UTRECHT UNIVERSITY

Department of Information and Computing Science

---

**Msc. Thesis Artificial Intelligence**

# Performance of LLM-written text detectors across domains and under adversarial attack

**First examiner:**

Prof. Dr. Albert Gatt

**Second examiner:**

Dr. Marijn Schraagen

**Candidate:**

Sjors Lockhorst

Student #: 2165112

June 26, 2024

# Acknowledgements

I would like to thank Prof. Dr. Albert Gatt for his mentorship and guidance throughout my thesis project. From the initial idea to the very end, Albert's guidance and wisdom has helped, guided and inspired me tremendously. Thanks to Dr. Marijn Schraagen for agreeing to be my second supervisor and for his valuable insights and questions during my proposal phase.

Thanks to Sjoerd and Leonardo from my thesis group for our weekly meetings. Sharing perspectives on our similar yet distinct topics was extremely valuable and interesting. Thanks to Frenk and Raoul for our shared study session and peer support during this thesis project. Thanks to Will and Thomas for our AI discussion group meetings. One such meeting inspired a section of this thesis.

A big thanks to my friends and family for supporting me on this journey. Thanks to Malique for coming up with the plan that eventually led me to where I am now, and for hosting me to work on my thesis in Zürich. Thanks to Stefan for taking such great interest in my work, and for lending me his GPU to run the first test experiments on. Thanks to Kristina for our shared thesis session in the library in Amsterdam.

This section is hardly big enough to express my gratitude to the love of my life, Sara. She's inspired and motivated me to pursue this Masters, and believed in me at times when I didn't. She's been there for me every step of the way. I dedicate this work to her.

**Abstract**

Large Language Models (LLMs) have greatly improved the diversity and quality of machine-generated text. So much so that humans score at chance when distinguishing human-written texts from LLM-generated texts. Associated risks include accelerating phishing, disinformation, fraudulent product reviews, academic dishonesty, and spam. Detecting LLM-generated text could prove crucial in mitigating these risks. Many detectors have been proposed, however, past work has mainly focused on building detectors within one domain, on the output of one LLM. The most performant detector seems to be a fine-tuned masked Language Model (LM) with a classification head. But these detectors struggle with several issues such as lack of interpretability, difficulty in generalizing to unseen domains, and lack of robustness to adversarial attacks. This study sheds a light on the performance and robustness of various LLM-generated text detectors across 10 different domains, as well as investigate if robustness can be improved through data augmentation. We provide interpretable baselines for each domain, as well as a comparison between a fine-tuned LM trained on all domain data and an in-domain fine-tuned LM. We first show that a fined-tuned LM detector trained on multiple domains indeed has trouble generalizing to an unseen domain. We then show that performance of various detectors varies between domains. In some domains a detector trained on all domains leads to better performance, while on others fine-tuning within domain is better. We then attack detectors in different domains with a character level attack and paraphrasing attack, and show that models are of variable robustness depending on the domain. We finally show that our fine-tuned LM detector trained on student-written essays, can be made robust to character level attacks through data augmentation, most effectively by adding paraphrases to the training data.

# Contents

# 1. Introduction

Recently, Natural Language Generation (NLG) models models have taken a significant step forward in the diversity and quality of machine-generated text. Large Language Models (LLMs) are the current state-of-the-art (SOTA) NLG models. LLMs are neural language models based on the transformer architecture [1]. The popularity of these LLMs surged when OpenAI's ChatGPT was made available to the public [2]. Its excellent performance, chat bot form, easy-to-use web interface and free usage made it the fastest growing consumer application in history [3]. Other LLMs were quickly made available to the public, either by open-sourcing their weights or by deploying them behind an API [4]. The combination of the capabilities and widespread access to these models, could however pose several threats including but not limited to; accelerating phishing, disinformation, fraudulent product reviews, academic dishonesty, and spam [5]. Detecting whether a given text is written by a human or by AI could prove vital for mitigating these risks [5]–[7]. A reason for concern in detection of LLM-generated text, is that humans are currently performing at chance at the task of labeling whether a text is generated by a LLM or written by a human [8], [9]. It seems our only hope is to create automatic systems that can detect the increasingly subtle differences between human and LLM-written texts.

Such LLM-written text detectors exist in many varieties. However, many existing detectors have been trained and evaluated only on specific domains of text [10], [11]. A 'domain' of text is roughly defined as a corpus of text gathered from a specific source, e.g. Wikipedia, PubMed, news articles from a certain newspaper, or a certain subreddit. Studies have shown that detectors struggle to generalize beyond the domain of their training data, which makes it unlikely that such domain specific detectors will generalize well to other domains [11]–[13]. Fine-tuned language models (LMs) seem to attain high accuracy within domain when fine-tuned on that domain [8], [9],

but they are susceptible to adversarial attacks [12], [14]–[16]. Adversarial attacks can noise the text at character level (e.g. strategically adding typos [17]), at word level (e.g. similar word substitution [18]), or at the sentence/paragraph level (e.g. paraphrasing attacks [15]). Such adversarial attacks have shown to decrease the accuracy of various detectors to below chance level [14]–[16], [19].

It is thus important to identify reliable and robust detection methods for LLM-generated text, and to map out how this varies across domains of text. Reliable, meaning we want a reasonable trade off between false positives and true positives. Robust, since potential bad actors have control over the input to the detector, and might be incentivized to perturb their LLM-generated text such that detection is evaded (see Appendix C for a list of commercial tools that aim to do just that). Can such a reliable and robust detector be created across the board for all text, or do we have a better chance at creating one detector per domain? Furthermore, it should be investigated if detectors robustness to adversarial attacks could feasibly be improved by data augmentation. This study will aim to address these issues by answering the following research questions:

1. How well do LLM-generated text detectors perform across different domains of text?
    1.1 How well do LLM-generated text detectors perform out-of-distribution?
2. How robust to adversarial attacks are detectors, and does this robustness vary across domains of text?
3. Does robustness to adversarial attacks of LLM-written text detectors improve when attacked texts are included in their training data?

## 1.1 Overview

In this study we will explore the detectability of LLM-generated text, by domain and under adversarial attack. First of all, a literature study is conducted, (chapter 2). It will establish a basic understanding of how LLMs

are trained and how they generate texts. Then, a brief overview of LLM-generated text detection will be discussed, touching upon human performance, rising questions around theoretical feasibility, and various automatic LLM-generated text detectors. Lastly different types of adversarial attacks and their effects on the LLM-generated text detection task will be discussed. This literature study is followed by a method section (chapter 3), which explains the experimental setup of the conducted experiments. Choices for the dataset, detection models, adversarial attacks and data augmentation are explained. The results section (chapter 4) guides the reader through the most important experimental results. These results are then discussed and the research questions are answered (chapter 5). Finally, limitations and suggestions for future work are discussed (chapter 6).

# 2. Literature review

The literature review touches upon different topics. Section 2.1 will review the fundamentals of large language models, to get a good understanding of how the models work that generate the input to the detectors. Section 2.2 will outline the LLM-generated text detection task. Section 2.3 will cover various adversarial attacks that can be performed against detectors, and to what effect. Section 2.4 will cover previous work on the influence of domains on detectability of LLM-generated text. Finally section 2.5 will conclude the findings of the literature review.

## 2.1 Large Language Model Fundamentals

As of the time of writing, the overwhelming majority of state-of-the-art NLG models are based on the transformer architecture [1] [5]. We will focus specifically on the transformer-based LLMs, since LLMs currently produce the highest quality text [4]. The upcoming overview will be a brief summary of some important concepts related to LLMs and is based on the excellent LLM survey by Zhao et al. [4]. Understanding some key ideas in how LLMs are trained, and how they generate text, will prove to be vital in understanding different detection approaches discussed in Section 2.2.

### 2.1.1 Tokenization

Tokenization can be seen as "the initial phase in NLP" [20]. Tokenization is the process of splitting sentences into individual sub units of text, called tokens. These tokens are then seen as the vocabulary, and each token in the vocabulary is represented with a unique integer identifier. Tokens were initially assumed to be words, defined as 'space-separated substrings', as is common in many European languages. However modern tokenizers view tokens as sub-words [21]. Moreover, tokenizers are increasingly learned

from data rather than defined by a set of rules. Byte-pair encoding (BPE) [22] adjusted for natural language [23] is often used for this purpose. The token set starts of as each character that occurs in a corpus. BPE then finds the most common pairs and merges these characters into a new token. This happens iteratively for a certain amount of steps, which yields the final token vocabulary. Byte-level BPE (BBPE) [24] is a variant of BPE that uses bytes rather than characters as atomic units of text, which was used by GPT-2 [21]. A variation on BPE tokenization is WordPiece [25]. WordPiece merges not based on the total frequency of byte pairs, but on character $n$-grams that maximize the probability of the data. WordPiece tokenization is used in the BERT family language models [26], [27]. SentencePiece [28] is a software library that provides tokenization based on BBPE combined with a unigram Language Model. It picks the byte-pairs based on their overall likelihood in a simple unigram language model. This implementation also includes tokens that span across word boundaries, making it a universal tokenizer for all languages and thus popular for multilingual language modelling, since many languages don't have natural white space delimiters between words (e.g. Chinese, Japanese).

### 2.1.2 Pre-training

The tokens resulting from tokenization can be used to train a language model. While LLMs can be trained to perform many tasks, we will focus on the autoregressive language generation task. Such generative language models predict the next token based on a sequence of previous tokens, also referred to as the 'context'. The process of training an LLM on this next token prediction task, is commonly called pre-training. Formally, an autoregressive language model in pre-training will maximize the objective function:

$$\mathcal{L}_{LM}(\mathbf{x}) = \sum_{i=1}^{n} \log P(x_i|\mathbf{x}_{1:i-1}), \tag{2.1}$$

where $\mathbf{x} = \{x_1, x_2, \cdots, x_n\}$ is a sequence of tokens, $x_i$ is the current token to predict, and $\mathbf{x}_{1:i-1} = \{x_1, x_2, \cdots, x_{i-1}\}$, is the sequence of all preceding tokens also referred to as the context. Essentially, the objective is to model the likelihood of a next token given the context tokens, as closely to the training data distribution as possible.

Pre-training is usually done by having the model generate the distribution $P(x|\mathbf{x}_{1:i-1})$ for all $x \in \mathcal{V}$, and calculating the cross-entropy loss between this distribution and the actual next token in the text. This loss is then back-propagated through the network, tuning the parameters towards more accurate prediction.

This language modelling objective is the same for previous implementations of language models such as $n$-gram models or RNNs. What sets LLMs apart from these approaches is enormous scale at which they are trained, hence the name **Large** Language Model. This is mainly due to the ability to parallalize the training of transformer-based models on modern high-performance GPUs, and the widespread availability of text data to train on. This allows for the efficient training of models with hundreds of billions of parameters or more, on huge datasets spanning from books and web data to code. This scaling up of the number of parameters, amount of computational power and dataset size has shown in practice to greatly improve performance [4].

### 2.1.3 Decoding

Once pre-training is done, the pre-trained language model can be used to autoregressively generate text. This process of leveraging the language model to generate text, is called decoding. Formally, we want the LLM to complete the context $\mathbf{x}_{1:m}$ with some new text of $n$ tokens, to obtain the full text $\mathbf{x}_{1:m+n}$ [29]. It's assumed that models compute the joint probability $P(\mathbf{x}_{1:m+n})$ using

the unidirectional left-to-right decomposition of token probabilities:

$$P(\mathbf{x}_{1:m+n}) = \prod_{i=1}^{m+n} P(x_i|\mathbf{x}_{1:i-1}), \tag{2.2}$$

This unidirectional left to right decomposition sets LLMs apart from language models like BERT [26], which predict a masked token in a sequence in a bidirectional fashion.

#### 2.1.3.1 Greedy

Perhaps the most intuitive decoding method would be to pick the token with highest probability at each decoding step $i$:

$$x_i = \underset{x \in \mathcal{V}}{\text{argmax}} P(x|\mathbf{x}_{1:i-1}). \tag{2.3}$$

Selecting the highest probability token at each step however, might overlook sequences of tokens that are more probable overall [4]. Selecting the most probable next token at each step is rarely used as it can only generate one sequence of tokens for each given input, and the generations tend to be repetitive and of low-quality [30].

Intuitively we would thus like to find the most probable sentence overall. With a large enough vocabulary this is unfortunately intractable for transformer based models [29]. Beam search [31] is a heuristic method to trim down the combinatorial space of possible highest probability sentences. While variants exist, the basic idea of beam search is a heuristic breadth-first search. At any given step in the search tree, only the $\beta$ nodes with the highest heuristic value (probabilities in our case) are kept at any given point in the search tree [32]. If we would thus set $\beta = |\mathcal{V}|$, we would end up exploring the entire combinatorial space and if we would set $\beta = 1$, we would effectively get equation 2.3. Once beam search has finished, it

returns the highest overall probability for that given beam size $\beta$. However as it turns out, the most probable sentence might not be ideal for generation. It's been shown that for purposes of open-ended text generation, beam search tends to produce common phrases and repetitive text from the training set [33].

#### 2.1.3.2 Sampling based approaches

A more commonly used decoding technique for open-ended text generation, involves sampling the distribution at each decoding step $i$

$$x_i \sim P'(x|\mathbf{x}_{1:i-1}), \tag{2.4}$$

where $P'(x|\mathbf{x}_{1:i-1})$ is some distribution derived from $P(x|\mathbf{x}_{1:i-1})$.

One approach would be random sampling, sampling a random token out of the entire distribution. We would simply apply equation 2.4, where $P'(x|\mathbf{x}_{1:i-1}) = P(x|\mathbf{x}_{1:i-1})$. Completely random sampling can however generate very unlikely words, which can lead to errors. Especially when using sub-word tokenization, where for example a word *can't* might be tokenized as tokens *ca* and *n't*. A model might produce *ca* but not the subsequent *n't* to complete the full word creating ungrammatical generations [33].

A way of limiting the likelihood that such errors occur during decoding, is top-$k$ sampling [33]. At each time step $i$, only the $k$ most likely tokens given by $P(x|\mathbf{x}_{1:i-1})$ are sampled. Let the set of the $k$ most likely tokens be $\mathcal{V}^{(\text{top-}k)}$. Now the probabilities for the top-$k$ tokens won't sum to one, so they need to be rescaled. This is done by dividing all the token probabilities for each token in $\mathcal{V}^{(\text{top-}k)}$ by the sum of their probabilities

$p' = \sum_{x \in \mathcal{V}^{(\text{top-}k)}} P(x|\mathbf{x}_{1:i-1})$, giving:

$$P'(x|\mathbf{x}_{1:i}) = \begin{cases} P(x|\mathbf{x}_{1:i-1})/p' & \text{if } x \in \mathcal{V}^{(\text{top-}k)}, \\ 0 & \text{otherwise.} \end{cases} \tag{2.5}$$

We then sample following equation 2.4. Since this $k$ is a constant value this might not be optimal for all contexts . If $k$ is small, it might create generic and bland contexts, while if $k$ is large the options start to include tokens that have low likelihood [29].

Nucleus sampling [29], also called top-$p$ sampling, offers a solution by having the distribution to sample from $P'(x|\mathbf{x}_{1:i-1})$ be of dynamic size relative to the original distribution $P(x|\mathbf{x}_{1:i-1})$. Top-$p$ sampling, samples from the smallest set of tokens that has a cumulative probability above $p$. Formally, given a distribution $P(x|\mathbf{x}_{1:i-1})$, the top-$p$ vocabulary $\mathcal{V}^{(\text{top-}p)} \subset \mathcal{V}$ is the smallest strict subset of $\mathcal{V}$ such that

$$\sum_{x \in \mathcal{V}^{(\text{top-}p)}} P(x|\mathbf{x}_{1:i-1}) \geq p. \tag{2.6}$$

Let $p' = \sum_{x \in \mathcal{V}^{(\text{top-}p)}} P(x|\mathbf{x}_{1:i-1})$. Now we sample according to equation 2.4, where

$$P'(x|\mathbf{x}_{1:i}) = \begin{cases} P(x|\mathbf{x}_{1:i-1})/p' & \text{if } x \in \mathcal{V}^{(\text{top-}p)}, \\ 0 & \text{otherwise.} \end{cases} \tag{2.7}$$

Along with sampling-based methods, a temperature parameter is often introduced to the decoding phase. Temperature sampling is a technique used to adjust the probability distribution from which words are sampled.

This adjustment can either increase the likelihood of sampling more probable events or allow for more diversity by giving less likely events a higher chance of being chosen [29]. The temperature parameter $t$ typically ranges from $(0, \infty)$. When $t = 1$, the distribution remains unchanged, representing standard sampling. As $t$ decreases towards 0 (but not reaching 0), the model increasingly favors higher probability events, leading to more predictable and less diverse outputs. Conversely, as $t$ increases above 1, the model starts to give more weight to less likely events, enhancing diversity at the cost of coherence, increasing the likelihood of errors. This temperature mechanism can be used in conjunction with any of the previously mentioned sampling methods. It is commonly employed in SOTA models, often in combination with top-$p$ sampling (e.g. LLama2 uses $p = 0.9, t = 0.1$ [34]). The temperature parameter thus controls the trade-off between the diversity and the quality of the generated text.

### 2.1.4 Conditional text generation

Once the pre-training phase has been completed, and a decoding strategy is chosen, a LLM can generate text. The generation will be an autoregressive continuation of a given sequence, the context $\mathbf{x}_{1:i-1} = \{x_1, x_2, \cdots, x_{i-1}\}$. At generation step $i$, the probability distribution $P(x|\mathbf{x}_{1:i-1})$ will be obtained from the LLM, and the chosen decoding strategy will produce token $x_i$. This process is repeated, but the new context is now $\mathbf{x}_{1:i} = \{x_1, x_2, \cdots, x_{i-1}, x_i\}$, the old context including the previously generated token. This new context is then passed to the LLM to obtain new distribution $P(x|\mathbf{x}_{1:i})$, from which new token $x_{i+1}$ is decoded. This continues on until some special end of sequence marker (e.g. <|endoftext|> in the GPT-x family) is sampled, which terminates the autoregressive process.

We call generation unconditional when no initial context is given to the model. Since no context is given, the generated text can be anything and highly depends on what the model was trained on and the decoding strategy. The distribution is only skewed towards words that are often used as the first word in a sentence across the entire training set. To make the LLM

generate useful text for a specific task, it needs to be given some initial context, also referred to as a 'prompt'. A LLM that is only pre-trained, will just continue the prompt, predicting what tokens are the most likely to follow it. For a LLM to generate useful text, it needs to be able to follow the instructions given in the prompt rather than just generate likely next tokens.

### 2.1.5 Instruction following

To enable the ability in LLMs to follow specific instructions, instruction-tuning can be employed. Instruction-tuning is the process of fine-tuning a pre-trained language model on a set of natural language instructions. These instruction templates often include a task description, some (optional) examples of how to execute the task, and a formulation of the correct answer to the task. Datasets containing such formatted instructions for instruction-tuning LLMs exist in many varieties [4], [35]. These datasets can vary based on what instructions they contain, and how they are formatted. Some datasets are specifically tailored to chat data for example, to make LLMs that are instruction-tuned on it respond to prompts in a chatbot fashion. Datasets initially all comprised of human-written data but are increasingly synthetically generated by LLMs. Fine-tuning a pre-trained language model on these natural language instructions, nudges the model's next token predictions towards generating texts similar to those in the instruction dataset [35]. Given task instructions in natural language form, LLMs have been found to quickly generalize to handle new tasks [35].

### 2.1.6 Aligning

During pre-training, LLMs might have extracted certain textual patterns from the data that the creators of the model don't want their model to repeat in their generation, for example for ethical or legal reasons. Furthermore, researchers want the model to generate text that is of utility to potential users. LLMs are prone to generate text that is likely, but not factual. This might lead to the model generating false information, also called hallucination. To address these issues, the model should be *aligned* to human preferences. Of-

ten this process of aligning a LLM is done by means of Reinforcement Learning from Human Feedback (RLHF). In their original paper where RLHF for LLMs is introduced, Ouyang et al. describe their process of creating instruct-GPT out of the pre-trained model GPT-3 [36]. First they collect a set of written instructions from labelers. They then employ instruction-tuning, to fine-tune the model towards following those instructions. After obtaining the instruction-tuned model, they prompt the model with many different prompts. For each prompt, they have the model generate many alternative responses. Human labelers are asked to rank the responses in order of their preference. A reward model (RM) is trained to predict these human preferred outputs. The LLM is then optimized using a reinforcement learning setup. At each step, a prompt is sampled from a dataset. The LLM generates a response, which gets fed to the RM. The RM predicts a score for this LLM generated response, stating how 'human-preferred' this output is. The LLMs weights are updated with the intent of optimizing the reward from the RM. The resulting LLM generates more human-like text, and is less likely to generate text that the human labelers have discouraged. They show that RLHF is highly effective in biasing models towards high quality text. The model resulting from instruction-tuning and RLHF, Instruct-GPT, scores higher than GPT-3 across many benchmarks, while having 100$\times$ less parameters.

### 2.1.7 Summary

The above sections have laid out some relevant mechanics in how LLMs are trained, and how their generative process works. First the pre-training phase equips the LLM with a model of language, which optimizes for the next word prediction task. This has an encoding phase; learning a representation of language in the models weights, and a decoding phase; turning the probability distribution into a sequence of tokens. The most probable text isn't always what humans want LLMs to generate however [29], which is why sampling methods are preferred over greedy methods in SOTA LLMs. It is desirable for LLMs to follow human instructions based on a given prompt, and the responses to these instructions should be written in a way

that 'looks good' to human labelers. To this end, LLMs are instruction-tuned and RLHF is used to bias the LLM towards generating human preferred outputs. With this knowledge knowledge on how LLMs generate text, we move to discussing how their outputs can be detected.

## 2.2 LLM-generated text detection

LLM-generated text detection can be formulated as a binary classification task. Given a natural language text $\mathbf{x}$, an LLM-generated text detector $\mathcal{D}$ classifies the text as AI-written or human-written [6]:

$$\mathcal{D}(\mathbf{x}) = \begin{cases} 1 & \text{if } \mathbf{x} \text{ generated by a LLM} \\ 0 & \text{if } \mathbf{x} \text{ written by a human} \end{cases} \tag{2.8}$$

Furthermore there exist two settings for detection; the white-box and black-box setting [7], [37]. In the white-box setting, the detector has some level of access to the LLM, either full or partial access. This allows detection methods to leverage different parts of the model (e.g. the logits) to make predictions [11], [19]. In the black-box setting, the detector has no access to the LLM that generated the texts in our dataset. Within the black-box setting, one can either know which model has generated the texts, or this can be unknown. In a real life scenario, the most likely setting is a black-box setting with any number of unknown models that might have generated texts. Bad actors trying to pose as a human by using an LLM are unlikely to kindly provide information about the LLM they used for generation. Therefore this literature review focuses on detection methods that are at least theoretically generalizable to a black-box unknown model setting. This thus excludes detection methods like watermarking or methods using logits of a known source model etc.

### 2.2.1 Metrics

To compare detectors, we need a way of formalizing how performance is measured. Since we are dealing with a classification task, we have many performance metrics to chose from. The most basic of metric is perhaps accuracy, the fraction of the whole dataset that was classified correctly. But accuracy doesn't provide any information about the amount of false positives or true positives. A standard confusion matrix can help give insights into this, which gives all true positives (TP), true negatives (TN), false positives (FP) and false negatives (FN). Since our detector $\mathcal{D}$ labels LLM-generated texts as 1, a positive sample is a LLM-generated text, and a negative sample is a human-written text. Since this trade-off between true and false positives is important in LLM-generated text detection, the area under the curve of the Receiver Operating Characteristic (AUROC) is often reported as the most important metric [7]. This metric expresses the ratio of true and false positives across different classification thresholds for the model and is given by:

$$AUROC = \int_0^1 \frac{TP}{TP + FP} \, d\left(\frac{FP}{FP + TN}\right). \tag{2.9}$$

In automatic detectors we can see how different thresholds would impact the performance of the detector, by outputting the probability that some text $\mathbf{x}$ has a certain label, $P_{\mathcal{D}}(y = 1|\mathbf{x}) = 1 - P_{\mathcal{D}}(y = 0|\mathbf{x})$. However in human performance, there is not really such a thing as a 'threshold' for decision. So in human performance, we often see studies reporting accuracy.

### 2.2.2 Human performance

Before resorting to automatic methods, we will discuss research on the ability of humans to detect texts written by LLMs. If this is an easy task for humans, there might not be any need for automatic detectors. As we'll see

however, humans rarely score higher than chance on different variations of the detection task, illustrating the importance of developing automatic detectors.

One paper proposes a benchmarking dataset for detecting LLM-generated vs. human-written text called Turingbench [9]. The dataset consists of human-written news articles in English from different news outlets. The titles of the articles, plus the desired length of the article are then used to prompt several LLMs, with the prompt to write a news article based on the title. Human performance on the detection of LLM-generated text in the Turingbench dataset is roughly at chance.

