arXiv:2305.19713v1 [cs.CL] 31 May 2023

Red Teaming Language Model Detectors with Language Models

Zhouxing Shi*, **Yihan Wang***, **Fan Yin***, **Xiangning Chen**, **Kai-Wei Chang**, **Cho-Jui Hsieh**
University of California, Los Angeles
{zshi, yihanwang, fanyin20, xiangning, kwchang, chohsieh}@cs.ucla.edu
*Alphabetical order

Abstract

The prevalence and high capacity of large language models (LLMs) present significant safety and ethical risks when malicious users exploit them for automated content generation. To prevent the potentially deceptive usage of LLMs, recent works have proposed several algorithms to detect machine-generated text. In this paper, we systematically test the reliability of the existing detectors, by designing two types of attack strategies to fool the detectors: 1) replacing words with their synonyms based on the context; 2) altering the writing style of generated text. These strategies are implemented by instructing LLMs to generate synonymous word substitutions or writing directives that modify the style without human involvement, and the LLMs leveraged in the attack can also be protected by detectors. Our research reveals that our attacks effectively compromise the performance of all tested detectors, thereby underscoring the urgent need for the development of more robust machine-generated text detection systems.

1 Introduction

Large language models (LLMs), such as ChatGPT (OpenAI, 2023b), and PaLM (Chowdhery et al., 2022), have demonstrated human-like capabilities to generate high-quality content, follow instructions, and respond to user queries. Although LLMs can improve the working efficiency of humans, they also pose several ethical and safety concerns, when it becomes harder to differentiate between text written by a human and text generated by an LLM. For example, LLMs may be inappropriately used for academic plagiarism or creating misinformation at large scale (Zellers et al., 2019). Therefore, it is important to develop reliable approaches to protecting LLMs and detecting the presence of AI-generated texts to mitigate the abuse of LLMs.

Towards this end, prior works have developed methods for automatically detecting text generated by LLMs. The existing methods mainly fall into three categories: 1) Classifier-based detectors by training a classifier, often a neural network, from supervised data with AI-generated/human-written labels (Solaiman et al., 2019; OpenAI, 2023a); 2) Watermarking (Kirchenbauer et al., 2023) by injecting patterns into the generation of LLMs such that the pattern can be statistically detected but imperceptible to humans; 3) Likelihood-based detector, e.g., DetectGPT (Mitchell et al., 2023), by leveraging the log-likelihood of the generated texts.

However, as recent research demonstrates that text classifiers are vulnerable (Iyyer et al., 2018; Ribeiro et al., 2018; Alzantot et al., 2018), we suspect that these detectors are not reliable under adversarial manipulations of AI-generated texts. To stress-test the reliability of the detectors, we red team and attack the detectors by prompting an LLM-based generative model. We modify and generate texts that become more challenging for the detectors. We develop two methods. In the first method, we prompt an LLM to generate candidate substitutions of words in an LLM-generated text. We then substitute certain words and choose replacements either in a query-free way or through a query-based evolutionary search (Alzantot et al., 2018), in order to bypass the detection. Our second method is for instruction-tuned LLMs such as ChatGPT (OpenAI, 2023b). We search for an instructional prompt on a small subset of training data and fix the prompt at the test time. The prompt instructs the LLM to write in a style such that the generated texts are hard to be detected.

There are concurrent works (Sadasivan et al.,

Work in progress.
Code will be released at https://github.com/shizhouxing/Red-Teaming-LM-Detectors-with-LM.

2023; Krishna et al., 2023) that evade detectors by paraphrasing AI-generated texts. However, they assume that the paraphrasing models are *not protected* by a detector; therefore, it is natural that the paraphrased text can not be recognized by the detector. In contrast, we consider a challenging setting, where we assume that the LLM leveraged for generating attacks is *protected*, meaning it also has a detection mechanism in place. This assumption imposes a realistic constraint on the attacker, as it is possible that all public LLMs are protected in the future. Furthermore, we also consider using the original LLM for text generation itself to jailbreak the detection mechanism, showing that malicious users can bypass the detectors.

We systematically test the three types of detectors, ranging from statistical approaches to commercial APIs. Our results reveal that all the detectors are vulnerable under the proposed attack mechanisms, and the detection performance drops significantly. These findings suggest the current detectors are not reliable and shed light on the discussion about how to build trustworthy detectors. We suggest possible defense strategies in the conclusion section and leave the exploration of defenses to future work.

2 Related Work

Detectors for AI-generated text. Recent detectors for AI-generated text mostly fall into three categories. First, classifier-based detectors are trained with supervised data to distinguish human-written text and AI-generated text. For example, the AI Text Classifier developed by OpenAI (OpenAI, 2023a) is a fine-tuned language model. Second, watermarking methods introduce distinct patterns into AI-generated text, allowing for its identification. Among them, Kirchenbauer et al. (2023) randomly partition the vocabulary into a greenlist and a redlist during the generation, where the division is based on the hash of the previously generated tokens. The language model only uses words in the greenlists, and thereby the generated text has a different pattern compared to human-written text which does not consider such greenlists and redlists. Third, DetectGPT (Mitchell et al., 2023) uses the likelihood of the generated text for the detection, as they find that text generated by language models tends to reside in the negative curvature region of the log probability function. Consequently, they define a curvature-based criterion for the detection.

Methods for red-teaming detectors. As the detectors emerge, there are also several works showing that the detectors may be evaded to some extent, typically by paraphrasing the text (Sadasivan et al., 2023; Krishna et al., 2023). However, they need additional paraphrasing models which are typically unprotected models that are much weaker than the original LLM. Besides paraphrasing, Kirchenbauer et al. (2023) also discussed attacks against watermarking detectors with word substitutions generated by a masked language model such as T5 (Raffel et al., 2020) which is a relatively weaker language model, and thus it may generate attacks with lower quality.

