GradEscape: A Gradient-Based Evader Against AI-Generated Text Detectors

Wenlong Meng[†] Shuguo Fan[†] Chengkun Wei[†,*] Min Chen[≀] Yuwei Li[♮§]
Yuanchao Zhang[‡] Zhikun Zhang[†,*] Wenzhi Chen[†]

[†]*Zhejiang University* [≀]*Vrije Universiteit Amsterdam* [♮]*National University of Defense Technology*
[§]*Anhui Province Key Laboratory of Cyberspace Security Situation Awareness and Evaluation* [‡]*Mybank, Ant Group*

{mengwl, fanshuguo, weichengkun, zhikun, chenwz}@zju.edu.cn
m.chen2@vu.nl, liyuwei@nudt.edu.cn, yuanchao.zhang@mybank.cn

Abstract

In this paper, we introduce GradEscape, the first gradient-based evader designed to attack AI-generated text (AIGT) detectors. GradEscape overcomes the undifferentiable computation problem, caused by the discrete nature of text, by introducing a novel approach to construct weighted embeddings for the detector input. It then updates the evader model parameters using feedback from victim detectors, achieving high attack success with minimal text modification. To address the issue of tokenizer mismatch between the evader and the detector, we introduce a warm-started evader method, enabling GradEscape to adapt to detectors across any language model architecture. Moreover, we employ novel tokenizer inference and model extraction techniques, facilitating effective evasion even in query-only access.

We evaluate GradEscape on four datasets and three widely-used language models, benchmarking it against four state-of-the-art AIGT evaders. Experimental results demonstrate that GradEscape outperforms existing evaders in various scenarios, including with an 11B paraphrase model, while utilizing only 139M parameters. We have successfully applied GradEscape to two real-world commercial AIGT detectors. Our analysis reveals that the primary vulnerability stems from disparity in text expression styles within the training data. We also propose a potential defense strategy to mitigate the threat of AIGT evaders. We open-source our GradEscape for developing more robust AIGT detectors.[1]

1 Introduction

The past years have witnessed the powerful capability of large language models (LLMs) in natural language generation (NLG) tasks [1–4]. However, AI-generated text (AIGT) may be used for malicious purposes, such as fabricating plausible yet false information [5], automating the production of spam and phishing email [6], and creating biased or discriminatory content [7]. Furthermore, many studies have revealed that AIGT is difficult to distinguish by humans [8, 9].

To mitigate the misuse of AIGT, researchers have proposed a number of methods for AIGT detection [10–15]. The most widely used approach is fine-tuning a language model to build a binary classifier, which has been demonstrated effective by many studies [11, 16, 17]. Owing to its ease of employment and independence from the generation process, LM fine-tuning-based detector has become the preferred approach for real-world detection products, including Sapling [18], Scribbr [19], and GPTZero [20].

In parallel, there has been growing interest in developing evasion techniques designed to generate adversarial texts that bypass detection. Evasion techniques are essential for uncovering vulnerabilities in AI detection models, improving their robustness, and simulating real-world attacks [21, 22]. These techniques enable researchers to stress test detection systems, enhance resilience against adversarial inputs, and anticipate future threats to build more secure AI systems. Existing evasion techniques fall into two categories: perturbation-based and paraphrase-based methods. Perturbation-based evaders modify keywords in the text that significantly influence detection outcomes [23, 24], while paraphrase-based evaders employ LMs to rewrite the text [8, 14].

However, our analysis reveals notable shortcomings in current AIGT evaders, particularly regarding compromised text integrity and quality (see Section 6.2). Specifically, perturbation-based methods [23, 24] often degrade the natural flow of the text, leading to awkward or grammatically incorrect sentences. Paraphrase-based strategies [8, 14], while maintaining grammatical correctness, tend to make extensive and uncontrollable changes to the original text's meaning or style, resulting in a loss of the original intent and key details.

Gradient-based methods offer a promising solution to the issues of text integrity and quality. For example, in the image domain, gradient-based adversarial attacks are commonly employed to create imperceptible perturbations that alter a model's output [25, 26]. *However, the discrete nature of text presents a significant challenge for applying such methods in*

*Corresponding authors.
[1]Code is available at https://doi.org/10.5281/zenodo.15586856.

the text domain because of undifferentiable computation.

Our Contributions. To fill this gap, in this paper, we propose the first gradient-based evader, named GradEscape. GradEscape fine-tunes a *sequence-to-sequence* (seq2seq) model and transforms it into a paraphrase, which enables AIGT to be paraphrased in such a way that can be recognized by detectors as human-generated text while maintaining syntactic and semantic integrity.

To solve the undifferentiable problem, we intercept the token probability output of the seq2seq model and use the token probabilities at each position to weight the embedding dictionary of the victim detector. These weighted embeddings can then be fed into the detector. To address the issue of tokenizer mismatch between the evader and the detector, we propose a *warm-started evader* method, where a pretrained seq2seq model is built using the same tokenizer as the detector model. These methods overcome the challenge posed by the discrete nature of text by operating in a continuous embedding space, enabling seamless integration of gradient-based techniques.

We formulate our GradEscape as an optimization problem with two objectives: (1) the *function constraint*, which focuses on fooling the victim detector, and (2) the *consistency constraint*, aimed at maintaining the text integrity and quality. We implement these two constraints with dedicated losses and transform the evader model training to an unsupervised manner that requires only AIGT rather than human-generated text as training data. Specifically, to enforce function constraint, we feed the probability output of the seq2seq model into the frozen victim detector, subsequently optimizing the seq2seq model to elicit human-label predictions from the detector. For the consistency constraint, we employ pairwise cross-entropy loss to maintain syntactic similarity and leverage text encoder models to ensure semantic coherence.

In the opaque model scenario, a more strict and practical situation where an attacker has only query access to the victim detector, we develop an *opaque model attack*. This attack involves constructing a *surrogate detector* and then training the evader against it. We determine the detector's architecture through a novel *tokenizer inference attack*, which speculates a model's tokenizer to infer its architecture. Then we construct similar parameters via *model extraction attacks* [27–29]. To our knowledge, *we are the first to exploit inherent vulnerabilities of LM tokenizers to inference model privacy.*

We conduct experiments on 4 AIGT detection datasets and compare GradEscape with 4 state-of-the-art evaders. When the ROUGE score, a widely used text consistency metric, is set to 0.9, our method consistently achieves higher evasion rates across all datasets, surpassing the performance of other evaders, including one powered by an 11B LLM. GradEscape achieves this with only 139M parameters. In the opaque model attack scenario, using just 2,000 queries, our GradEscape achieves evasion rates that are 90% as effective as when the attacker has full knowledge of the victim detector. We also apply GradEscape against two real-world commercial AIGT detectors. With a query cost of $10, GradEscape achieves an average evasion rate of 0.617.

Table 1: Language model taxonomy.

LM Type	Architecture	Applied Tasks	Roles
CLM	Decoder Only	NLG, NLU	Generator, Detector
MLM	Encoder Only	NLU	Detector
Seq2seq	Encoder-Decoder	NLG	Evader

Furthermore, to mitigate the threat posed by GradEscape, we propose a novel black-box, detector-independent defense that removes the subtle modifications made by evaders. Our defense leverages an off-the-shelf LLM to paraphrase input text before detection, ensuring that all inputs share a consistent expression style. With this defense, we successfully reduce the evasion rate to 20%. In essence, our contributions can be summarized as follows:

- We present GradEscape, the first gradient-based evader, for attacking AIGT detectors. GradEscape leverages victim detectors' gradient to update its model parameters, which allows GradEscape to achieve stronger attacks with a small model size. At the same time, we introduce a warm-started evader technique, making GradEscape adaptable for attacking detectors built on arbitrary LM architecture.
- We introduce the opaque model attack, enabling the adaptation of GradEscape to scenarios where the attacker has only query access to the victim detector. Opaque model attack comprises a novel tokenizer inference technique that allows for probing the detector's model architecture through crafting dedicated queries.
- We evaluate GradEscape on 4 datasets against 3 popular LMs. We reproduce 4 state-of-the-art AIGT evaders and compare them with GradEscape. Experimental results demonstrate that our GradEscape outperforms existing evaders in various scenarios. Additionally, we successfully applied our GradEscape against two real-world commercial AIGT detectors.
- We find that the threat arises from differences in text expression style within the training set. To address this, we propose a novel, back-box, detector-independent defense that neutralizes subtle modifications made by evaders.

2 Preliminaries

2.1 Language Models

A language model is a probabilistic model of natural languages. It is usually a Transformer [30] model trained on large amounts of raw text in a self-supervised fashion. This process is generally referred to as *pretraining* to distinguish it from downstream training for adapting specific tasks. Based on the approach of pretraining, LMs can be categorized into three types: causal language models, masked language models, and sequence-to-sequence models, as shown in Table 1.

Casual Language Models (CLMs). During pretraining,

CLMs predict the next token based on the preceding input. The loss function for a given token sequence $\{x_1, x_2, \ldots, x_n\}$ and model parameters θ is:

$$L_\theta = -\sum_{i=1}^{n-1} \log \mathrm{P}_\theta \left(x_{i+1} \mid x_1, x_2, \ldots, x_i\right) \quad (1)$$

CLMs consist only of the decoder part of the Transformer, featuring a unidirectional attention mechanism. CLMs possess strong text generation capabilities [31, 32].

Masked Language Models (MLMs). Represented by BERT [33], MLMs aim to minimize the loss of masked token prediction. A certain proportion of tokens (generally 15%) are randomly selected and masked in the input sequence. The model then predicts these masked tokens with a loss:

$$L_\theta = -\sum_{x_m \in \mathcal{M}} \log \mathrm{P}_\theta \left(x_m \mid x_1, x_2, \ldots, x_{m-1}, x_{m+1}, \ldots, x_n\right), \quad (2)$$

where $\mathcal{M}$ is the set of masked tokens. MLMs utilize only the encoder part of the Transformer and feature a bidirectional attention mechanism. They are primarily used for *natural language understanding* (NLU) tasks. With a similar number of parameters, MLMs demonstrate superior performance on NLU tasks compared to CLMs [33, 34].

Sequence-to-Sequence Models (Seq2seq Models). Seq2seq models have two inputs: a source sequence $\{x_1, x_2, \ldots, x_n\}$ and a target sequence $\{y_1, y_2, \ldots, y_k\}$ during pretraining. The training objective is to minimize the prediction loss of the target sequence:

$$L_\theta = -\sum_{j=1}^{k} \log \mathrm{P}_\theta \left(y_j \mid y_1, y_2, \ldots, y_{j-1}, \mathbf{C}\right), \quad (3)$$

where $\mathbf{C} = \mathrm{E}_\theta \left(x_1, x_2, \ldots, x_n\right)$ is the context vector generated by the encoder part of the seq2seq model. Seq2seq models consist of both the encoder and decoder parts of the Transformer. While CLMs are primarily used for general generation tasks, seq2seq models are mainly utilized for specific generation tasks, such as machine translation [35] and summarization [36].

2.2 AIGT Detection

AIGT detection techniques can be divided into *post-hoc* and *proactive* methods, depending on whether they require interaction with the generation process. Proactive methods require actions during generation, while post-hoc methods do not.