Another study showed that the performance of humans is dependent on the decoding method the LLM uses as well as the length of the text [38]. The human-written and LLM-generated texts are not specified to be within any specific domain of text, they are just specified as coming from the training distribution of GPT-2. Humans were more accurately able to detect whether a text was human or LLM-generated, the longer the text was. However at the full length of 192 tokens, the average accuracy was still only 71.4%. Interestingly, text generated with top-$k$ sampling (with $k = 40$), is harder for humans to distinguish from human-written than top-$p$ sampling ($p = 0.96$ and $p = 1.0$). This is in contradiction to their findings for automated detectors, which have a easier time detecting top-$k$ sampling and a harder time with top-$p$.

This poor performance of humans begs the question if humans can be trained to perform better at the task. One study found that the initial accuracy of human labelers was around chance for detecting human vs. GPT-3 output [8]. They tested three domains of text: stories, news articles and recipes. Deliberately training the labelers was found to increase the accuracy of labelers to 55% across domains, which was not a significant increase in performance. Within domains there also wasn't a significant increase in labelers performance.

In summary, studies show that humans have great difficulty in discriminating between human-written and LLM-generated. Even deliberate train-

ing for this task was shown to not sufficiently improve their skill level beyond chance level. Further reason for pessimism is that many studies that have tested for human performance, have tested against the output of GPT-2. The SOTA LLMs are even better at generating human-like text, providing the intuition that this task has gotten harder rather than easier for humans. Since it is unlikely that humans will be able to reliably differentiate between human-written and LLM-generated text, there is a urgent demand for accurate, robust and reliable automatic detectors that can be used at scale. In many NLP tasks human performance is seen as the gold standard that sets the bar for NLP systems to reach. In LLM-generated text detection it merely provides a hint towards the difficulty of the task.

### 2.2.3 Can LLM-generated text reliably be detected?

Before we embark on the search for good automatic detection, we must show that automatic LLM-generated text detection is at least theoretically possible. Especially since the possibility of such reliable and robust automatic detectors has recently been questioned. One recent paper provides the 'impossibility result' in which they prove that as LLMs become more capable at mimicking the distribution in human text, reliable detection of such LLM-generated texts will become increasingly harder and inevitably impossible in the limit [16]. Further reason for skepticism about the feasibility of reliable LLM-generated text detection, is that OpenAI published a AI-written text classifier, which they revoked as of July 2023 due to its low rate of accuracy [39]. This thus begs the question, can LLM-generated text be reliably detected [16] ?

As we will find out in Section 2.2.5, many detectors leverage the assumption that a text written by a human has properties that are distinguishable from the properties of a LLM-generated text. Often this assumption is extended to assuming that human-written text follows a different distribution than LLM-generated text. A detector is essentially a discriminator between those two distributions, and it's accuracy is bound by the overlap of these distributions. Sadasivan et al. [16] argue that a reliable detector, meaning

a detector with high true positive and low false positive rate, would need a large discrepancy between human-written and LLM-generated distributions. To be precise, they prove that for a detector to have an AUROC of at least 0.9, the total variance distance between human and LLM-generated texts, should be more than 0.5. This total variance (TV) between distributions is approximated by training a RoBERTa [27] classifier on the LLM-generated classification task. The difference between the true positive and false negative rate for LLM-generated text is taken as the TV between distributions. The authors claim that it is unlikely that the TV distance between human and LLM-generated texts distributions will remain above 0.5 as LLMs become more capable at modeling the human distribution of text. They further their argument by showing that the TV between the distributions can be drastically lowered by adversarially attacking LLM-generated text through paraphrasing the text many times (see Section 2.3.3). They show that the performance of a whole range of different detectors degrades to below chance level once the text has been recursively paraphrased five times.

While an interesting and valuable insight in itself, the way the problem is framed is a vast oversimplification. The paper lumps all of LLM-generated and human text on a big pile. But this fairly raises the question, over 'which humans' [40] are we trying to generalize? In their 2024 study, Atari et al. claim that much of the existing literature largely ignores that humans are a cultural species with substantial psychological diversity around the globe that is not fully captured within the training data of LLMs. In an experiment, they let ChatGPT fill in the World Values Survey (WVS), a survey designed to monitor cultural values, issues of justice, moral principles, attitudes toward corruption, accountability and risk, migration, national security, global governance, gender, family, religion, poverty, education, health, security, social tolerance, trust, and institutions. They find that ChatGPTs performance on this survey most resembles people of Western Educated Industrialized Rich Democratic (WEIRD) countries. It can off course be argued that this is an arbitrary feature of OpenAI's own bias. But since all LLMs need textual training data that's available at scale on the internet, we

know that a large chunk of the population isn't represented in this data. As Atari et al. mention, the United Nations estimated that almost half the worlds population doesn't have access to the internet as of 2023 [40].

It is good to know that such variance between distribution has to be in a certain order of magnitude to create a robust detector, as Sadasivan et al. [16] point out in their paper. However the currently available human-written text data is clearly not representative of humankind, and neither are the people involved in creating instructions for instruction tuning, or the labelers who label LLM responses through RLHF. Any claims about the distribution of human text, should thus be taken with some level of skepticism. It seems more likely that many distributions exist, which differ across cultures, languages, domains of text, etc. Additionally, as Section 2.2.5.1 will outline, LLMs still have systematic flaws which could be exploited, even at their current scale. It is not evident that scaling up LLMs further will make these flaws disappear.

Whether reliable, accurate and robust detection is feasible remains an open question. This study will aim to address some of these questions, including specifically the impact of paraphrasing attacks on detectors and potential ways to increase robustness to them.

### 2.2.4 Human-assisted methods

Some frameworks for aiding humans in labeling texts accurately as LLM vs. human-written have been proposed. One paper proposed the Giant Language Model Test Room (GLTR) [41]. This method annotates words in the text by highlighting them, based on statistical methods that detect common artifacts in the text, across common sampling schemes. The paper shows that for detecting GPT-2 output, it boosts human performance from 54% to 72%.

Another paper proposed the Scarecrow framework, and is focused on detecting GPT-3 output [42]. In this approach, the authors have defined common errors that LLMs make based on academic findings (e.g. incoherency, bad math, hallucinated 'facts'). Using this as a starting point, they

crowd-sourced annotation to get a label set that is salient to non-expert labelers. This label set could be used to label data in the future, by relatively untrained labelers. The paper provides the community with the annotation scheme, but it remains to be seen if the accuracy of training labelers to use this labeling technique enhances their performance in detecting LLM vs. human-written text.

### 2.2.5 Automatic detection

A whole range of automatic LLM-written text detectors has been proposed in the literature, too many to include here. Some great surveys have been published that give a good overview of the many detectors that have been proposed [5]–[7], [37]. To limit the scope of this literature review, we will focus only on training-based models that can be used in a black-box, unknown model setting. We will specifically look at types of detectors: feature based detectors, which can be used as a baseline for detection and are inherently explainable, and fine-tuned LM detectors, which seem to attain highest performance while being relatively simple to implement and train.

#### 2.2.5.1 Feature based detectors

Language exhibits patterns that can be captured using statistical techniques. These statistical features are known to vary across different domains of text [43], but also across authors, allowing models to perform the task of authorship attribution [37]. Differences in stylometric features have also been found to exist between human-written text and LLM-generated text [44], which can be leveraged to predict if a text was written by a LLM or a human. Using such features makes detection transparent, explainable and offers insights into characteristic behaviour of LLMs [45]. Furthermore, it's been shown that using features alongside a neural approach leads to classifiers being more robust against adversarial attacks [14]. Features specific to the LLM-generated text detection task are formulated to capture common limitations that LLMs exhibit in general, making them ideal for a black-box setting where the source LLM is unknown. Whether a feature based detector is trained to classify the outputs of one known LLM, or multiple

unknown LLMs depends on the composition of the dataset on which a classifier is finally trained. In previous work on feature based LLM-generated text detection, both datasets with LLM-samples from multiple LLMs [45] and just one LLM [46] have been used as training data.

A simple baseline that is commonly used in NLP for classification, is term frequency inverse document frequency (TF.IDF) with unigrams and bigrams as features for a logistic regression classifier [5]. OpenAI shared such a baseline in their report when releasing the 1.5B parameter GPT-2 model. When setting temperature to 1, meaning the whole distribution is sampled at random and left unchanged, the accuracy of the baseline ranged from 88% on their 124M parameter model to 74% on the 1.5B model [46]. When top-$k$ sampling with $k = 40$, the accuracy improves to between 97% and 93% respectively. Another study tested more model sizes of GPT-2, as well as outputs of GPT-3 and Grover [45]. They report similar results as stated in the OpenAI report, larger models GPT-2 models are generally harder to detect with the TF.IDF baseline and total random sampling is easier to detect than top-$k$ sampling for $k = 40$. For the newer models GPT-3 and Grover, they report an AUROC of 83.7% on GPT-3 and 76.4% on Grover. As of the time of writing, we haven't been able to find a study that replicates a TF.IDF baseline for SOTA LLMs.

One feature of human text is that its word occurrences follows Zipf's law [47]: the frequency of a word is inversely proportional to its rank in an ordering of words by frequency. In LLMs however, this can vary according to the decoding strategy that is used [5]. The deviance of a text from the expected distribution according to Zipf's law, has been used as a feature and has been show to have some predictive power in detecting the output of a transformer-based translation tool [44], as well as the output of GPT-2 [14].

Fröhling et al. trained detectors with perhaps the most comprehensive set of features for the LLM-generated text detection task [45]. Their features can be divided up into four different types of errors LLMs commonly make.

1. Lack of syntactic diversity: Named Entity tags, Part-of-Speech tags,

and coreference clusters.

2. Repetitiveness of words: Number of stop words, unique words, and the frequency of the most common words.
3. Lack of coherence: Assesses coherence by tracking entity roles and transitions, analyzing content-word co-occurrences, and measuring topic consistency through information loss in text simplification.
4. Lack of purpose: Features from the lexicon-package empath, which analyses text across 200 gold standard topics and emotions [48].

These features are then used all together with different statistical classifiers (Logistic Regression, SVM, Random Forest and Neural Network). They continue with a neural network for further analysis, since it had superior performance on GPT-2 output of all model sizes. In further analyses they found that detectors did generalize between models with the same architecture and decoding strategy but different complexity. They found no generalizability between models with different decoding strategies. They observe that their classifier beats the TF.IDF LR baseline in detecting text that used full distribution random sampling, but underperforms the baseline when top-$k$ sampling is used. The paper also implements ensemble classifiers, which take into account both the TF.IDF LR model, as well as the feature based classifier. They show that these ensemble models do consistently outperform the TF.IDF baseline, with a 0.864 AUROC on GPT-3 and 0.805 on Grover.

#### 2.2.5.2 Fine-tuned Language Models

A common approach in LLM-generated text detection, is fine-tuning smaller language models as classifiers. Commonly, masked language models like RoBERTa [27] are used to this end. Again these approaches are in the black-box setting, where it is dependent on the composition of the dataset how many different LLMs are included. OpenAI published such a RoBERTa based LLM-generated text detector with high accuracy of approximately 95% in detecting GPT-2 text [46]. These fine tuned model thus outperform both the TF.IDF baseline as well as feature based classifiers, in the case of

Fröhling et al. by 18 percent points on accuracy [45]. The OpenAI report states that when top-$p$ decoding is used, text is hardest to detect overall but training the classifier on text generated with top-$p$ decoding, generalizes the best to other decoding strategies. The paper by Ippolito et al. [38] mentioned in Section 2.2.2, replicates the finding with their fine-tuned BERT [26] model and juxtaposes these results to human performance, which is conversely higher on top-$p$ and lower on top-$k$. They also find that training on top-$p$ generalizes best to other decoding strategies, while training on top-$k$ or random decoding doesn't generalize well enough to stay above chance level on other decoding strategies. In general, these models exhibit great performance at face value with around a 95% accuracy on different detection tasks.

However, these models lack robustness as they tend to overfit to the training data, making them of limited use across domains of text and on unseen data [6], [10], [11], [49]. The implications of different domains of text in relation to fine-tuned LM detectors will be discussed in more detail in Section 2.4. Perhaps most worrisome of all, various adversarial attacks have been shown to degrade performance below chance level, which will be discussed in Section 2.3.

## 2.3 Adversarial attacks

The detectors described in Section 2.2.5.2 are considered the SOTA LLM-generated text detectors. It has been shown however that different adversarial attacks degrade their performance to below chance level [14]–[16], [19]. This is problematic since a detector is specifically used in an adversarial setting, where a bad actor tries to pass a LLM-generated text as human-written. All serious threats of LLM-generated text as outlined in the threat model by Crothers et al. are of such a nature [5]. Bad actors are thus incentivized to use such attacks to perturb LLM-generated texts in order to evade detection.

In order to create robust detectors to mitigate such threats, it should be studied which perturbations will fool LLM-generated text detectors and

what could be done to mitigate the effects of such attacks. Without robustness to adversarial attacks, detectors will be of little utility in real-life, high risk scenarios. For example, if LLM-generated text detectors can detect ChatGPT-generated essays, but these detectors aren't robust to paraphrasing attacks, a student could just paste their ChatGPT-generated essay into a tool that paraphrases their text (e.g. one of the tools in Appendix C) and evade detection. We will review the types of adversarial attacks that exist, so we can understand how they change text, and how this could confuse detectors.

### 2.3.1 Character-level attacks

Attacks in this category operate at the character level. One type of attack is a misspelling attack [14], [37]. This type of attack leverages the fact that LLMs are unlikely to make spelling mistakes or 'typos'. Strategic typos that are commonly made by humans are added, making it more likely for detectors to classify a noised LLM-generated text as human-written. Another attack is the white space attack, which removes white space between certain words [37].

Character level attacks leverage the first part of the detectors pipeline, tokenization. Before a detector can make a prediction, it must tokenize the input, as described in Section 2.1.1. Slight variations in spelling will likely still be readable to us humans, but they will lead to different tokens being inputted to the automatic detector. Such a small variation could cause a vastly different internal representation of the sequence within the detector (e.g. the TF.IDF representation, stylometric features or weights within the fine-tuned LM), which could in turn lead to misclassification of the text.

### 2.3.2 Word-level attacks

Different methods in this category, swap words for other words, while preserving some level of semantic similarity. One such attack is TextFooler [18], which replaces words with similar words in the BERT embedding space, based on cosine similarity. Another approach is to flip the sentiment of a

word. That is, if a word is classified as having negative connotation make it have positive connotation and vice-versa [50]. Named entities can also be swapped by another irrelevant entity from the same type [50].

Word-level attacks leverage the differences in the internal state of detectors when one token or more tokens are changed in the model. Detectors might recognize certain pattern in sentences and their correlation to corresponding classes (human vs. LLM). These patterns are then broken by an adversarial attack that changes words strategically, possibly causing the model to miss classify inputs.

However, such attacks seem less plausible to be used in a real life scenario, since changing individual words could greatly effect the meaning of the sentence. Such attacks might not only fool detectors, but also perturb the text such that it no longer serves the original purpose.

### 2.3.3 Paraphrasing attacks

Paraphrasing attacks involve paraphrasing entire sequences of text, with the goal of keeping semantics highly similar. Krishna et al. [15] propose a model called DIPPER, which is a paraphrasing model that takes paragraph level context into account. They show that the performance of various detectors (including a fine-tuned LM) is decreased to below chance level when evaluating on LLM-generated texts paraphrased with DIPPER. Another approach is by Sadasivan et al., who publish their paraphrasing approach along with the impossibility theorem discussed in Section 2.2.3 [16]. They recursively rewrite a text some amount of times with a paraphrasing model, showing that after five iterations, all tested detectors have a performance below chance level on detecting the paraphrased texts. A more simple way is to use an LLM for paraphrasing through a prompt (e.g. Can you paraphrase the following text: *text to paraphrase here*), as was done by Li et al. [11]. A popular model for paraphrasing has been trained specifically to paraphrase as good as ChatGPT, by training a model on inputs and paraphrased outputs from ChatGPT [51].

Paraphrasing attacks leverage the difference in the internal representa-

tion of a detector of two sentences with different sentence structure, but equivalent semantics. A detector may have started to correlate a certain token order and certain word choices in the training data with that text being LLM-generated or human-written. Paraphrasing the text removes some of these patterns by scrambling the word order and word choice, potentially causing misclassification, while keeping semantics highly similar.

## 2.4 LLM-generated text detection and domains of text

Previous work has mainly focused on creating a detector for a single domain [10], [11]. Recently however, more work has been published about the difference in detector performance between domains. One study by Wang et al. created a multi-generator, multi-lingual and multi-domain dataset to evaluate different models [10]. They include domains such as Wikipedia, WikiHow, Reddit ELI5, arXiv, peer read, as well as some other non-English domains. They prompt various LLMs to write Wikipedia articles, to write abstracts for academic papers, to write an academic paper review based on the title and the abstract of the paper, and to answer questions from Reddit's ExplainLikeImFive subreddit. This results in a dataset with the original human-written texts, and the LLM-generated ones. They show that a RoBERTa model fine-tuned for classification is the most accurate model for in-domain classification. The accuracy varies between domains however, ranging from 0.936 on the Reddit ELI5 domain to the 0.996 on the Wikipedia domain. They show that the performance generalizes poorly to other domains that the detectors weren't trained on. The authors don't investigate the cross-domain in the multi-generator setting at the same time. All their models are only trained on the outputs of one specific model. They also don't subject any of their models to adversarial attacks to test for robustness.

Another study by Li et al. attempts to do 'deepfake detection in the wild' [11]. They too acknowledge that previous work has been too focused on de-

tection of texts in one domain from one LLM. To test detection in a more practical scenario (e.g. 'in the wild') where a detector faces texts from various domains and LLMs, they create a dataset which includes 10 different domains (Table 3.1) and 27 different LLMs (Table 3.2). They fine-tune a Longformer [52] model as a LLM-generated text detector, in various settings: fixed-domain and fixed-model, fixed-domain arbitrary-models, arbitrary-domains fixed-model and arbitrary-domains arbitrary models. Furthermore they have test beds to test the performance of a detector on unseen-models and unseen-domains. In the domain-specific cross-model setting, their Longformer detector scores on average over all domains: an AUROC of 0.99 and an average recall of 93.51%, whereas in the cross-domain cross-model setting, the Longformer detector has an AUROC of 0.99 with an average recall of 90.53%. When generalizing to an unseen domain, Longformer scores an AUROC of 0.93 with an average recall of 68.40%. They however tweak their decision boundary based on the ROC curve, which boosts the performance on unseen domain to an average recall of 81.78%. The authors conclude that the transferability of detection capabilities to out-of-distribution scenarios, including unseen domains, remains uncertain and a crucial challenge [11]. Additionally, the authors perform a paraphrasing attacks to test for robustness. They use `gpt-3.5-turbo` as their paraphraser. They paraphrase both human-written and LLM-generated texts, after which they label both as LLM-generated texts. Their Longformer detector scores an AUROC of 0.75 and an average recall of 62.92%, showing that their model isn't robust to paraphrasing. Furthermore, they make domains either 'specific' or 'arbitrary', but they don't look at individual differences between domains, and their relative performance. They also don't explore the potential variation of robustness to paraphrasing across the various domains.

Finally, a concurrent work by He et al. [49] explores the performance and robustness to various attacks of various detectors, on three domains, high-school and university level essays, posts from the subreddit WritingPrompts, and news articles from Reuters. They create a dataset for the LLM-generated text detection task, by prompting 6 LLMs (Table 3.3) to write news articles, stories, or news articles following a given headline. Unlike

in Li et al. [11], they always detect the outputs of one specific LLM. They too find that fine-tuning a LM for classification, leads to the best performance within various domains and LLMs. Their BERT-based [26] detector scores F1 scores on all domains and LLMs above 0.950, even scoring close to 1.000 on many domain and LLM combinations. They observe that detection methods trained on different datasets may have different transferability to the other datasets. For example training their fine-tuned BERT on essays and evaluating it on news articles, has an F1 score of only 0.672, while training on essays and evaluating on WritingPrompts data leads to an F1 score of 0.983. Lastly, they evaluate the adversarial robustness of different detectors on different domains, as generated by different LLMs. They attack the models using character-level attacks by inserting random whitespaces in the text with 1% probability. Word-level attacks are performed through swapping words with synonyms in an adversarial way [53], [54]. Paraphrasing attacks are performed using the ChatGPT paraphraser [51] from Huggingface. For their fine-tuned LM they find that paraphrasing the text has generally little to non effect on detector performance. Likewise inserting random whitespaces doesn't seem to fool the detector. Most effective was the word-level adversarial attack. Since the paper doesn't provide statistics about the amount of perturbations the algorithm had to perform to fool the detector, it's hard to tell if those output texts are still human readable. It is interesting to see that paraphrasing had no effect in this study. This could be because they used a different paraphrasing model, or because paraphrasing attacks are less effective against fine-tuned LMs in these specific domains.

## 2.5 Conclusion & gap identification

This literature review glanced over several important topics surrounding LLM-generated text detection. First of all we looked at how LLMs generate language. We then turned to human performance on the task of LLM-generated text detection and concluded that humans don't score better chance. We touched upon rising questions from the literature about the feasibility of the task, and determined that detection is not necessarily theoretically im-

possible. We then discussed some approaches have been tried in detecting LLM-generated vs. human written texts in a black-box setting. We continued by surveying the adversarial attacks have been shown to degrade detector performance. We closed off the literature review by highlighted some recent work on domain differences in detecting LLM-generated text.

Evidently some levels of analysis are still missing from the literature when it comes to the issue of domain and adversarial attacks. No study seems to look at the general detectability of multiple unknown LLMs which generated text within a certain domain and the differences between domains specifically. When creating for example a detector for university level essays, we know the domain of text we are expecting as input to our detector, but it is unknown which LLMs students might attempt to use to pass LLM-generated essays off as their own. While it is infeasible to include all LLMs in the training data, detector performance generalize much better to unseen LLMs than to unseen domains [10], [11], providing the intuition that training within a specific domain, on the output of multiple LLMs might have the highest chance of creating a reliable in-domain detector. Training on a small number of LLMs might also mean that a detector could overfit to the specific decoding strategy used by those LLMs, which doesn't always generalize to other decoding strategies [38]. Furthermore, by comparing how detectable LLM-generated texts are across various domains, we could learn more about the problem of detection, in general and per domain. Establishing baselines by training inherently explainable models might prove insightful here, which is missing from many existing studies.

It is clear that more research should be conducted to identify the best approach in creating a robust, reliable LLM-generated text detector. Creating one detector for all domains of text, and all possible LLMs seems challenging. The best performing models, fine-tuned LMs, struggle to generalize beyond their training data and suffer from vulnerability to adversarial attacks. More fundamentally, viewing all of human-written text as one 'human-distribution' seems like a vast simplification that likely does not hold. It is more likely that many such distributions exist, and that they vary across many dimensions, one of which is the domain that the text belongs

to. Perhaps this should inform us to turn to a more fine-grained analysis when it comes to detection. Rather than trying to create a reliable and robust LLM-generated text detector for all 'human' vs. LLM-generated text, let's first explore if we can create such reliable and robust detectors within various domains.

A benefit of building domain-specific detectors, is that they seem practically very feasible to create. One would need human-written texts representative of the domain, and access to LLMs to generate LLM-generated texts, resulting in a fully labelled dataset to train on. Fine-tuning a LM as a detector on such a dataset is very feasible and cost effective due to the low price of compute and the relatively small model size. It would however then be important to know how reliable these detectors are for specific domains, and how robust these in-domain detectors are to various adversarial attacks.

We see that such a cross-model, domain-specific setting in combination with adversarial attacks is lacking from existing works. Such an analysis could tell us more about patterns in human and LLM-generated texts that detectors are learning per domain, and what perturbations break those patterns to evade detection. This would also allow to explore defenses against such attacks, such as data augmentation through adding adversarial samples to the train set.

This study hopes to provide empirical insight into these questions by showing which approach is most reliable and robust to adversarial attack, for various domains of text. Additionally it attempts to show if data augmentation could improve robustness of detectors.

# 3. Method

## 3.1 Data

The main dataset that was used was the MAGE dataset [55]. It contains texts text from 10 different domains including opinion statements, news article writing, question answering, story generation, commonsense reasoning, knowledge illustration and scientific writing. A breakdown of all domains can be found in Table 3.1. The MAGE dataset contains text written by humans and text generated by one out of 27 LLMs, see Table 3.2 for the LLMs that were used. The authors create LLM-generated texts using three different prompt strategies:

- Continuation prompts: they prompt the LLM to finish a human-written text after being given the initial 30 words,
- Topical prompts: they prompt the LLM to write a story on a certain topic,
- Specified prompts; they prompt the LLM with specified information about the text source from which to generate (e.g. Reddit or Yelp).

The original training set consists of a total of 319k texts, with a validation and test set of 56.7k texts each. The original 'test beds' that the dataset creators provide, won't be used here, as they provide different test settings that don't align with the research questions of this study. Each data split contains texts from all 27 LLMs.

An additional dataset was selected to evaluate the out-of-distribution performance of detectors, the student essay domain from the MGTBench dataset [49]. It contains essays of high-school and college level on several academic topics, gathered from a public website called Ivypanda. To obtain the generated texts the authors prompt ChatGPT-turbo to generate a prompt based on the essay text. This prompt is then filled into the following tem-

plate: "Write a story in K words to the prompt <prompt>", which is used to prompt the 7 LLMs to generate essays. See Figure 3.3 for the LLMs that were used. Contrary to the MAGE dataset, these prompts are made available in the dataset and can be used, for example to enhance paraphrasing (by conditioning the paraphrase on the prompt with DIPPER [15]). In total, the dataset includes 1000 human-written and 6774 LLM-generated essays. In the case of evaluating detectors out-of-distribution, the total dataset is used for evaluation. In the case of data augmentation on essays (discussed in Section 3.4), we set aside a random sample of 20% for validation, and 10% for testing.