Adversarial Examples in NLP. Red-teaming and attacking detectors for testing their reliability are also relevant to works on adversarial examples in NLP. Word substitution is a commonly used strategy in generating textual adversarial examples (Alzantot et al., 2018; Ren et al., 2019; Jin et al., 2020). Language models such as the BERT (Devlin et al., 2019) have also been used for generating word substitutions (Shi and Huang, 2020; Li et al., 2020; Garg and Ramakrishnan, 2020). In this work, we demonstrate the effectiveness of using the latest LLMs for generating high-quality word substitutions, and our query-based word substitutions are also inspired by the genetic algorithm in (Alzantot et al., 2018; Yin et al., 2020). For our instructional prompt, it is relevant to recent works that prompt LLMs to red team LLMs themselves (Perez et al., 2022) rather than detectors in this work. In addition, we fix a single instructional prompt at test time, which is partly similar to universal triggers in adversarial attacks (Wallace et al., 2019; Behjati et al., 2019), but unlike them constructing an unnatural sequence of tokens as the trigger, our prompt is natural and it is added to the input for the generative model rather than the detector directly.

3 Settings and Overview

We consider a large language model G that conditions on an input context or prompt $\mathbf{X}$ and generates an output text $\mathbf{Y} = G(\mathbf{X})$. In this work, we use upper-case characters such as $\mathbf{X}$ to denote a sequence of tokens $\mathbf{X} = [\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_m]$, where m is the sequence length. The model may be protected by a detector $f(\mathbf{Y}) \in [0, 1]$ that predicts whether

Attack	Perturbation type	Test-time Queries		Applicability		
		G'	f	Classifier	Watermarking	Likelihood
Query-free word substitutions	Output	✓	-	✓	✓	✓
Query-based word substitutions	Output	✓	✓	✓	-	✓
Instructional Prompts	Input	-	-	✓	-	-

Table 1: Properties of various attack methods proposed in this paper and their applicability to various detectors. "Test-time queries" indicates whether each method requires querying G' or f for multiple times at test time.

$\mathbf{Y}$ is generated by the language model G where higher $f(\mathbf{Y})$ score means more likely to be generated by a language model. We use τ to denote a detection threshold such that $\mathbf{Y}$ is considered AI-generated if $f(\mathbf{Y}) \geq \tau$.

In this work, we consider three categories of detectors: (1) classifier-based detectors, (2) watermarking detectors, and (3) likelihood-based detectors. For classifier-based detectors, a text classifier $f(\mathbf{Y})$ is trained on a labeled dataset with G-generated and human-written texts. For watermarking detectors, G is modified from a base generator G_0 with a watermarking mechanism W, denoted as $G = W(G_0)$, and a watermark detector $f(\mathbf{Y})$ is constructed to predict whether $\mathbf{Y}$ is generated by an LLM watermarked with W. Specifically, we consider the watermarking mechanism in Kirchenbauer et al. (2023). For likelihood-based detectors, they estimate the LLM-generated score $f(\mathbf{Y})$ based on the output logits of G. Specifically, we consider DetectGPT (Mitchell et al., 2023). We consider a model G as *protected* if there is a detector $f(\mathbf{Y})$ in place to protect the model from inappropriate usage.

To stress test the reliability of those detectors in this setting, we develop red-teaming techniques to generate texts that can bypass a detector using an LLM that is also protected by this detector. This differs from previous methods that requires a separate unprotected paraphrasing model (Sadasivan et al., 2023; Krishna et al., 2023). We use G' to denote the protected LLM used for generating attacks, where G' may also be the same as G if applicable, and we consider the attack from two aspects:

- **Output perturbation** that directly perturbs the original output $\mathbf{Y}$ and generates a perturbed output $\mathbf{Y}'$.
- **Input perturbation** that perturbs the input $\mathbf{X}$ into $\mathbf{X}'$ as the new input, leading to a new output $\mathbf{Y}' = G(\mathbf{X}')$.

In both cases, we aim to minimize $f(\mathbf{Y}')$ so that the new output $\mathbf{Y}'$ is wrongly considered as human-written by the detector f. Meanwhile, we require that $\mathbf{Y}'$ has a quality similar to $\mathbf{Y}$ and remains a plausible output to the original input $\mathbf{X}$. For our attack algorithms, we also assume that the detector f is black-box, and only the output scores are visible, but not internal parameters.

We propose to attack the detectors in two different ways. In Section 4, we construct an output perturbation by replacing some words in $\mathbf{Y}$, where we prompt a protected LLM G' to obtain the new substitution words, and we then build query-based and query-free attacks respectively with these word substitutions. In Section 5, if G is able to follow instructions, we search for an instructional prompt from the generation by G and append the prompt to $\mathbf{X}$ as an input perturbation, where the instructional prompt instructs G to generate texts in a style making it hard for the detector to detect.

Table 1 summarizes our methods and their applicability to different detectors. At test time, instructional prompts are fixed and thus totally query-free. For word substitutions, they require querying G' multiple times to generate word substitutions on each test example; the query-free version does not repeatedly query f while the query-based version also requires querying f multiple times. In practice, we may choose between these methods depending on the query budget and their applicability to the detectors.

4 Attack with Word Substitutions

To attack the detectors with output perturbations, we aim to find a perturbed output $\mathbf{Y}'$ that is out of the original detectable distribution. This is achieved by substituting certain words in $\mathbf{Y}$. To obtain suitable substitution words for the tokens in $\mathbf{Y}'$ that preserve the naturalness and semantic meaning, we utilize a protected LLM denoted as G'. For each token in $\mathbf{Y}$ denoted as $\mathbf{y}_k$, we use $s(\mathbf{y}_k, \mathbf{Y}, G', n)$ to denote the process of generating at most n word substitution candidates for $\mathbf{y}_k$ given the context in $\mathbf{Y}$ by prompting G', and $s(\mathbf{y}_k, \mathbf{Y}, G', n)$ outputs a set of at most n words. Note that not every word can be substituted, and $s(\mathbf{y}_k, \mathbf{Y}, G', n)$ can be an empty set if it is not suitable to replace $\mathbf{k}$. We will discuss how to generate the word substitution candidates using G' in Section 4.1.