Post-hoc Detectors. The most common post-hoc detector is a binary classifier obtained by fine-tuning an LM. We refer to this type of detector as a deep learning-based detector. This simple yet effective method has been validated by various studies [16, 37, 38]. Beyond vanilla fine-tuning, researchers proposed improvements. For instance, CheckGPT [17] freezes the LM while adding a BiLSTM head to enhance training efficiency; MPU [11] employs a multi-scaling module and *positive unlabeled* (PU) loss to increase detection accuracy for short texts; DEMASQ [12] utilizes an energy-based model that involves a regulation of drumhead vibrations. Another type of post-hoc detector is statistic-based, which trains regression models based on statistical information from the text. For example, GLTR [39] tracks the probability of each token's occurrence; DetectGPT [40] perturbs the target text and then compares changes in perplexity. Deep learning-based detectors generally have higher detection accuracy than statistic-based detectors, but require a large training set. Statistic-based detectors perform better in few-shot scenarios.

Proactive Detectors. Unlike post-hoc detectors, proactive detectors require modifications to the LLM inference process. Therefore, this type of detector typically needs maintenance from the LLM service provider, who then provides a detection API to users. There are two main types of proactive detectors: watermark-based and retrieval-based. Watermark-based detectors inject invisible watermarks into the text during generation and then measure the strength of these watermarks during detection [15, 41–43]. For example, KGW [15] randomly selects a set of "green" tokens before a word is generated and then softly promotes the probability of green tokens during sampling. For detection, KGW calculates z-scores based on the number of green tokens. Retrieval-based detector [14] stores historically generated text in a database. Later, text samples are evaluated by matching against this database.

In this paper, we focus on post-hoc detectors since they have been employed by real-world products. Watermark-based detectors jeopardize text quality and are extremely vulnerable to paraphrasing attacks [8, 14]. Retrieval-based detectors require the storage of user data, which raises data privacy concerns and violates the European GDPR [44] law. Furthermore, storing all historical data is impractical, considering the vast amount of text generated daily. OpenAI once stated that GPT3 alone produces 4.5 billion words per day.[2]

2.3 Evasion Attacks

AIGT detection is a combat. The adversary may be aware of the existence of the detector and utilize an evader to bypass the detection by editing the generated text without changing its meaning. Existing evasion techniques lie in two main lines: perturbation-based and paraphrase-based.

Perturbation-based Evaders. These evaders alter specific words in the target text, aiming to remove important words that trigger AIGT detection. Random perturbation (RP) [23] replaces random words in the text with synonyms that preserve the semantic content. DFTFooler [23] found that AIGT usually has a lower perplexity than human-generated

[2] https://openai.com/blog/gpt-3-apps

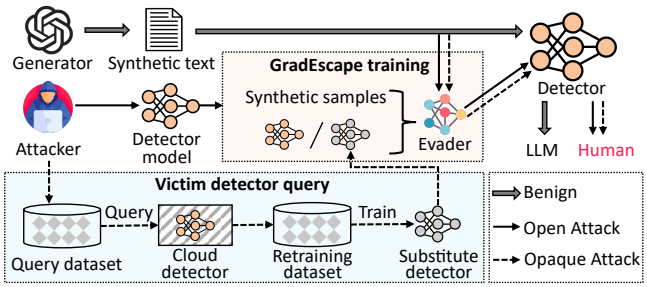


Figure 1: Attack scenarios.

text. Therefore, DFTFooler replaces the top-N most confidently predicted words according to a backend LM. Different from the above two black-box evaders, TextFooler [24] is a gray-box evader necessitating queries to the target detector. TextFooler finds important words to replace and the best replacement words by querying the detector model. Since TextFooler utilizes detector information, it performs a better attack performance than RP and DFTFooler when perturbing the same number of words. However, in our experiment, TextFooler needs 500-700 queries to complete one sample attack, which makes it impractical in real-world settings.

Paraphrase-based Evaders. These evaders paraphrase the target text. They assume that distinctive patterns in the output of LLMs are the primary factors that trigger AIGT detection. DIPPER [14] is an 11B seq2seq model fine-tuned on PAR3 dataset [45], which contains multiple English translations of non-English novels aligned at a paragraph level. DIPPER can paraphrase a paragraph of text with specific settings for lexicon diversity and order diversity. SentPara [8] is a lightweight paraphrase-based evader. The process begins by dividing the input text into individual sentences. Following this, SentPara applies a pre-existing sentence-level paraphrase model to rephrase each sentence sequentially. In the final step, it reassembles these paraphrased sentences into a coherent whole. SentPara only requires a model with 222M parameters to execute the attack, which is 1/50 the size of DIPPER.

Flaws of Existing Evaders. Perturbation-based evaders' word-changing strategy can significantly reduce the fluency and readability of the text. Paraphrase-based evaders' modification magnitude is hard to control. Although DIPPER provides lexicon and order knobs, there are only five settings to choose from. Some researchers also proposed attack methods involving the addition of special symbols in the text [46, 47]. However, these special symbols can be easily filtered out, making them difficult to use in real-world scenarios.

3 Threat Model

Attacker's Goal. We assume that the attacker is an LLM user and intends to apply the LLM-generated content in environments equipped with AIGT detectors. The attacker aims to employ an automated evader to modify the LLM-generated text in a way that circumvents detection by the AIGT detector. Simultaneously, the attacker wishes for the evader's modifications not to compromise the syntactic and semantic integrity and fluency of the text. For instance, consider a scenario in which the attacker is a fraudster who is using the LLM to craft phishing emails. Aware of the AIGT detection feature in the victim's email system, the attacker may employ an evader to alter the generated phishing email, allowing it to bypass the detection mechanism.

Attacker's Capabilities. The attacker can access public text datasets, such as WikiText [48], and open-source LMs, such as BERT [33] and BART [49]. We require the attacker to possess a *synthetic dataset*, which contains solely LLM-generated text. Note that, unlike detector training, which requires both LLM-generated and human-generated text, GradEscape only needs LLM-generated samples. Collecting the synthetic dataset is trivial for the attacker, as they can generate it using their own LLM. For the accessibility to the victim detector, we consider two scenarios: *open model attack* and *opaque model attack*, as depicted in Figure 1.

In open model attack scenarios, we assume the attacker knows the victim detector's model architecture and parameters. This situation occurs with open-source detectors. For instance, OpenAI released their GPT2 output detector [50] on Hugging Face. Open model attacks are also practical when the attacker knows the detector's architecture and training set.

In the opaque model attack, we further illustrate that GradEscape is also effective when the attacker can only query the victim detector, e.g., via an API, and obtain the query results. The attacker has a query budget N_q that limits the maximum number of queries that can be made. We assume that the attacker possesses $N_q/2$ human-generated samples and $N_q/2$ AI-generated samples for querying. By querying, the attacker can deduce the victim detector's model architecture and use query results to train a substitute model. As a specific LM often corresponds to a particular tokenizer, the attacker only needs to identify the tokenizer used by the detector to infer its model architecture. The attacker can craft inputs that are tokenized identically under the specific tokenizer but differently under others and then query the victim detector with these inputs. By observing whether the detector's response scores differ, the attacker can determine whether the detector is utilizing that specific tokenizer. We elaborate the tokenizer inference attack in Section 5. Based on the responses from queries, the attacker can construct a retraining dataset to train a surrogate victim model and then use the surrogate model and the synthetic dataset to train the GradEscape evader.

4 GradEscape Methodology

In this section, we will detail how to employ GradEscape to attack open models. We will elaborate on how GradEscape handles opaque models in Section 5.

4.1 Overview

The goal of AIGT evasion is to revise AIGT to bypass detection while maintaining as much syntactic and semantic integrity as possible. We can deem an evader as a transformation $\mathcal{F}_\theta$ whose input and output are both texts, where θ represents its model parameters. Formally, given the victim detector D_V and input $\{x_i\}_{i=1}^n$, the training objective of the evader model can be expressed as:

$$\min \quad \mathcal{L}_{D_V}\left(\mathcal{F}_\theta\left(\{x_i\}_{i=1}^n\right), y_{\text{human}}\right), \tag{4}$$

$$\text{s.t.} \quad \text{dis}_{syn}\left(\mathcal{F}_\theta\left(\{x_i\}_{i=1}^n\right), \{x_i\}_{i=1}^n\right) < T_{syn}, \tag{5}$$

$$\text{dis}_{sem}\left(\mathcal{F}_\theta\left(\{x_i\}_{i=1}^n\right), \{x_i\}_{i=1}^n\right) < T_{sem}. \tag{6}$$

Here, $\mathcal{L}_{D_V}$ is the cross-entropy loss of detector D_V; dis_{syn} and dis_{sem} measure the *syntactic distance* and *semantic distance* between the input and output of $\mathcal{F}_\theta$, respectively, while T_{syn} and T_{sem} represent the *syntactic threshold* and *semantic threshold*, respectively. T_{syn} constrains the extent to which $\mathcal{F}_\theta$ can alter the text's structure, while T_{sem} ensures that $\mathcal{F}_\theta$ does not change the meaning of the text significantly. Since seq2seq models outperform CLMs on specific NLG tasks with the same parameter size [51–53], we choose the seq2seq model as our evader $\mathcal{F}$.

A significant challenge in evader training is the lack of parallel data, where each input text is paired with a corresponding output text, necessary for supervised fine-tuning of $\mathcal{F}_\theta$. Constructing such a parallel dataset entails a lot of manual effort. To solve this dilemma, we transform evader training into an unsupervised learning method by reformulating Equation 4 to Equation 6 into distinct optimization objectives. To implement the function constraint (Equation 4), we pass the gradient of the detector's classification loss to the evader. To enforce the consistency constraint (Equation 5 and Equation 6), we introduce two losses to increase the syntactic and semantic similarity between the paraphrased and the input text.

The above training strategy is also part of our opaque model attacks, where the attacker does not have access to the internal parameters or architecture of the victim detector. In this scenario, once a substitute detector is obtained, the attacker can use the substitute detector as D_V to train evaders in the same manner. We elaborate on the process of obtaining substitute detectors in Section 5.

4.2 Evader Training Details

We formulate our GradEscape as an optimization problem and use gradient descent to solve it. In particular, we transform Equation 4 to Equation 6 into *classification loss*, *label loss*, and *semantic loss*, respectively. Classification loss serves as the functional constraint, while label loss and semantic loss act as the consistency constraint. Through classification loss, the evader can acquire new knowledge from the detector model's gradients to overcome the challenge of contribution collapse. The training procedure of GradEscape is illustrated in Figure 2.

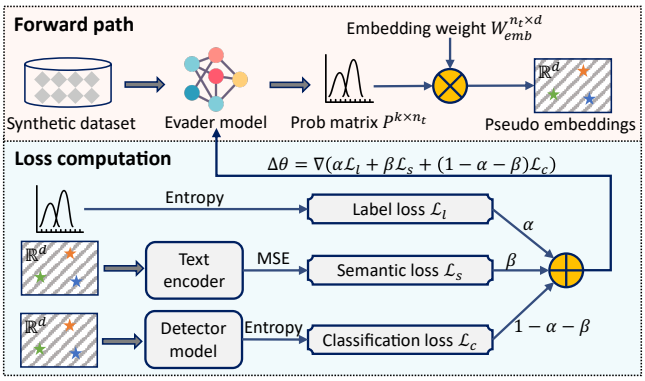


Figure 2: GradEscape training procedure.