**Table 3.1:** Different domains of text that are used in this study.

| **Abbreviation** | **Explanation** |
|---|---|
| CMV | Opinion statements collected from the Reddit subreddit "ChangeMyView". |
| Yelp | Yelp reviews. |
| XSum | News articles from the BBC from 2010 to 2017 on various topics (News, Politics, Sports, Weather, Business, Technology, Science, Health, Family, Education, Entertainment and Arts). |
| TLDR | News articles collected from hyperlinks in a tech newsletter. |
| ELI5 | Questions and answers collected from the 'explainlikeimfive' subreddit. |
| WP | Stories based on prompts from the Reddit WritingPrompts subreddit. |
| ROCT | Collection of commonsense short stories, written by Amazon Mechanical Turk workers. |
| HellaSwag | Dataset for assessing commonsense Natural Language Inference of models. Contains step-by-step instructions from Wikihow and captions of events in videos. |
| SQuAD | Wikipedia paragraphs from the SQuAD QA context. |
| SciGen | Abstracts of scientific articles from SciGen. |
| Essays | Student-written essays from Ivypanda. Only domain that is is retrieved from MGTBench [49] rather that MAGE [55] |

| Organization | Model | Variants |
|---|---|---|
| OpenAI | GPT | text-davinci-002, text-davinci-003, gpt-turbo-3.5 |
| Meta | LLaMA | 6B, 13B, 30B, 65B |
| GLM | 130B | - |
| Google | FLAN-T5 | small, base, large, xl, xxl |
| Facebook | OPT | 125M, 350M, 1.3B, 2.7B, 6.7B, 13B, 30B, iml-1.3B, iml-30B |
| BigScience | T0 | T0-3B, T0-11B |
| BigScience | BLOOM | BLOOM-7B1 |
| EleutherAI | GPT-J | GPT-J-6B |
| EleutherAI | GPT-NeoX | GPT-NeoX-20B |

**Table 3.2:** The various LLMs that were used in the MAGE dataset [55].

| Organization | Model |
|---|---|
| OpenAI | gpt-turbo-3.5 |
| GLM | ChatGLM |
| Databricks | Dolly |
| Open-source | GPT4All |
| Stability AI | StableLM |
| Anthropic | Claude |

**Table 3.3:** The various LLMs used in the MGTBench dataset [49].

## 3.2 Detectors

Detectors of varying complexity were used in an attempt to detect LLM-generated text. In this section, we will describe each of them.

### 3.2.1 Most common class baseline

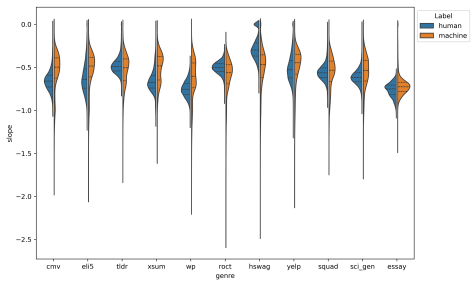
Since our datasets have an imbalanced class distribution, it is meaningful to compare detectors relative to the majority class. To provide this comparison, a 'most common' detector was added across all domains and per-domain. This detector always predicts the majority class, which is LLM-generated in all domains except for Yelp, which has more human-written texts.

### 3.2.2 Features

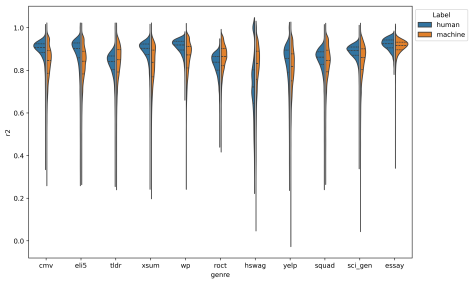
The features from the literature are almost entirely based on the study by Crothers et al. [14]. Crothers et al. use different collections of features: frequency features, complex phrasal features, fluency features and consistency features. A selection of these features were included as features in our detectors. All included features were extracted at dataset level. Figure 3.1 shows the distributions of each feature, for each domain, for both human and LLM-generated text.

The frequency features measure the deviance of the lemma frequencies in a text from the expected distribution based on a Zipfian distribution. To this end texts are tokenized and tokens are turned into lemma form to account for conjugation variance (e.g. walking $\rightarrow$ walk). Then, the frequencies of lemmas are counted and ordered from highest to lowest. A linear regression is fitted, where the ranks of the lemmas are used as the independent variables, $\mathbf{X}$, and the lemma log frequencies are the dependent variable, $\vec{y}$. The resulting features are the slope, $r^2$ and the Mean Square Error (MSE) of the fitted linear regression. Figures showing the Zipfian distribution of the top 300 most frequent terms for LLM-generated and human-written texts per domain are added to Appendix B, Figure B.1.

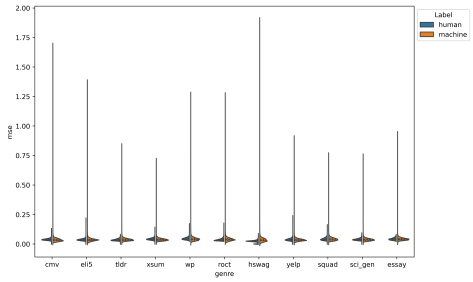
The complex phrasal features are measured as the amount of occurrences of certain phrases in the text, such as English idioms, English cliche phrases, or English archaisms. E.g. an English idiom phrase is "in the nick of time", or a cliche phrase "throw in the towel". Both English idioms and English cliche phrases were used as features, but archaisms were dropped, since the original list of archaisms was no longer available. Omitting archaisms is unlikely to make a meaningful difference in detection accuracy, since Crothers et al. showed that the feature archaisms had little predictive power [14].



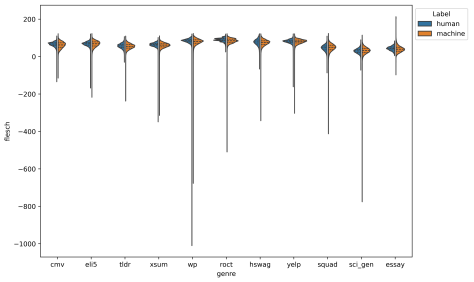

**(a)** Slope



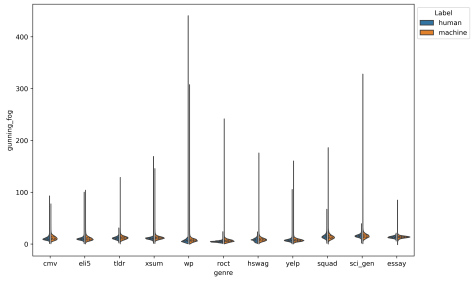

**(b)** $r^2$



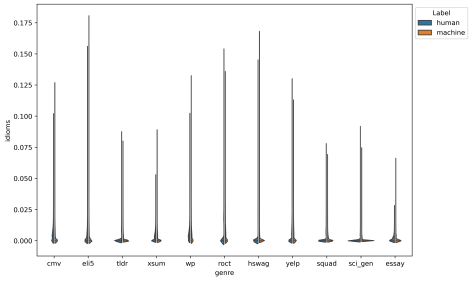

**(c)** Mean Squared Error (MSE)



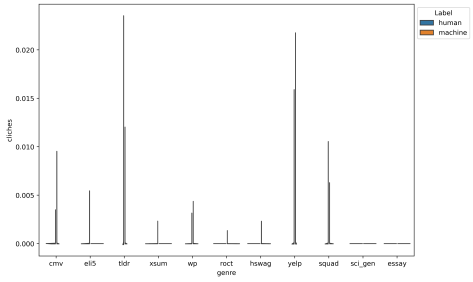

**(d)** Flesch



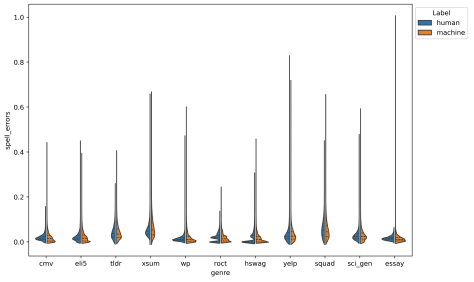

**(e)** Gunning-Fog



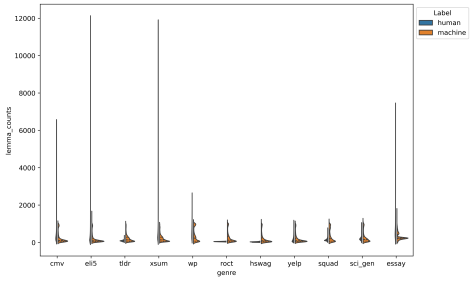

**(f)** English Idioms




**(g)** English Cliches




**(h)** Spelling Errors




**(i)** Lemma count

**Figure 3.1:** Distribution of each used feature, by domain and split out human vs. LLM

To obtain the phrasal features, we first lemmatize all idiom and cliche phrases. We then lemmatize each text in our dataset and count all occurences of the idioms and cliche phrases in them. These counts are normalized by the total amount of lemmas in each text, to obtain the 'idiom ratio' and 'cliche ratio' features.

The fluency features are measured by the Gunning-Fog index [56] and the Flesch index [57], both measuring how readable a text is. The Gunning-Fog index ranging from a score of 6, representing a required reading level of a sixth grader, to a score of 17, which would require a reading level of a college graduate. The Flesch index ranges from a score of 100, requiring a reading level of a fifth grader, to a score of 0, requiring university graduate level to understand. While these are the semantic interpretation of these scores, their values are theoretically unbounded. A text can become arbitrarily hard to read. The scores of Gunning-Fog can exceed 17, and the score of Flesch can drop below 0 (as visible in Figures 3.1d and 3.1e). The Flesch index and the Gunning-Fog index were both added as features.

Finally for the consistency features, Crothers et al. use the ratio of phrasal verbs to the total number of words, as well as coreference ratios. These features require relatively a large amount of compute to extract, since they rely on seperate neural models for Part-Of-Speech (POS) tagging and dependency parsing in the case of the verb phrase ratio, and coreference resolution models in the case of the coreference ratios. To assess their usability, a stratified subsample was taken out of the full set, using 10% of the data of each domain (total $N = 15344$). On this subset, feature based detectors both including and excluding these consistency features were fitted, to see if adding consistency features increased performance significantly and would thus be worth the additional computational expense. The results were added to Appendix B. In Figure B.2 we can see that including the consistency features did not systematically improve performance. The consistency features were thus excluded as they were considered to not be worth the extra computational cost.

Two features were added besides the features used by Crothers et al.,

the amount of spelling mistakes and the lemma counts. First of all, the amount of spelling errors was added as a feature. Where humans can easily make a typo or a grammar mistake, LLMs have often been subjected to instruction tuning and RLHF, which penalize for making such mistakes. This could theoretically make LLMs less likely to produce such errors, thus allowing us to leverage the amount of spelling errors as a meaningful feature for prediction. The amount of spell errors were counted using a package called `Pyspellchecker` [58]. Each lemma in the text was checked with the spellchecker, and the final feature was the total amount of spell errors normalized by the total amount of lemmas in the text. The lemma counts were added as a feature, to see if a certain bias existed in the dataset, where for example human texts were always of a different length than LLM written texts. If such a bias were to exist, the weight for this feature in the logistic regression should give us an intuition how instructive it is.

After extracting features, some outliers were discovered. An outlier here, is a feature $z$-score of either lower than -3 or higher than 3. Upon further inspection it became noticeable that some texts had outliers in multiple features, and that these texts often looked odd, containing a repeating pattern of words. Such repeating patterns are a common problem in the decoding phase of LLMs. Since such outputs hardly fool a human, they were seen as noisy for the classification task, and dropped from the dataset. All texts that had an outlier on at least three feature values were dropped, resulting in a total of 775 texts dropped, out of which 39 human-written and 736 LLM-generated. The final set of features and their correlations on the whole MAGE dataset can be seen in Figure 3.2.

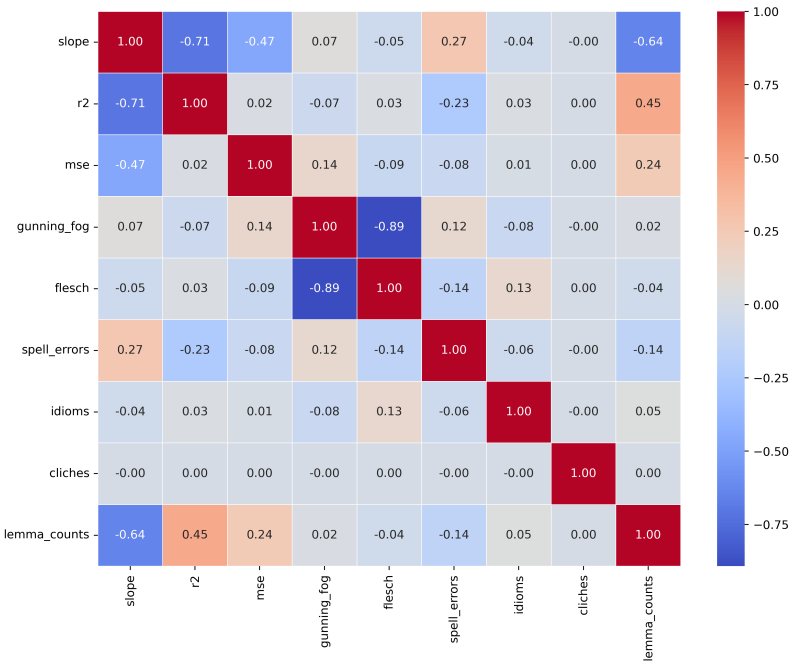



**Figure 3.2:** Correlation matrix of features.

### 3.2.3 Term Frequency Inverse Document Frequency (TF.IDF)

Another set of features that was used are TF.IDF features. TF.IDF features with a logistic regression have often been used as a baseline for NLP classification tasks. Establishing such a baseline is useful as it allows other detectors to be interpreted relative to it, in the case of this study regarding performance and adversarial robustness. The included terms were unigrams and bigrams, with a minimum document frequency of 5% and a maximum document frequency of 95%.

### 3.2.4 Logistic regression detectors

The features above were used to fit logistic regression classifiers. For each domain, a classifier based on only the Crothers et al. features, a classifier based on TF.IDF features, and one based on the combination of both were fitted. All input features were normalized to have a mean of 0 and unit variance. Each of these logistic regression classifiers was fit on these scaled features, using 5-fold cross validation to find the best regularization hyperparameter $C$.

While there are more accurate classifiers, logistic regression was chosen for it's inherent interpretability. These detectors were created as interpretable baselines for the other detectors, not as the best possible detector given these features.

### 3.2.5 Neural models

Additionally, a masked LM with a classification head on top was fine-tuned to the classification task. Li et al. [55] offer a detector based on the Longformer model [52], which they fine-tuned as a LLM-generated text detector on all domains. The authors however don't offer a fine-tuned detector per domain. To test the performance of fine-tuning a LM in-domain, we fine-tuned DeBERTaV3-base [59] for the classification task. DeBERTaV3-base was chosen as it currently represents the SOTA for masked language modelling [59]. It was chosen over the larger DeBERTaV3-large variant to keep training feasible. We chose to use a model with a shorter context window than Longformer, for the same reason, to keep training and inference feasible given the scope of this study. Longformer supports a context window up to 4098, but in their detector Li et al. use a max sequence length of 2048. DeBERTaV3-base has a max context length of 512 tokens. To get an idea of the influence of the sequence length per domain, see Table 3.4. It shows the fraction of texts longer than 512 tokens per domain. In these texts, Longformer will have access to a larger part of the text than DeBERTa.

Another important distinction is that Li et al. [55] give a custom decision boundary for classification rather than just predicting the class with the highest probability. They argue that this helps the model to generalize to unseen domains. This decision boundary, also called threshold, is used in all further experiments that involve the Longformer model. All detectors that we've fit, use the default decision boundary of predicting the most common class. This essentially means that Longformer has picked another point on the ROC curve, and thus has a different trade off between the FPR and the TPR. We have decided to stick with the default decision boundary, but obviously a different decision boundary could be set in our detectors

**Table 3.4:** Per domain, the percentage of texts with more than 512 tokens, tokenized with DeBERTa tokenizer.

| Domain | > 512 tokens |
|---|---|
| cmv | 17.59% |
| eli5 | 14.84% |
| tldr | 7.73% |
| xsum | 14.36% |
| wp | 35.33% |
| roct | 12.27% |
| hswag | 10.63% |
| yelp | 6.59% |
| squad | 16.09% |
| sci gen | 13.28% |

too to attain a certain desired FPR and TPR. We provide ROC curves for all detectors to display the performance at every possible decision boundary, which is the most fair comparison between detectors.

The implementation of both the Longformer from Li et al. and our DeBERTa, were taken from the Transformers library from Hugginface [60]. The library offers various out-of-the-box architectures that serve various purposes, one of which is sequence classification. Such a model is a combination of a backbone (DeBERTa/Longformer in this case) plus fully connected layers on top as a classification head.

Table 3.5, shows the model characteristics and training setup for all in-domain DeBERTa detectors, as compared to the all-domain Longformer detector by Li et al. [55].

**Table 3.5:** Comparison between Longformer detector by Li et al. [55] and our DeBERTa detector. Per model it shows the total number of parameters, the maximum sequence length (in tokens), the dimensions of the last hidden state, batch size, learning rate and weight decay.

| Model | # Params | Seq len | Last hidden | bs | lr | wd |
|---|---|---|---|---|---|---|
| Longformer | 148M | 2048 | 768 | 16 | 0.00005 | - |
| DeBERTa | 184M | 512 | 768 | 16 | 0.00004 | 0.01 |

To prevent overfitting on the train set, a pass through the validation set was done after 20% of each epoch. After each pass the weights that pro-

duced the lowest validation loss were saved. Early stopping was employed after at least 8 passes through the validation set without improvement of the validation loss. The final model was the model that resulted in the minimal loss on the validation set.

#### 3.2.5.1 Exploring the embedding space

To get an intuition of the internal representation of the texts in the embedding space of DeBERTa, a stratified sample of 10% of each domain was taken from the MAGE dataset. These texts were then embedded by extracting the `[CLS]` token's last hidden state as an embedding from the DeBERTa model. A UMAP [61] projection was used to project down these 768 embedding dimensions to a 2D plane for visualisation. Figure 3.3 highlights the differences between domain, with no discrimination between LLM-generated or human-written text. We can see some amount of clustering per domain, from which we can infer that there is some amount of meaningful separation in the representation of texts from different domains. Figure 3.4 highlights the differences between LLM-generated and human-written text overall. Here we see a lack of separability between the two classes, hinting at the difficulty of creating an all-domain detector. The highlighted differences between LLM-generated and human-written texts split out per domain are included in Figure 3.5 and 3.6. We see that each domain looks quite different in the embedding space, when comparing LLM-generated to human-written text. Some domains have a more clear separation between classes (e.g. Yelp), where other domains seem to have more separation (e.g. SQuAD). These preliminary explorations of the embedding space seem to justify a further focus on the differences in detectability of LLM-generated text between domains.

## 3.3 Adversarial attacks

Two adversarial attacks were performed, a character-level attack with DeepWordBug [62] and a paraphrasing attack using the DIPPER [15] paraphrasing model. As a target for adversarial attack, a random sample of 100 LLM-

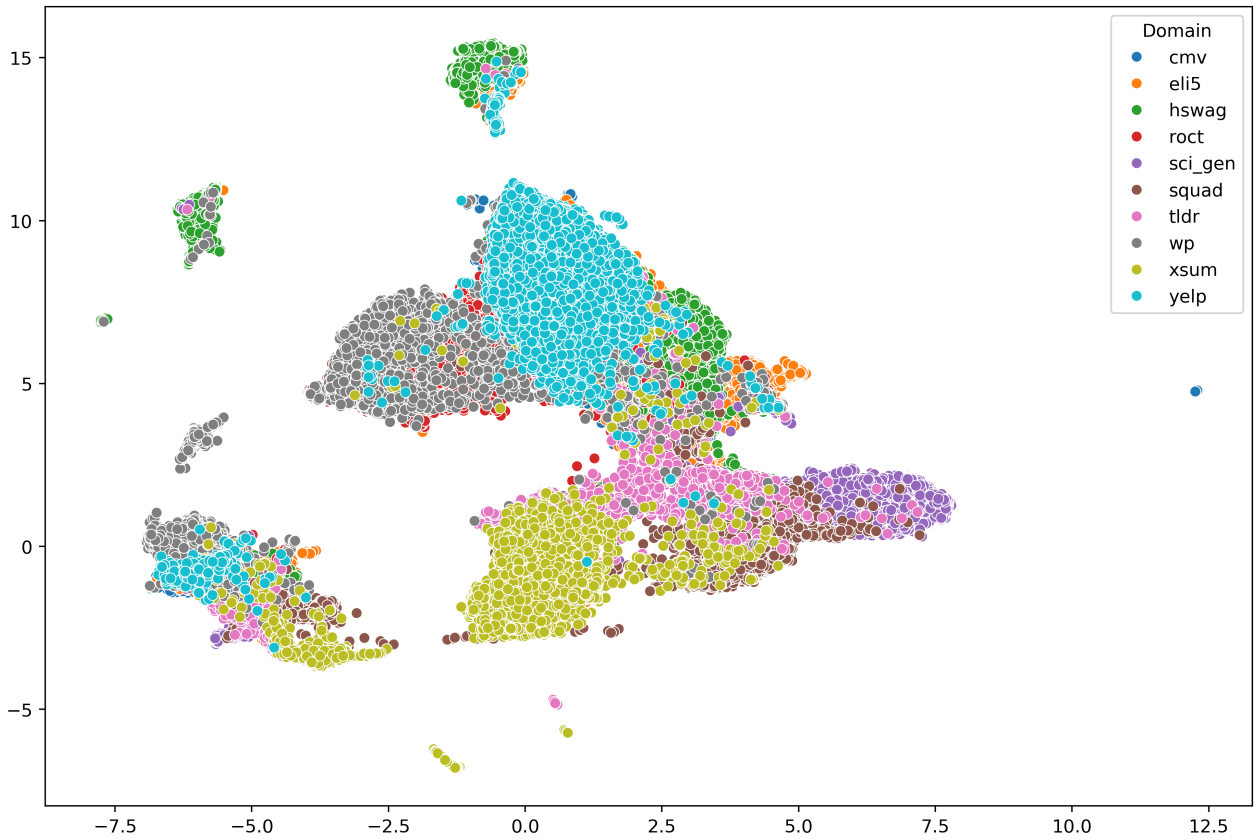
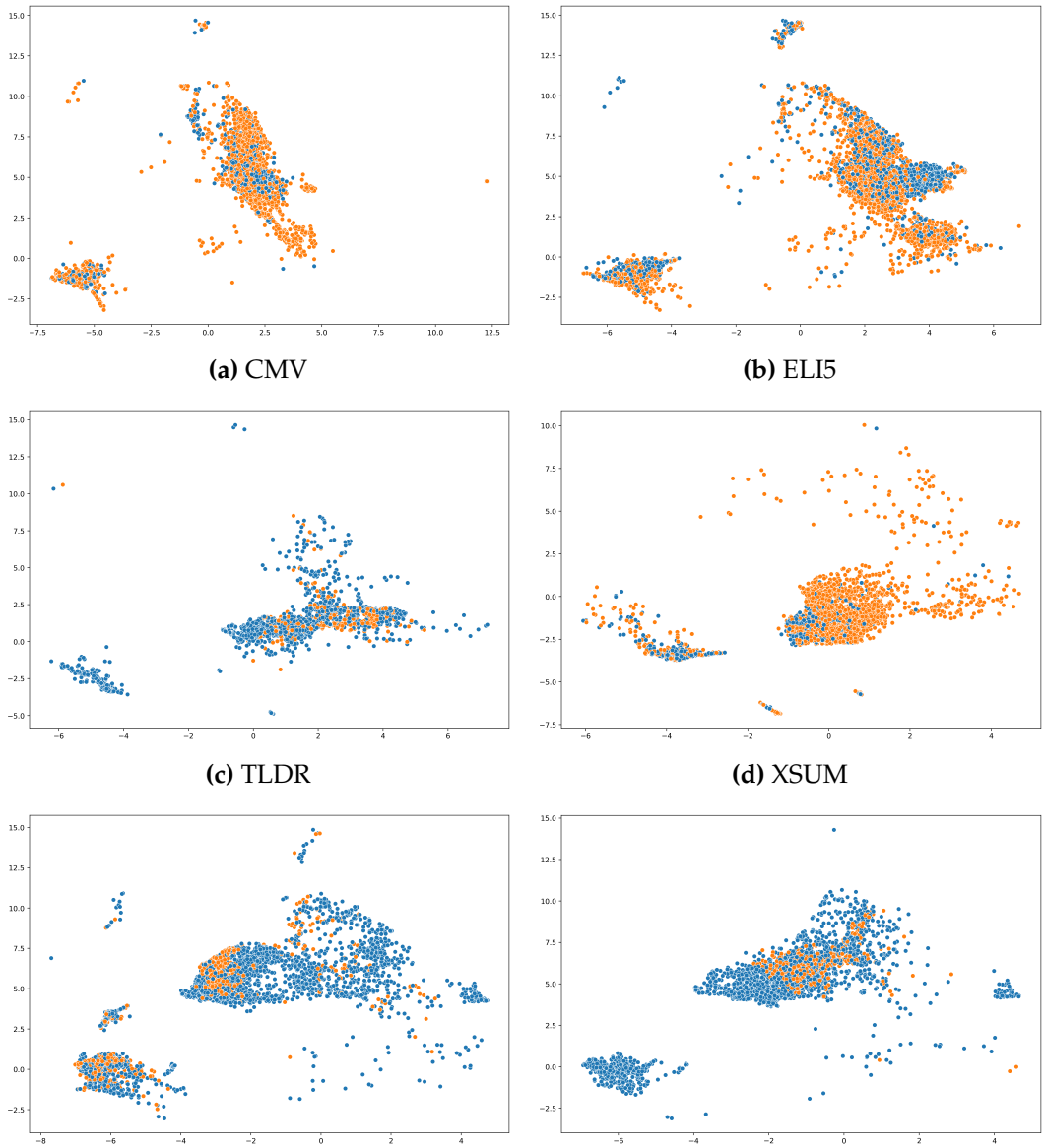



