4.1 Classification results

The primary metrics we focus on during evaluation are the accuracy on the validation dataset and the F1 score, which provide insights into the model's overall performance and the performance in particular cases. As shown in Figure 4.1, the most effective model among those tested is the baseline LR variation, with a learning rate of 0.001. Subsequently, the LR & MLP variation demonstrates promising results, followed by the LR variation with a learning rate of 0.0001. Surprisingly, the baseline model and the LR variation with a learning rate of 0.01 exhibit identical performances.

It's evident from our findings that the baseline model has notable deficiencies. Despite achieving a somewhat acceptable validation accuracy, its F1 score suffers significantly, primarily due to the presence of the exploding gradient problem. This issue prevents the model from learning effectively, resulting in false predictions. From visually inspecting the prediction output we can see that all of the sentences are non-toxic. As previously noted, approximately 64% of our data is non-toxic text hence if a model predicts all of the text to be non-toxic it can lead to deceiving conclusions about the model's performance. This also explains why the model sometimes get a F1_score of 0 since the F1_score can be defined as:

$$\text{F1\_score} = \frac{\text{recall} \cdot \text{precision}}{\text{recall} + \text{precision}} \tag{1}$$

In this case the precision is 0 leading to the F1 score of 0. In Section 5, we discuss on why such

| Model | Changes | Performance |
|---|---|---|
| Baseline | None | Acc: 64%, F1: 0.00 |
| LR Variation | lr = 0.01 | Acc: 64%, F1: 0.00 |
| LR Variation | lr = 0.001 | Acc: 92%, F1: 0.89 |
| LR Variation | lr = 0.0001 | Acc: 84%, F1: 0.80 |
| LR & MLP Variation | lr = 0.001, 8 * num_features | Acc: 88%, F1: 0.84 |

Figure 4: Model Results

errors maybe have occurred in our model.

### 4.2 Text generation results

| **Toxic text output** |
|---|
| <\|startoftext\|><\|1\|> they should all be held to account..... |
| <\|startoftext\|><\|1\|> what a stupid comment. if i had that much my children would kill me. a couple bucks that would have to pay for something they didn't need. |
| <\|startoftext\|><\|1\|> it is not just the press and the liberal media who hate trump. it is the political class of America that hates the media and this is the key to destroying the country |
| **Nontoxic text output** |
| <\|startoftext\|><\|0\|> they should all be held to account..... |
| <\|startoftext\|><\|0\|> maybe he should be more than just an admitted racist. |
| <\|startoftext\|><\|0\|> it is not just the government that does not have the resources to manage. it is the community that has to step up and take responsibility for managing the community. so much so that the residents are forced to have their elected representatives look after their citizens and do what is right for them. there is an ongoing effort to reduce the burden of the people and make people feel like something exists no matter what. it s not about them not them. it s about them. |

The loss function used was the default loss function of the Trainer class, and it was calculated as 3.33 for evaluation and 2.8 for training. The final loss being above 1 is a sign that the model does not perform as it should. However, it appears from the output that the model does successfully produce text that is not only coherent but that would also potentially be classified as toxic by human annotators. The nontoxic output on the other hand is slightly more questionable, and it is interesting that the first generated output is the exact same as for the toxic output.

## 5 Discussion

### 5.1 The amount of training data

When we decided to train our own model from scratch we had to accept some tradeoffs in model performance simply due to the computational resources and data that was available to us. Typically, transformers use extremely large text corpora for pretraining. As mentioned above, the gpt-2 model is 1.5 Billion parameters. Gpt-3 [2] on the other hand has 175 Billion parameters, and the original training set is estimated to contain 45 Terabytes of compressed plaintext. The revolutionary performance of the gpt models is directly tied to their massive training data sizes. Even BERT [4] was trained on 3 Terabytes of data. Through learning as much language as possible they are able to handle ambiguity and a large variety of language understanding tasks. In comparison, our model having trained on only 23 Megabytes (or 90k lines of text) pales in comparison. Thus it is only expected that its accuracy does not reach that of large pretrained models.

## 5.2 Overfitting

Our models may have encountered overfitting issues, demonstrated by training accuracies reaching 97-99%, while prediction accuracies and F1 scores on unseen test data were notably lower. This indicates potential problems stemming from either an excessive number of epochs in our training loop or preprocessing steps that didn't generalize well to new data. Consequently, it's likely that some models overfitted to the specific patterns within the training data. This is supported by manual validation and word cloud analysis, revealing prevalent themes related to U.S. politics. As a result, the model might have learned to associate certain political affiliations, such as Trump, with toxicity.

## 5.3 Vocabulary Size Reduction

Our early models faced a significant challenge due to its overly large vocabulary size. Despite the vocabulary's extensive range of words, the trained model failed to recognize a big portion of tokens in the test set. In worst cases, some sentences were filled completely with "unknown" tokens, essentially providing no information about the sentence, resulting in random guessing.