General attack objective. The objective of attacking f with word substitutions can be formulated as a minimization problem given a substitution budget ϵ:

$$\begin{aligned} & \mathbf{Y}' = \arg\min_{\mathbf{Y}'} f(\mathbf{Y}'), \qquad (1) \\ \text{s.t.} \quad & \mathbf{y}'_k \in \{\mathbf{y}_k\} \cup s(\mathbf{y}_k, \mathbf{Y}, G', n), \\ & \sum_{k=1}^{m} \mathbb{1}(\mathbf{y}_k \neq \mathbf{y}'_k) \leq \epsilon m. \end{aligned}$$

Here we aim to find an optimally perturbed output $\mathbf{Y}'$ that minimizes the predicted score $f(\mathbf{Y}')$ among all possible $\mathbf{Y}'$. Each word in the perturbed output $\mathbf{y}'_k$ is either the unperturbed word $\mathbf{y}_k$ or selected from the word substitution candidates $s(\mathbf{y}_k, \mathbf{Y}, G', n)$, and the total number of perturbed words is at most ϵm. To solve the minimization problem in Eq. (1), we consider both query-free and query-based substitutions respectively. For query-based substitutions, we use the evolutionary search algorithm (Alzantot et al., 2018; Yin et al., 2020) originally for generating adversarial examples in NLP, with details in Appendix A. And we also design query-free substitution methods in Section 4.2. We may choose between query-based substitutions and query-free substitutions depending on whether we may query f for multiple times.

4.1 Generating Word Substitution Candidates

Table 2 shows the prompts we use when G' is ChatGPT and LLaMA respectively and outputs by the LLMs. ChatGPT is able to follow instructions, and thus our prompt is an instruction asking the model to generate substitution words, and multiple words can be substituted simultaneously. For LLaMA which cannot directly follow instructions, we expect it to generate a text completion following our prompt, where the prompt is designed such that a plausible text completion consists of suggested substitution words, and we replace one word at each time.

ChatGPT
Given this sentence: "*The scientists said the findings could help improve treatments and lead to new ways of preventing breast cancer spreading. Experts have described the findings as "exciting".* **", for each word in "***findings, exciting***", give 10 substitution words that do not change the meaning of the sentence. Return each word and its substitutions in one line, in the format of "word:substitutions"**
findings: results, discoveries, outcomes, conclusions, observations, data, evidence, findings, research, studies exciting: thrilling, exhilarating, stimulating, electrifying, gripping, captivating, invigorating, inspiring, breathtaking, exciting
LLaMA
"*The scientists said the findings could help improve treatments and lead to new ways of preventing breast cancer spreading. Experts have described the findings as "exciting".*" **The word** *"exciting"* **in the above sentence is equivalent to:** **a)** "interesting" b) "surprising" c) "unusual"

Table 2: Prompts for generating word substitution candidates using ChatGPT and LLaMA respectively and the outputs by the LLMs. Text in bold denotes templates in the prompts. Text in italic denotes a text to be perturbed or words to be replaced for a given example. Text in blue denotes the generated word substitutions.

4.2 Query-free Substitutions

For the query-free attack, we apply word substitution on random tokens in $\mathbf{Y}$ to attack DetectGPT and classifier-based detectors. For watermarking detectors, we further design an effective query-free attack utilizing the properties of the detection method.

Specifically, we consider the watermarking mechanism introduced in Kirchenbauer et al. (2023). A watermarked LLM generates a token with modified predicted logits at position $i+1$: $g(\mathbf{y}_{i+1}|[\mathbf{y}_1, ..., \mathbf{y}_i]) = g_0(\mathbf{y}_{i+1}|[\mathbf{y}_1, ..., \mathbf{y}_i]) + \delta$ if the candidate token $\mathbf{y}_{i+1}$ is in the greenlist decided by a hash function $h([\mathbf{y}_1, ..., \mathbf{y}_i])$. Here we use the lower-case g_0 to denote the logits outputs of a generative model G_0 and g for the watermarked version G. δ is an offset value pre-defined by the watermarking. Therefore, a text generated by a watermarked LLM tends to have more greenlist tokens and $f(\mathbf{Y})$ calculates the score with the count of greenlist tokens in $\mathbf{Y}$.

Therefore, given a fixed substitution budget ϵ, we aim to identify and substitute more greenlist tokens to reduce the total count of greenlist tokens. We achieve this with a two-stage algorithm. At the first stage, we sort all tokens in $\mathbf{Y}$ by the prediction entropy estimated by a language model M, which can be a weaker model than G as we only use the entropy as a heuristic score. The prediction entropy is calculated with the output probability among all the possible vocabulary. As the watermarking offset δ is applied on the decoding process, a token with higher entropy is easier to be affected by watermarking. At the second stage, we pick ϵm tokens with highest entropy and use a watermarked LLM G' to generate word substitutions as introduced in Section 4.1.

5 Attack by Instructional Prompts

In this section, we build attacks by perturbing the input prompt to encourage LLMs to generate texts that is difficult to be detected. In particular, we focus on LLM-based generative models that can follow instructions and classifier-based detectors. We consider ChatGPT (OpenAI, 2023b) as the generative model G and OpenAI AI Text Classifier (OpenAI, 2023a) as the detector f. The OpenAI AI Text Classifier is a fine-tuned neural network, while neural networks have been shown to be vulnerable to distribution shifts in NLP literature (Miller et al., 2020; Awadalla et al., 2022). Therefore, we aim to shift the generated text to a different distribution where the detector is more likely to fail, while making the generated text still a plausible output to the input. We achieve this by searching for an additional prompt $\mathbf{X}_p$ appended to the original input $\mathbf{X}$, which forms a new input $\mathbf{X}' = [\mathbf{X}, \mathbf{X}_p]$ to G. In particular, $\mathbf{X}_p$ consists of $\mathbf{X}_{\text{ins}}$ and $\mathbf{X}_{\text{ref}}$, where $\mathbf{X}_{\text{ins}}$ is an instruction asking the model to follow the writing style of reference $\mathbf{X}_{\text{ref}}$.

Searching for $\mathbf{X}_p$. We search for $\mathbf{X}_p$ on a small subset of training examples with n examples $\mathbf{X}_1, \mathbf{X}_2, \cdots, \mathbf{X}_n$. We assume that we can query the detector f for multiple times during search time. After an effective $\mathbf{X}_p$ is found, it can be applied universally on all inputs from this dataset at test time. The objective of the search is:

$$\arg\min_{\mathbf{X}_p} \frac{1}{n} \sum_{i=1}^{n} \mathbb{1}(f(G([\mathbf{X}_i, \mathbf{X}_p])) \geq \tau), \qquad (2)$$

which aims to minimize the average detection rate for the new outputs generated with $\mathbf{X}_p$ appended to the input.