**Figure 3.3:** UMAP projection into 2D of DeBERTa embeddings. Sample of 10% of each domain.


**Figure 3.4:** UMAP projection of 10% of the data, shown is LLM-generated (in orange) vs. human-written (in blue).

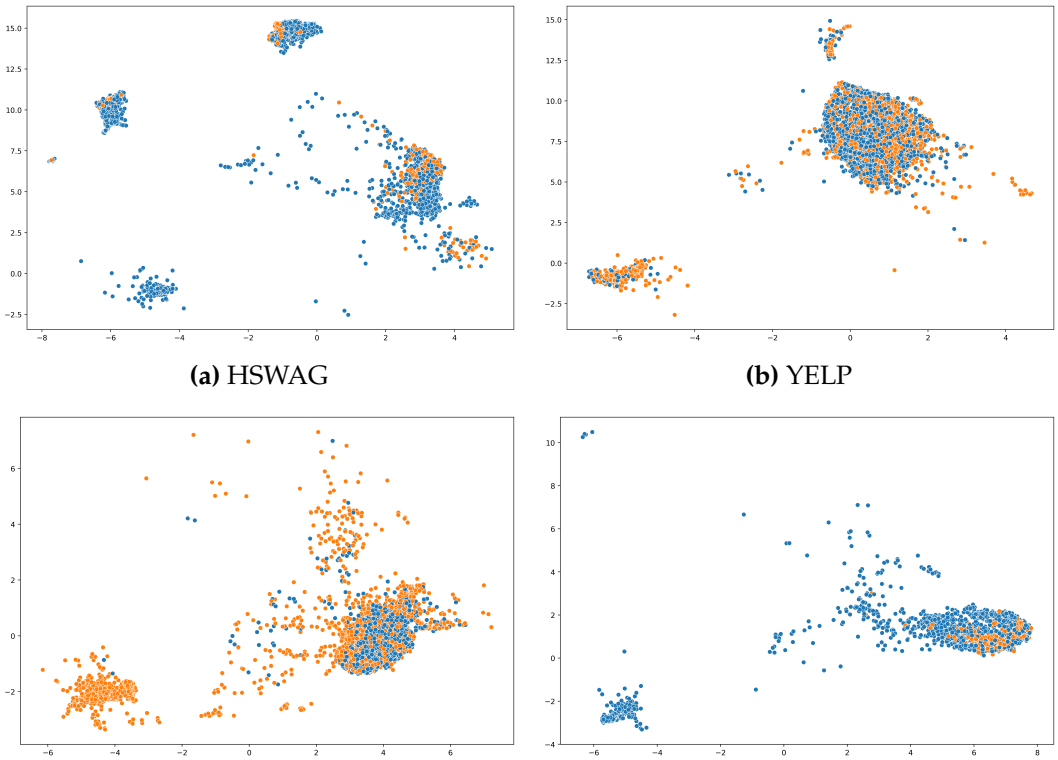


**(a)** CMV **(b)** ELI5

**(c)** TLDR **(d)** XSUM

**(e)** WP **(f)** ROCT

**Figure 3.5:** UMAP projection of embeddings of texts split out by domain, labelled as LLM-generated (in orange) vs. human-written (in blue).



**(a)** HSWAG **(b)** YELP

**(c)** SQUAD **(d)** SCI_GEN

**Figure 3.6:** UMAP projection of embeddings of texts split out by domain, labelled as LLM-generated (in orange) vs. human-written (in blue).

generated samples were taken from the test set, for each attack. The modest size of this sample was informed by the relatively large computational resources needed to perform these attacks. Only LLM-generated samples were attacked because an adversary is likely to want to pass off a LLM-generated text as human written, but unlikely to want to pass of a human-written text as LLM-generated.

### 3.3.1 Character level attack

To attack the LLM-texts at the character level, DeepWordBug was chosen [62]. DeepWordBug needs white-box access to the model that it is attacking. It first determines which tokens are the most important to perturb, based on the difference in the probability that the model assigns to each perturbed input. DeepWordBug then perturbs the most important tokens in the text by swapping, substituting, deleting or inserting characters. The final goal is to have the model under attack misclassify a text with as little character edits as possible. Different detectors can thus be subjected to different perturbations, depending on which edits DeepWordBug finds most effectively confuse that detector. See Figure 3.7 for an example of a successful DeepWordBug attack. Texts where a detector already predicts the wrong class without perturbations, are skipped.

**Original Text:**
(...) It seems to me like one has more impact than another depending on how well they resonate within our life experience. (...)

**Perturbed Text:**
(...) It sems to me like one has more impact than another depning on how well they resonate within our life experience. (...)

**Figure 3.7:** Example of a successful DeepWordBug attack. Attacked text taken from an attack on our ChangeMyView DeBERTa detector.

### 3.3.2 Paraphrasing attack

To perform paraphrasing attacks, the DIPPER paraphrasing model [15] was used. It allows control over the paraphrased output through lexical diversity with parameter $L$, and the order diversity with parameter $O$. Additionally, it allows the insertion of a conditional 'prompt' to guide the generation of the paraphrase. The MAGE dataset [55] didn't contain such a prompt so it was left out during paraphrasing. The MGTBench dataset [49] provides the essay domain, includes the prompt that was used to generate each LLM-generated text. For the essay domain we thus leveraged this prompt to guide the paraphrasing generation. In all usages of DIPPER, both the lexical diversity parameter $L$ and the order diversity parameter $O$ were set to 60, as Krishna et al. [15] show that these settings produce paraphrases that are the most effective in evading detection while retaining semantic similarity with the original text.

An important difference with DeepWordBug is that paraphrasing is done without access to a detector. The paraphrases are generated only given the input text, parameters $O$ and $L$, and an optional prompt. The generated paraphrases are thus the same for each detector. See Figure 3.8 for an example paraphrase.

## 3.4 Data augmentation

Only the essay domain from MGTBench [49] was used to see what the impact of data augmentation on adversarial robustness was. This decision was made to make experimentation feasible. To do data augmentation effectively, we need to add a meaningful amount of augmented texts to the train set. If we add too few augmented texts, the robustness might not increase much, but if we add too many, the performance on the original texts might decrease. One study by Li et al. shows that in increasing adversarial robustness, 10% seems to be a good balance between performance on the task and increased robustness [63]. We thus desire such a fraction to perform data augmentation. In the case of the MAGE [55] dataset that would

**Original Text:**
"Yes, it is a loophole, no that doesn't change our position." Saint-Peter impatiently drummed his fingers on his lecturn, dreaming of the day he'd be allowed to retire. "But I may have something else for you... It's not going to work with your current technology but..." He went silent as she turned and looked at him in surprise after hearing about what was possible from her. He had been working hard trying to find any new way through this mess when something suddenly occurred to him: If they used their power against each other then perhaps there would be less chance of them being found out than if one or both just tried doing so without using magic themselves?

**Paraphrased Text:**
Saint-Pierre impatiently tapped his fingers on the lectern, dreaming of the day when he would be allowed to retire. - But I have something else to tell you... Yes, it is a loophole, no, it does not change our position. He had been working hard to find a new way out of this mess, when suddenly he thought of something: if they used their power against each other, perhaps they would be less likely to be discovered than if one or both of them had tried to do it without magic. It won't work with your current technology, but...

**Figure 3.8:** Example of a text before and after paraphrasing. Text originally generated using OPT-30B within the WritingPrompts domain by the authors of MAGE [55]. The text was then paraphrased using DIPPER [15] by us.

imply augmenting 31900 texts, which is beyond the scope of this study. By limiting to essays we only have to augment 610 texts, which can feasibly be done.

For data augmentation we again only sample LLM-generated texts to perturb. For character-level data augmentation, the package `nlpaug` [64] was used, which provides functionality to swap characters with higher probability if they are closer to each other on a QWERTY keyboard. This is meant to simulate human-like typos. Note that we're not using attacked texts from DeepWordBug [62] for data augmentation, but that we are instead using random character perturbations. This was done because it is computationally much faster, since we don't need white-box access to any detector. For `nlpaug`, the probability of generating a typo was set to 10% per token, with a max of one typo per token. The data augmentation module allowed for a custom tokenizer, so the DeBERTa tokenizer was used in the data augmentation process. A random sample of 10% of the all texts in the train set are augmented using this method.

For data augmentation using paraphrasing, DIPPER was used. A sample 10% of the LLM-generated texts from the training set was recursively paraphrased three passes, each saved individually. DIPPER allows an optional prompt to guide generation, which we provided, since MGTBench [49] includes the prompts that were used to generate each text. With these three passes of paraphrasing, we create four different train sets. A set with one paraphrasing pass, two passes, three passes, and an even split between one/two/three passes. Another train set was created to evaluate the added effect of augmenting the train set both with typos and paraphrases. This train set contained the mix paraphrases of three different levels along with all texts noised with typos. Notably this is the only train set where 20% of data is augmented.

Now that we have these different train sets, we fit DeBERTa detectors on each train set in exactly the same way as was done prior. Once we obtained these detectors, we evaluated their performance and robustness. Evaluating the robustness to character level attacks was again done by attacking

detectors with DeepWordBug, on a random sample of 100 LLM-generated texts from the test set. Evaluating the robustness to paraphrasing attacks is done by evaluating detector performance on the full test set (human-written and LLM-generated), plus paraphrased LLM-generated texts. To obtain the paraphrases for evaluation, all LLM-generated texts from the test set (672 texts) were paraphrased recursively three times. Evaluating on such a test set of both the original test set and the paraphrased LLM-generated texts provides a balanced view of both the robustness to paraphrasing and the performance on the original detection task. Three test sets were created by adding each paraphrasing level to the test set. An additional test set with a mix of all three levels of paraphrasing was created, in which we randomly sampled one of the paraphrasing levels for each LLM-generated text in the test set. This mix of paraphrased texts was again added to the original test set, to create a fourth test set. A final test set was created, by adding all unperturbed LLM-generated texts to the original test set. This was done to evaluate detectors on the original performance only, while preserving the same label distribution as all other test sets.

Recursive paraphrasing has been done previously by Sadasivan et al. [16] in a slightly different way. They paraphrase in 5 rounds, and always use the previous round $p-1$ as input to paraphrase, and $p-2$ as the prompt. In our case this wasn't done, each paraphrase pass was paraphrased on $p-1$, but prompted with the original generation prompt. We also only paraphrase 3 passes to keep running experiments feasible within the scope of this study.

## 3.5 Evaluation & Metrics

For each detector, we will report the Area Under the ROC Curve (AUROC), the accuracy of the model, and the weighted averages of the precision, recall and F1 score. In the case of evaluating on all domains at once, and on the out-of-domain essay domain, we will provide confusion matrices. This is done to more specifically zoom in on the impact of the custom descision boundary of Longformer on the true positive ratio (TPR) and false positive

ratio (FPR) of the model, both in-domain and out-of-domain. In all other cases, the ROC curves and AUROC are given, allowing evaluation of detector performance for any decision threshold.

## 3.6 Ethical statement

The Ethics and Privacy Quick Scan of the Utrecht University Research Institute of Information and Computing Sciences was conducted, see Appendix A for its results.

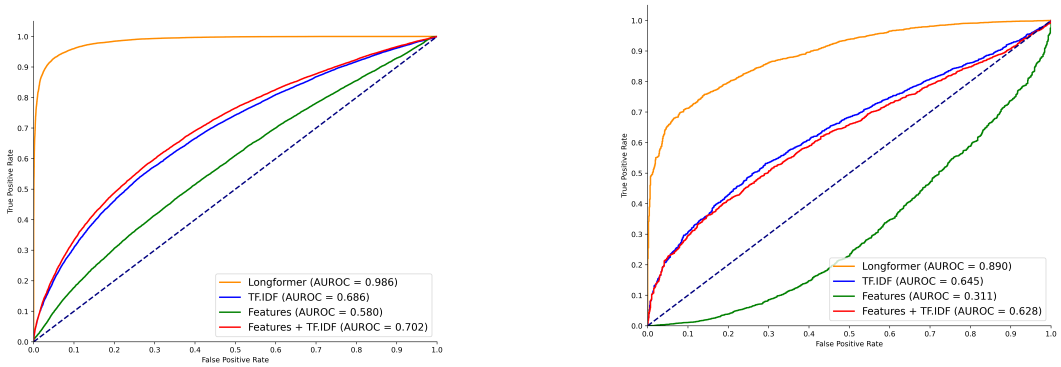
# 4. Results

## 4.1 Performance

Table 4.1 shows the performance of detectors that have been trained using the entire MAGE dataset [55]. This was done to compare the Longformer detector to our feature-based baselines. We see that the Longformer detector far outperforms the baseline detectors. Within the baseline detectors, we see that combining the features and TF.IDF increases overall performance compared to both separate detectors. However, the feature based detectors barely score above our baseline of classifying each text as LLM-written.

**Table 4.1:** Performance of detectors on all domains when trained on all those domains, except for essays.

| Detector | AUROC | Accuracy | Precision | Recall | F1 |
|---|---|---|---|---|---|
| Longformer | **0.985** | **0.876** | **0.898** | **0.876** | **0.874** |
| Features + TF.IDF | 0.702 | 0.646 | 0.647 | 0.646 | 0.646 |
| TF.IDF | 0.685 | 0.632 | 0.633 | 0.632 | 0.632 |
| Features | 0.580 | 0.558 | 0.558 | 0.558 | 0.557 |
| Most common | - | 0.650 | - | - | - |




**(a)** Evaluated on all MAGE domains.

**(b)** Evaluated on out-of-domain essays from MGTBench.

**Figure 4.1:** ROC curves for detectors trained on all domains from the MAGE dataset.

Although Longformer has a quite high AUROC of 0.985, we see that the

other metrics aren't very high. This is result of the chosen decision boundary by Li et al., see Fig 4.1a for the ROC curves. When we look at the confusion matrix for the Longformer in Appendix B, Figure B.3, we see that the chosen decision boundary gives a FPR of $\frac{164}{164+28571} \approx 0.57\%$, and a TPR of $\frac{21138}{6853+21138} \approx 75.52\%$.

To evaluate the out-of-distribution (OOD) performance of Longformer when compared to our baselines, we evaluated them on the essay domain from the MGTBench dataset [12]. Table 4.2 shows the results and Figure 4.1b shows the ROC curves. Similarly to the in-distribution results, we see that Longformer outperforms the feature-based and TF.IDF detectors. We that see the AUROC of Longformer falls by 0.098 compared to the in-distribution setting. Additionally we see that the detector only based on the features performs worse than predicting the most common class, the TF.IDF and TF.IDF + features detectors score slightly better than predicting the most common class. At the given threshold Longformer has a FPR of $\frac{160}{160+840} = 16\%$ and a TPR of $\frac{5182}{5182+1592} \approx 76.50\%$, see Appendix B, Figure B.4 for the confusion matrix.

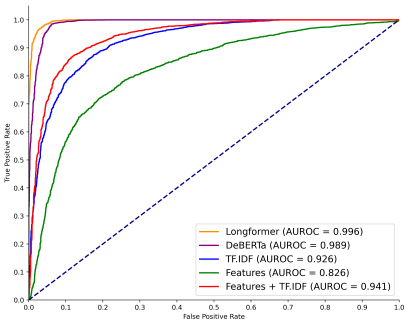
**Table 4.2:** Performance of detectors trained on entire MAGE dataset, evaluated on out-of-distribution essays domain.

| Detector | AUROC | Accuracy | Precision | Recall | F1 |
|---|---|---|---|---|---|
| Longformer | **0.887** | **0.774** | **0.889** | **0.774** | **0.808** |
| Features + TF.IDF | 0.628 | 0.493 | 0.827 | 0.493 | 0.567 |
| TF.IDF | 0.644 | 0.488 | 0.835 | 0.488 | 0.560 |
| Features | 0.311 | 0.551 | 0.734 | 0.551 | 0.623 |
| Most common | - | 0.871 | - | - | - |

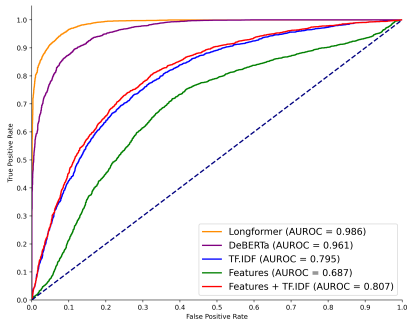
Now for the results within domains. We will compare the best detectors trained on all data (Longformer), with various detectors that have been trained only in in-domain (Logistic Regression baselines and DeBERTa). Table 4.3 shows the results, and Figures 4.2 and 4.3 show the ROC curves per domain for each detector.

**Table 4.3:** Performance comparison between training a general detector on all domains (Longformer) vs. training a detector in-domain (all other detectors). *Yelp is the only domains where the most common class is human-written.

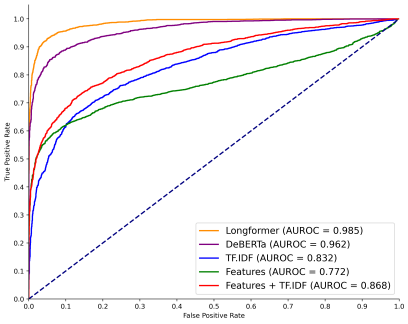
| Domain | Classifiers | AUROC | Accuracy | Precision | Recall | F1 |
|---|---|---|---|---|---|---|
| cmv | Longformer | **0.996** | 0.895 | 0.912 | 0.895 | 0.894 |
| | DeBERTa | 0.989 | **0.962** | **0.942** | **0.985** | **0.963** |
| | Features + TF.IDF | 0.941 | 0.875 | 0.876 | 0.875 | 0.875 |
| | TF.IDF | 0.926 | 0.848 | 0.848 | 0.848 | 0.848 |
| | Features | 0.826 | 0.757 | 0.767 | 0.757 | 0.755 |
| | Most common | - | 0.736 | - | - | - |
| eli5 | Longformer | **0.986** | 0.824 | **0.869** | 0.824 | 0.818 |
| | DeBERTa | 0.961 | **0.879** | 0.838 | **0.942** | **0.887** |
| | Features + TF.IDF | 0.807 | 0.740 | 0.740 | 0.740 | 0.740 |
| | TF.IDF | 0.795 | 0.724 | 0.725 | 0.724 | 0.724 |
| | Features | 0.687 | 0.659 | 0.659 | 0.659 | 0.658 |
| | Most common | - | 0.580 | - | - | - |
| hswag | Longformer | 0.982 | 0.860 | 0.888 | 0.860 | 0.856 |
| | DeBERTa | **0.988** | **0.969** | **0.976** | **0.960** | **0.968** |
| | Features + TF.IDF | 0.912 | 0.851 | 0.854 | 0.851 | 0.851 |
| | TF.IDF | 0.858 | 0.794 | 0.796 | 0.794 | 0.793 |
| | Features | 0.872 | 0.833 | 0.848 | 0.833 | 0.831 |
| | Most common | - | 0.759 | - | - | - |
| roct | Longformer | **0.983** | 0.886 | 0.907 | 0.886 | 0.885 |
| | DeBERTa | 0.980 | **0.957** | **0.958** | **0.955** | **0.956** |
| | Features + TF.IDF | 0.878 | 0.834 | 0.844 | 0.834 | 0.833 |
| | TF.IDF | 0.845 | 0.795 | 0.802 | 0.795 | 0.794 |
| | Features | 0.855 | 0.846 | 0.872 | 0.846 | 0.843 |
| | Most common | - | 0.764 | - | - | - |
| sci gen | Longformer | 0.981 | **0.909** | **0.916** | 0.909 | **0.908** |
| | DeBERTa | **0.982** | 0.888 | 0.816 | **0.983** | 0.892 |
| | Features + TF.IDF | 0.894 | 0.817 | 0.817 | 0.817 | 0.817 |
| | TF.IDF | 0.841 | 0.764 | 0.764 | 0.764 | 0.763 |
| | Features | 0.807 | 0.769 | 0.778 | 0.769 | 0.766 |
| | Most common | - | 0.709 | - | - | - |
| squad | Longformer | **0.991** | 0.866 | 0.893 | 0.866 | 0.863 |
| | DeBERTa | 0.977 | **0.918** | **0.944** | **0.887** | **0.915** |
| | Features + TF.IDF | 0.862 | 0.780 | 0.784 | 0.780 | 0.779 |
| | TF.IDF | 0.778 | 0.710 | 0.711 | 0.710 | 0.709 |
| | Features | 0.785 | 0.762 | 0.780 | 0.762 | 0.758 |
| | Most common | - | 0.543 | - | - | - |
| tldr | Longformer | **0.985** | **0.894** | **0.909** | 0.894 | **0.893** |
| | DeBERTa | 0.962 | 0.864 | 0.811 | **0.941** | 0.871 |
| | Features + TF.IDF | 0.868 | 0.794 | 0.796 | 0.794 | 0.794 |
| | TF.IDF | 0.832 | 0.763 | 0.764 | 0.763 | 0.762 |
| | Features | 0.772 | 0.761 | 0.776 | 0.761 | 0.756 |
| | Most common | - | 0.758 | - | - | - |
| wp | Longformer | **0.997** | 0.930 | 0.937 | 0.930 | 0.930 |
| | DeBERTa | 0.995 | **0.966** | **0.943** | **0.991** | **0.967** |
| | Features + TF.IDF | 0.957 | 0.895 | 0.895 | 0.895 | 0.895 |
| | TF.IDF | 0.944 | 0.871 | 0.871 | 0.871 | 0.871 |
| | Features | 0.816 | 0.743 | 0.749 | 0.743 | 0.742 |
| | Most common | - | 0.711 | - | - | - |
| xsum | Longformer | **0.988** | **0.890** | **0.907** | 0.890 | 0.889 |
| | DeBERTa | 0.969 | 0.887 | 0.839 | **0.958** | **0.894** |
| | Features + TF.IDF | 0.885 | 0.809 | 0.809 | 0.809 | 0.809 |
| | TF.IDF | 0.859 | 0.783 | 0.783 | 0.783 | 0.783 |
| | Features | 0.821 | 0.760 | 0.762 | 0.760 | 0.760 |
| | Most common | - | 0.743 | - | - | - |
| yelp | Longformer | **0.984** | 0.812 | 0.861 | **0.812** | 0.804 |
| | DeBERTa | 0.952 | **0.869** | **0.927** | 0.794 | **0.855** |
| | Features + TF.IDF | 0.751 | 0.687 | 0.688 | 0.687 | 0.686 |
| | TF.IDF | 0.741 | 0.676 | 0.677 | 0.676 | 0.675 |
| | Features | 0.623 | 0.624 | 0.626 | 0.624 | 0.620 |
| | Most common * | - | 0.592 | - | - | - |



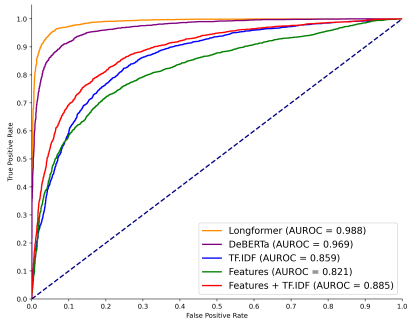

**(a)** CMV



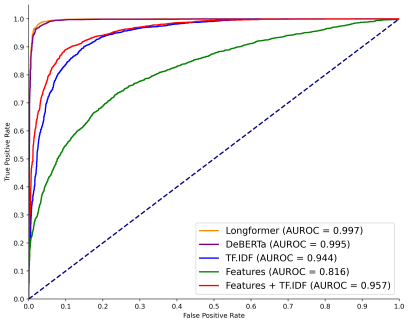

**(b)** ELI5



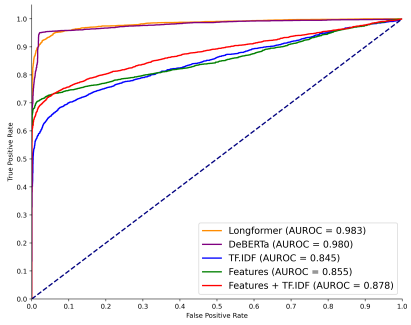

**(c)** TLDR




**(d)** XSUM

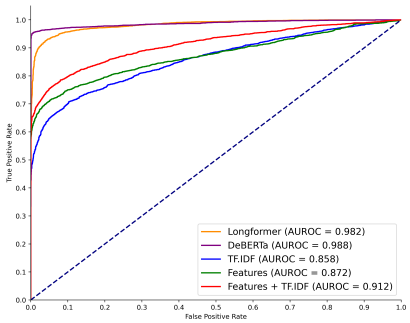



**(e)** WP

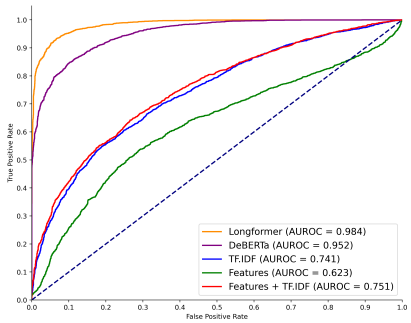



**(f)** ROCT

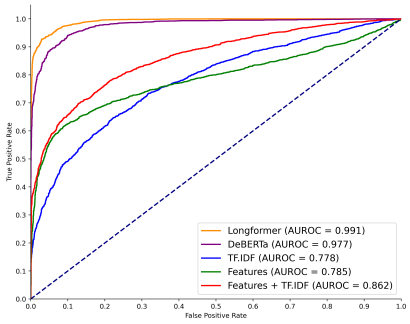
**Figure 4.2:** ROC curve of detectors per domain. Longformer trained on all data vs. other detectors trained in-domain.