Function Constraint. The purpose of the classification loss is to allow the evader to generate output that spoofs the victim detector. In the white-box scenario, one can calculate the classification loss by using the output of $\mathcal{F}_\theta$ as the input of D_V through Equation 4. This method has been widely used in the *compute vision* (CV) domain (e.g., *generative adversarial network* (GAN)) to craft deepfake images. Nevertheless, it is not commonly used in the NLP domain. Due to the discrete nature of text, the process of sampling text from the probabilities output by LMs is non-differentiable, hindering the backpropagation of gradients. To overcome the problem of non-differentiability, we employ a technique named *pseudo embeddings* illustrated in Figure 2. Its main idea is to use matrix multiplication to replace the lookup operation in the token embedding layer. Before passing LMs, a text would first be segmented into tokens and converted into token IDs (see examples in Table 2). These token IDs are then used to look up corresponding token embeddings in the embedding matrix $W_{emb}^{n_t \times d}$. Here, n_t is the number of tokens in the LM dictionary, and d is the dimension of token embeddings. The pseudo-embedding technique replaces token IDs with token probability vectors as inputs, using matrix multiplication to generate their embeddings. This technique has been proven effective in recent research on text backdoor detection [54, 55], text backdoor attack [56], and text adversarial attack [57]. Based on pseudo embeddings, our classification loss can be formulated as follows:

$$\mathcal{L}_c = \mathcal{L}_{ce}\left(D_V\left(P^{k \times n_t} \cdot W_{emb}^{n_t \times d}\right), y_{\text{human}}\right), \tag{7}$$

where $\mathcal{L}_{ce}$ stands for cross-entropy loss while $P^{k \times n_t} = \mathcal{F}_\theta\left(\{x_i\}_{i=1}^n\right)$ is the probability matrix output by the evader. k is the output sequence length. y_{human} represents the label of human-generated texts. We perform post-processing on $P^{k \times n_t}$ to enable the multiplication of $P^{k \times n_t}$ and $W_{emb}^{n_t \times d}$ in practical implementations. The key idea of post-processing is to reassign the probability assigned to tokens generated by the evader model but missing from the detector model. We redistribute this probability to tokens in the detector model. The

details are provided in Appendix B.

Consistency Constraint. In *reinforcement learning from human feedback* RLHF [58], the pairwise KL divergence between the output probability distribution of policy and reference model is used as the consistency constraint. In AIGT evasion scenarios, our objective slightly differs from RLHF. Instead of aiming for the trained model's output to be close to the original model, our goal is for the output of the trained model to be close to the input. Therefore, we use the distribution between the output and the input as our label loss. KL divergence is equivalent to cross-entropy when the label is a one-hot probability distribution. As a result, our label loss is formulated as follows:

$$\mathcal{L}_l = \frac{\sum_{i=1}^{n} \mathcal{L}_{ce}\left(P_i, x_i\right)}{n}. \tag{8}$$

One advantage of our label loss is that we do not need to load a reference model on the GPU, which saves half the memory overhead and allows GradEscape to run on consumer-level GPUs.

In addition to label loss, our consistency constraint incorporates a semantic loss. It is essential because the attacker does not hope the evader will change the text semantics significantly. A single label loss is not enough, as a few important word changes can completely alter the meaning of a text, such as changing "yes" to "no", or "love" to "hate". We employ *sentence transformers* [59] to encode both input and output of $\mathcal{F}_\theta$ into vectors. Then we use an MSE loss to constrain their $l2$ distance, which is

$$\mathcal{L}_s = \mathrm{MSE}\left(E_{st}\left(\{x_i\}_{i=1}^{n}\right), E_{st}\left(P^{k\times n_t} \cdot W_{emb}^{n_t \times d}\right)\right). \tag{9}$$

Here $E_{st}(\cdot)$ means the input's feature vector obtained by sentence transformers. We employ a similar pseudo-embedding technique used in classification loss to make the process of computing the output's feature vector differentiable. In our implementation of GradEscape, $E_{st}\left(\{x_i\}_{i=1}^{n}\right)$ is computed prior to training, as the calculation of $E_{st}\left(\{x_i\}_{i=1}^{n}\right)$ does not involve gradient propagation. This pretraining computation reduces training overhead and avoids loading two E_{st} on the GPU.

Solving Optimization Problem. After defining the three loss terms $\mathcal{L}_l$, $\mathcal{L}_s$, and $\mathcal{L}_c$, we can formulate our GradEscape to an optimization problem by combine them together:

$$\min_{\theta} \mathcal{L} = \alpha \cdot \mathcal{L}_l + \beta \cdot \mathcal{L}_s + (1-\alpha-\beta) \cdot \mathcal{L}_c, \tag{10}$$

where α and β are two hyperparameters to balance the consistency constraint. A larger α places greater emphasis on syntactic similarity, while a larger β emphasizes semantic similarity. By ensuring that the sum of the coefficients for the tree loss terms equals 1, we can eliminate the impact of the learning rate while tuning α and β. Through Equation 10, we can solve the optimization problem using vanilla gradient descent. During the training process, we freeze the text encoder and the detector model, as they do not require optimization.

4.3 Warm-started Evader

Unfortunately, the application of the pseudo embeddings technique encounters the tokenizer mismatch challenge in real-world scenarios, as it implicitly requires the evader and the detector to use the same tokenizer, enabling direct multiplication of P and W_{emb}. However, for CLMs and MLMs, it is not always possible to find corresponding seq2seq models that use the same tokenizer. Bagdasaryan et al. [56] proposed a *token re-mapping* algorithm to resolve the tokenizer mismatch issue between BART and GPT2. However, the token re-mapping algorithm can only address the issue of mismatched token IDs, but not the differences in tokenization methods. As shown in Table 2, BART and GPT2 differ only in the token IDs. They both use *byte-pair encoding* (BPE) tokenization method. The token re-mapping algorithm cannot concatenate BART with BERT, as BERT utilizes *WordPiece* tokenization method. As far as we know, there is no pretrained seq2seq model that uses the WordPiece tokenizer.

In this paper, we propose a warm-started evader method based on the *warm-started encoder-decoder* technique [60] to enable the construction of evaders against any type of detector LM. The warm-started encoder-decoder technique is a way to initialize an encoder-decoder model with pretrained encoder and/or decoder-only checkpoints (e.g., BERT, GPT2) to skip the costly pretraining. It initializes the parameters shared between the encoder-decoder model and the standalone encoder/decoder model using the standalone model's parameters. Parameters unique to the encoder-decoder model will be randomly initialized. In essence, this technique provides a mapping from the parameters of stand-alone encoder/decoder models to the parameters of a seq2seq model:

$$f_{ws} : (\theta_{enc}/\theta_{dec}, \theta_{enc}/\theta_{dec}) \mapsto \theta_{s2s}. \tag{11}$$

We first initialize a seq2seq model using the detector LM and train the seq2seq model to a repeater by SFT. Specifically, our warm-started evader method includes three steps:

Step1: *Model Initialization.* Use function f_{ws} and pretrained detector model to initialize the evader model θ_{eva}.

Step2: *Data Collection.* Collect public corpora, such as WikiText [48] and BookCorpus [61], to form a repeater dataset.

Step3: *Repeater Training.* Sample text from repeater dataset and update θ_{env} by SFT. The output label is a copy of the input so that the model can repeat the input.

In our experiments of BERT, repeater training can be finished within 5,000 steps with a batch size of 16. After getting the warm-started evader, the attacker can carry out GradEscape training following Section 4.2.

Table 2: Tokenization examples of LMs on the sentence of "*Different LMs possess different tokenizers.*"

LM	Tokenizer	Tokens	Token IDs
BART	BPE	'Different', 'ĠL', 'Ms', 'Ġpossess', 'Ġdifferent', 'Ġtoken', 'izers', '.'	44863, 226, 13123, 15256, 430, 19233, 11574, 4
RoBERTa	BPE	'Different', 'ĠL', 'Ms', 'Ġpossess', 'Ġdifferent', 'Ġtoken', 'izers', '.'	44863, 226, 13123, 15256, 430, 19233, 11574, 4
GPT2	BPE	'Different', 'ĠL', 'Ms', 'Ġpossess', 'Ġdifferent', 'Ġtoken', 'izers', '.'	40341, 406, 10128, 8588, 1180, 11241, 11341, 13
BERT	WordPiece	'Different', 'L', '##Ms', 'possess', 'different', 'token', '##izer', '##s', '.'	14380, 149, 25866, 10192, 1472, 22559, 17260, 1116, 119

5 Opaque Model Attack

In this section, we elaborate on how GradEscape handles opaque model scenarios, where the attacker only has query access to the victim detector. Our key idea is to train a surrogate detector to mimic the victim detector. The attacker first conducts a tokenizer inference attack to get the model architecture, then uses a model extraction attack to train the surrogate detector. After that, the attacker can train GradEscape against the surrogate detector according to Section 4.

Tokenizer Inference Attack. This attack exploits the fact that current LMs are bound with specific tokenizers. Although GPT2 and RoBERTa both employ the BPE tokenizer, their special tokens differ. Specifically, GPT2 tokenizer lacks `<pad>` and `<unk>`, which exist in RoBERTa tokenizer. This discrepancy offers attackers a method to deduce the victim detector's tokenizer and infer the model architecture. The main thought of our tokenizer inference attack is to query the detector using inputs that are tokenized identically by a particular tokenizer but differently by others. The attacker can then ascertain if the victim's detector uses the particular tokenizer by comparing the returned scores. Here we provide two methods to craft inference inputs:

- *Padding Spoof.* The attacker can add pad tokens to the end of the input text. LM tokenizers usually pad the text to ensure inputs have equal length. The LM will ignore correct pad tokens, otherwise, it will be tokenized to other tokens. For instance, RoBERT pad token `<pad>` would be tokenized as ["<", "pad", ">"] by GPT2 tokenizer and BERT tokenizer.
- *Space Spoof.* The attacker can replace spaces with LM-specific tokens in the input text and observe the response scores before and after replacement. Different LM tokenizers handle spaces differently. BERT ignores spaces, while LLaMA2 denotes spaces by "▁". "Space spoof" and "Space▁spoof" will both be tokenized as ["▁Space", "▁spo", "of"] by LLaMA2 tokenizer, but would be respectively tokenized as ["Space", "s", "##po", "##of"] and ["`[UNK]`"] by BERT tokenizer.

Model Extraction Attack. Model extraction attack aims to construct a substitute model that closely approximates the functionalities or the parameters of the target victim model. It has been proven effective to steal online models by previous literatures [27–29]. As discussed in Section 3, model extraction attack consists of three steps: query dataset construction, victim model query, and substitute model training. In this paper, we employ the vanilla label-only model extraction attack, where the victim detector only exposes predicted labels. We construct our query dataset using texts drawn from the same distribution as the evader's training dataset, which may differ from the distribution of the detector's training dataset. We ensure that the query dataset does not overlap with either the evader's or the detector's training datasets. Then, we use this dataset to query the victim detector and utilize its predictions to label data, creating the retraining dataset. Finally, we use the retraining dataset to fine-tune the pretrained model whose architecture is known from the tokenizer inference attack. After obtaining the substitute model, the attacker can use it to compute the classification loss, derive gradients, and subsequently update the evader's parameters. There are also other opportunities for improvement in model extraction attacks. However, this area is orthogonal to the research presented in this paper, so we only consider the label-only method in our study. Also, we find that vanilla label-only method can achieve commendable attack performance (see Section 6.3).