Algorithm 1 Search for the attacking prompt

```
input Training data $\mathbf{X}_1, \cdots, \mathbf{X}_n$; generative model $G$ and detector $f$; initial instruction $\mathbf{X}_{\text{ins},0}$.
output An attacking instructional prompt $\mathbf{X}_p$.
  $\mathbf{X}_{\text{ins}} \leftarrow \mathbf{X}_{\text{ins, 0}}$
  $\mathcal{O} \leftarrow$ PriorityQueue()
  GenerateAndDetect("", "")
  for $t = 1, \cdots, T$ do
    $\mathbf{X}_{\text{ref}} \leftarrow$ SearchForReference($\mathbf{X}_{\text{ins}}$)
    $\mathbf{X}_{\text{ins}} \leftarrow$ SearchForInstruction($\mathbf{X}_{\text{ins}}, \mathbf{X}_{\text{ref}}$)
  $\mathbf{X}_p \leftarrow [\mathbf{X}_{\text{ins}}, \mathbf{X}_{\text{ref}}]$
  return $\mathbf{X}_p$
  function GenerateAndDetect($\mathbf{X}_{\text{ins}}, \mathbf{X}_{\text{ref}}$)
    for $i = 1, \cdots, n$ do
      $\mathbf{Y}_i \leftarrow G([\mathbf{X}_i, \mathbf{X}_{\text{ins}}, \mathbf{X}_{\text{ref}}])$
      $\mathcal{O}$.push(key $= f(\mathbf{Y}_i)$, value $= \mathbf{Y}_i$)
    return Detection rate and average score
  function SearchForReference($\mathbf{X}_{\text{ins}}$)
    for $i = 1, \cdots, K$ do
      $\mathbf{C}_{\text{ref}}^{(i)} \leftarrow \mathcal{O}$.pop()
    $Best \leftarrow \arg\min\{$
        GenerateAndDetect($\mathbf{X}_{\text{ins}}, \mathbf{C}_{\text{ref}}^{(i)}$)$\}_{i=1}^{K}$
    return $\mathbf{C}_{\text{ref}}^{(Best)}$
  function SearchForInstruction($\mathbf{X}_{\text{ins}}, \mathbf{X}_{\text{ref}}$)
    for $i = 1, \cdots, K$ do
      $\mathbf{C}_{\text{ins}}^{(i)} \leftarrow$ Paraphrase($\mathbf{X}_{\text{ins}}$)
    $Best \leftarrow \arg\min\{$
        GenerateAndDetect($\mathbf{C}_{\text{ins}}^{(i)}, \mathbf{X}_{ref}$)$\}_{i=1}^{K}$
    return $\mathbf{C}_{\text{ins}}^{(Best)}$
```

We illustrate our algorithm searching for $\mathbf{X}_p$ in Algorithm 1. We use G to generate various $\mathbf{X}_{\text{ins}}$ and $\mathbf{X}_{\text{ref}}$ in each iteration and try to search for an optimal $\mathbf{X}_p = [\mathbf{X}_{\text{ins}}, \mathbf{X}_{\text{ref}}]$ following the objective in Eq. (1). Initially, we set $\mathbf{X}_{\text{ins}}$ as a manually written instruction, "Meanwhile please imitate the writing style and wording of the following passage:". An initial value for $\mathbf{X}_{\text{ref}}$ is not necessary. We also create and initialize a priority queue $\mathcal{O}$ with n initial outputs generated from the n training examples without $\mathbf{X}_p$. $\mathcal{O}$ sorts its elements according to the detection scores from f and prioritize those with lower scores. In each iteration of the search, we have two steps:

- Updating $\mathbf{X}_{\text{ref}}$: We pop the top-K candidates from $\mathcal{O}$. For each candidate, we combine it with the current $\mathbf{X}_{\text{ins}}$ respectively as the potential candidates for $\mathbf{X}_p$ in the current iteration.
- Updating $\mathbf{X}_{\text{ins}}$: We instruct model G to generate K variations of the current $\mathbf{X}_{\text{ins}}$, inspired by Zhou et al. (2022) for automatic prompt engineering. And we combine them with the current $\mathbf{X}_{\text{ins}}$ respectively as the potential candidates for $\mathbf{X}_p$.

For both of these two steps, we take the best candidate $\mathbf{X}_p$ according to Eq. (2). When generating $G([\mathbf{X}_i, \mathbf{X}_p])$ in Eq. (2), we push all the generated outputs to $\mathcal{O}$ as the candidates for $\mathbf{X}_{\text{ref}}$ in the later rounds. We take T iterations and return the final $\mathbf{X}_p = [\mathbf{X}_{\text{ins}}, \mathbf{X}_{\text{ref}}]$ to be used at test time.

6 Experiments

6.1 Experimental Settings

Generative Models and Detectors. We consider a wide range of LLM-based generative models with detectors protecting the models. For the generative model G, we consider GPT-2-XL (Radford et al., 2019), LLaMA-65B (Touvron et al., 2023), and ChatGPT (OpenAI, 2023b). For the detectors, watermarking and DetectGPT are applied to both GPT-2-XL and LLaMA-65B but not ChatGPT; classifier-based detectors include a fine-tuned RoBERTa-Large detector (Solaiman et al., 2019) for GPT-2 texts; and the OpenAI AI Text Classifier (OpenAI, 2023a) for ChatGPT texts. We use either LLaMA-65B or ChatGPT as G' for generating perturbations in the attack. When G is LLaMA-65B or ChatGPT, we simply use itself as $G' = G$. And when G is GPT-2-XL, we use ChatGPT as G' when the classifier-based detector or DetectGPT is used, and we use LLaMA-65B as G' when watermarking are used, as we add watermarking to ChatGPT which is not open-source. Table 3 summarizes the protected LLM G' used for all the generative models and the detectors in the experiment.

Datasets We use two types of datasets in our experiments including text completion and instructional datasets. For text completion datasets, we use XSum (Narayan et al., 2018) and WikiText (Merity et al., 2016). We take the first sentence for XSum and the first 20 tokens for WikiText as the input prompt to the generative models. And we also use an instructional dataset, ELI5 (Fan et al., 2019), which is a long-form question-answering dataset collected from Reddit. To test the RoBERTa-Large detector specifically for detecting GPT-2 texts, we also adopt the GPT-2 output dataset (Solaiman et al., 2019). Since the OpenAI AI Text Classifier requires the text to contain at least 1000 characters, we filter all the datasets and only retain examples with human reference containing at least 1000 characters. We use the first 100 examples in the shuffled test set for each dataset.