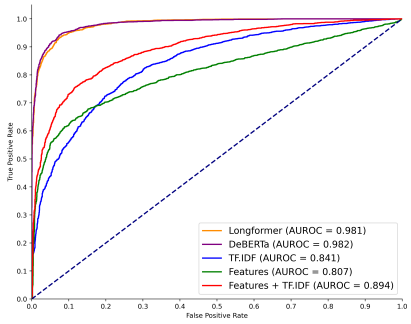

**(a)** HSWAG




**(b)** YELP




**(c)** SQUAD




**(d)** SCI_GEN

**Figure 4.3:** ROC curve for detectors per domain. Longformer trained on all data vs. other detectors trained in-domain.

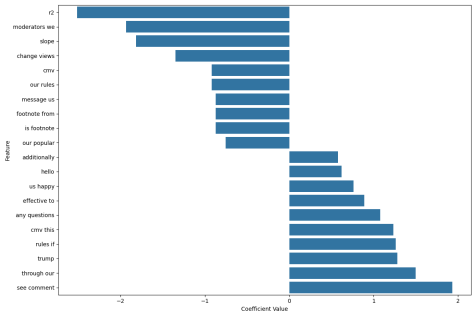
We see that in general, Longformer has high performance on each individual domain that it was trained on compared to the in-domain detectors. The AUROC of Longformer is highest on all domains except SciGen and HellaSwag. In terms of accuracy, Longformer is best in 3 domains, while the in-domain DeBERTa is best in 7 of the domains. These differences in accuracy between Longformer and DeBERTa can be quite dramatic, for example on HellaSwag DeBERTa outperforms Longformer by $\approx$ 10.9%. On F1 DeBERTa outperforms Longformer in 8 out of the 10 domains. But, this is at the default decision boundary of predicting the most probable class. When we look at the ROC plots in Figures 4.2 and 4.3, we see that in the case of ELI5, TLDR, XSum, Yelp and SQUaD, DeBERTa would have a much lower TPR than Longformer when constraining the FPR through a custom decision boundary. Longformer would thus outperform the in-domain DeBERTa when a lower FPR is desired.

For the feature based and TF.IDF detectors, we see that they perform vastly better in this in-domain setting than they did in the all domain setting. We see that combining both features and TF.IDF into one detector consistently outperforms the detectors that use them individually.

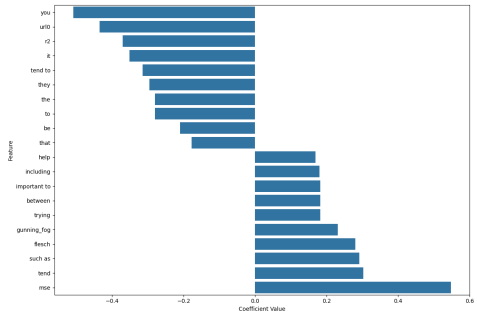
### 4.1.1 Features importance

When we look at the top 10 coefficients from each class of our features + TF.IDF detectors in Figure 4.4, we see that across domains, different features are off different importance to detection.

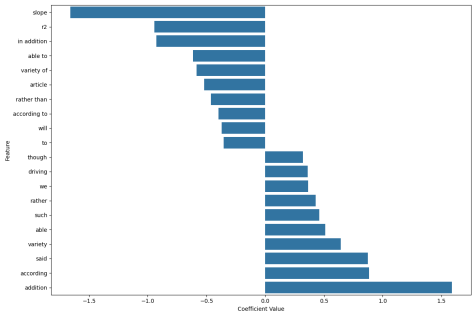
The most common features across domains are the fluency features: slope and $r^2$. In all domains except HellaSwag, a higher $r^2$, thus a better fit with a Zipfian distribution, leads to a text being more likely to be human-written. In most domains, (all except XSum), a larger slope means that a text is more likely to be human-written. This means that a text that contains relatively more uncommon words is more likely to be human-written than LLM-generated. Put another way, humans sample more words from the long tail of infrequent words of the Zipfian distribution when compared to LLMs, which was also found in previous work [41]. The Mean Squared Error (mse) seems to be of variable importance across detector, where in some domains it is big factor pointing towards LLM-generation like HellaSwag, while in others it has a marginal weight only like XSum.



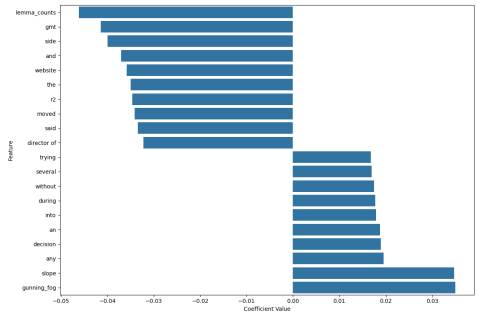

**(a)** CMV



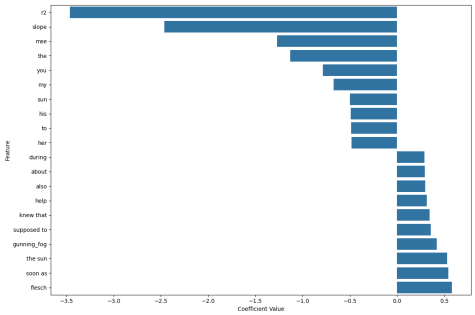

**(b)** ELI5



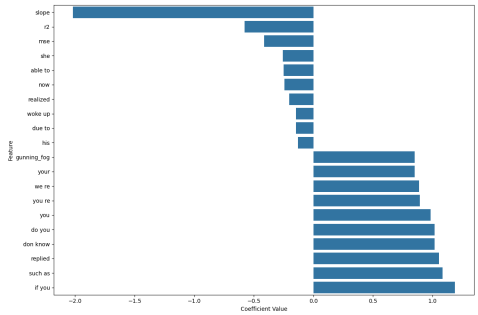

**(c)** TLDR



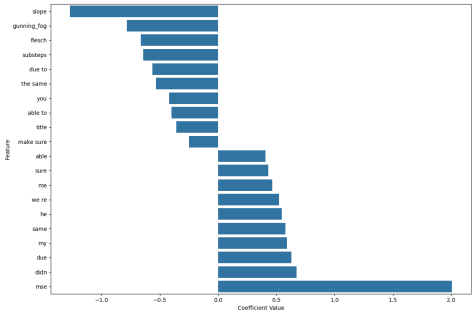

**(d)** XSUM



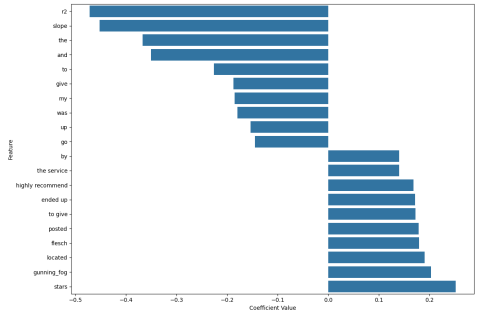

**(e)** WP



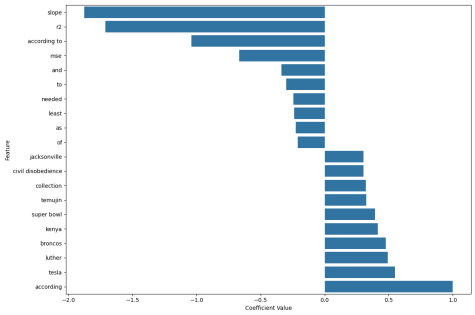

**(f)** ROCT



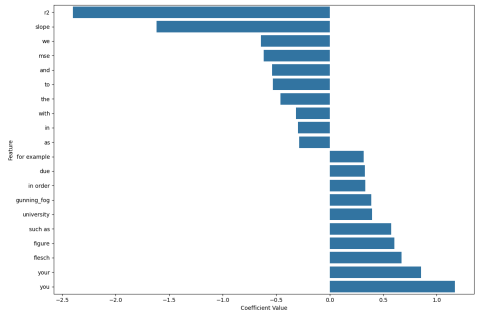

**(g)** HSWAG




**(h)** YELP




**(i)** SQUAD




**(j)** SCI_GEN

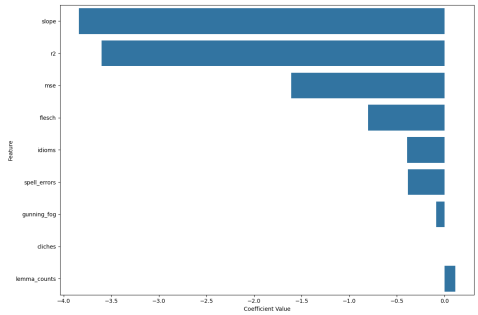


**Figure 4.4:** Top 10 features for human-written (lower coefficients) vs. LLM-generated (higher coefficients), of the TF.IDF + features detectors.

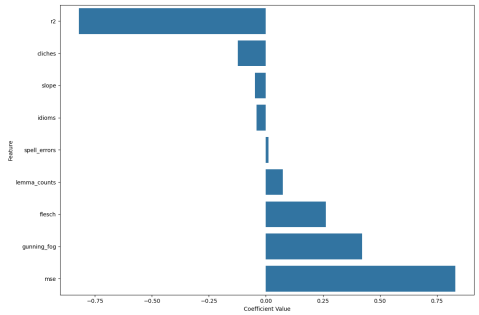
The fluency features are a bit less intuitive. In many domains, we see that texts with a higher Gunning-Fog fluency score, are more likely to be generated by an LLM. However in some domains, also a higher Flesch readability score makes a text more likely to be LLM-generated. This is counter-intuitive since the features are strongly inversely correlated (see Figure 3.2), a higher Gunning-Fog scores means that a text is harder to read while a higher Flesch score means it is easier to read. The unintuitive coefficients are most likely due to the large class imbalance. There are so many more LLM-generated texts than human-written texts, that the logistic regression assigns a relatively high coefficient to a non-sparse feature, in order to predict that a text is LLM-written often enough. For the length of the sequence, measured by the feature lemma counts, we see that it is only a strong predictor for XSum, where humans tend to write shorter texts than LLMs.

Then we see that the features English idioms, English cliches, and spelling errors never end up in the top 10 most important features for either class. Looking at the coefficients of the feature only detectors in Figure 4.5, we can see that indeed, English idioms and English cliches have low coefficients, with the exception of Yelp reviews, where humans seem to use a lot more cliches than LLMs.

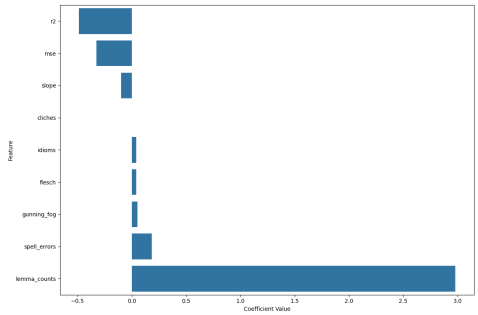
In most domains, the amount of spelling mistakes has a small negative coefficient, indicating the humans make slightly more spelling mistakes than LLMs. We see a variable coefficient of lemma counts, where in some domains LLMs generate longer texts than humans write, while in others humans write longer texts. The top 10 most important for the TF.IDF detectors are added to Appendix B, Figure B.5.



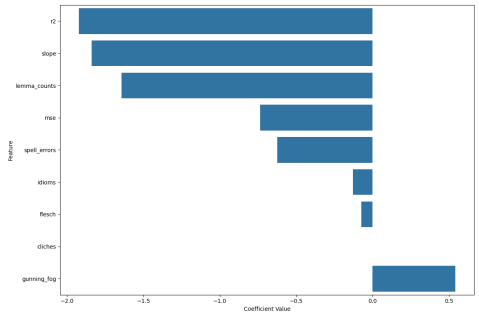

**(a)** CMV



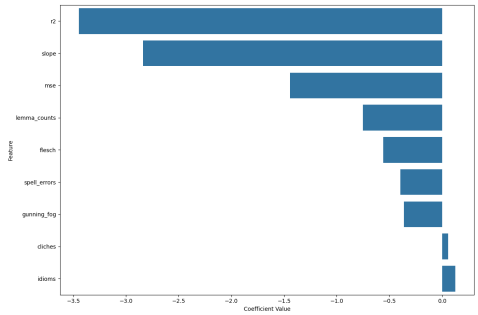

**(b)** ELI5



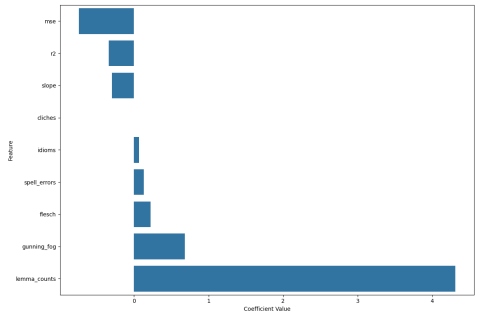

**(c)** TLDR



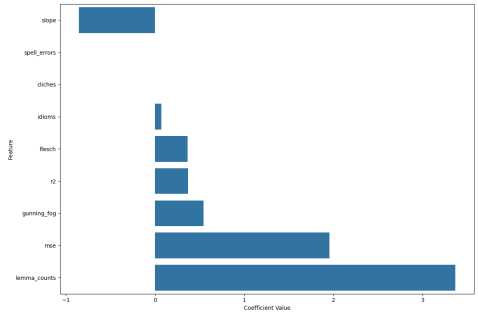

**(d)** XSUM



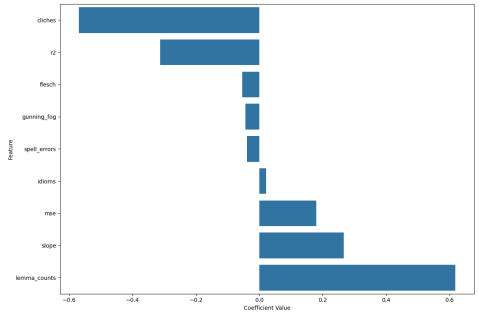

**(e)** WP



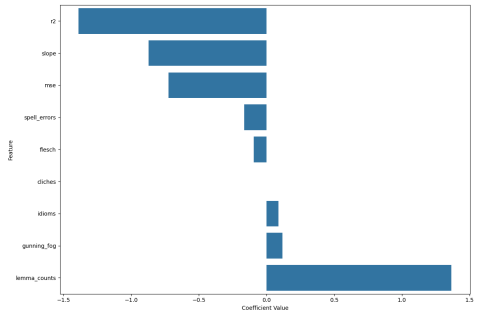

**(f)** ROCT



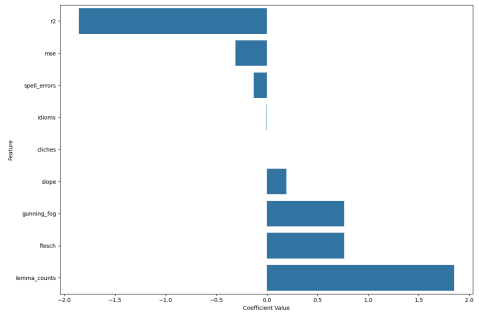

**(g)** HSWAG




**(h)** YELP




**(i)** SQUAD




**(j)** SCI_GEN

**Figure 4.5:** All 9 features for human-written (lower coefficients) vs. LLM-generated (higher coefficients), of the feature detectors.

In general we see variability in performance between domains, across all detectors. For example the best detector on the Yelp domain has a F1 score of 0.855, while the best detector on the WritingPrompts domain scores 0.967. The variability in performance of detectors across domains can also clearly be seen from the ROC curves in Figure 4.2.

## 4.2 Robustness

In evaluating robustness, we leave out the all-domain feature based and TF.IDF detectors. They perform so poorly that even if these detectors were robust they wouldn't be useful. We evaluate the robustness of all detectors always per domain, where Longformer is the only detector that was trained on all domains, and all other detectors were only trained in-domain.

### 4.2.1 Character level

The results of DeepWordBug can be found in Table 4.4. The columns mean the following: Domain is the domain under attack, Acc is the original accuracy on the unperturbed texts, Attack Acc is the accuracy under attack, Success % is the percentage of successful attacks, and Avg. perturb is the average amount of words that were perturbed in successful attacks.

**Table 4.4:** Adversarial robustness of detectors against DeepWordBug attack. Results are reported on 100 randomly sampled LLM-generated texts from the test set. Best in **bold**.

| Domain | Approach | Acc | Attack Acc | Success % | Avg. perturb |
|---|---|---|---|---|---|
| cmv | Longformer | 0.980 | 0.254 | 0.740 | **9.48** |
| | DeBERTa | **1.000** | **0.330** | **0.670** | 4.98 |
| | Features + TF.IDF | 0.813 | 0.114 | 0.860 | 5.21 |
| | TF.IDF | 0.840 | 0.160 | 0.810 | 4.01 |
| | Features | 0.763 | 0.237 | 0.690 | 6.37 |
| eli5 | Longformer | 0.952 | 0.200 | 0.790 | **10.40** |
| | DeBERTa | **0.962** | **0.960** | **0.090** | 7.47 |
| | Features + TF.IDF | 0.781 | 0.219 | 0.720 | 6.56 |
| | TF.IDF | 0.794 | 0.254 | 0.680 | 4.64 |
| | Features | 0.741 | 0.163 | 0.780 | 6.80 |
| hswag | Longformer | **0.935** | 0.402 | 0.570 | 11.36 |
| | DeBERTa | 0.909 | **0.836** | **0.080** | **15.51** |
| | Features + TF.IDF | 0.833 | 0.433 | 0.480 | 13.30 |
| | TF.IDF | 0.800 | 0.488 | 0.390 | 5.55 |
| | Features | 0.870 | 0.783 | 0.100 | 9.00 |
| roct | Longformer | 0.980 | 0.275 | 0.720 | 9.55 |
| | DeBERTa | **0.990** | **0.990** | **0.000** | **-** |
| | Features + TF.IDF | 0.877 | 0.342 | 0.610 | 9.99 |
| | TF.IDF | 0.820 | 0.541 | 0.340 | 6.78 |
| | Features | 0.877 | 0.342 | 0.610 | 14.61 |
| sci gen | Longformer | **1.000** | 0.430 | 0.570 | **12.50** |
| | DeBERTa | 0.971 | **0.612** | **0.370** | 7.81 |
| | Features + TF.IDF | 0.800 | 0.176 | 0.780 | 6.37 |
| | TF.IDF | 0.758 | 0.197 | 0.740 | 4.82 |
| | Features | 0.746 | 0.187 | 0.750 | 7.25 |
| squad | Longformer | **0.971** | **0.379** | 0.610 | **12.57** |
| | DeBERTa | 0.840 | 0.311 | 0.630 | 6.89 |
| | Features + TF.IDF | 0.746 | 0.276 | 0.630 | 6.91 |
| | TF.IDF | 0.699 | 0.273 | 0.610 | 3.11 |
| | Features | 0.725 | 0.348 | **0.520** | 8.36 |
| tldr | Longformer | **0.990** | 0.267 | 0.730 | 10.03 |
| | DeBERTa | 0.963 | **0.644** | **0.330** | **12.50** |
| | Features + TF.IDF | 0.735 | 0.213 | 0.710 | 8.85 |
| | TF.IDF | 0.769 | 0.246 | 0.680 | 4.67 |
| | Features | 0.769 | 0.292 | 0.620 | 7.54 |
| wp | Longformer | 0.990 | **0.396** | **0.600** | **9.29** |
| | DeBERTa | **1.000** | 0.220 | 0.780 | 6.79 |
| | Features + TF.IDF | 0.909 | 0.282 | 0.690 | 5.34 |
| | TF.IDF | 0.855 | 0.308 | 0.640 | 4.34 |
| | Features | 0.671 | 0.228 | 0.660 | 7.33 |
| xsum | Longformer | **1.000** | 0.270 | 0.730 | **10.67** |
| | DeBERTa | 0.952 | **0.533** | **0.440** | 8.49 |
| | Features + TF.IDF | 0.893 | 0.223 | 0.750 | 6.80 |
| | TF.IDF | 0.870 | 0.339 | 0.610 | 3.73 |
| | Features | 0.787 | 0.354 | 0.550 | 7.91 |
| yelp | Longformer | **0.926** | **0.278** | **0.700** | 7.24 |
| | DeBERTa | 0.826 | 0.248 | **0.700** | 3.85 |
| | Features + TF.IDF | 0.714 | 0.143 | 0.800 | 5.14 |
| | TF.IDF | 0.730 | 0.153 | 0.790 | 4.74 |
| | Features | 0.662 | 0.192 | 0.710 | **9.05** |

First of all we see that Longformer isn't robust to attacks by DeepWordBug. In every domain, the accuracy of Longformer drops greatly under

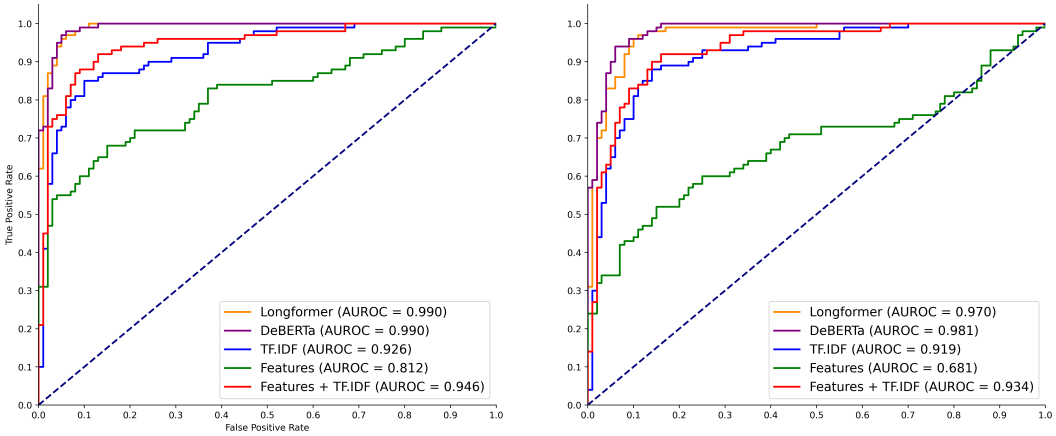
attack. While DeBERTa is also not robust in many domains, DeBERTa holds up almost perfectly on ELI5, HellaSwag and perfectly on ROCT, with low attack success rates and high accuracy under attack. The logistic regression baselines show great drops in accuracy under attack and are thus not very robust. We see that in general, the robustness of detectors to character perturbations of DeepWordBug varies between domains. The attacks are generally less successful on HellaSwag and more successful on Yelp for example.

### 4.2.2 Paraphrasing

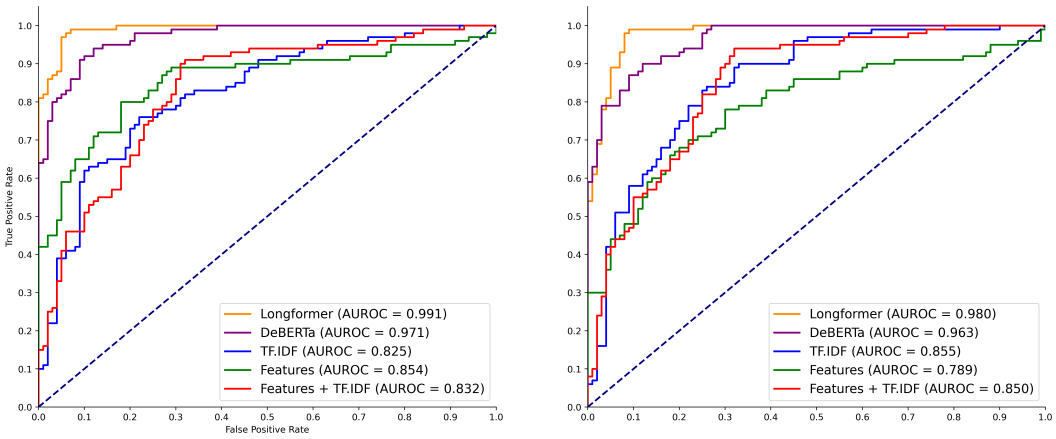
Results for the paraphrasing attack with DIPPER can be found in Figures 4.6, 4.7 and 4.8. The figures show the detectors ROC curve on 100 sampled LLM-generated texts, and 100 sampled human-written texts, both from the test set. Each figure shows the ROC curve of the detectors on the original texts on the left, and on the right the ROC curve of the detectors where all LLM-generated texts have been paraphrased with DIPPER.