6 Evaluation

In this section, we first examine the efficacy of GradEscape against three types of LM classifiers in open model scenarios. Second, we demonstrate that GradEscape can maintain its attack capability in opaque model scenarios. Third, we conduct dataset transfer experiments to test its transferability. Fourth, we assess the computation and memory costs of GradEscape. Fifth, we apply GradEscape to two real-world commercial detectors to demonstrate its practicality.

Additionally, several aspects of our evaluation are documented in the appendix due to space limitations: (1) We assess the ability of GradEscape to compromise advanced detectors, which feature enhanced model architecture or training strategy, in open model scenarios (Appendix C.2). (2) Our evaluation also considers GradEscape against these advanced detectors in opaque model scenarios (Appendix D.1). (3) We test our GradEscape against statistic-based detectors through model querying (Appendix D.2). (4) We examine the semantic consistency of texts generated by GradEscape using both GPT4 annotation and human evaluation, and compare the results with four baseline evaders (Appendix F).

6.1 Experimental Setup

Datasets. We consider four AIGT datasets for evaluation, namely GROVER News [62], HC3 [16], GPA [17], and GPTWiki Intro [63]. To ensure diversity, we consider both

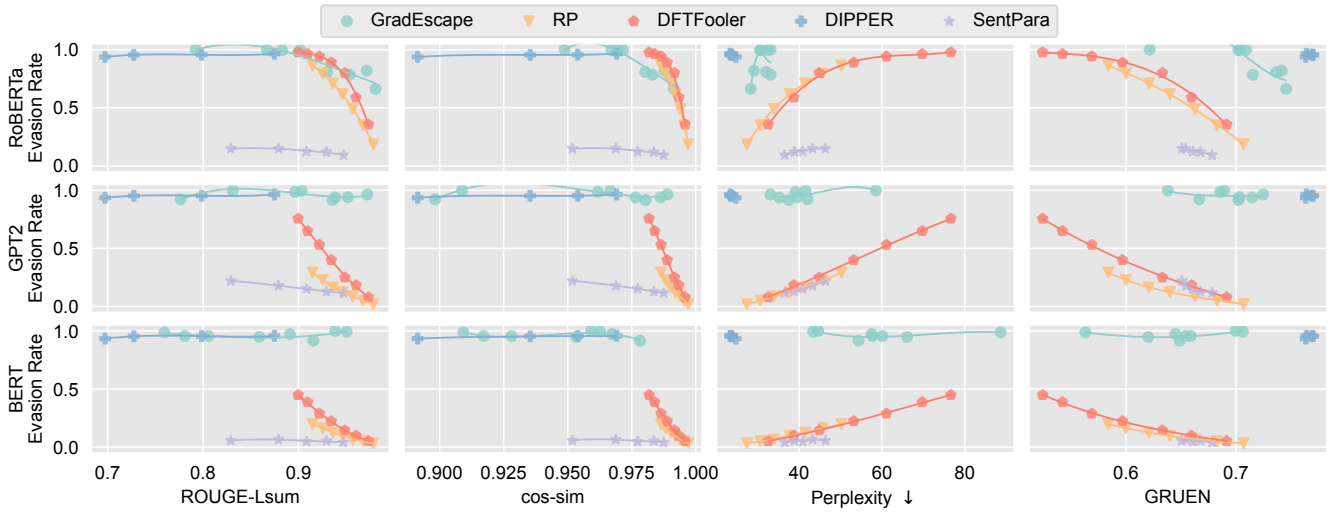


Figure 3: Evasion rates versus text quality metrics on GROVER News dataset.

aligned and unaligned LLMs, and consider three tasks: text completion, question-answering, and instruction-writing. Due to the space limitation, we defer detailed information of datasets to Appendix A.

Cai et al. [46] found that the differences in format between human-generated text and machine-generated text lead detectors to rely on text format for prediction, thereby reducing the robustness of the detector. To eliminate the influence of text format, we clean the text before training detectors. Specifically, we remove line breaks, standardize the use of punctuation, and delete modal particles.

Metrics. We use *evasion rate* (ER) to assess evaders' attack utility. Given that the attacker aims to maintain syntactic and semantic similarity while ensuring readability, we employ four additional text integrity and quality metrics: *ROUGE*, *cos-sim*, *perplexity*, and *GRUEN*. ROUGE is for syntactic similarity; cos-sim is for semantic similarity; while perplexity and GRUEN are for readability. We defer the details of metrics to Appendix A due to space limitations.

Setups of Detectors and Evaders. In this paper, we select three basic detectors, two advanced detectors, and one statistics-based detector as victim detectors. We also choose four representative evaders as baselines. The configurations of detectors and evaders largely follow the original papers. For brevity, we defer the detailed settings in Appendix A.

Implementation. We use HuggingFace Accelerate [64] to implement our GradEscape. Followers can easily run GradEscape using distributed techniques such as Microsoft's DeepSpeed [65] and Nvidia's Megatron-LM [66]. In this work, we use a 2 × A100-80GB GPU node for all experiments. Our node possesses 32 CPU cores and 1TB of memory.

Disscussion. Note that our GradEscape does not require the synthetic dataset to share the same distribution as the dataset used to train the detector. Instead, GradEscape only requires the synthetic dataset to have the same distribution as the texts targeted for evasion. As discussed in Section 3, this requirement is easily met since both of them can be generated from the attacker's LLM. In our evaluations, we primarily focus on attacking detectors with the same dataset distribution as evaders because they are harder to evade. Otherwise, detectors would suffer from low transferability [23]. Section 6.4 demonstrates that GradEscape is also effective when the data distribution of evaders and detectors does not match.

GradEscape can be easily extended to attack LLM-based detectors. In this paper, we do not consider fine-tuning LLMs as detectors because we find that LLMs tend to overfit and achieve lower accuracy. For example, RoBERTa-Base achieves 0.99 accuracy on HC3 and 0.91 accuracy on GROVER, while LLaMA3-8B achieves 0.96 on HC3 and 0.88 on GROVER despite its significant training and inference overhead.

6.2 Open Model Attack

Attack Setup. In this section, we select three popular LMs, RoBERTa, GPT2, and BERT, as victim detectors. The detectors' performance before being attacked is shown in Table 7 (Appendix C.1). All detectors have low baseline evasion rates ($< 2\%$), which means nearly all machine-generated texts are correctly classified. Since BART shares the same tokenizer with RoBERTa, we use BART as the evader model to attack RoBERTa. We use the token re-mapping technique to re-map BART's logits output for attacking GPT2. For BERT, we first build a warm-started evader model using the technique described in Section 4.3. We randomly choose 15,000 texts from each of the AlpacaGPT4 [67], OpenWebText [68], and WikiText [48] datasets to form the repeater dataset. Using this repeater dataset, we fine-tune the warm-started encoder-decoder model for 5,000 steps with a batch size of 16. We

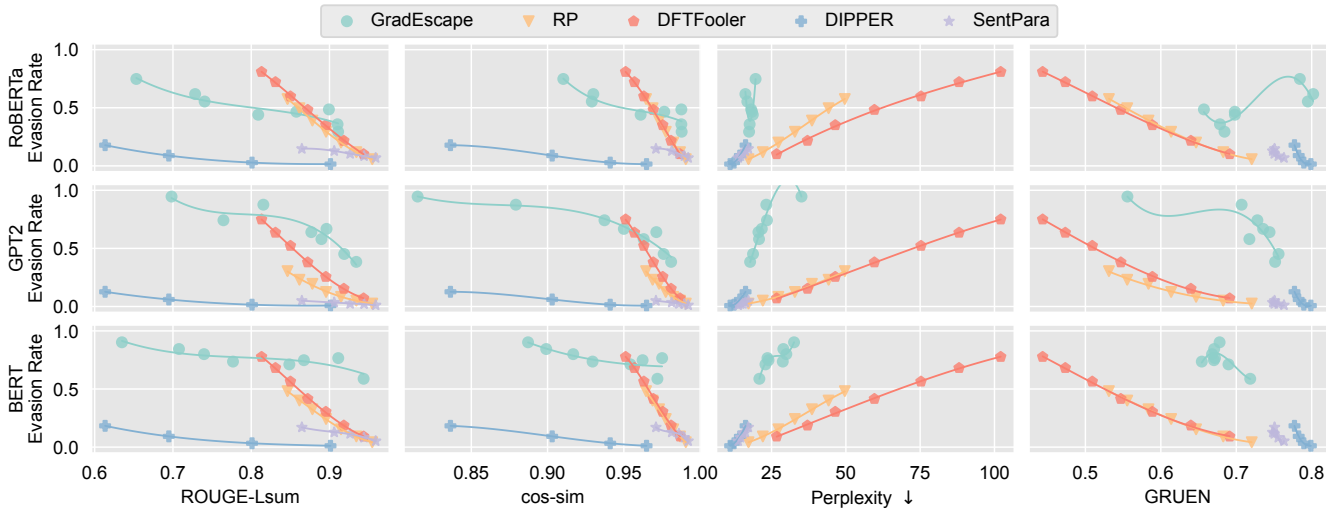


Figure 4: Evasion rates versus text quality metrics on HC3 dataset.

choose 9,000 samples for each dataset to train GradEscape. This evader training set does not overlap with the detector training set. The detailed evader training settings are shown in Table 6. After training evaders, we employ beam search and set the number of beams to 4 to generate paraphrased texts. We compare our GradEscape with 4 state-of-the-art evaders. We adjust α and β to alter the degree of consistency constraint. For baseline evaders, we adjust the knobs they provide.

Results. To consider text quality while comparing the attack utility of evaders, we plot evasion rate versus text quality metrics in Figure 3 and Figure 4 for GROVER News and HC3, respectively. Due to space limitations, we defer the results of GPA and GPTWiki to Appendix C.1. For ROUGE, cos-sim, and GRUEN, a position further toward the upper right is preferable, whereas for Perplexity, a position further toward the upper left is preferable. We can observe that GradEscape is superior to other evaders in most cases. Although RP and DFTFooler have a higher evasion rate than GradEscape under some ROUGE and cos-sim values, they significantly reduce text readability, indicated by high Perplexity and low GRUEN scores. We find that DIPPER only achieves a high evasion rate (> 0.8) on the GROVER dataset, with its evasion rate on other datasets being quite low (< 0.4). Note that DIPPER is an 11B model, whereas our GradEscape only has 139M parameters. GradEscape also achieves a high evasion rate in attacks against BERT detectors, even though we use a warm-started encoder-decoder model as the base model instead of an off-the-shelf seq2seq model. This demonstrates that our warm-started evader technique can effectively address the issue of tokenizer mismatch. An interesting finding is that for GradEscape, the evasion rate has a clear linear relationship with ROUGE and cos-sim, but this linear relationship is not significant with Perplexity and GRUEN. This indicates that the greater the degree of text modification by GradEscape, the easier it is to attack successfully, but the degree of modification does not have a necessary impact on text readability. We show example outputs of GradEscape in Table 16 of Appendix C.1. We find that on GROVER News, which is easy to attack, GradEscape only disturbs a few words like a perturbation-based evader, while on more challenging datasets like HC3, GradEscape significantly changes the way of narration like other paraphrase-based evaders.