One reason behind the large vocabulary could be explained by the informal nature of online communication and the prevalence of misspellings. A single word can occur in various forms, including slang, abbreviations, or typographical errors, thereby increasing the necessity for more advanced preprocessing methods. Simply treating each word as a distinct token proved insufficient in capturing the semantics of many sentences.

In comparison, top-performing transformer-encoder models pretrained on a single language typically maintain a vocabulary size less than 40,000 [9]. For instance, BERT has a vocabulary of 37,000 tokens, nearly half the size of our model's vocabulary [4]. Most transformer encoders utilize a technique known as subword tokenization, such as WordPiece or Byte-Pair Encoding, where the vocabulary is constructed based on the most common subword units found in the training set, starting from individual characters. This approach enables the model to recognize new words in the test data by breaking them down into subword units according to the subword vocabulary.

To combat the overly large vocabulary size, we tested removing tokens below some minimum frequency. We settled on removing all tokens that occurred less than ten times. This reduced the vocabulary size from 60k to around 15k, resulting in a substantially sped up training process, where the quadratic time complexity of the encoder seems to be the bottleneck. Removing the infrequent words could also decrease the chance of overfitting, which is a common problem for transformers.

We attempted to implement Byte-Pair-Encoding subword tokenization from scratch in our training process after identifying the problem. However, with only a couple days left to finish the project, and our implementation taking approximately 20h to process all training, test, and validation data, we were not able to finish BPE. We should have done more research on the preprocessing steps at the beginning, but it was hard to identify such problems before our models were ready, as there has not been much relevant research about overly large vocabulary sizes. Subword tokenisation could have improved the final score by a large margin, as many of the mislabeled sentences had only a few known tokens by the vocabulary.

## 5.4 Future Iterations

It is clear from the performance of our model that there is something lacking in it compared to the efficiency of pretrained models. The first step to increase the robustness would be to follow in the steps of the pretrained models by giving our model more data to be trained on.

Data augmentation is a very promising way of having more data. By using data augmentation techniques, we can synthetically generate additional samples, effectively expanding the diversity of the dataset without necessitating the collection of additional data. For instance, recent advancements in data augmentation have en-

| Example | Test data input | Tokenized |
|---|---|---|
| 1 | I couldn't disagree more with this column; Canadians have moved on and don't care... | [CLS, disagree, more, with, this, have, moved, on, and, care,...] |
| 2 | I get the odd feeling Klastri the head of the ACLU of... | [CLS, get, the, odd, feeling, ", the, head, of, the, of, ",...] |
| 3 | "Had"... and is now outed as a hypocrite. | [CLS, ..., "had", ..., and, ..., is, now, outed, as, a, EOS] |

Figure 5: Example: Too many words are lost in the translation, suggesting of incorrect preprocessing steps

abled the generation of knowledge-specific data, tailoring the augmentation process to introduce variations relevant to the domain of interest [5]. This method can help in diversifying our dataset and exposing the model to a broader spectrum of variations and contexts.

Another way to improve our model as previously discussed would be to use transfer learning. By using a pretrained model that was trained to an adjacent task would help improve our model [1]. By fine-tuning pretrained models or using them as feature extractors, we can help our model with in understanding of language structures and semantics, facilitating more effective learning and adaptation.

In addition, knowledge specific textual perturbations are also a great way to improve model performance. By exposing the model to perturbed inputs designed to mislead its predictions, we can train it to become more adept at discerning genuine patterns from noisy inputs [10].

As a binary classifier our model also doesn't take into account varying nuances of toxicity. A model that not only would detect toxicity but also measures how toxic a piece of text is would be the next step up for these models.

As previously mentioned in the report, some of these techniques were attempted but our efforts were thwarted due to time constraints. For future iterations, these techniques can be leveraged along with BytePair Encoding to help improve our models perfomance that could possibly rival that of pretrained models.

## 5.5 Text generation

The output of our text generation model is a promising first step, but based on the loss values it can be said that this first iteration of the model is still at a very experimental level. Based on our interpretation of the output it also appears that it has learned certain associations with more left-leaning opinions labelled as non-toxic (the "admitted racist" output presumably referring to Donald Trump), and right-leaning opinions being labelled as toxic (the "liberal media" sentence). Due to the lack of any other evaluation metrics or benchmarks for this task it is also difficult to say with conviction whether it performs as expected. It also appears that the vocabulary constructed by the tokenizer is 50k, which is definitely too large. It could be that similarly to our classifier, there has been an issue with misspellings and slang words being added separately to the vocabulary. A visual inspection of the preprocessed data reveals that punctuation had escaped the normalization step and thus it can be assumed that special characters have also been added to the vocabulary. Creating a custom vocabulary with constraints could have solved this issue.