We observe that in general, paraphrasing has a variable effect on the various detectors per domain. On HellaSwag for example (Figure 4.8a), we see that the AUROC of Longformer decreases after paraphrasing, while the AUROC of DeBERTa and the features + TF.IDF detectors increase. On SQUaD (Figure 4.8c), we see all the AUROC of all detectors dip. Conversely on WritingPrompts (Figure 4.7b), we see that some detectors dip slightly in AUROC, while others have an increased AUROC on paraphrases.

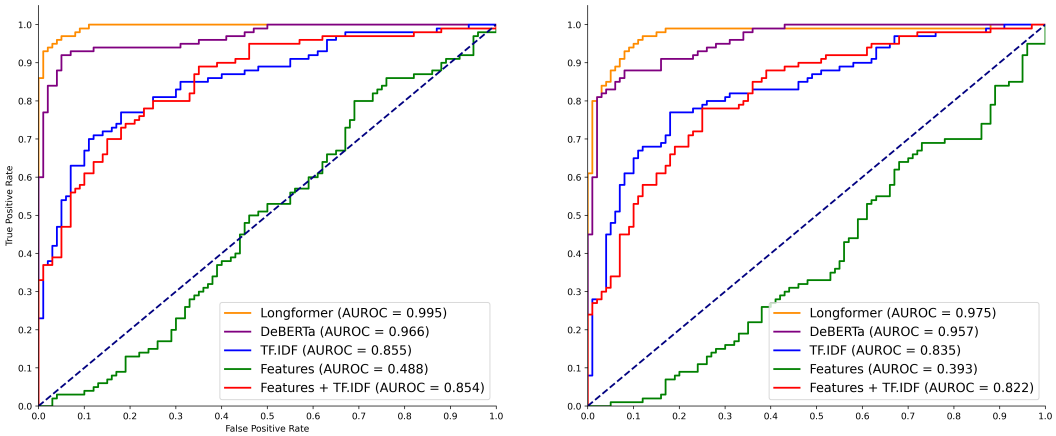
However a change in AUROC doesn't tell the full story. It also matters where the ROC curve changes. We see that paraphrasing lowers the TPR specifically at lower FPR rates for most domains and most detectors. This means that if a requirement of a detector is a low FPR, the TPR can still suffer dramatically, even if the total AUROC doesn't decrease much. As an example, if we'd want a 1% FPR on SQUaD for Longformer, we could attain a TPR of 92% on the original texts. On paraphrases Longformer could only attain a TPR of 71% at that same FPR of 1%. Quite a big difference given that the AUROC on paraphrases decreased by only 0.021.




**(a)** CMV

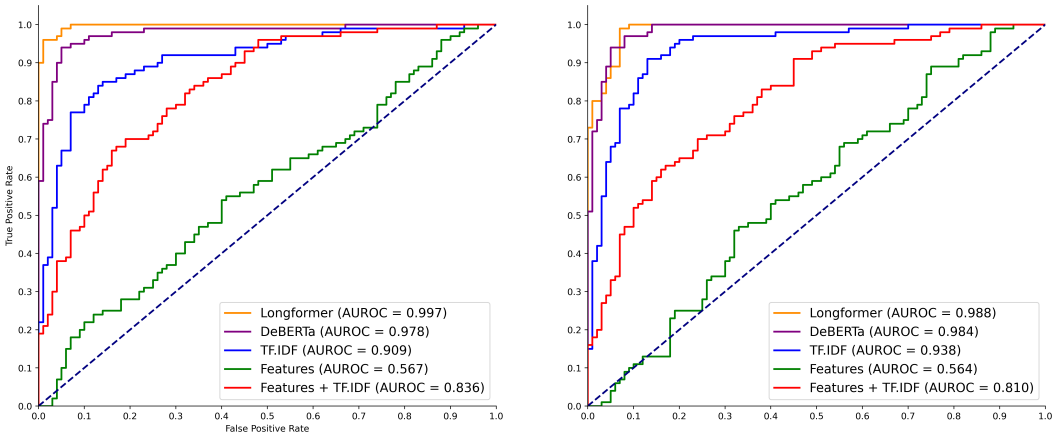



**(b)** ELI5

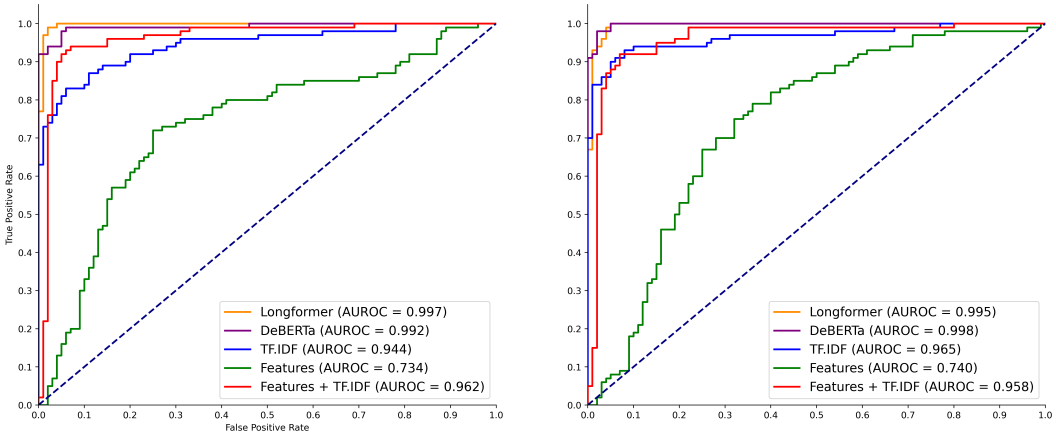



**(c)** TLDR

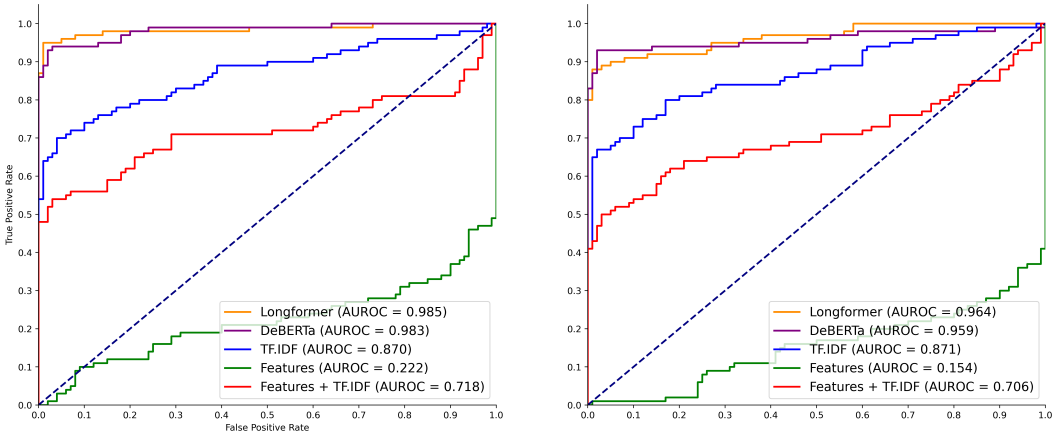
**Figure 4.6:** ROC curves of detectors under paraphrasing attack. Sample of 100 human-written and 100 LLM-generated texts. Left shows the unattacked ROC curves, the right shows the ROC curves when LLM-generated texts are paraphrased.




**(a)** XSUM

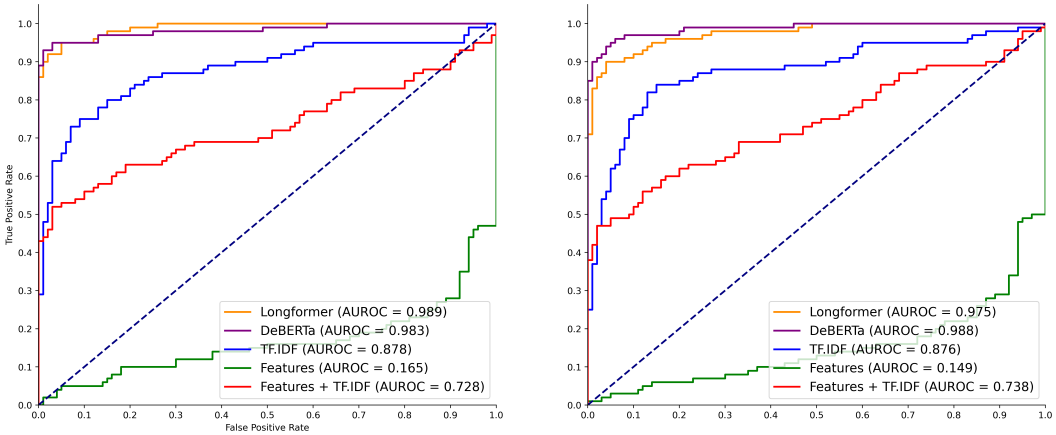



**(b)** WP

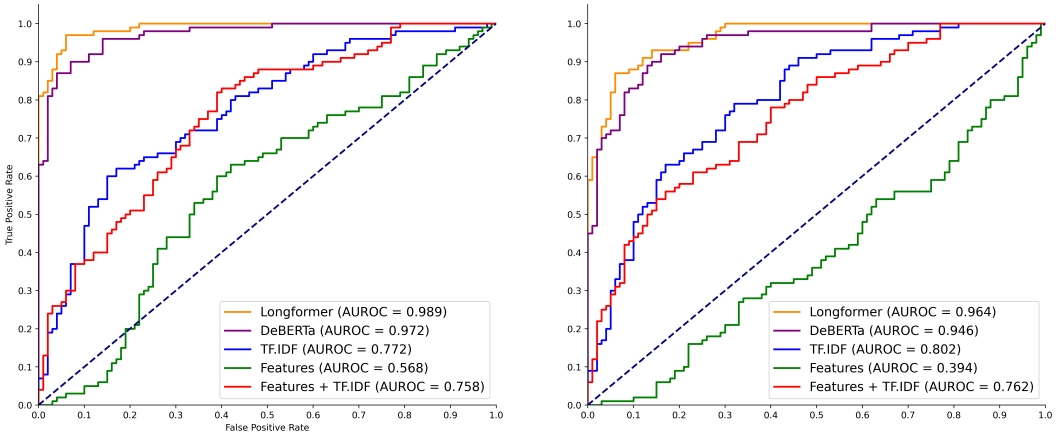



**(c)** ROCT

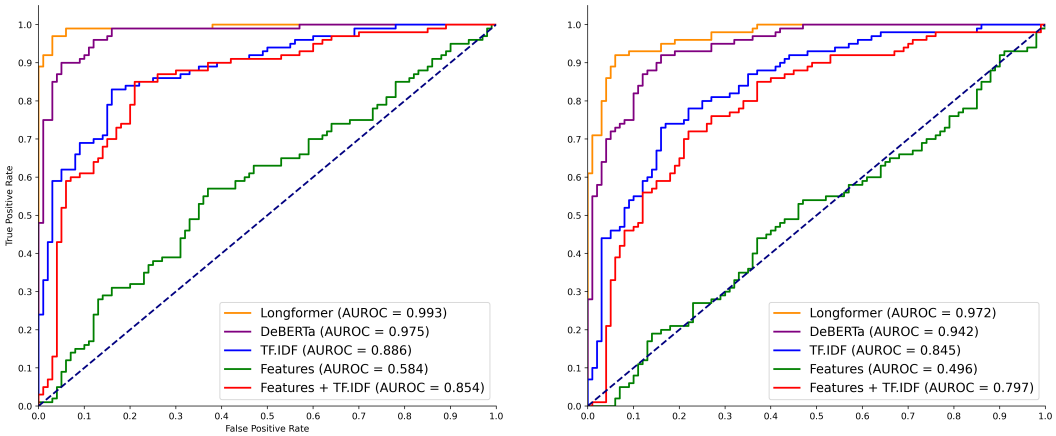
**Figure 4.7:** Figure 4.6 continued.




**(a)** HSWAG

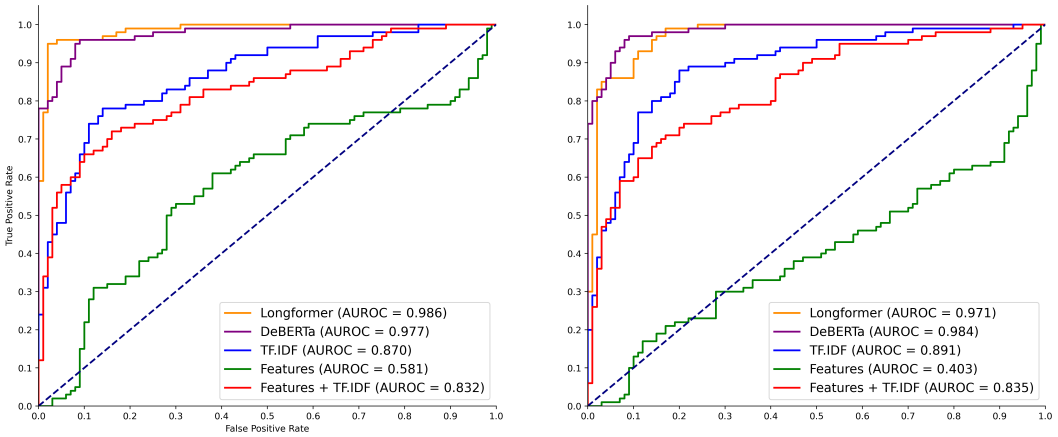



**(b)** YELP




**(c)** SQUAD

**Figure 4.8:** Figure 4.6 continued.




**(a)** SCI GEN

**Figure 4.9:** Figure 4.6 continued.

But again, we don't see this pattern on all domains and all detectors. On WritingPrompts, DeBERTa is completely robust to paraphrasing. The AUROC even increases slightly on paraphrases. The detector scores a TPR of 92% at a fixed FPR of 1% both on the original texts and paraphrases. Differences within a domain, between detectors can also be vast. For example in the SciGen domain, Longformer attains a TPR of 77% at a fixed FPR of 1%, which is decreased to a TPR of 45% on paraphrases at that same fixed FPR of 1%. Meanwhile DeBERTa attains a TPR of 78% which increases to 80% on paraphrases, at the same fixed 1% FPR.

Domain thus matters to the robustness of detectors to paraphrasing. Different detectors are variably robustness to paraphrasing per domain, and different detectors vary in robustness within domain. When detectors aren't robust to paraphrasing, we see that specifically the TPR at a low FPR decreases.

## 4.3 Data Augmentation

In this section we summarize the results of attempting to detect against adversarial attacks through data augmentation. First we will go over the results for the character level attack using DeepWordBug, followed by the results of paraphrasing. Data augmentation was done by either adding typos, paraphrases, or both to the train set of a detector.

### 4.3.1 Character level attacks

The robustness results of the DeBERTa LLM-generated essay detector to DeepWordBug for various data augmentation settings, can be found in Table 4.5.

**Table 4.5:** Adversarial robustness of DeBERTa detector to DeepWordBug, different data augmentation, best in **bold**. The column 'Data augmentation' refers to the type of data augmentation that was done in the training data.

| Data augmentation | Acc | Attack Acc | Success % | Avg. perturb |
|---|---|---|---|---|
| - | 0.990 | 0.505 | 0.490 | 0.062 |
| Typos | 0.990 | 0.921 | 0.070 | 0.047 |
| Paraphrase (1 + 2 + 3) | **1.000** | **0.960** | **0.040** | 0.065 |
| Paraphrase (1 + 2 + 3) + typos | 0.990 | 0.891 | 0.100 | **0.066** |

We see that DeBERTa is not very robust to the character perturbations that DeepWordBug performs, when it's been trained on the original train set without any data augmentation. Its accuracy drops from 0.990 to 0.505. Doing data augmentation by adding typos to the train set, makes the resulting detector much more robust to attack, resulting in the same original accuracy, and an accuracy of 0.921 under attack. The best level of data augmentation for robustness to the DeepWordBug attack is done by adding various levels of paraphasing, which attains an original accuracy of 1.000 and an accuracy under attack of 0.960. Adding both typos and paraphrase levels leads to worse adversarial robustness than adding paraphrases or typos separately, but needs on average more perturbations to be fooled than the unaugmented detectors. We can see that data augmentation with paraphrases dramatically improves the adversarial robustness of the DeBERTa detector to DeepWordBug attacks within the essay domain.

### 4.3.2 Paraphrasing

We first of all establish the extent to which our DeBERTa detector to detect LLM-generated essays is robust to paraphrasing, when no data augmentation is used. In Table 4.6 we see that the unaugmented detector is generally robust to paraphrasing already, with performance even going up on texts

that were paraphrased multiple times recursively. We also see that augmenting the data with paraphrases of various levels and retraining the detector increases the accuracy and recall, but decreases the precision slightly, resulting in a higher F1-score. Augmenting with paraphrases also makes the detectors perform more similarly on all levels of paraphrasing, e.g. there is less variance between the detectors performance on the different paraphrase levels. Adding typos seems to have almost no effect on the performance on paraphrased text when compared to the detector that was trained without data augmentation.

**Table 4.6:** Performance of DeBERTa with different levels of data augmentation, on test set containing various degrees of paraphrased texts.

| Aug | # DIPPER passes | AUROC | Accuracy | Precision | Recall | F1 |
|---|---|---|---|---|---|---|
| - | 0 | 1.000 | 0.988 | 1.000 | 0.987 | 0.993 |
| | 1 | 1.000 | 0.987 | 1.000 | 0.986 | 0.993 |
| | 2 | 1.000 | 0.989 | 1.000 | 0.988 | 0.994 |
| | 3 | 1.000 | 0.992 | 1.000 | 0.992 | 0.996 |
| | (1 + 2 + 3) | 1.000 | 0.989 | 1.000 | 0.988 | 0.994 |
| 1 | 0 | 0.999 | 0.993 | 0.996 | 0.997 | 0.996 |
| | 1 | 0.999 | 0.994 | 0.996 | 0.999 | 0.997 |
| | 2 | 0.999 | 0.994 | 0.996 | 0.999 | 0.997 |
| | 3 | 0.999 | 0.994 | 0.996 | 0.999 | 0.997 |
| | (1 + 2 + 3) | 0.999 | 0.994 | 0.996 | 0.999 | 0.997 |
| 2 | 0 | 0.998 | 0.992 | 0.993 | 0.999 | 0.996 |
| | 1 | 0.998 | 0.992 | 0.993 | 0.999 | 0.996 |
| | 2 | 0.998 | 0.992 | 0.993 | 0.999 | 0.996 |
| | 3 | 0.998 | 0.992 | 0.993 | 0.999 | 0.996 |
| | (1 + 2 + 3) | 0.998 | 0.992 | 0.993 | 0.999 | 0.996 |
| 3 | 0 | 0.999 | 0.994 | 0.996 | 0.997 | 0.997 |
| | 1 | 0.999 | 0.994 | 0.996 | 0.998 | 0.997 |
| | 2 | 0.999 | 0.995 | 0.996 | 0.999 | 0.997 |
| | 3 | 0.999 | 0.995 | 0.996 | 0.999 | 0.997 |
| | (1 + 2 + 3) | 0.999 | 0.995 | 0.996 | 0.999 | 0.997 |
| (1 + 2 + 3) | 0 | 0.999 | 0.991 | 0.995 | 0.996 | 0.995 |
| | 1 | 0.999 | 0.992 | 0.995 | 0.996 | 0.996 |
| | 2 | 0.999 | 0.992 | 0.995 | 0.997 | 0.996 |
| | 3 | 0.999 | 0.992 | 0.995 | 0.997 | 0.996 |
| | (1 + 2 + 3) | 0.999 | 0.992 | 0.995 | 0.997 | 0.996 |
| typos | 0 | 0.999 | 0.990 | 0.998 | 0.991 | 0.994 |
| | 1 | 0.999 | 0.987 | 0.998 | 0.988 | 0.993 |
| | 2 | 0.999 | 0.989 | 0.998 | 0.990 | 0.994 |
| | 3 | 0.999 | 0.991 | 0.998 | 0.993 | 0.995 |
| | (1 + 2 + 3) | 0.999 | 0.989 | 0.998 | 0.990 | 0.994 |
| (1 + 2 + 3) + typos | 0 | 0.999 | 0.990 | 0.999 | 0.991 | 0.995 |
| | 1 | 0.999 | 0.993 | 0.999 | 0.994 | 0.996 |
| | 2 | 0.999 | 0.993 | 0.999 | 0.994 | 0.996 |
| | 3 | 0.999 | 0.993 | 0.999 | 0.994 | 0.996 |
| | (1 + 2 + 3) | 0.999 | 0.993 | 0.999 | 0.994 | 0.996 |

# 5. Discussion

In this section we will discuss the results and put them in broader light of the existing literature, as well as answer the research questions.

## 5.1 Detector performance on all domains & generalizing to unseen domain

When training on all domain data simultaneously, we see that the fine-tuned Longformer greatly outperforms the feature-based baselines. Whether the longformer detector is good enough for detection 'in the wild' depends on what is considered good enough for some real life use case. At the chosen threshold, we attain a conservative 0.57% FPR, but also only a TPR of 75.52%, meaning about a quarter of all LLM-generated texts go undetected.

Tweaking the decision boundary to set a different ratio between TPR and FPR (as per the ROC curve in Figure 4.1a) is possible, but could have an impact on the detectors robustness to adversarial attack in various domains, especially when trying to attain a low FPR. This can be seen in Figure 4.6, 4.7, 4.8 for various domains, where paraphrasing specifically harms the TPR at a low FPR.

The logistic regression baselines when trained on all domains, generalize poorly out-of-domain. The Longformer detector performs reasonably well compared to the baselines, however, the resulting FPR and TPR still result in an unreliable detector. Out of all essay, the Longformer detector would falsely accuse 16% of students of using LLMs to write their essays while still missing 23.5% of actually LLM-generated essays. We thus confirm once more the difficulty for fine-tuned LM detectors to generalize to unseen domains.

## 5.2 Fine-tuning on all-domains vs. in-domain

We see that fine-tuning an LM within a domain (DeBERTa) performs very well versus fine-tuning on all domains (Longformer). While Longformer outperforms DeBERTa on AUROC in most domains, DeBERTa often gets very close. In 5 out of 10 domains, DeBERTa is within 0.01 of the AUROC of Longformer or scores higher (CMV, HellaSwag, ROCT, SciGen and WritingPrompts). While DeBERTa has more parameters, it has only a quarter of the context size compared to Longformer (see Table 3.5). The DeBERTa detector is thus able to extract enough information out of this limited context window to attain similar performance to the all-domain Longformer.

There are also domains where Longformer is clearly more performant than the in-domain DeBERTa (Yelp, XSum, TLDR, ELI5, and SQuAD). One reason could be that the longer context length plays a role here, another that Longformer learned more abstract, domain-invariant features of LLM-generated text from training on all domains, that are leveraged in these domains. However if we look at Table 3.4, we see that the Yelp and TLDR are the domains with the smallest fraction of texts above 512 tokens, yet Longformer outperforms DeBERTa there. Conversely WritingPrompts has many longer texts, while DeBERTa and Longformer score comparably on that domain. It thus seems that the effect of context length too, is not entirely trivial across all domains. More context isn't always needed for better performance in all domains while it might still might help in others. Both the fact that Longformer retains some accuracy out-of-domain and the fact that it outperforms DeBERTa on shorter texts, gives the intuition that Longformer has indeed learned some abstract domain-invariant features from training on all domains, that the in-domain DeBERTa detectors have no access to.

## 5.3 Detector performance by domain

The performance of the best detector per domain varies quite drastically. This further emphasizes the importance of domain in LLM-generated text

detection. This implies that any LLM-generated text detector should be evaluated against a myriad of domains, as generalization of performance across different domains is clearly not self-evident. For example, fine-tuning a LM for detection, varies in accuracy between 0.969 by fine-tuning on HellaSwag, and 0.869 by fine-tuning on Yelp, a full 10% difference.

It's interesting to see that the fine-tuned HellaSwag detector has the highest performance, since HellaSwag is a dataset used to evaluated the common-sense reasoning ability of LLMs, a task at which they are known to struggle [4]. It's also interesting to see that detection works well when detecting within the domain of creative writing, e.g. the WritingPrompts and ROCT domains. It seems as though creative writing is still something in which LLMs struggle to write indistinguishably from humans.

## 5.4 Robustness

We see variation between domains on adversarial robustness. DeBERTa is robust to DeepWordBug in some domain, and not in others. Longformer is not robust to DeepWordBug attacks in any domain. This could be due to the larger context window, giving the DeepWordBug more possibilities for perturbations that could confuse the detector. Another explanation could be that a detector trained on multiple detectors suffers from decreased robustness when compared to an in-domain detector. In general we see that DeepWordBug attacks are generally more effective on some domains than others. The attack success rate of Yelp is really high for example, while the attack success rate is much lower on HellaSwag.