6.3 Opaque Model Attack

Attack Setup. We choose 2,000, 4,000, and 6,000 samples from each dataset to query the victim detector, respectively. These query data do not overlap with the training data of the victim detectors. Besides the query attack, we consider a scenario where the attacker does not need to query when the attacker has data from the same distribution as the detector training set to train a surrogate detector. We choose 12,000 samples from each dataset as the *shadow dataset*. These shadow datasets also do not overlap with the detector training set. For each set of experiments, we collect 6 results and adjust α and β to ensure the ROUGE scores are between 0.85 and 0.9. Due to heavy overhead, we only conduct gray-box experiments on GROVER News and HC3.

Results. Our gray-box experimental results are shown in Figure 5. From the figure, we can see that 2,000 queries are sufficient to achieve an evasion rate close to that of a white-box scenario. Especially on the GROVER News dataset, 2,000 queries can achieve an evasion rate close to 100%. The query dataset for 2,000 is only $1/6$ the size of the detector training set. We find that on the HC3 dataset, the fewer queries there are, the more unstable the evasion rate becomes. This may be due to the surrogate model being trained with little data, leading to a tendency to overfit. And evaders trained on an overfitted surrogate model would have poor generalization. Note that in this paper, we use a vanilla label-only model

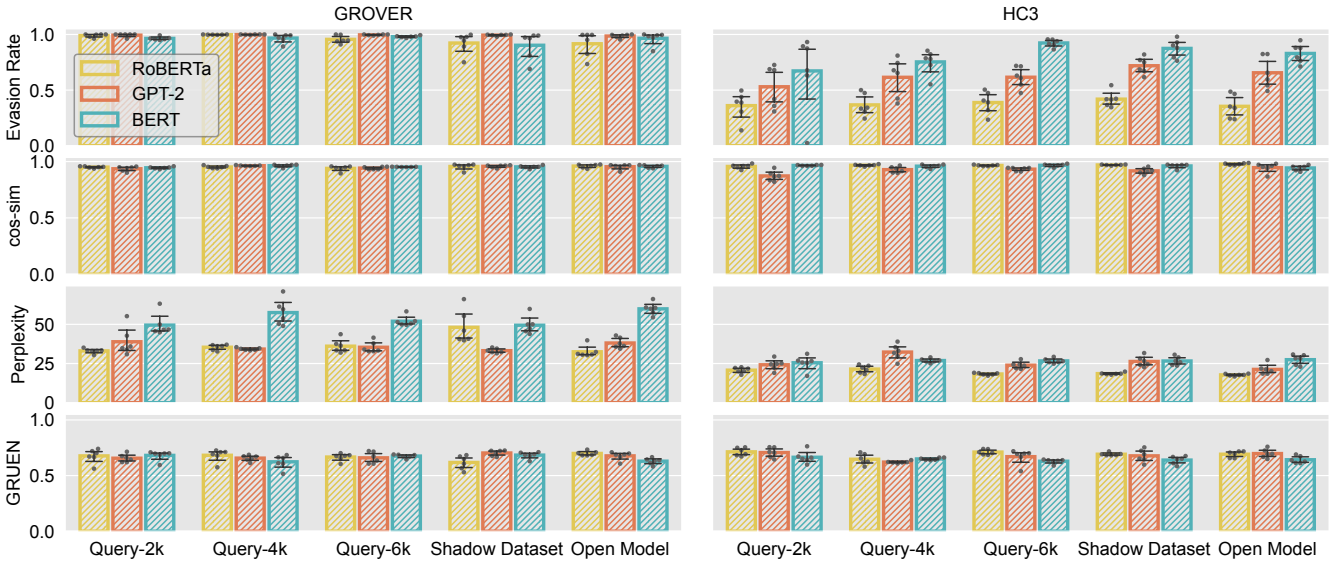


Figure 5: Opaque model attack results.

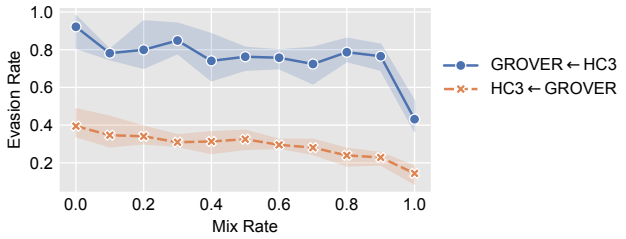


Figure 6: Evasion rates under target AIGT distribution shift. Mix rate is the proportion of external dataset texts mixed into the synthetic training dataset. "GROVER←HC3" denotes training on GROVER mixed with HC3 texts, and vice versa.

extraction attack to construct the surrogate model. Attackers could use improved methods to enhance the generalization of the surrogate model. Another observation is that query attacks decrease the quality of generated text when attacking RoBERTa and GPT2 detectors. But this decrease is within 5%, which is acceptable. We find that shadow dataset attacks can achieve an evasion rate and text quality close to that of a white-box scenario, which enables GradEscape attack without direct access to the detector.

6.4 Dataset Transferability

In this section, we evaluate the capability of GradEscape to generalize across datasets with different distributions. Specifically, we consider two scenarios: (1) the distribution of the evader's training data differs from that of the AIGT targeted for editing; and (2) the distribution of the evader's training data differs from that of the detector's training data.

Transfer to Unseen AIGT. GradEscape relies on a synthetic dataset, composed exclusively of texts generated by LLMs, to train evaders. As discussed in Section 3, the attacker can easily construct synthetic datasets by using LLMs that produce target texts intended for editing. Here, we examine an extreme scenario in which the attacker does not have sufficient AIGT samples that share the same distribution as the target texts to construct a synthetic dataset. We simulate this situation by mixing texts from external datasets into the synthetic training data while keeping the test set unchanged. In particular, we choose GROVER and HC3 datasets and mix in texts from the other dataset to train evaders. Then we measure evasion rates against RoBERTa at a fixed ROUGE score around 0.9.

We change the proportion of mixture and report evasion rates in Figure 6. We vectorize the datasets using TF-IDF with an n-gram range of 2–3 and measure the cosine similarity between mixed and original datasets, which gradually decreases from 1.0 to 0.495 as the mix rate increases. Results show that evasion rates decline with increasing mix rates but remain above half of the original one when the mix rate is below 1.0. A significant drop occurred only when the mix rate reached 1.0, i.e., training exclusively on external datasets. However, this situation is unlikely, as attackers can at least include the targeted texts as the evader's training data.

Transfer to Heterogeneous Detectors. In Section 5, we discuss how attackers can easily obtain the model architecture of the detector; thus, aligning the surrogate model's architecture with that of the victim detector is straightforward. However, attackers may occasionally find that the data they intend to use for an attack does not align with the training data of the victim detector. For example, an attacker may wish to use AI to assist in writing a paper, but the victim detector is trained on a question-answering dataset. Below, we assess GradEscape's effectiveness in such a situation where the distributions of evaders' and detectors' datasets significantly vary. We choose surrogate model training data and evader training data from the same dataset. We select RoBERTa, MPU, and GLTR as

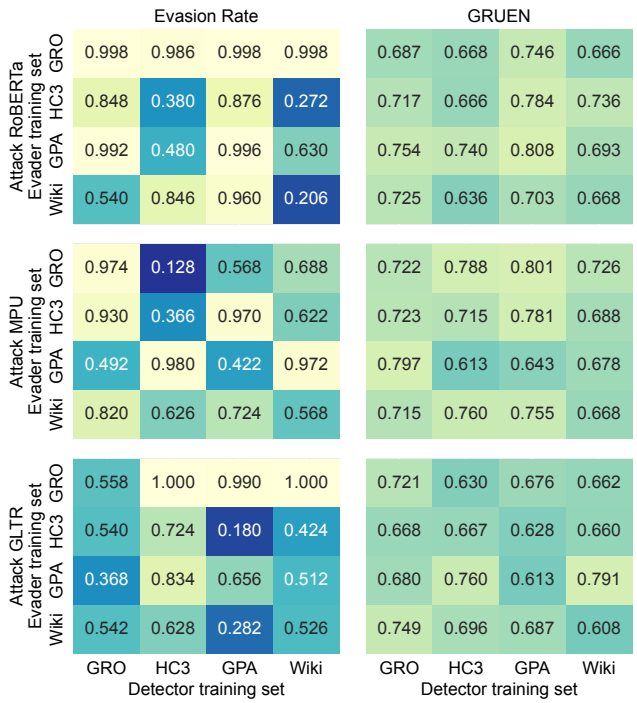


Figure 7: Evasion results under detector distribution shift.

our victim detectors since they perform best in their respective categories. For GLTR, we train a logistic regression model using its word-rank features for classification. Our GLTR implementation is the same as [23] and [38]. We adjust α and β to achieve a target ROUGE of approximately 0.9 and measure the evader's evasion rate against the victim detector.

We report the evasion rates and GRUENs in Figure 7. We discover that using different training sets for evader and detector typically results in higher evasion rates without significantly impacting the text quality. This may be due to the inherently poor data transferability of machine learning models, which leads to the subpar performance of the victim detector when identifying texts that are not from the same distribution as its training set. Evasion rates are more closely related to the evader's training set than to the victim detector training set. For instance, GROVER News, a dataset that is easy to attack, achieves nearly 100% evasion rates across all victim detectors, while HC3, a dataset that is more challenging to attack, has lower evasion rates on all victim detectors. To further assess GradEscape's transferability against detectors trained on heterogeneous datasets, we compare its performance with four baseline evaders. Due to the space limitation, we defer the detailed results in Appendix D.3. While distribution discrepancies degrade GradEscape's performance, it still outperforms baselines in many cases.

6.5 Cost Analysis

The overhead of an evader includes computational expenses and GPU memory usage. We measure the overhead for GradEscape and other baseline evaders. When measuring DIPPER, SentPara, and GradEscape, we set the batch size to 1 and use only one GPU. The results are shown in Table 3. GradEscape has the smallest GPU memory footprint and the shortest computation time among all evaders. While DIPPER boasts considerable attack utility, its inference requires more than 40GB of GPU memory, rendering it impractical for consumer-grade graphics cards. Our GradEscape offers the advantages of a smaller model size and lower training costs. In our experiments, with a batch size of 64 and set for 5 epochs, GradEscape could complete its training within 50 minutes. Due to GradEscape's minimal space requirements, attackers could potentially distribute the evader to sub-attackers for use.

Table 3: Evaders' computation and GPU memory overhead. We measure the time required to complete attacking 100 samples and the maximum GPU memory footprint.

Evader	RP	DFTFooler	DIPPER	SentPara	GradEscape
Time	382s	1182s	412s	329s	105s
GPU Mem	3.51GB	9.54GB	43.51GB	4.43GB	2.48GB

6.6 Real-world Case Studies

In this section, we apply GradEscape to two real-world AIGT detection services provided by Sapling [18] and Scribbr [18] to showcase the vulnerability of existing AI-text detection methods. We first assess the performance of these two detectors across our four datasets in the absence of attacks. The results are presented in Table 13, located in Appendix E. We observed that the performance of the two real-world detectors varies across different datasets. For instance, their detection accuracy on HC3 is higher than 0.91, but they exhibit low recall on GROVER News, meaning they struggle to identify GROVER-generated texts. For each detector, we select two datasets where the performance is comparatively better for conducting attacks. We also compare GradEscape with four baselines against above two real-world detectors. The experimental results show that GradEscape achieves state-of-the-art evaion rates, and Sapling is more robust than Scribbr when confronting evaders. Due to the space limitation, we defer the comparison to Appendix E.