Metrics We use several metrics for the detectors under attacks. Area Under the Receiver Operating Characteristic Curve (AUROC) scores summarize the performance of detectors under various thresholds. A detection rate (DR) is the true positive rates under a fixed threshold (positive examples mean LLM-generated texts), where we either tune the threshold to meet a particular false positive rate or follow the original thresholds of the detectors. For query-based word substitutions, we also use Attack Success Rate (ASR) which computes the rate that the attack successfully flips the prediction by the detector, out of all the postive examples on which the detector originally predicts correctly.

6.2 Attack with Word Substitutions

We apply word substitution-based attack on all the three detection methods including DetectGPT, classifier-based detectors, and watermarking. In each setting, we assume that both G and G' are protected by the same detector f.

Attack against DetectGPT For experiments on attacking DetectGPT, we follow Mitchell et al. (2023) to prompt GPT-2-XL with the first 30 tokens from the samples and ask LLMs to generate the rest. We set the maximum changes to be 10% of the sequence except for stop words, which leads to around 10 substituted tokens. For evolutionary searching, this requires 100 queries per instance with population size of 10. DetectGPT uses an external T5-3B model to do mask infilling that generates the perturbations and we fix the mask rate to be 15%. We adopt a more realistic cross-model setting, where we use GPT-

Generative Model	Classifier-based Detector	Watermarking	DetectGPT
GPT-2-XL	ChatGPT	LLaMA-65B	ChatGPT
LLaMA-65B	-	LLaMA-65B	LLaMA-65B
ChatGPT	ChatGPT	-	-

Table 3: The protected LLM G' used in generating perturbations for each generative model G and the detectors. "-" indicates a combination of the generative model and the detector is not applicable.

Attack	AUROC	
	XSum	Wiki
Unattacked	92.5	69.4
Paraphrasing	45.7	37.5
Query-free Substitutions	25.9	31.2
Query-based Substitutions	**3.9**	**6.1**

Table 4: AUROC (%) for DetectGPT (Mitchell et al., 2023). We compare DetectGPT before and after various attacks under the cross-model setting where the base model is GPT-2-XL and the detection model that estimates the likelihood is GPT-Neo.

Neo (Black et al., 2021) as the detection model to estimate the log-likelihood.

The results are shown in Table 4. On XSum and WikiText, DetectGPT's AUROC drops below random guessing to 25.9% and 31.2% respectively, after query-free substitutions which randomly select substitutions from the candidate pool. The AUROC scores further drop to only 3.9% and 6.1% respectively after the query-based evolutionary search. Note that both of the methods change around 10 tokens in a 1000 character paragraph. This demonstrates that using the likelihood of machine-generated texts may not be robust against malicious word changes.

Attack against Classifier-based Detectors We experiment with the the two public classifier-based detectors: a RoBERTa-large model fine-tuned for detecting GPT-2 texts (Mitchell et al., 2023), and the OpenAI AI text detector (OpenAI, 2023b). For all the experiments, we keep the maximum number of substitutions to be 20% of the total lengths of paragraphs except for stop words.

Results for attacking GPT-2 text detector are shown in Table 5. We find that the attack success rate (ASR) on detecting GPT-2 texts is close to 0 for both paraphrasing and query-free substitutions. We hypothesize that this is because the detector is specifically trained on detecting GPT-2 texts, it is hard to remove the patterns leveraged by those detectors by randomly selecting word substitutions or paraphrasing without querying the detector. Our evolutionary searching-based substitutions achieve much better ASR compared to the query-free methods.

For the OpenAI AI Text Classifier shown in Table 7, query-free attacks are able to decrease the detection AUROC by 18.9 and 28.1 points on XSum and ELI5, respectively, while query-based ones further decrease them by 45.4 and 55.6 points to lower than random. Comparison with the attack with instructional prompts and more details are discussed in Section 6.3.

Attack	ASR
Paraphrasing	4%
Query-free Substitutions	2%
Query-based Substitutions	**68%**

Table 5: Attack Success Rate (ASR) for OpenAI RoBERTa-Large detector for GPT-2 texts (Mitchell et al., 2023).

Attack against Watermarking We implement watermarking mechanism introduced in (Kirchenbauer et al., 2023) on two language models including GPT-2-XL and LLaMA-65B. We use a T5-Large model to estimate the prediction entropy for each token and select 20% of tokens in the initial output $\mathbf{Y}$. We use a watermarked LLaMA to suggest word substitutions with the prompt introduced in Table 2. We use $\delta = 1.0$ and $\gamma = 0.5$ in watermarking implementation. We only keep the word substitutions with fewer than 4 tokens to avoid invalid substitutions. We report AUROC and detection rates in Table 6. For detection rates, we set the threshold value to keep the false positive rate for human texts equal to 1%. As we can observe from the table, detection rate can be significantly decreased after query-free word substitution attack. We show an example for query-free attack against watermarking detectors in Table 9.

Generative Model	Dataset	Unattacked		Query-free Substitution	
		AUROC	DR	AUROC	DR
GPT-2-XL	XSum	97.9	81.0	87.7	36.0
	WikiText	97.4	81.0	89.7	54.0
LLaMA-65B	XSum	88.9	22.0	70.2	9.0
	WikiText	92.6	73.2	81.3	43.3

Table 6: Attack against watermarking detector. We report both AUROC scores (%) and the detection rates (DR) (%) under the threshold value when false positive rate for human texts is 1%.

6.3 Attack with Instructional Prompts

Method	XSum		ELI5	
	AUROC	DR	AUROC	DR
Unattacked	88.8	30.0	87.1	54.0
ChatGPT Paraphrasing	82.4	12.0	73.6	27.0
Query-free Substitution	69.9	2.0	59.0	2.0
Query-based Substitution	43.4	**0.0**	**31.5**	**0.0**
Instructional Prompts	**11.2**	**0.0**	43.6	18.0

Table 7: AUROC scores (%) and detection rates (DR) (%) of the OpenAI AI Text Classifier on the original outputs by ChatGPT and outputs with various attacks respectively.