We see that robustness of detectors to DIPPER paraphrases varies across domains and across detectors. Furthermore we see that in the case where detectors aren't robust to paraphrasing, that their TPR drops at lower FPR rates. This raises the issue of the generalizing statement that 'paraphrasing evades detection' [15]. We have shown that this is definitely the case for some domains, but not for all. The issue of domain was initially not addressed by Sadasivan et al. [16], but their paper has since been updated with experiments of paraphrasing attacks on multiple different domains.

They too see variability in the success paraphrasing attacks on detectors and they observe that fine-tuned LMs are generally more robust to paraphrasing than other detectors. They do however, still evaluate the performance on one LLM to generate texts per domain.

In the DIPPER paper [15], Krishna et al evaluate the effectiveness of their attacks on two domains ELI5, and Wikipedia articles (similar to SQuAD). We indeed show that robustness of detectors is poor on both ELI5 and SQuAD. However we also show robustness of detectors is much better on WritingPrompts and ROCT for example. In the case of WritingPrompts, our in-domain DeBERTa detector is completely robust to paraphrasing.

The Longformer classifier is never completely robust to paraphrasing attacks. Again we think this might be attributable to either the increased context size or a possible decrease in robustness when training a detector on all domains.

## 5.5 Data augmentation

We see that the DeBERTa essay detector is not initially robust to DeepWordBug, but augmenting with either typos or paraphrases, boosts the robustness greatly. Augmenting with paraphrases is more effective than augmenting with typo's which also raises the unattacked accuracy of the detector. Data augmentation could be a effective and practical way to boost robustness to character level attacks. Importantly this was only shown for the essay domain. These results should be evaluated per domain, as the attack accuracy per domain varies greatly, and perhaps different domains also vary in how effective data augmentation is.

While the DeBERTa essay detector is already robust to paraphrasing attacks, we see that data augmentation with paraphrases increases the recall at cost of the precision, finally resulting in a slightly higher F1 score. Since the DeBERTa detector trained on unaugmented data is already robust, it's hard to tell from this experiment if data augmentation will greatly improve the performance. In this particular case we're quickly hitting the 'ceiling'

with the detector.

## 5.6 Answering of Research Questions

We will now return to the research questions and briefly answer them based on our discussed results.

*How well do LLM-generated text detectors perform across different domains of text?*

Performance varies quite substantially across domains. A detector trained on multiple domains scores variably on each individual domain it was trained on. Fine-tuning a detector within a domain had a different maximum performance per domain. Detecting LLM-generated texts is considerably harder in some domains than others.

*How well do LLM-generated text detectors perform out-of-distribution?*

Detectors perform poorly OOD. A detector trained on a whole range of domains performs poorly on an unseen domain, where LLM-generated texts are generated by different LLMs. Even with a custom the decision boundary set to specifically support better OOD performance.

*How robust to adversarial attacks are detectors, and does this robustness vary across domains of text?*

When looking at character based attacks, we see that most detectors have poor adversarial robustness DeepWordBug. A notable exception is in-domain fine-tuned DeBERTa detectors in specific domains. In general we see great variability in the robustness to DeepWordBug across domains. For paraphrasing attacks, we see that adversarial robustness is more nuanced. In most domains we see that paraphrasing causes a dip in the AUROC of the detector, but not in all. When the AUROC does dip, we see that the total amount of decrease in AUROC is generally not large, but that specifically the TPR at a low FPR is affected. In some domains, an in-domain detector has increased performance on paraphrased texts compared to the original texts. Adversarial robustness to paraphrasing varies substantially across domains.

*Does robustness to adversarial attacks of LLM-written text detectors improve when attacked texts are included in their training data?*

This question can only be answered with respect to the essays domain on which we ran our experiments. Our in-domain LLM-generated essay detector was initially not robust to DeepWordBug attacks. Through data augmentation typos or paraphrases we could make the detector robust. Data augmentation with paraphrases worked the best, boosting both the original accuracy and the accuracy under attack. Our in-domain LLM-generated essay detector was initially already highly accurate and robust to paraphrasing, so data augmentation had minimal effect. Data augmentation with paraphrasing did boost the overall F1-score on both the original and paraphrased texts.

# 6. Limitations & future work

Writing a thesis in a research field as dynamic as natural language processing, is bound to have limitations. This section describes some of those limitations, as well as propose directions for future research.

## 6.1 Latest generation LLMs

The field is moving at an incredibly high pace. The scaling of compute, dataset size and amount of parameters seems to hold, as bigger and more capable models are released. The datasets used in this study don't contain the most recent SOTA models (such as GPT-4 [65], Google Gemini [66], and many others). The findings in this study might not necessarily generalize to these newer more capable models. Future work should focus on generating new LLM-generated texts with the latest LLMs and adding them to datasets.

## 6.2 Context length

When comparing the Longformer detector with the DeBERTa in-domain detector, it can be hard to attribute the differences in performance and robustness to only the all domain vs. in-domain setting, since the detectors also vary in context length. This study doesn't directly address the influence of longer context length on performance and robustness of detectors. This might prove crucial as the context length of LLMs is increasing rapidly, in some cases up to millions of tokens (e.g. Google's Gemini model [66]). Furthermore, when a detector has a limited context window, users could write a text within the context limit themselves and have an LLM continue their text, which falls out of the detector context window. In our essay example, students could just write a beginning to an essay consisting of 512 tokens, after which they prompt an LLM the continue their essay. Future work is

needed on the influence of different context lengths on both performance and robustness across various domains.

## 6.3 Collaboration between human and LLM

All of our efforts have focused on detecting if a text was either human-written or LLM-generated. But we don't know how detection would work in cases where humans write colaboratively with LLMs. Intuitively this feels like the way in which many people use these tools to write, rather than generating full texts in a zero-shot manner. The Real or Fage Text (RoFT) framework as proposed by Dugan et al. [67] might be a step in the right direction here. They introduce an evaluation task based on detecting the boundary at which a text that starts off as human-written transitions to being LLM-generated [67]. While they only use the task for human evaluation, it would be interesting to see how automatic detectors can preform at this task. Existing work on automated LLM-generated text detection doesn't seem to provide an answer to the question of the detectability of such a mix of distributions.

## 6.4 Model size

In this study, DeBERTa-base was used, where DeBERTa-large was also available. It has been shown that the larger variants of these LMs perform better on the LLM-generated text detection task. Sadasivan et al. [16] find that in their attacks when comparing different detectors, a RoBERTa-large detector is much more robust to attack than RoBERTa-base in different domains. This begs the question also, if an even larger model, would be even more robust. One idea for future work here would be to test QLoRA [68] fine-tuning an LLM to the classification task. This could be done relatively easily with the open-source LLMs available on Huggingface [60]. Perhaps the much richer internal language model LLMs have obtained through pre-training can be leveraged to more reliable and robust detection of LLM-generated text.

## 6.5 Sampling variance

Most adversarial attacks were performed on a small sample of 100 out of approximately 4000 samples in the test set of each domain, for the MAGE dataset [55]. It could be that some of the variance in robustness between domains is merely sampling variance. There is a chance that the robustness to attacks in certain domains was higher or lower, due to by chance sampling easier or harder texts to attack.

## 6.6 Dominance of the English language

It's non trivial that findings from this study will generalize beyond the English language. Some work has been done on the multi-lingual and multi-domain setting [10], but this remains an understudied area. Future work should focus on creating multilingual datasets and evaluating detector performance and robustness in various languages.

## 6.7 Data augmentation on robust detector/domain

Since our DeBERTa was trained on MGTBench [12] essays was already quite robust to paraphrasing without data augmentation, it doesn't show all too much about the possible improvement by using data augmentation in a domain where detectors are not robust to paraphrasing. In hindsight it would have been better to apply data augmentation to a domain that has worse robustness to paraphrasing, to show the extent to which this can be improved by data augmentation. Future work should be done on data augmentation to improve robustness to paraphrasing in domains that where detectors are not robust.

## 6.8 Generalization within domains

While we've trained detectors within certain domains by training it on some data from that domain, it's hard to say how representative these domain

dataset are of the whole domain. In the case of essays, we train on only 1000 human written essays. A 'domain' of text can in some cases be tricky to define. Essays for one course might follow vastly different distributions that essays from another course. People in primary school write different essays than Philosophy PhDs. In some cases, 'domain' might still not be the right level of abstraction to encapsulate a human-written vs. LLM-generated distribution.

## 6.9 Limited amount of attack strategies

Another limitation is that only for both character perturbations and paraphrasing, only one attack strategy was used. Many other attacks exist, including but not limited to homoglyph attacks, using an LLM directly to paraphrase texts, or commercial tools as listed in Appendix C. These techniques might perturb the text differently and have a different effect on detector performance than shown in this study. Detectors that are shown to be robust to attacks in this study might not be robust to different attacks. Further work should be done to asses how effective different attacks are in different domains. This might involve reverse-engineering how commercial detection evaders work and experimenting with potential defenses.

## 6.10 The need for high-quality data

To ensure that detection is possible, we need high-quality human-written texts from various different domains, in different languages. This data should be prevented from entering into the training data of LLMs, to rule out any data leakage. This would make sure that the human-written texts aren't trivially reproduced by the next generation LLMs. Having this data leak could also allow adversaries to exploit the differences in distributions. A bad actor could for example fine-tune open-source LLMs to generate more human-like texts based on such data.

LLM-generated texts could be generated relatively easily, from such a dataset of human-written texts, as has been done to create the datasets in

this study. Existing datasets often have a class imbalance skewed towards LLM-generated texts as it is so much easier, cheaper and faster to generate text with an LLM when compared to having humans write texts. How to create such a high-quality dataset with a wide variaty of high-quality human-written text data, prevent LLMs from being trained on it, put it in the hands of good actors and out of the hands of bad actors, is a practical problem that we don't see a simple solution for unfortunately.

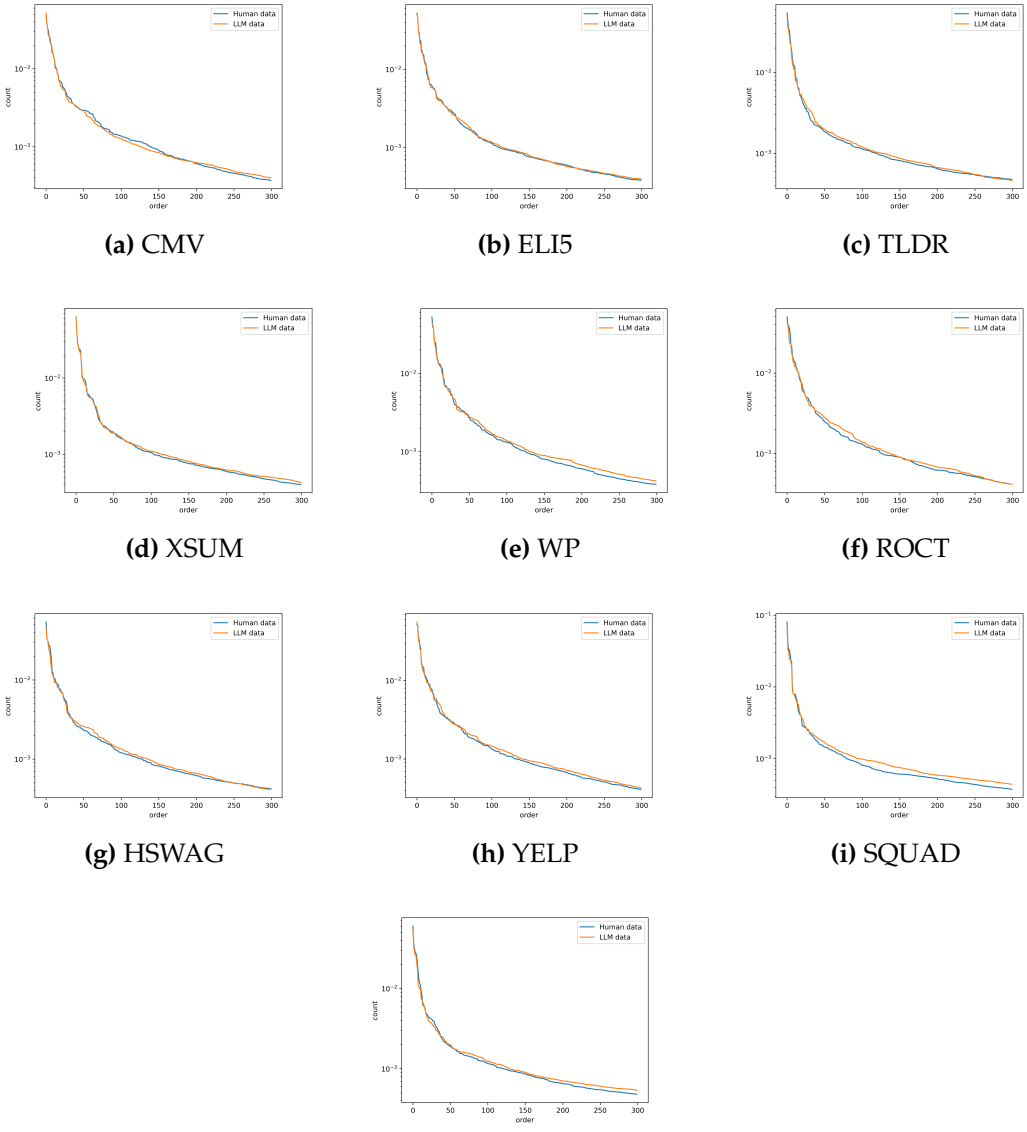
## 6.11 Contrastive learning

Our detectors all use the cross-entropy loss during learning. However such a loss doesn't maximize the distance between positive and negative samples in the embedding space [69]. Contrastive learning seems like an interesting avenue of research. It might help create better representations of human-written and LLM-generated text and thus better detectors. Some promising work has already been done in this area [70], [71]. It would be interesting to see more work in the multi-domain setting combined with adversarial attacks.

# A. Ethics & Privacy scan Utrecht University

The Ethics and Privacy Quick Scan of the Utrecht University Research Institute of Information and Computing Sciences classifies this research as low-risk with no fuller ethics review or privacy assessment required.

# B. Additional Tables and Figures




**(a)** CMV **(b)** ELI5 **(c)** TLDR

**(d)** XSUM **(e)** WP **(f)** ROCT

**(g)** HSWAG **(h)** YELP **(i)** SQUAD

**(j)** SCI_GEN

**Figure B.1:** Ordered frequencies of each word, ranked from highest to lowest. Frequencies are displayed in a log space. Each plot shows human-written vs. LLM-generated texts distributions for one domain.

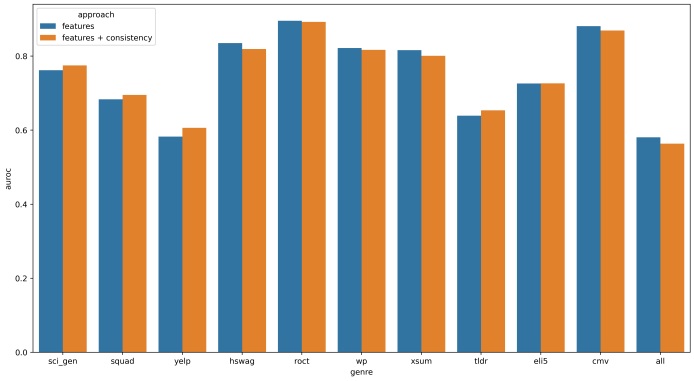



**Figure B.2:** AUROC of a logistic regression classifier based on features from the literature, including vs. excluding consistency features on a subsample of N=15344

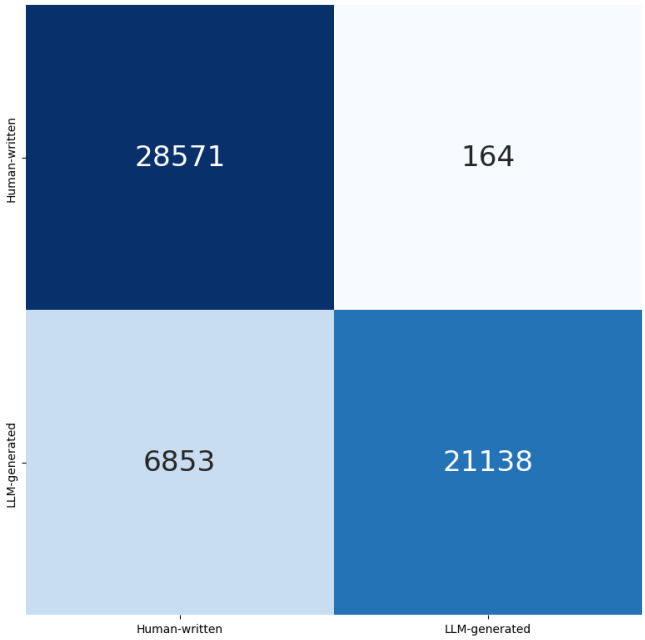



**Figure B.3:** Confusion matrix of Longformer on entire MAGE dataset, given the custom decision boundary.

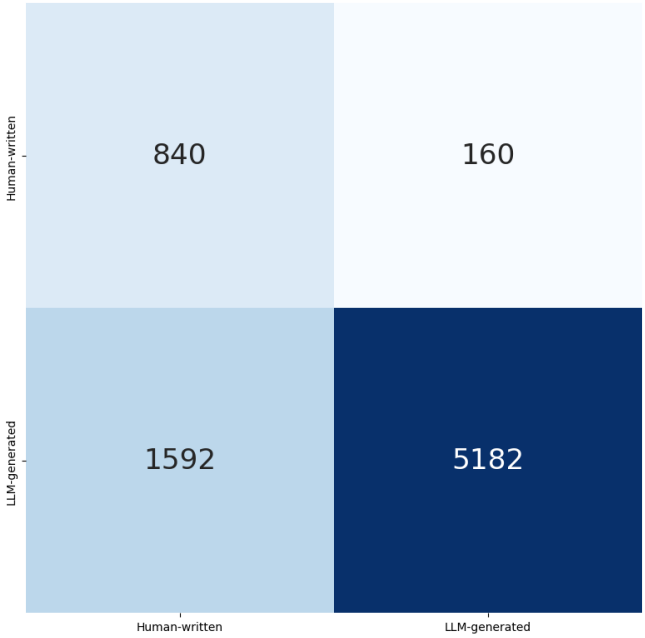
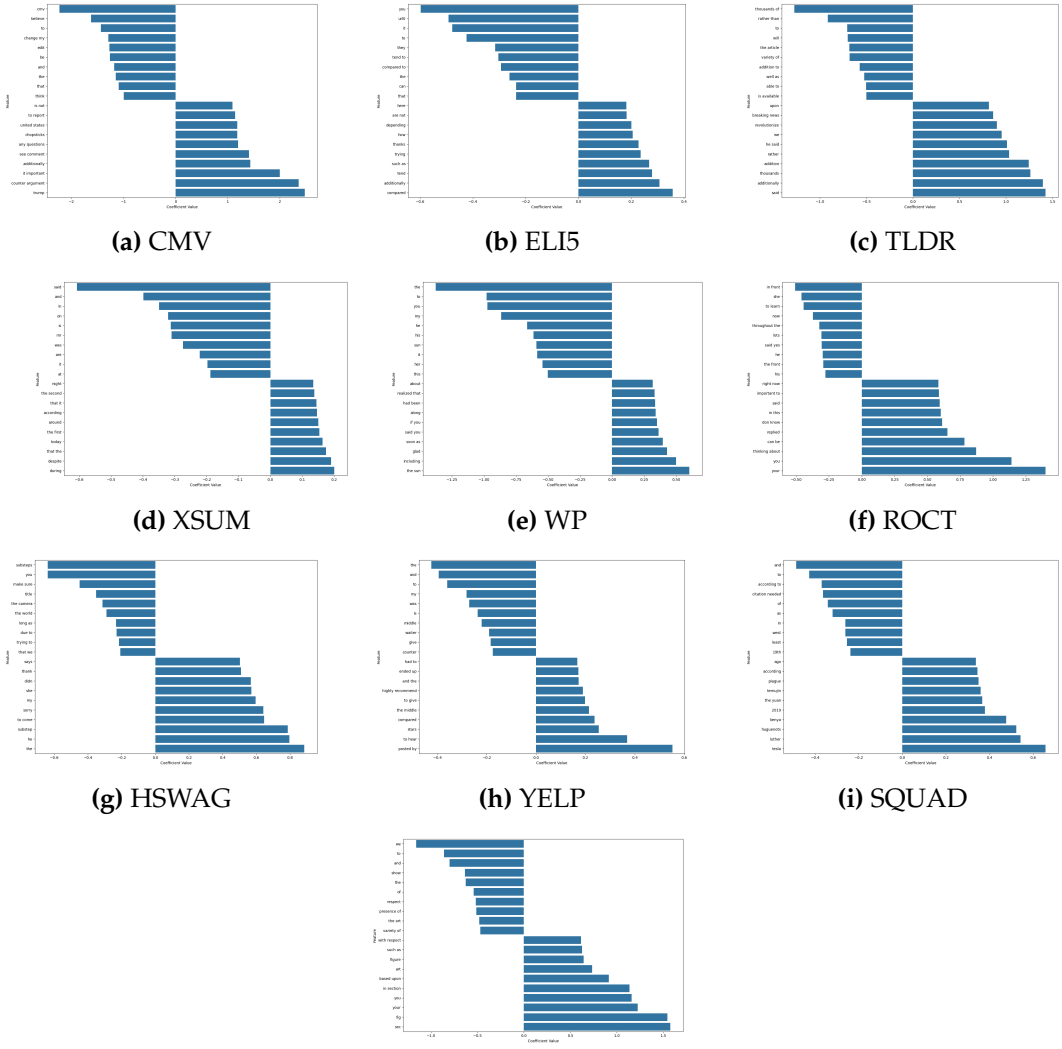



**Figure B.4:** Confusion matrix of Longformer OOD essay domain, given the custom decision boundary.



**(a)** CMV **(b)** ELI5 **(c)** TLDR

**(d)** XSUM **(e)** WP **(f)** ROCT

**(g)** HSWAG **(h)** YELP **(i)** SQUAD

**(j)** SCI_GEN

**Figure B.5:** Top 10 features for human-written (lower coefficients) vs. LLM-generated (higher coefficients), of the TF.IDF detector for each domain.

# C. Commercial AI detection evasion tools

The relevance of evading detection through adversarial attacks is evident by the abundant availability of commercial tools that aim to achieve this goal (items are clickable):

- undetectable.ai
- stealthgpt.ai
- writehuman.ai
- stealthwriter.ai
- conch.ai

# Bibliography


[1] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, *Attention Is All You Need*, Aug. 2023. DOI: 10.48550/arXiv.1706.03762. arXiv: 1706.03762 [cs]. (visited on 01/11/2024).

[2] *Introducing ChatGPT*, https://openai.com/blog/chatgpt. (visited on 01/11/2024).

[3] K. Hu and K. Hu, "ChatGPT sets record for fastest-growing user base - analyst note," *Reuters*, Feb. 2023. (visited on 01/11/2024).

[4] W. X. Zhao, K. Zhou, J. Li, T. Tang, X. Wang, Y. Hou, Y. Min, B. Zhang, J. Zhang, Z. Dong, Y. Du, C. Yang, Y. Chen, Z. Chen, J. Jiang, R. Ren, Y. Li, X. Tang, Z. Liu, P. Liu, J.-Y. Nie, and J.-R. Wen, *A Survey of Large Language Models*, Nov. 2023. DOI: 10.48550/arXiv.2303.18223. arXiv: 2303.18223 [cs]. (visited on 01/10/2024).

[5] E. N. Crothers, N. Japkowicz, and H. L. Viktor, "Machine-Generated Text: A Comprehensive Survey of Threat Models and Detection Methods," *IEEE Access*, vol. 11, pp. 70 977–71 002, 2023, ISSN: 2169-3536. DOI: 10.1109/ACCESS.2023.3294090. (visited on 01/11/2024).

[6] J. Wu, S. Yang, R. Zhan, Y. Yuan, D. F. Wong, and L. S. Chao, *A Survey on LLM-generated Text Detection: Necessity, Methods, and Future Directions*, Oct. 2023. DOI: 10.48550/arXiv.2310.14724. arXiv: 2310.14724 [cs]. (visited on 01/16/2024).

[7] X. Yang, L. Pan, X. Zhao, H. Chen, L. Petzold, W. Y. Wang, and W. Cheng, *A Survey on Detection of LLMs-Generated Content*, Oct. 2023. DOI: 10.48550/arXiv.2310.15654. arXiv: 2310.15654 [cs]. (visited on 01/11/2024).

[8] E. Clark, T. August, S. Serrano, N. Haduong, S. Gururangan, and N. A. Smith, *All That's 'Human' Is Not Gold: Evaluating Human Evaluation of Generated Text*, Jul. 2021. DOI: 10.48550/arXiv.2107.00061. arXiv: 2107.00061 [cs]. (visited on 01/14/2024).

[9] A. Uchendu, Z. Ma, T. Le, R. Zhang, and D. Lee, *TURINGBENCH: A Benchmark Environment for Turing Test in the Age of Neural Text Generation*, Sep. 2021. DOI: 10.48550/arXiv.2109.13296. arXiv: 2109.13296 [cs]. (visited on 01/17/2024).