Attack Sapling. Sapling offers AIGT detection service through an HTTP API and a free online demo. Both the HTTP API and the online demo provide the probability that a given text is fake, which we refer to as detection confidence (DC) in this paper. When DC exceeds 0.5, we classify the text as AI-generated. Initially, we leverage its online demo to execute a tokenizer inference attack, aiming to deduce the architecture of its detector. We begin by creating a text with a medium DC, which can be easily done by merging a segment of human-generated text with a piece of AIGT. Following this, we conduct insertion tests and observe the DC variation. We get the following observations: a) Inserting spaces before punctuations, DC changes; b) Appending `<pad>`s

Table 4: Evasion results of real-world case studies.

Detector	Dataset \ ATK	ROUGE ✓	cos-sim ✓	PPL↓ –	PPL↓ ✓	GRUEN –	GRUEN ✓	$\overline{\text{DC}}$↓ –	$\overline{\text{DC}}$↓ ✓	ER –	ER ✓
Sapling	HC3	0.852	0.965	9.2	25.7(+16.5)	0.801	0.685(-0.116)	0.999	0.550(**-0.449**)	0.000	0.448(**+0.448**)
	GPTWiki	0.861	0.960	12.8	27.7(+14.9)	0.758	0.670(-0.088)	0.965	0.422(**-0.543**)	0.026	0.578(**+0.552**)
Scribbr	HC3	0.870	0.965	9.2	19.7(+10.5)	0.801	0.749(-0.052)	0.982	0.378(**-0.604**)	0.018	0.785(**+0.767**)
	GPA	0.884	0.913	13.7	28.5(+14.8)	0.829	0.691(-0.138)	0.992	0.496(**-0.496**)	0.008	0.658(**+0.650**)

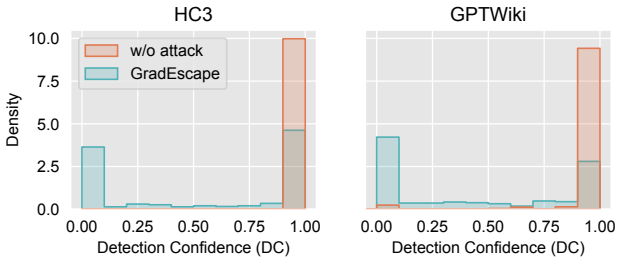


Figure 8: DC histogram of Sapling.

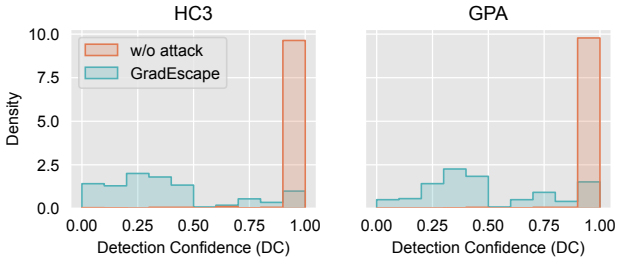


Figure 9: DC histogram of Scribbr.

after the text, DC does not change; c) Prepending `<|endoftext|>`s to the text, DC changes. These observations are illustrated in Figure 19 of Appendix E. From the above three observations, we can conclude three key points: (1) Sapling detector's tokenizer is sensitive to spaces; (2) Its tokenizer employs `<pad>` as the padding token; (3) Its tokenizer does not utilize `<endoftext>` as the padding token. Among the three detector LM architectures considered in this paper (i.e., BERT, RoBERTa, and GPT2), RoBERTa is the only one that meets all the aforementioned criteria. Though we cannot assert Sapling detector is definitively trained on a RoBERTa model, RoBERTa should be appropriate as Sapling detector's surrogate model since they share similar tokenizers.

After determining the appropriate surrogate model architecture, we construct our retraining dataset. Given Sapling detector's minimum input requirement of 50 words, we filter out examples shorter than 50 words from the training sets of HC3 and GPTWiki, then randomly select 2,000 samples to form our query datasets. We utilize Sapling's HTTP API to conduct the querying process. The querying costs for HC3 and GPTWiki are $10.10 and $10.15, respectively. We retain only the labels obtained from the queries to build our retraining dataset, on which we fine-tune the surrogate model. Subsequently, we train an evader on this surrogate model, maintaining ROUGE scores within the range of 0.85 to 0.90. The attack results are depicted in Table 4, and the distributions of DC are presented in Figure 8. Our results demonstrate that, while slightly compromising text quality, GradEscape is able to reduce the DC of half the texts from around 1 to near 0, achieving an evasion rate of approximately 0.5. We present an attack example in Figure 20.

Attack Scribbr. Scribbr also offers a free online demo that outputs DC values. Following the same steps with Sapling, we conduct a tokenizer inference attack on the Scribbr online demo. However, we find that all three types of insertions affect the DC, unlike Sapling. At this point, we are unable to identify a tokenizer that matches Scribbr's tokenization method. We suspect this is due to Scribbr employing a statistic-based detector, or possibly using a combination of statistic-based and deep learning-based detectors for its assessments. In Appendix D.2, we show that GradEscape is also effective against statistic-based detectors. Here, we still choose RoBERTa as the surrogate model despite the inconsistency of tokenization.

Similar to Sapling, Scribbr imposes a restriction on the input length, requiring between 25 to 500 words. We filter the HC3 and GPA training sets to exclude examples outside this range and then choose 2,000 examples to form our query dataset. Unlike Sapling, Scribbr does not offer an API. To implement the querying process, we employ Selenium [69] to write browser automation scripts, assisting us in collecting DC values output by the online demo. Subsequently, we train the surrogate model and evader using the same approach as with Sapling. The attack results are shown in Table 4, with the distribution of DC illustrated in Figure 9. We find that after the attack, the DC distribution for Scribbr is more uniform. GradEscape achieves an evasion rate of over 0.6 on both datasets, which indicates GradEscape is still effective when the detector architecture is not assured. An example of the attack is depicted in Figure 21 of Appendix E.

7 Potential Defense

From the aforementioned attack experiments, we can know that deep learning-based methods are susceptible to subtle variations in textual expression. An evader can modify the text's expression to bypass detection while preserving the underlying semantics. Several text adversarial attacks [24, 70] exploit this vulnerability by substituting essential words with

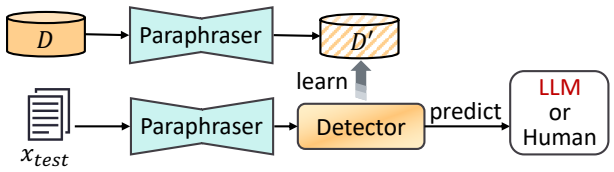


Figure 10: Active paraphrase.

Table 5: Comparison of detection utility between vanilla detector (VD) and paraphrase detector (PD).

	Accuray		FPR		ER_b(FNR)	
	VD	PD	VD	PD	VD	PD
GROVER	0.917	0.609(**-0.308**)	0.148	0.643(**+0.495**)	0.016	0.131(**+0.115**)
HC3	0.995	0.944(**-0.051**)	0.006	0.100(**+0.094**)	0.004	0.010(**+0.006**)
GPA	0.994	0.981(**-0.013**)	0.012	0.018(**+0.008**)	0.000	0.019(**+0.019**)
GPTWiki	0.983	0.942(**-0.041**)	0.034	0.104(**+0.070**)	0.000	0.010(**+0.010**)

their synonyms, thereby inducing NLP models to generate incorrect outputs. Ideally, detectors should evaluate texts based solely on semantics. However, in our training dataset, there is a consistent difference in expression between AIGT and human-generated text. For instance, human-generated text tends to be more colloquial and may include spelling errors. This discrepancy makes it easier for detectors to recognize and use these differences in expression as criteria for detection.

Proposed Defense. We propose employing active paraphrasing prior to building detectors, as illustrated in Figure 10. For a training dataset D, we utilize a paraphraser to rephrase each $x \in D$ to obtain x'. Along with the label y, this forms the new dataset D'. We then train our detector using D'. During the inference stage, any test text x_{test} is first processed by the paraphraser and subsequently by the detector. This paraphrasing step helps minimize discrepancies in text expression during both training and testing phases, thereby allowing the detector to focus exclusively on semantics. We use Llama-3-8B-Instruct [71] as our paraphrase, and set the user prompt "Rewrite the following text for me:\n\n{*input text*}". Our defense strategy can be integrated with any post-hoc detectors, as it only involves modifications to the data preprocessing stage. Notably, our defense remains robust in open model attack scenarios, where the attacker has complete knowledge of both the detector and the paraphraser. Under such conditions, the attacker is unable to train a gradient-based evader because the detector and paraphrase could employ different tokenizers, so the pseudo-embedding technique is not applicable. In our GradEscape, we address this challenge by warm-starting an encoder-decoder model as a repeater. However, there is no technique available to initialize a paraphrase model without considerable overhead.

Evaluation. We employ RoBERTa to build our detectors and initially assess the impact of active paraphrasing on detection utility. The experimental results are presented in Table 5. We observe that, except for GROVER dataset, the accuracy decrease induced by active paraphrasing in the other three datasets remains below 5%, and the evasion rate increases by less than 2%. Active paraphrasing significantly impacts the detection utility on GROVER dataset, possibly due to the smaller semantic differences between AI-generated and human-generated text. This also explains the susceptibility of the GROVER dataset to attack, as revealed in Section 6.2 and Section 6.3. Experimental results on defense utility are shown in Figure 11. Active paraphrasing effectively reduces the evasion rate to below 20% on GROVER dataset and below 10% on other datasets, with the only exception of dealing with our GradEscape on GPA dataset. While active defenses can mitigate most attacks, employing LLM-based paraphrasing for training sets and inputs incurs substantial computational and time overheads, along with a subtle reduction in detection utility. Whether to use this defensive approach requires careful consideration by defenders.

Comparison with Adaptive Defenses. We compare our proposed defense with two adaptive defenses, assuming that the detector developer is aware of GradEscape. The first adaptive defense is *adversarial training* [72]: the detector developer collects texts generated by the evader, augments the original training dataset with these texts labeled as LLM-generated, and then retrains the detector on the augmented dataset. The second adaptive defense is *ensemble detectors* [73]. In this approach, the developer trains multiple detectors using different models, and the final detection result is obtained by aggregating the outputs from each model. We apply soft voting, which averages the predicted probabilities of all models, as the ensemble method. For evader training, we construct substitute detectors using RoBERTa by querying the ensemble detectors. The experimental results are shown in Figure 12. As more adversarial examples are added, adversarial training significantly lowers evasion rates, but it requires over 100 examples to outperform active paraphrasing. Because adversarial training is reactive, the developer must first suffer some attacks, while active paraphrasing provides proactive defense. We also find that adversarial training does not transfer well between different evaders. For example, on the GROVER dataset, 200 adversarial examples from one GradEscape evader reduce the evasion rate to 0.110, but using a different GradEscape evader increases it back to 0.712. This is due to the randomness in GradEscape's training. Ensemble detectors are less effective; with three detectors, the evasion rate only drops to half of the original on HC3. This suggests that different models trained on the same data may share similar weaknesses.