We conduct experiments for our attack using instructional prompts on ChatGPT as the generative model with the OpenAI AI Text Classifier as the detector. The detector is `model-detect-v2` accessible in OpenAI APIs. Its output contains five classes, including "likely", "possibly", "unclear if it is", "unlikely" and "very unlikely", with thresholds 0.98, 0.90, 0.45, and 0.10 respectively. We follow these thresholds and use a threshold of 0.9 to determine detection rates. In Algorithm 1, we search for the instructional prompt using $n = 50$ training examples, $T = 5$ iterations, and $K = 5$ candidates for references and instructions respectively in each iteration.

Table 7 shows the results. We also show our prompts and an example on ELI5 with various attacks in Appendix B. Our instructional prompts significantly reduce the the AUROC scores and detection rates compared to the unattacked version. We also find that paraphrasing with ChatGPT itself can also somewhat downgrade the detection but it is much less effective than our instructional prompts. Compared to word substitutions, the attack with instructional prompts achieves lower AUROC and 0 detection rate on XSum; on ELI5, the attack with instructional prompts has lower AUROC score than query-free word substitution but not query-based word substitution, and it has higher detection rates. Nevertheless, unlike word substitutions, the attack with instructional prompts does not query G' or f for multiple times, and thus it is more efficient while also effective.

7 Conclusion and Discussion

In this work, we studied the reliability of representative AI text detectors from three different categories: classifier-based, likelihood-based, and watermarking. We proposed two methods to prompt LLMs to modify texts and make them harder to be detected. The limitations revealed in our experiments urge the design of a more reliable detection mechanism.

As an initial discussion around the defense against the word substitution attacks proposed in the paper, we discuss a few possible defense strategies. First, by fine-tuning a more specific classifier-based detector based on the target model. As we show in the experiments, if a classifier-based detector is specifically fine-tuned for detecting a target model (RoBERTa-Large for GPT-2-XL in this paper), it would be much more robust against query-free modifications. Second, by combining lexical watermarking with text likelihood estimation. This is based on the intuition that if a word substitution attack successfully evades a watermarking detector, it would need to change around 20% tokens from the greenlist tokens to redlist tokens that might not be of large probability. Thus, one may apply a watermarked LLM (Kirchenbauer et al., 2023) to a suspected text and then check the perplexity or the likelihood of all redlist tokens to detect machine-generated texts. More work on defense strategies will be deferred to future endeavors.

Acknowledgement

We thank UCLA-NLP for their invaluable feedback. The work is supported in part by CISCO.

References

Moustafa Alzantot, Yash Sharma, Ahmed Elgohary, Bo-Jhang Ho, Mani Srivastava, and Kai-Wei Chang. 2018. Generating natural language adversarial examples. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 2890–2896, Brussels, Belgium.

Anas Awadalla, Mitchell Wortsman, Gabriel Ilharco, Sewon Min, Ian Magnusson, Hannaneh Hajishirzi, and Ludwig Schmidt. 2022. Exploring the landscape of distributional robustness for question answering models. *arXiv preprint arXiv:2210.12517*.

Melika Behjati, Seyed-Mohsen Moosavi-Dezfooli, Mahdieh Soleymani Baghshah, and Pascal Frossard. 2019. Universal adversarial attacks on text classifiers. In *ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 7345–7349. IEEE.

Sid Black, Leo Gao, Phil Wang, Connor Leahy, and Stella Biderman. 2021. GPT-Neo: Large Scale Autoregressive Language Modeling with Mesh-Tensorflow. If you use this software, please cite it using these metadata.

Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, et al. 2022. Palm: Scaling language modeling with pathways. *arXiv preprint arXiv:2204.02311*.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota.

Angela Fan, Yacine Jernite, Ethan Perez, David Grangier, Jason Weston, and Michael Auli. 2019. ELI5: long form question answering. In *Proceedings of the 57th Conference of the Association for Computational Linguistics, ACL 2019, Florence, Italy, July 28- August 2, 2019, Volume 1: Long Papers*, pages 3558–3567. Association for Computational Linguistics.

Siddhant Garg and Goutham Ramakrishnan. 2020. Bae: Bert-based adversarial examples for text classification. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 6174–6181.

Mohit Iyyer, John Wieting, Kevin Gimpel, and Luke Zettlemoyer. 2018. Adversarial example generation with syntactically controlled paraphrase networks. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 1875–1885.

Di Jin, Zhijing Jin, Joey Tianyi Zhou, and Peter Szolovits. 2020. Is bert really robust? a strong baseline for natural language attack on text classification and entailment. In *Proceedings of the AAAI conference on artificial intelligence*, volume 34, pages 8018–8025.

John Kirchenbauer, Jonas Geiping, Yuxin Wen, Jonathan Katz, Ian Miers, and Tom Goldstein. 2023. A watermark for large language models. *arXiv preprint arXiv:2301.10226*.

Kalpesh Krishna, Yixiao Song, Marzena Karpinska, John Wieting, and Mohit Iyyer. 2023. Paraphrasing evades detectors of ai-generated text, but retrieval is an effective defense. *arXiv preprint arXiv:2303.13408*.

Linyang Li, Ruotian Ma, Qipeng Guo, Xiangyang Xue, and Xipeng Qiu. 2020. Bert-attack: Adversarial attack against bert using bert. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 6193–6202.

Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. 2016. Pointer sentinel mixture models.

John Miller, Karl Krauth, Benjamin Recht, and Ludwig Schmidt. 2020. The effect of natural distribution shift on question answering models. In *International Conference on Machine Learning*, pages 6905–6916. PMLR.

Eric Mitchell, Yoonho Lee, Alexander Khazatsky, Christopher D. Manning, and Chelsea Finn. 2023. Detectgpt: Zero-shot machine-generated text detection using probability curvature.

Shashi Narayan, Shay B. Cohen, and Mirella Lapata. 2018. Don't give me the details, just the summary! topic-aware convolutional neural networks for extreme summarization. *ArXiv*, abs/1808.08745.

OpenAI. 2023a. Ai text classifier.

OpenAI. 2023b. Chatgpt. `https://openai.com/blog/chatgpt/`. Accessed on May 3, 2023.