As a first look into the world of generative AI this task was extremely insightful. While generating toxic content may not have a place in content moderation, it might be useful for e.g. a data augmentation task as mentioned above. With some tweaks in the model construction and more robust evaluation metrics this task could potentially reveal a lot of new insight on toxic discourse. It is unfortunate that due to time constraints in this project and much time spent

on researching the more ambitious detoxification idea, we were not able to further this concept beyond the first experimental stages.

# 6 Conclusion

Our report outlined a working model for toxicity detection on a labelled dataset. We delved deep into the construction of both a transformer and an MLP neural network. As both parts of the model were programmed from scratch, we provided insights into every decision that was made in their construction. In our experiments section we detailed different tuning parameters and our reasoning for them.

Our lr variation model shows potential with 92% accuracy and an F1_score of 0.89. Although our model may not reach the performance of large pretrained models there is still ample room for growth. Future iterations of our model hold a lot of promise.

As part of extensive experiments we also researched other toxicity related language tasks, and ended up finetuning a pretrained gpt-2 model for text generation. The output of the model indicates that such a model may successfully learn the difference between toxic and non-toxic text and may be used to generate such in future research.

# 7 Division of Labor

Mikael: Researched, modelled, and programmed the encoder-neural network model. Researched and chose the variable parameters, optimizer and loss function. Planned and wrote the pipeline which includes loading the data, constructing the vocabulary, setting up the models, entering the training loop, calculating the accuracy and F1-score on a dataset, saving the model, and making the predictions. Made tests to ensure that dataloader functions correctly. Worked on the collate function, added preprocessing steps for the dataloader, and wrote the vocabulary trimming function. Implemented byte-pair-encoding preprocessing, that wasn't used in the end.

Marjaana: Implemented custom dataloader, which included the vocabulary class, preprocessing functions, dataset class, and a collate function. Wrote the code for loading the data and constructing the vocabulary. Coded a function to translate encoded sentences back to text, as well as tests for the dataloader steps. Coded the wordcloud script. Researched and attempted to implement the detoxification concept. Researched and implemented finetuned text generator.

Syed: General code debugging, getting the code to work and code deployment on the Aalto Triton cluster with GPUs. Running all the models along with hyperparameter optimization, running model predictions and checking model performance. Researched model performance deficiencies and how to increase robustness.

# References


[1] Zaid Alyafeai, Maged Saeed AlShaibani, and Irfan Ahmad. "A survey on transfer learning in natural language processing". In: *arXiv preprint arXiv:2007.04239* (2020).

[2] Tom B. Brown et al. *Language Models are Few-Shot Learners*. July 22, 2020. arXiv: 2005.14165[cs]. URL: http://arxiv.org/abs/2005.14165 (visited on 04/25/2024).

[3] Boris Dayma. *huggingtweets/huggingtweets-demo.ipynb at master · borisdayma/huggingtweets*. GitHub. 2020. URL: https://github.com/borisdayma/huggingtweets/blob/master/huggingtweets-demo.ipynb (visited on 04/25/2024).

[4] Jacob Devlin et al. *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding*. 2019. arXiv: 1810.04805 [cs.CL].

[5] Steven Y. Feng et al. "A Survey of Data Augmentation Approaches for NLP". In: *arxiv* (Jan. 2021). DOI: 10.18653/v1/2021.findings-acl.84. URL: https://doi.org/10.18653/v1/2021.findings-acl.84.

[6] Noam Lapidot-Lefler and Azy Barak. "Effects of Anonymity, Invisibility, and Lack of Eye-contact on Toxic Online Disinhibition". In: *Computers in Human Behavior* 28.2 (Mar. 2012), pp. 434–443. DOI: 10.1016/j.chb.2011.10.014. URL: https://www.sciencedirect.com/science/article/pii/S0747563211002317?casa_token=JamGdC8HQnwAAAAA:ZjjxLjMKo5UMoLy48rXYWd7AvsF_dYRREm4LLexlAy8LVcg6zkrkifUx-flYJWrcORxhDgwRE4Q.

[7] Alec Radford et al. "Language Models are Unsupervised Multitask Learners". In: (2018).

[8] Colin Raffel et al. *Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer*. Sept. 19, 2023. arXiv: 1910.10683[cs, stat]. URL: http://arxiv.org/abs/1910.10683 (visited on 04/17/2024).

[9] Ashish Vaswani et al. "Attention Is All You Need". In: *CoRR* abs/1706.03762 (2017).

[10] Yunxiang Zhang et al. "Interpreting the robustness of neural NLP models to textual perturbations". In: *arXiv preprint arXiv:2110.07159* (2021).

# 8 Appendix

Github link:
https://github.com/smaikkeli/snlp_toxicity