8 Related Work

Text Revision. Our work can be regarded as a special form of text revision. The purpose of text revision is to alter certain properties of a text without changing its semantic meaning. Its applications include text detoxification [74, 75], grammatical error correction [76, 77], text style transfer [78, 79], and

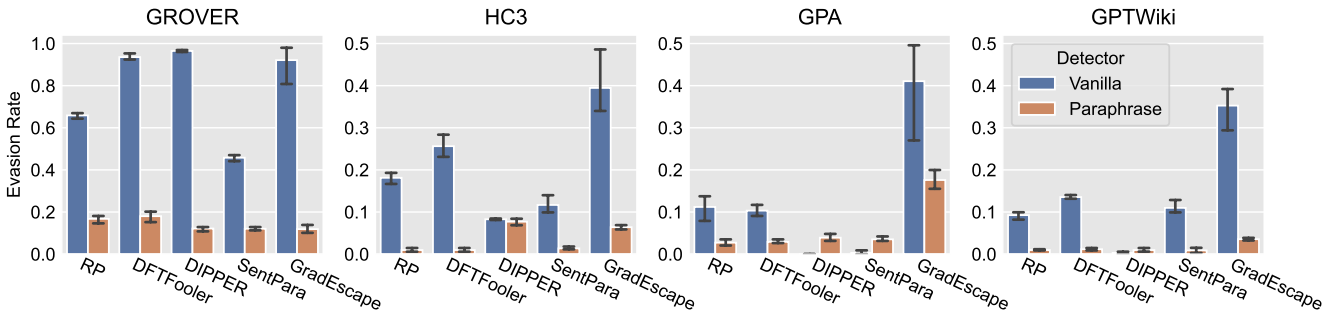


Figure 11: Defense effectiveness of our active paraphrase.

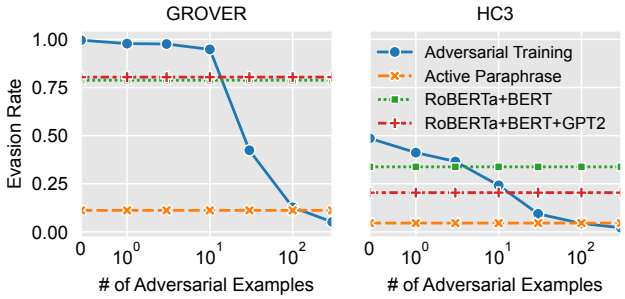


Figure 12: Comparison of active paraphrasing, adversarial training, and ensemble detectors.

even code editing [80]. In GradEscape, the property being altered ensures the text evades the target AIGT detector. Most text revision methods rely on supervised learning and require parallel datasets, which are difficult for evasion attackers to obtain. Existing non-parallel text revision methods are limited by their performance [51] or the need for a large target corpus [81]. We address the above challenge through unsupervised learning, where the evader's parameters are updated based on the gradients provided by the detectors.

Textual Adversarial Attack. Our work can also be deemed as a textual adversarial attack against detector models. Textual adversarial attacks aim to induce the victim model to produce erroneous output (e.g., misclassification) by perturbing the original text. According to a recent study in this field [82], textual adversarial attacks can be categorized into three categories in terms of attacker accessibility, which are blind, decision-based, score-based, and gradient-based. BadCharacters [83], which replaces characters with their homoglyphs, is a blind attack. SEAR [84] leverages a paraphrasing model to generate adversarial examples that help debug models by extracting rules. TextFooler [24] employs a score-based strategy to identify keywords and replace them effectively. On the other hand, TextBugger [70] also relies on returned scores for selecting replacements but discerns important words using gradients. GBDA [57] is the first pure gradient-based adversarial attack in the NLP domain, utilizing gradients to pinpoint both crucial words and their alternatives. Our GradEscape can also be categorized as a gradient-based adversarial attack. However, distinctively, GradEscape eliminates the necessity for attackers to have access to the target or surrogate model after finishing evader training. Thus, an attacker can distribute GradEscape evader, and anyone who acquires the evader can conduct attacks in a plug-and-play manner.

LM Alignment. Our attack is analogous to LM alignment, but it aligns with detector predictions rather than human preferences. LM alignment techniques can be categorized into online and offline training, depending on the need for interaction between policy and reward. The most well-known online training method is RLHF [58], which initially conducts SFt and then optimizes the SFT model using PPO algorithm [85] based on the feedback from a reward model. Some work uses language model feedback (RLAIF) as an alternative to human feedback [86–88]. However, online training often faces optimization instability [89, 90] and sensitivity to hyperparameters [91, 92]. Some researchers have proposed using SFT to replace the RL process [89, 93, 94]. DPO [89] transforms the reward modeling stage into a preference learning stage, while DRPO [94] introduces an odds ratio penalty to the SFT loss. These methods typically employ contrastive pairwise data. POR [95] and RRHF [96] improve preference learning with ranked data.

9 Conclusion

We propose a novel gradient-based AIGT evader, named GradEscape, to paraphrase AI-generated text and then bypass post-hoc AIGT detectors. By efficiently exploiting vulnerabilities in AIGT detectors, GradEscape achieves high evasion rates with a small model size. We develop a warm-started evader technique, enabling GradEscape to attack detectors of any LM architecture. Through tokenizer inference attack and model extraction attack, we successfully extend GradEscape into gray-box scenarios. Extensive experiment results have confirmed that GradEscape's attack utility surpasses that of state-of-the-art AIGT evaders while ensuring text quality. To mitigate the threat of GradEscape, we further propose a new black-box defense that eliminates potential harmful modifications in the input text.

10 Acknowledgements

We thank our anonymous reviewers for their valuable feedback. This research received support from the National Natural Science Foundation of China under Grant No. 62302441, 62441618. This project was also supported by Open Fund of Anhui Province Key Laboratory of Cyberspace Security Situation Awareness and Evaluation, Ant Group, and the Key Research and Development Program Project of Ningbo Grant No. 2025Z029.

11 Open Science

In accordance with the open science policy, we publicly release all artifacts necessary to reproduce the results in this paper, including our experimental datasets, source code, and four evader models targeting real-world AIGT detectors. Although Scribbr has updated its service to employ stronger detectors, we provide a webarchive file so that readers can access the previous version of Scribbr's service and replicate our experiments about Scribbr. All stages of our experimentation are available, including data preprocessing, detector training, evader training, and metric computation. Additionally, we provide a Python library named `AIGT` to streamline the replication of existing AIGT detectors, thereby promoting transparency and reproducibility in AI-generated text detection research. Our code, datasets, and models can be accessed through https://doi.org/10.5281/zenodo.15586856.

12 Ethical Considerations

Our research aims to expose the vulnerabilities in current AIGT detectors, assisting developers in creating more robust systems. We weighed the potential benefits and risks of this research and concluded our strategies can effectively mitigate such risks.

Disclosure and Potential Misuse. Similar to other adversarial attack studies [83, 97, 98], the newly discovered vulnerabilities in our work may provide insight for potential attackers. Nonetheless, we believe that responsible disclosure is generally more beneficial than withholding such findings. To prevent potential misuse, our team has established strict internal regulations and requirements. All code of GradEscape was developed and maintained solely by the first author, who is the only person with access to it. We stored our trained evaders on a server that requires public key authentication to log in, lest they be stolen by unauthorized parties. No individuals outside our team have access to the server.

Possible Defense. We propose an active paraphrasing defense method to neutralize the adversarial modifications made by evaders. Experimental results show that this approach can reduce the evasion rate to below 0.2 at the cost of additional computational overhead for paraphrasing. For overhead-sensitive scenarios, developers may opt for a perplexity-based filter, as our findings suggest that strong attacks often increase text perplexity. However, perplexity filters can cause a lot of false positives when the input is from a diverse source [99]. A recent study introduced certified robustness techniques into NLP classification models [100], offering rigorous guarantees against textual adversarial attacks.

Responsible Real-world Deployment and Human Evaluation. We implement several measures to ensure our real-world studies do not harm actual systems or users. For the dataset, we exclusively choose publicly available corpora. The human-generated texts were sourced from open-access websites such as Reddit, Wikipedia, and OpenWebText. Querying cloud-based AIGT detectors with these texts does not involve private data. When querying Sapling, we employed the official API and paid a total of $20.25 for building evaders. For Scribbr, we employed an automated script to simulate user interactions with the WebUI. To avoid placing an unnecessary load on Scribbr's infrastructure, all queries were sent at night in U.S. Eastern Time. They have updated their service models. Therefore, publishing our work poses no severe harm to these platforms. For human evaluation, we consulted and obtained approval from our IRB and paid our crowdworkers at a rate exceeding the minimum hourly wage in our hiring region.

References

[1] OpenAI, "GPT-4 technical report," *CoRR*, vol. abs/2303.08774, 2023. [Online]. Available: https://doi.org/10.48550/arXiv.2303.08774

[2] R. Anil, A. M. Dai, O. Firat, M. Johnson, D. Lepikhin, A. Passos, S. Shakeri, E. Taropa, P. Bailey, Z. Chen *et al.*, "Palm 2 technical report," *arXiv preprint arXiv:2305.10403*, 2023.

[3] OpenAI, 2022. [Online]. Available: https://openai.com/blog/chatgpt

[4] Github, 2022. [Online]. Available: https://github.com/features/copilot

[5] J. A. Leite, O. Razuvayevskaya, K. Bontcheva, and C. Scarton, "Detecting misinformation with llm-predicted credibility signals and weak supervision," *arXiv preprint arXiv:2309.07601*, 2023.

[6] S. Abdelnabi, K. Greshake, S. Mishra, C. Endres, T. Holz, and M. Fritz, "Not what you've signed up for: Compromising real-world llm-integrated applications with indirect prompt injection," in *Proceedings of the 16th ACM Workshop on Artificial Intelligence and Security*, 2023, pp. 79–90.

[7] K.-C. Yeh, J.-A. Chi, D.-C. Lian, and S.-K. Hsieh, "Evaluating interfaced llm bias," in *Proceedings of the 35th Conference on Computational Linguistics and Speech Processing (ROCLING 2023)*, 2023, pp. 292–299.

[8] V. S. Sadasivan, A. Kumar, S. Balasubramanian, W. Wang, and S. Feizi, "Can ai-generated text be reliably detected?" *arXiv preprint arXiv:2303.11156*, 2023.

[9] C. Chen and K. Shu, "Can llm-generated misinformation be detected?" in *The Twelfth International Conference on Learning Representations*, 2023.

[10] W. Zhong, D. Tang, Z. Xu, R. Wang, N. Duan, M. Zhou, J. Wang, and J. Yin, "Neural deepfake detection with factual structure of text," in *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2020, pp. 2461–2470.

[11] Y. Tian, H. Chen, X. Wang, Z. Bai, Q. Zhang, R. Li, C. Xu, and Y. Wang, "Multiscale positive-unlabeled detection of ai-generated texts," *arXiv preprint arXiv:2305.18149*, 2023.

[12] K. Kumari, A. Pegoraro, H. Fereidooni, and A.-R. Sadeghi, "Demasq: Unmasking the chatgpt wordsmith," *arXiv preprint arXiv:2311.05019*, 2023.

[13] X. Hu, P.-Y. Chen, and T.-Y. Ho, "Radar: Robust ai-text detection via adversarial learning," *Advances in Neural Information Processing Systems*, vol. 36, pp. 15 077–15 095, 2023.