Ethan Perez, Saffron Huang, Francis Song, Trevor Cai, Roman Ring, John Aslanides, Amelia Glaese, Nat McAleese, and Geoffrey Irving. 2022. Red teaming language models with language models. *arXiv preprint arXiv:2202.03286*.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. 2019. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. 2020. Exploring the limits of transfer learning with a unified text-to-text transformer. *The Journal of Machine Learning Research*, 21(1):5485–5551.

Shuhuai Ren, Yihe Deng, Kun He, and Wanxiang Che. 2019. Generating natural language adversarial examples through probability weighted word saliency. In *Proceedings of the 57th annual meeting of the association for computational linguistics*, pages 1085–1097.

Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. 2018. Semantically equivalent adversarial rules for debugging nlp models. In *Proceedings of the 56th annual meeting of the association for computational linguistics (volume 1: long papers)*, pages 856–865.

Vinu Sankar Sadasivan, Aounon Kumar, Sriram Balasubramanian, Wenxiao Wang, and Soheil Feizi. 2023. Can ai-generated text be reliably detected? *arXiv preprint arXiv:2303.11156*.

Zhouxing Shi and Minlie Huang. 2020. Robustness to modification with shared words in paraphrase identification. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 164–171, Online.

Irene Solaiman, Miles Brundage, Jack Clark, Amanda Askell, Ariel Herbert-Voss, Jeff Wu, Alec Radford, Gretchen Krueger, Jong Wook Kim, Sarah Kreps, et al. 2019. Release strategies and the social impacts of language models. *arXiv preprint arXiv:1908.09203*.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurelien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. 2023. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*.

Eric Wallace, Shi Feng, Nikhil Kandpal, Matt Gardner, and Sameer Singh. 2019. Universal adversarial triggers for attacking and analyzing nlp. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 2153–2162.

Fan Yin, Quanyu Long, Tao Meng, and Kai-Wei Chang. 2020. On the robustness of language encoders against grammatical errors. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 3386–3403, Online. Association for Computational Linguistics.

Rowan Zellers, Ari Holtzman, Hannah Rashkin, Yonatan Bisk, Ali Farhadi, Franziska Roesner, and Yejin Choi. 2019. Defending against neural fake news. *Advances in neural information processing systems*, 32.

Yongchao Zhou, Andrei Ioan Muresanu, Ziwen Han, Keiran Paster, Silviu Pitis, Harris Chan, and Jimmy Ba. 2022. Large language models are human-level prompt engineers. *arXiv preprint arXiv:2211.01910*.

A Query-based Evolutionary Search

We attach the algorithm for the evolutionary search in Algorithm 2.

Algorithm 2 Genetic attack (Alzantot et al., 2018)

```
input Original generated text $\mathbf{Y} = [\mathbf{y}_1 \cdots \mathbf{y}_m]$; substitution word candidates for each token $\mathbf{y}_k$ ($k \in [m]$) as $s(\mathbf{y}_k, \mathbf{Y}, G', n)$; detector $f$; parameters for the genetic search, including population size $N_p$ and total number of generations $N_G$.
output $\mathbf{Y}'$ that minimizes $f(\mathbf{Y}')$.
$P^{(0)} \leftarrow$ PriorityQueue$()$.
for $i = 1, 2, \cdots, N_p$ do
    $k \leftarrow$ Random$(1, 2, \cdots, m)$
    $\mathbf{y}'_k \leftarrow$ Random$(s(\mathbf{y}_k, \mathbf{Y}, G', n))$
    $\mathbf{Y}'_k \leftarrow [\mathbf{y}_1, \cdots, \mathbf{y}'_k, \cdots, \mathbf{y}_m]$
    $P^{(0)}$.push(key$=f(\mathbf{Y}'_k)$, value$=\mathbf{Y}'_k$)
for $g = 1, 2, \cdots, N_G - 1$ do
    $P^{(g)} \leftarrow$ PriorityQueue$()$.
    for $i = 1, 2, \cdots, N_p$ do
        $\mathbf{Y}^{(g-1)}_{\text{elite}} \leftarrow P^{(g-1)}$.pop(),
        $P^{(g)}$.push(key$=f(\mathbf{Y}^{(g-1)}_{\text{elite}})$, value$=\mathbf{Y}^{(g-1)}_{elite}$)
        $prob \leftarrow$ Normalized sample probability with $f(P^{(g-1)}_i)$
        for $j = 2, 3, ..., p$ do
            $c_1 \leftarrow$ sample from $P^{(g-1)}$ with $prob$
            $c_2 \leftarrow$ sample from $P^{(g-1)}$ with $prob$
            $c \leftarrow$ CrossOver($c_1$, $c_2$) {For each position, randomly select the token from $c_1$ or $c_2$.}
            $c_{mut} \leftarrow$ Mutate($c$) {Randomly select a position and a candidate to modify $c$.}
            $P^{(g)}$.push(key$=f(c_{mut})$, value$=c_{mut}$)
return $P^{(N_G)}$.pop()
```

B Details on ChatGPT Experiments

In Table 8, we show an example on ELI5 including the original input and output, as well as attacked texts by various methods. In Table 10, we show prompts for querying ChatGPT before using on attack with instructional prompts, and we show our searched instructional prompts in Table 11.

C Attack Examples against Watermarking

We also show some qualitative examples for attacks against watermarking in Table 9.