[10] Y. Wang, J. Mansurov, P. Ivanov, J. Su, A. Shelmanov, A. Tsvigun, C. Whitehouse, O. M. Afzal, T. Mahmoud, A. F. Aji, and P. Nakov, *M4: Multi-generator, Multi-domain, and Multi-lingual Black-Box Machine-Generated Text Detection*, May 2023. DOI: 10.48550/arXiv.2305.14902. arXiv: 2305.14902 [cs]. (visited on 02/29/2024).

[11] Y. Li, Q. Li, L. Cui, W. Bi, Z. Wang, L. Wang, L. Yang, S. Shi, and Y. Zhang, *MAGE: Machine-generated Text Detection in the Wild*, May 2024. arXiv: 2305.13242 [cs]. (visited on 06/13/2024).

[12] X. He, X. Shen, Z. Chen, M. Backes, and Y. Zhang, *MGTBench: Benchmarking Machine-Generated Text Detection*, Jan. 2024. arXiv: 2303.14822 [cs]. (visited on 01/30/2024).

[13] Y. Wang, J. Mansurov, P. Ivanov, J. Su, A. Shelmanov, A. Tsvigun, C. Whitehouse, O. M. Afzal, T. Mahmoud, A. F. Aji, and P. Nakov, *M4: Multi-generator, Multi-domain, and Multi-lingual Black-Box Machine-Generated Text Detection*, May 2023. DOI: 10.48550/arXiv.2305.14902. arXiv: 2305.14902 [cs]. (visited on 01/30/2024).

[14] E. Crothers, N. Japkowicz, H. Viktor, and P. Branco, "Adversarial Robustness of Neural-Statistical Features in Detection of Generative Transformers," in *2022 International Joint Conference on Neural Networks (IJCNN)*, Jul. 2022, pp. 1–8. DOI: 10.1109/IJCNN55064.2022.9892269. arXiv: 2203.07983 [cs]. (visited on 01/17/2024).

[15] K. Krishna, Y. Song, M. Karpinska, J. Wieting, and M. Iyyer, *Paraphrasing evades detectors of AI-generated text, but retrieval is an effective defense*, Oct. 2023. DOI: 10.48550/arXiv.2303.13408. arXiv: 2303.13408 [cs]. (visited on 01/11/2024).

[16] V. S. Sadasivan, A. Kumar, S. Balasubramanian, W. Wang, and S. Feizi, *Can AI-Generated Text be Reliably Detected?* Jun. 2023. DOI: 10.48550/arXiv.2303.11156. arXiv: 2303.11156 [cs]. (visited on 01/14/2024).

[17] M. Wolff and S. Wolff, *Attacking Neural Text Detectors*, Jan. 2022. DOI: 10.48550/arXiv.2002.11768. arXiv: 2002.11768 [cs]. (visited on 01/14/2024).

[18] D. Jin, Z. Jin, J. T. Zhou, and P. Szolovits, "Is BERT Really Robust? A Strong Baseline for Natural Language Attack on Text Classification and Entailment," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 34, no. 05, pp. 8018–8025, Apr. 2020, ISSN: 2374-3468. DOI: 10.1609/aaai.v34i05.6311. (visited on 01/30/2024).

[19] H. Stiff and F. Johansson, "Detecting computer-generated disinformation," *International Journal of Data Science and Analytics*, vol. 13, no. 4, pp. 363–383, May 2022, ISSN: 2364-4168. DOI: 10.1007/s41060-021-00299-5. (visited on 01/27/2024).

[20] J. J. Webster and C. Kit, "Tokenization as the Initial Phase in NLP," in *COLING 1992 Volume 4: The 14th International Conference on Computational Linguistics*, 1992. (visited on 01/18/2024).

[21] S. J. Mielke, Z. Alyafeai, E. Salesky, C. Raffel, M. Dey, M. Gallé, A. Raja, C. Si, W. Y. Lee, B. Sagot, and S. Tan, *Between words and characters: A Brief History of Open-Vocabulary Modeling and Tokenization in NLP*, Dec. 2021. DOI: 10.48550/arXiv.2112.10508. arXiv: 2112.10508 [cs]. (visited on 01/18/2024).

[22] P. Gage, "A new algorithm for data compression," *The C Users Journal archive*, Feb. 1994. (visited on 01/18/2024).

[23] R. Sennrich, B. Haddow, and A. 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)*, K. Erk and N. A. Smith, Eds., Berlin, Germany: Association for Computational Linguistics, Aug. 2016, pp. 1715–1725. DOI: 10.18653/v1/P16-1162. (visited on 01/18/2024).

[24] C. Wang, K. Cho, and J. Gu, *Neural Machine Translation with Byte-Level Subwords*, Dec. 2019. DOI: 10.48550/arXiv.1909.03341. arXiv: 1909.03341 [cs]. (visited on 01/18/2024).

[25] M. Schuster and K. Nakajima, "Japanese and Korean voice search," in *2012 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, Mar. 2012, pp. 5149–5152. DOI: 10.1109/ICASSP.2012.6289079. (visited on 01/18/2024).

[26] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding*, May 2019. DOI: 10.48550/arXiv.1810.04805. arXiv: 1810.04805 [cs]. (visited on 01/25/2024).

[27] Y. Liu, M. Ott, N. Goyal, J. Du, M. Joshi, D. Chen, O. Levy, M. Lewis, L. Zettlemoyer, and V. Stoyanov, *RoBERTa: A Robustly Optimized BERT Pretraining Approach*, Jul. 2019. DOI: 10.48550/arXiv.1907.11692. arXiv: 1907.11692 [cs]. (visited on 01/25/2024).

[28] T. Kudo and J. Richardson, "SentencePiece: A simple and language independent subword tokenizer and detokenizer for Neural Text Processing," in *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, E. Blanco and W. Lu, Eds., Brussels, Belgium: Association for Computational Linguistics, Nov. 2018, pp. 66–71. DOI: 10.18653/v1/D18-2012. (visited on 01/18/2024).

[29] A. Holtzman, J. Buys, L. Du, M. Forbes, and Y. Choi, *The Curious Case of Neural Text Degeneration*, Feb. 2020. DOI: 10.48550/arXiv.1904.09751. arXiv: 1904.09751 [cs]. (visited on 01/18/2024).

[30] D. Ippolito, N. Carlini, K. Lee, M. Nasr, and Y. W. Yu, *Reverse-Engineering Decoding Strategies Given Blackbox Access to a Language Generation System*, Sep. 2023. DOI: 10.48550/arXiv.2309.04858. arXiv: 2309.04858 [cs]. (visited on 01/23/2024).

[31] Defense Technical Information Center, *Speech Understanding Systems. Summary of Results of the Five-Year Research Effort at Carnegie-Mellon University*. Aug. 1977. (visited on 01/21/2024).

[32] D. Furcy and S. Koenig, "Limited discrepancy beam search," in *Proceedings of the 19th International Joint Conference on Artificial Intelligence*, ser. IJCAI'05, San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., Jul. 2005, pp. 125–131. (visited on 01/21/2024).

[33] A. Fan, M. Lewis, and Y. Dauphin, "Hierarchical Neural Story Generation," in *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, I. Gurevych and Y. Miyao, Eds., Melbourne, Australia: Association for Compu-

tational Linguistics, Jul. 2018, pp. 889–898. DOI: 10.18653/v1/P18-1082. (visited on 01/21/2024).

[34] H. Touvron, L. Martin, K. Stone, P. Albert, A. Almahairi, Y. Babaei, N. Bashlykov, S. Batra, P. Bhargava, S. Bhosale, D. Bikel, L. Blecher, C. C. Ferrer, M. Chen, G. Cucurull, D. Esiobu, J. Fernandes, J. Fu, W. Fu, B. Fuller, C. Gao, V. Goswami, N. Goyal, A. Hartshorn, S. Hosseini, R. Hou, H. Inan, M. Kardas, V. Kerkez, M. Khabsa, I. Kloumann, A. Korenev, P. S. Koura, M.-A. Lachaux, T. Lavril, J. Lee, D. Liskovich, Y. Lu, Y. Mao, X. Martinet, T. Mihaylov, P. Mishra, I. Molybog, Y. Nie, A. Poulton, J. Reizenstein, R. Rungta, K. Saladi, A. Schelten, R. Silva, E. M. Smith, R. Subramanian, X. E. Tan, B. Tang, R. Taylor, A. Williams, J. X. Kuan, P. Xu, Z. Yan, I. Zarov, Y. Zhang, A. Fan, M. Kambadur, S. Narang, A. Rodriguez, R. Stojnic, S. Edunov, and T. Scialom, *Llama 2: Open Foundation and Fine-Tuned Chat Models*, Jul. 2023. DOI: 10.48550/arXiv.2307.09288. arXiv: 2307.09288 [cs]. (visited on 01/23/2024).

[35] R. Lou, K. Zhang, and W. Yin, *A Comprehensive Survey on Instruction Following*, Jan. 2024. DOI: 10.48550/arXiv.2303.10475. arXiv: 2303.10475 [cs]. (visited on 01/23/2024).

[36] L. Ouyang, J. Wu, X. Jiang, D. Almeida, C. L. Wainwright, P. Mishkin, C. Zhang, S. Agarwal, K. Slama, A. Ray, J. Schulman, J. Hilton, F. Kelton, L. Miller, M. Simens, A. Askell, P. Welinder, P. Christiano, J. Leike, and R. Lowe, *Training language models to follow instructions with human feedback*, Mar. 2022. DOI: 10.48550/arXiv.2203.02155. arXiv: 2203.02155 [cs]. (visited on 01/23/2024).

[37] A. Uchendu, T. Le, and D. Lee, *Attribution and Obfuscation of Neural Text Authorship: A Data Mining Perspective*, Mar. 2023. DOI: 10.48550/arXiv.2210.10488. arXiv: 2210.10488 [cs]. (visited on 01/25/2024).

[38] D. Ippolito, D. Duckworth, C. Callison-Burch, and D. Eck, *Automatic Detection of Generated Text is Easiest when Humans are Fooled*, May 2020. DOI: 10.48550/arXiv.1911.00650. arXiv: 1911.00650 [cs]. (visited on 01/16/2024).

[39] *New AI classifier for indicating AI-written text*, https://openai.com/blog/new-ai-classifier-for-indicating-ai-written-text. (visited on 01/26/2024).

[40] M. Atari, M. J. Xue, P. S. Park, D. Blasi, and J. Henrich, "Which Humans?," Jan. 2024. DOI: 10.31234/osf.io/5b26t. (visited on 01/24/2024).

[41] S. Gehrmann, H. Strobelt, and A. M. Rush, *GLTR: Statistical Detection and Visualization of Generated Text*, Jun. 2019. DOI: 10.48550/arXiv.1906.04043. arXiv: 1906.04043 [cs]. (visited on 01/17/2024).

[42] Y. Dou, M. Forbes, R. Koncel-Kedziorski, N. A. Smith, and Y. Choi, "Is GPT-3 Text Indistinguishable from Human Text? Scarecrow: A Framework for Scrutinizing Machine Text," in *Proceedings of the 60th*

*Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, S. Muresan, P. Nakov, and A. Villavicencio, Eds., Dublin, Ireland: Association for Computational Linguistics, May 2022, pp. 7250–7274. DOI: 10.18653/v1/2022.acl-long.501. (visited on 01/17/2024).

[43] D. Biber, *Variation across Speech and Writing*. Cambridge: Cambridge University Press, 1988, ISBN: 978-0-521-42556-8. DOI: 10.1017/CBO9780511621024. (visited on 02/15/2024).

[44] H.-Q. Nguyen-Son, N.-D. T. Tieu, H. H. Nguyen, J. Yamagishi, and I. E. Zen, "Identifying computer-generated text using statistical analysis," in *2017 Asia-Pacific Signal and Information Processing Association Annual Summit and Conference (APSIPA ASC)*, Dec. 2017, pp. 1504–1511. DOI: 10.1109/APSIPA.2017.8282270. (visited on 01/24/2024).

[45] L. Fröhling and A. Zubiaga, "Feature-based detection of automated language models: Tackling GPT-2, GPT-3 and Grover," *PeerJ Computer Science*, vol. 7, e443, Apr. 2021, ISSN: 2376-5992. DOI: 10.7717/peerj-cs.443. (visited on 01/24/2024).

[46] I. Solaiman, M. Brundage, J. Clark, A. Askell, A. Herbert-Voss, J. Wu, A. Radford, G. Krueger, J. W. Kim, S. Kreps, M. McCain, A. Newhouse, J. Blazakis, K. McGuffie, and J. Wang, *Release Strategies and the Social Impacts of Language Models*, Nov. 2019. DOI: 10.48550/arXiv.1908.09203. arXiv: 1908.09203 [cs]. (visited on 01/24/2024).

[47] G. K. Zipf, "Selected Studies of the Principle of Relative Frequency in Language," in *Selected Studies of the Principle of Relative Frequency in Language*, Harvard University Press, Dec. 1932, ISBN: 978-0-674-43492-9. DOI: 10.4159/harvard.9780674434929. (visited on 01/24/2024).

[48] E. Fast, B. Chen, and M. S. Bernstein, "Empath: Understanding Topic Signals in Large-Scale Text," in *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems*, ser. CHI '16, New York, NY, USA: Association for Computing Machinery, May 2016, pp. 4647–4657, ISBN: 978-1-4503-3362-7. DOI: 10.1145/2858036.2858535. (visited on 01/25/2024).

[49] X. He, X. Shen, Z. Chen, M. Backes, and Y. Zhang, *MGTBench: Benchmarking Machine-Generated Text Detection*, Jan. 2024. DOI: 10.48550/arXiv.2303.14822. arXiv: 2303.14822 [cs]. (visited on 02/29/2024).

[50] M. M. Bhat and S. Parthasarathy, "How Effectively Can Machines Defend Against Machine-Generated Fake News? An Empirical Study," in *Proceedings of the First Workshop on Insights from Negative Results in NLP*, A. Rogers, J. Sedoc, and A. Rumshisky, Eds., Online: Association for Computational Linguistics, Nov. 2020, pp. 48–53. DOI: 10.18653/v1/2020.insights-1.7. (visited on 01/30/2024).

[51] *Humarin/chatgpt_paraphraser_on_T5_base · Hugging Face*, https://huggingface.co/humarin/chatgpt_paraphraser_on_-T5_base, Apr. 2023. (visited on 06/13/2024).

[52] I. Beltagy, M. E. Peters, and A. Cohan, *Longformer: The Long-Document Transformer*, Dec. 2020. DOI: 10.48550/arXiv.2004.05150. arXiv: 2004.05150 [cs]. (visited on 01/30/2024).

[53] S. Ren, Y. Deng, K. He, and W. Che, "Generating Natural Language Adversarial Examples through Probability Weighted Word Saliency," in *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, A. Korhonen, D. Traum, and L. Màrquez, Eds., Florence, Italy: Association for Computational Linguistics, Jul. 2019, pp. 1085–1097. DOI: 10.18653/v1/P19-1103. (visited on 06/14/2024).

[54] J. X. Morris, E. Lifland, J. Y. Yoo, J. Grigsby, D. Jin, and Y. Qi, *TextAttack: A Framework for Adversarial Attacks, Data Augmentation, and Adversarial Training in NLP*, Oct. 2020. DOI: 10.48550/arXiv.2005.05909. arXiv: 2005.05909 [cs]. (visited on 01/17/2024).

[55] Y. Li, Q. Li, L. Cui, W. Bi, Z. Wang, L. Wang, L. Yang, S. Shi, and Y. Zhang, *MAGE: Machine-generated Text Detection in the Wild*, May 2024. DOI: 10.48550/arXiv.2305.13242. arXiv: 2305.13242 [cs]. (visited on 06/13/2024).

[56] R. Gunning, "The technique of clear writing," *(No Title)*, (visited on 04/03/2024).

[57] R. Flesch, "A new readability yardstick," *Journal of Applied Psychology*, vol. 32, no. 3, pp. 221–233, 1948, ISSN: 1939-1854. DOI: 10.1037/h0057532.

[58] *Pyspellchecker — pyspellchecker 0.8.2 documentation*, https://pyspellchecker.readthedocs.io/en/latest/. (visited on 04/30/2024).

[59] P. He, J. Gao, and W. Chen, *DeBERTaV3: Improving DeBERTa using ELECTRA-Style Pre-Training with Gradient-Disentangled Embedding Sharing*, Mar. 2023. DOI: 10.48550/arXiv.2111.09543. arXiv: 2111.09543 [cs]. (visited on 05/13/2024).

[60] T. Wolf, L. Debut, V. Sanh, J. Chaumond, C. Delangue, A. Moi, P. Cistac, T. Rault, R. Louf, M. Funtowicz, J. Davison, S. Shleifer, P. von Platen, C. Ma, Y. Jernite, J. Plu, C. Xu, T. L. Scao, S. Gugger, M. Drame, Q. Lhoest, and A. M. Rush, *HuggingFace's Transformers: State-of-the-art Natural Language Processing*, Jul. 2020. DOI: 10.48550/arXiv.1910.03771. arXiv: 1910.03771 [cs]. (visited on 05/13/2024).

[61] L. McInnes, J. Healy, and J. Melville, *UMAP: Uniform Manifold Approximation and Projection for Dimension Reduction*, Sep. 2020. DOI: 10.48550/arXiv.1802.03426. arXiv: 1802.03426 [cs, stat]. (visited on 02/09/2024).

[62] J. Gao, J. Lanchantin, M. L. Soffa, and Y. Qi, *Black-box Generation of Adversarial Text Sequences to Evade Deep Learning Classifiers*, May

2018. DOI: 10.48550/arXiv.1801.04354. arXiv: 1801.04354 [cs]. (visited on 02/15/2024).

[63] D. Li, Y. Zhang, H. Peng, L. Chen, C. Brockett, M.-T. Sun, and B. Dolan, *Contextualized Perturbation for Textual Adversarial Attack*, Mar. 2021. DOI: 10.48550/arXiv.2009.07502. arXiv: 2009.07502 [cs]. (visited on 02/28/2024).

[64] *Makcedward/nlpaug: Data augmentation for NLP*, https://github.com/makcedward/nlpaug. (visited on 06/06/2024).

[65] OpenAI, J. Achiam, S. Adler, S. Agarwal, L. Ahmad, I. Akkaya, F. L. Aleman, D. Almeida, J. Altenschmidt, S. Altman, S. Anadkat, R. Avila, I. Babuschkin, S. Balaji, V. Balcom, P. Baltescu, H. Bao, M. Bavarian, J. Belgum, I. Bello, J. Berdine, G. Bernadett-Shapiro, C. Berner, L. Bogdonoff, O. Boiko, M. Boyd, A.-L. Brakman, G. Brockman, T. Brooks, M. Brundage, K. Button, T. Cai, R. Campbell, A. Cann, B. Carey, C. Carlson, R. Carmichael, B. Chan, C. Chang, F. Chantzis, D. Chen, S. Chen, R. Chen, J. Chen, M. Chen, B. Chess, C. Cho, C. Chu, H. W. Chung, D. Cummings, J. Currier, Y. Dai, C. Decareaux, T. Degry, N. Deutsch, D. Deville, A. Dhar, D. Dohan, S. Dowling, S. Dunning, A. Ecoffet, A. Eleti, T. Eloundou, D. Farhi, L. Fedus, N. Felix, S. P. Fishman, J. Forte, I. Fulford, L. Gao, E. Georges, C. Gibson, V. Goel, T. Gogineni, G. Goh, R. Gontijo-Lopes, J. Gordon, M. Grafstein, S. Gray, R. Greene, J. Gross, S. S. Gu, Y. Guo, C. Hallacy, J. Han, J. Harris, Y. He, M. Heaton, J. Heidecke, C. Hesse, A. Hickey, W. Hickey, P. Hoeschele, B. Houghton, K. Hsu, S. Hu, X. Hu, J. Huizinga, S. Jain, S. Jain, J. Jang, A. Jiang, R. Jiang, H. Jin, D. Jin, S. Jomoto, B. Jonn, H. Jun, T. Kaftan, Ł. Kaiser, A. Kamali, I. Kanitscheider, N. S. Keskar, T. Khan, L. Kilpatrick, J. W. Kim, C. Kim, Y. Kim, J. H. Kirchner, J. Kiros, M. Knight, D. Kokotajlo, Ł. Kondraciuk, A. Kondrich, A. Konstantinidis, K. Kosic, G. Krueger, V. Kuo, M. Lampe, I. Lan, T. Lee, J. Leike, J. Leung, D. Levy, C. M. Li, R. Lim, M. Lin, S. Lin, M. Litwin, T. Lopez, R. Lowe, P. Lue, A. Makanju, K. Malfacini, S. Manning, T. Markov, Y. Markovski, B. Martin, K. Mayer, A. Mayne, B. McGrew, S. M. McKinney, C. McLeavey, P. McMillan, J. McNeil, D. Medina, A. Mehta, J. Menick, L. Metz, A. Mishchenko, P. Mishkin, V. Monaco, E. Morikawa, D. Mossing, T. Mu, M. Murati, O. Murk, D. Mély, A. Nair, R. Nakano, R. Nayak, A. Neelakantan, R. Ngo, H. Noh, L. Ouyang, C. O'Keefe, J. Pachocki, A. Paino, J. Palermo, A. Pantuliano, G. Parascandolo, J. Parish, E. Parparita, A. Passos, M. Pavlov, A. Peng, A. Perelman, F. d. A. B. Peres, M. Petrov, H. P. d. O. Pinto, Michael, Pokorny, M. Pokrass, V. H. Pong, T. Powell, A. Power, B. Power, E. Proehl, R. Puri, A. Radford, J. Rae, A. Ramesh, C. Raymond, F. Real, K. Rimbach, C. Ross, B. Rotsted, H. Roussez, N. Ryder, M. Saltarelli, T. Sanders, S. Santurkar, G. Sastry, H. Schmidt, D. Schnurr, J. Schulman, D. Selsam, K. Sheppard, T. Sherbakov, J.

Shieh, S. Shoker, P. Shyam, S. Sidor, E. Sigler, M. Simens, J. Sitkin, K. Slama, I. Sohl, B. Sokolowsky, Y. Song, N. Staudacher, F. P. Such, N. Summers, I. Sutskever, J. Tang, N. Tezak, M. B. Thompson, P. Tillet, A. Tootoonchian, E. Tseng, P. Tuggle, N. Turley, J. Tworek, J. F. C. Uribe, A. Vallone, A. Vijayvergiya, C. Voss, C. Wainwright, J. J. Wang, A. Wang, B. Wang, J. Ward, J. Wei, C. J. Weinmann, A. Welihinda, P. Welinder, J. Weng, L. Weng, M. Wiethoff, D. Willner, C. Winter, S. Wolrich, H. Wong, L. Workman, S. Wu, J. Wu, M. Wu, K. Xiao, T. Xu, S. Yoo, K. Yu, Q. Yuan, W. Zaremba, R. Zellers, C. Zhang, M. Zhang, S. Zhao, T. Zheng, J. Zhuang, W. Zhuk, and B. Zoph, *GPT-4 Technical Report*, Mar. 2024. DOI: 10.48550/arXiv.2303.08774. arXiv: 2303.08774 [cs]. (visited on 06/18/2024).

[66] G. Team, R. Anil, S. Borgeaud, *et al.*, *Gemini: A Family of Highly Capable Multimodal Models*, May 2024. DOI: 10.48550/arXiv.2312.11805. arXiv: 2312.11805 [cs]. (visited on 06/18/2024).

[67] L. Dugan, D. Ippolito, A. Kirubarajan, and C. Callison-Burch, "RoFT: A Tool for Evaluating Human Detection of Machine-Generated Text," in *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, Q. Liu and D. Schlangen, Eds., Online: Association for Computational Linguistics, Oct. 2020, pp. 189–196. DOI: 10.18653/v1/2020.emnlp-demos.25. (visited on 01/17/2024).

[68] T. Dettmers, A. Pagnoni, A. Holtzman, and L. Zettlemoyer, *QLoRA: Efficient Finetuning of Quantized LLMs*, May 2023. DOI: 10.48550/arXiv.2305.14314. arXiv: 2305.14314 [cs]. (visited on 06/18/2024).

[69] Y. Moukafih, N. Sbihi, M. Ghogho, and K. Smaili, "SuperConText: Supervised Contrastive Learning Framework for Textual Representations," *IEEE Access*, vol. 11, pp. 16 820–16 830, 2023, ISSN: 2169-3536. DOI: 10.1109/ACCESS.2023.3241490. (visited on 04/30/2024).

[70] X. Liu, Z. Zhang, Y. Wang, H. Pu, Y. Lan, and C. Shen, *CoCo: Coherence-Enhanced Machine-Generated Text Detection Under Data Limitation With Contrastive Learning*, Oct. 2023. DOI: 10.48550/arXiv.2212.10341. arXiv: 2212.10341 [cs]. (visited on 01/27/2024).

[71] A. Bhattacharjee, T. Kumarage, R. Moraffah, and H. Liu, *ConDA: Contrastive Domain Adaptation for AI-generated Text Detection*, Sep. 2023. DOI: 10.48550/arXiv.2309.03992. arXiv: 2309.03992 [cs]. (visited on 01/27/2024).