Original input
Please answer this question with at least 150 words: Why we don't yet have a cable provider that will let me pick and pay for ONLY the channels I want?
Original output
The reason why we do not yet have a cable provider that will allow choosing and paying for only the channels we want is that it may not be economically feasible for the providers. The cable providers typically negotiate content deals with media companies in packages or bundles. This packaging provides providers with a lower cost per channel, making it more profitable for them. The media companies also benefit from bundling their channels, as they can sell lower-demand channels as a package deal, ensuring broader visibility for their programs. In addition, some content providers may be hesitant to unbundle their channels, as they may be in lower demand. In contrast, other media companies may have popular channels that are not profitable if unbundled and offered on an individual basis. As a result, providers may lose money if they forced to offer channels individually, or the subscription fees may be too high, many cannot afford it. Although some content providers are trying out new delivery models, such as streaming, it may take time for the cable providers to adjust their policies to fit with the changing times.
ChatGPT Paraphrasing
It is currently not possible for consumers to choose and pay only for the channels they want from cable providers due to the economic feasibility for the providers. Typically, cable providers negotiate content deals with media companies in packages or bundles, which allows them to obtain a lower cost per channel and be more profitable. Bundling is also beneficial for media companies as they can sell lower-demand channels as a package deal, ensuring broader visibility for their programs. However, content providers may not be keen on unbundling their channels, primarily if they are in low demand, while other companies may have popular channels that are not financially viable if offered individually. Forcing providers to offer channels individually could result in them losing money, or the subscription fees may be unaffordable for consumers. Although some content providers are testing new delivery models such as streaming, cable providers may take time to adapt their policies to fit with the changing times.
Query-free Substitution
The **motive** why we do not yet have a cable provider that will allow choosing and paying for only the channels we want is that it may not be **monetarily** feasible for the providers. The cable providers typically negotiate content deals with media companies in **wrap** or bundles. This packaging provides providers with a lower **expense** per channel, making it more profitable for them. The media companies also benefit from **integrating** their channels, as they can sell lower-demand channels as a package deal, ensuring broader visibility for their programs. In addition, some content **suppliers** may be hesitant to unbundle their channels, as they may be in lower demand. In contrast, other media companies may have popular channels that are not profitable if unbundled and offered on an individual **underpinning**. As a result, providers may lose **finances** if they forced to offer channels **distinctly**, or the subscription fees may be too high, many can not afford it. Although some content providers are trying out new **transporting** models, such as streaming, it may take time for the cable providers to adjust their policies to fit with the **progressing** times.
Query-based Substitution
The reason why we do not yet have a cable provider that will allow **nominating** and paying for only the channels we want is that it may not be economically feasible for the providers. The **Cablegram** providers typically negotiate content deals with media companies in packages or bundles. This **parcel** provides providers with a lower cost per channel, making it more profitable for them. The media companies also benefit from packaging their channels, as they can sell lower-demand **avenue** as a package deal, ensuring broader visibility for their programs. In addition, some content providers may possibly be hesitant to unbundle their **transmission**, as they may be in **subordinate** demand. In contrast, other **press** companies may have popular channels that are not profitable if unbundled and offered on an individual basis. As a result, providers may **miss** money if they forced to offer **transmission autonomously**, or the subscription fees may be too high, many can not afford it. Albeit some content **contributors** are trying out new **handover** models, such as streaming, it may possibly take time for the cable providers to adjust their **approaches** to **adapt** with the textbfconverting times.
Instructional Prompts
Many cable providers have not yet implemented a system where customers can select and pay only for the channels they want. This has been a long-standing concern among cable subscribers who are paying for channels they don't watch. The reasons for this are complex and varied, but ultimately, it boils down to economic and legal issues. Cable providers operate on a business model that relies on bundling channels and packages in order to maximize profits by leveraging the content deals they have with media companies. It is much cheaper for cable companies to buy packages of channels than to buy individual channels. This means that any loss of profitability from individual customer choices is not something that many of them are willing to take on. Furthermore, cable companies are subjected to strict contracts and licensing agreements with media companies, and changing these agreements can be complicated, time-consuming, and costly. Cable companies are also subjected to stringent regulations from the Federal Communications Commission, which adds another layer of complexity to their operations. Overall, while the demand for a la carte cable has been growing, it remains to be seen if cable providers will take the steps necessary to make this a reality.

Table 8: An example from the ELI5 dataset. We show the original input and output, as well as the output after using various attacks.

Original output
The scientists said the findings could help improve treatments and lead to new ways of preventing breast cancer spreading. Experts have described the findings as “exciting”. Bone is the most common site for breast cancer to spread to. Once breast cancer reaches the bone, it can be treated but often is not curable. In experiments in mice, the Sheffield researchers found breast cancer cells were sending signals to the cells inside
Query-free Substitution
The **researchers** said the findings could help improve **drugs** and lead to new ways of **stopping** breast cancer **metastasizing**. Experts have **said** the discoveries as “**interesting**”. Bone is the most common **place** for **bone** cancer to **metastasize** to. Once **bone** cancer **spreads to** the bone, it can be treated but usually is not **can be treated**. In **research** in **the Sheffield researchers**, the Sheffield **Scientists** found **cancer** cancer cells were sending signals to the cells **in**.

Table 9: An example from the XSum dataset. We show the original output from watermarked LLaMA-65B, as well as the output after query-free word substitution attack.

Initial prompt for XSum	Please complete this passage with at least 150 words: $\{\mathbf{X}\}$
Initial prompt for ELI5	Please answer this question with at least 150 words: $\{\mathbf{X}\}$
Prompt for paraphrasing	Please paraphrase the following passage, with at least 200 words: $\{\mathbf{Y}\}$

Table 10: Prompts used for querying ChatGPT. Initial prompts are used for instructing ChatGPT to perform text completion or question answering on XSum and ELI5 respectively, and we also instruct ChatGPT to generate at least 150 words as the OpenAI AI Text Classifier cannot accept short texts. And the prompt for paraphrasing is used in Table 7 for paraphrasing $\mathbf{Y}$ into $\mathbf{Y}'$ directly.

$\mathbf{X}_p$ on XSum	During this period, we request that you adhere to the writing style and terminology employed in the provided excerpt. "Wetherspoon, the British pub chain, has announced plans to boost employee wages above the government’s National Living Wage as part of an initiative to reduce high staff turnover and enhance job satisfaction. This move will put hourly pay above the government’s standard by between 10p and 50p, exceeding £7.50 per hour for staff aged 25 and over. The pay rise will become effective throughout August and will apply to the chain’s 37,000 UK employees. Training and wage rises were both common themes identified in staff requests, according to Founder and Chairman, Tim Martin. However, the precise impact of this increase upon the company’s expenditure is unclear. Although staff wages will be brought closer to the Office for National Statistics’ average UK hourly wage of £11.22, Wetherspoon’s existing net annual profits of £77.8m have not been quantified with respect to this new policy. The move came after a year of controversy when Martin was previously criticized by the press for, among other things, instructing staff to remove the UK’s Evening Standard newspaper from his chain’s town centre premises."
$\mathbf{X}_p$ on ELI5	Meanwhile, please imitate the writing style and select the same words used in the following passage: *{*$\mathbf{X}_{ref}$ *redacted as it happens to be about a controversial political issue.}*

Table 11: Our searched instructional prompts on XSum and ELI5 respectively.