[14] K. Krishna, Y. Song, M. Karpinska, J. Wieting, and M. Iyyer, "Paraphrasing evades detectors of ai-generated text, but retrieval is an effective defense," *Advances in Neural Information Processing Systems*, vol. 36, 2024.

[15] J. Kirchenbauer, J. Geiping, Y. Wen, J. Katz, I. Miers, and T. Goldstein, "A watermark for large language models," in *International Conference on Machine Learning*. PMLR, 2023, pp. 17 061–17 084.

[16] B. Guo, X. Zhang, Z. Wang, M. Jiang, J. Nie, Y. Ding, J. Yue, and Y. Wu, "How close is chatgpt to human experts? comparison corpus, evaluation, and detection," *arXiv preprint arXiv:2301.07597*, 2023.

[17] Z. Liu, Z. Yao, F. Li, and B. Luo, "On the detectability of chatgpt content: benchmarking, methodology, and evaluation through the lens of academic writing," in *Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security*, 2024, pp. 2236–2250.

[18] "Sapling ai detector," https://sapling.ai/ai-content-detector, 2024, accessed on 06/01/2025.

[19] "Scribbr ai detector," https://www.scribbr.com/ai-detector/, 2024, accessed on 06/01/2025.

[20] "Gptzero," https://gptzero.me/, 2024, accessed on 06/01/2025.

[21] A. Fawzi, H. Fawzi, and O. Fawzi, "Adversarial vulnerability for any classifier," *Advances in neural information processing systems*, vol. 31, 2018.

[22] Y. Zhang, S. Hu, L. Y. Zhang, J. Shi, M. Li, X. Liu, W. Wan, and H. Jin, "Why does little robustness help? a further step towards understanding adversarial transferability," in *2024 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2024, pp. 3365–3384.

[23] J. Pu, Z. Sarwar, S. M. Abdullah, A. Rehman, Y. Kim, P. Bhattacharya, M. Javed, and B. Viswanath, "Deepfake text detection: Limitations and opportunities," in *2023 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2023, pp. 1613–1630.

[24] 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," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 34, no. 05, 2020, pp. 8018–8025.

[25] C. Xiao, B. Li, J.-Y. Zhu, W. He, M. Liu, and D. Song, "Generating adversarial examples with adversarial networks," in *Proceedings of the 27th International Joint Conference on Artificial Intelligence*, 2018, pp. 3905–3911.

[26] R. Zhang, J. Liu, Y. Ding, Z. Wang, Q. Wu, and K. Ren, ""adversarial examples" for proof-of-learning," in *2022 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2022, pp. 1408–1422.

[27] F. Tramèr, F. Zhang, A. Juels, M. K. Reiter, and T. Ristenpart, "Stealing machine learning models via prediction {APIs}," in *25th USENIX security symposium (USENIX Security 16)*, 2016, pp. 601–618.

[28] T. Orekondy, B. Schiele, and M. Fritz, "Knockoff nets: Stealing functionality of black-box models," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2019, pp. 4954–4963.

[29] Y. Chen, R. Guan, X. Gong, J. Dong, and M. Xue, "D-dae: Defense-penetrating model extraction attacks," in *2023 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2023, pp. 382–399.

[30] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017.

[31] A. Radford, K. Narasimhan, T. Salimans, I. Sutskever *et al.*, "Improving language understanding by generative pre-training."

[32] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, I. Sutskever *et al.*, "Language models are unsupervised multitask learners," *OpenAI blog*, vol. 1, no. 8, p. 9, 2019.

[33] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," *arXiv preprint arXiv:1810.04805*, 2018.

[34] 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," *arXiv preprint arXiv:1907.11692*, 2019.

[35] L. Xue, N. Constant, A. Roberts, M. Kale, R. Al-Rfou, A. Siddhant, A. Barua, and C. Raffel, "mt5: A massively multilingual pre-trained text-to-text transformer," *arXiv preprint arXiv:2010.11934*, 2020.

[36] K. Akiyama, A. Tamura, and T. Ninomiya, "Hie-bart: Document summarization with hierarchical bart," in *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Student Research Workshop*, 2021, pp. 159–165.

[37] D. Ippolito, D. Duckworth, C. Callison-Burch, and D. Eck, "Automatic detection of generated text is easiest when humans are fooled," in *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, 2020, pp. 1808–1822.

[38] X. He, X. Shen, Z. Chen, M. Backes, and Y. Zhang, "Mgtbench: Benchmarking machine-generated text detection," in *Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security*, 2024, pp. 2251–2265.

[39] S. Gehrmann, H. Strobelt, and A. M. Rush, "Gltr: Statistical detection and visualization of generated text," in *Annual Meeting of the Association for Computational Linguistics*. Association for Computational Linguistics (ACL), 2019.

[40] E. Mitchell, Y. Lee, A. Khazatsky, C. D. Manning, and C. Finn, "Detectgpt: Zero-shot machine-generated text detection using probability curvature," in *ICML*, 2023.

[41] S. Abdelnabi and M. Fritz, "Adversarial watermarking transformer: Towards tracing text provenance with data hiding," in *2021 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2021, pp. 121–140.

[42] X. He, Q. Xu, L. Lyu, F. Wu, and C. Wang, "Protecting intellectual property of language generation apis with lexical watermark," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 36, no. 10, 2022, pp. 10 758–10 766.

[43] J. Fairoze, S. Garg, S. Jha, S. Mahloujifar, M. Mahmoody, and M. Wang, "Publicly detectable watermarking for language models," *arXiv preprint arXiv:2310.18491*, 2023.

[44] P. Voigt and A. Von dem Bussche, "The eu general data protection regulation (gdpr)," *A Practical Guide, 1st Ed., Cham: Springer International Publishing*, vol. 10, no. 3152676, pp. 10–5555, 2017.

[45] K. Thai, M. Karpinska, K. Krishna, B. Ray, M. Inghilleri, J. Wieting, and M. Iyyer, "Exploring document-level literary machine translation with parallel paragraphs from world literature," in *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, 2022, pp. 9882–9902.

[46] S. Cai and W. Cui, "Evade chatgpt detectors via a single space," *arXiv preprint arXiv:2307.02599*, 2023.

[47] J. Gonzales, "Bypass gpt-zero using different fonts," https://gonzoknows.com/posts/Bypass-GPTZero-Using-Different-Fonts/, 2023.

[48] S. Merity, C. Xiong, J. Bradbury, and R. Socher, "Pointer sentinel mixture models," 2016.

[49] M. Lewis, Y. Liu, N. Goyal, M. Ghazvininejad, A. Mohamed, O. Levy, V. Stoyanov, and L. Zettlemoyer, "Bart: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension," *arXiv preprint arXiv:1910.13461*, 2019.

[50] I. Solaiman, M. Brundage, J. Clark, A. Askell, A. Herbert-Voss, J. Wu, A. Radford, G. Krueger, J. W. Kim, S. Kreps *et al.*, "Release strategies and the social impacts of language models," *arXiv preprint arXiv:1908.09203*, 2019.

[51] W. Du, V. Raheja, D. Kumar, Z. M. Kim, M. Lopez, and D. Kang, "Understanding iterative revision from human-written text," in *60th Annual Meeting of the Association for Computational Linguistics, ACL 2022*. Association for Computational Linguistics (ACL), 2022, pp. 3573–3590.

[52] Z. Li, Y. Wang, R. Fan, Y. Wang, J. Li, and S. Wang, "Learning to adapt to low-resource paraphrase generation," in *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, 2022, pp. 1014–1022.

[53] X. Sun, T. Ge, S. Ma, J. Li, F. Wei, and H. Wang, "A unified strategy for multilingual grammatical error correction with pre-trained cross-lingual language model," *arXiv preprint arXiv:2201.10707*, 2022.

[54] A. Azizi, I. A. Tahmid, A. Waheed, N. Mangaokar, J. Pu, M. Javed, C. K. Reddy, and B. Viswanath, "{T-Miner}: A generative approach to defend against trojan attacks on {DNN-based} text classification," in *30th USENIX Security Symposium (USENIX Security 21)*, 2021, pp. 2255–2272.

[55] Y. Liu, G. Shen, G. Tao, S. An, S. Ma, and X. Zhang, "Piccolo: Exposing complex backdoors in nlp transformer models," in *2022 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2022, pp. 2025–2042.

[56] E. Bagdasaryan and V. Shmatikov, "Spinning language models: Risks of propaganda-as-a-service and countermeasures," in *2022 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2022, pp. 769–786.

[57] C. Guo, A. Sablayrolles, H. Jégou, and D. Kiela, "Gradient-based adversarial attacks against text transformers," in *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, 2021, pp. 5747–5757.

[58] L. Ouyang, J. Wu, X. Jiang, D. Almeida, C. Wainwright, P. Mishkin, C. Zhang, S. Agarwal, K. Slama, A. Ray *et al.*, "Training language models to follow instructions with human feedback," *Advances in Neural Information Processing Systems*, vol. 35, pp. 27 730–27 744, 2022.

[59] N. Reimers and I. Gurevych, "Sentence-bert: Sentence embeddings using siamese bert-networks," 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)*. Association for Computational Linguistics, 2019.

[60] S. Rothe, S. Narayan, and A. Severyn, "Leveraging pre-trained checkpoints for sequence generation tasks," *Transactions of the Association for Computational Linguistics*, vol. 8, pp. 264–280, 2020.

[61] Y. Zhu, R. Kiros, R. Zemel, R. Salakhutdinov, R. Urtasun, A. Torralba, and S. Fidler, "Aligning books and movies: Towards story-like visual explanations by watching movies and reading books," in *The IEEE International Conference on Computer Vision (ICCV)*, December 2015.

[62] R. Zellers, A. Holtzman, H. Rashkin, Y. Bisk, A. Farhadi, F. Roesner, and Y. Choi, "Defending against neural fake news," in *Proceedings of the 33rd International Conference on Neural Information Processing Systems*, 2019, pp. 9054–9065.

[63] Aaditya Bhat, "Gpt-wiki-intro (revision 0e458f5)," 2023. [Online]. Available: https://huggingface.co/datasets/aadityaubhat/GPT-wiki-intro

[64] S. Gugger, L. Debut, T. Wolf, P. Schmid, Z. Mueller, S. Mangrulkar, M. Sun, and B. Bossan, "Accelerate: Training and inference at scale made simple, efficient and adaptable." https://github.com/huggingface/accelerate, 2022.

[65] J. Rasley, S. Rajbhandari, O. Ruwase, and Y. He, "Deepspeed: System optimizations enable training deep learning models with over 100 billion parameters," in *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2020, pp. 3505–3506.

[66] M. Shoeybi, M. Patwary, R. Puri, P. LeGresley, J. Casper, and B. Catanzaro, "Megatron-lm: Training multi-billion parameter language models using model parallelism," *arXiv preprint arXiv:1909.08053*, 2019.

[67] B. Peng, C. Li, P. He, M. Galley, and J. Gao, "Instruction tuning with gpt-4," *arXiv preprint arXiv:2304.03277*, 2023.

[68] A. Gokaslan and V. Cohen, "Openwebtext corpus," http://Skylion007.github.io/OpenWebTextCorpus, 2019.

[69] "Selenium," https://www.selenium.dev/, 2024, accessed on 06/01/2025.

[70] J. Li, S. Ji, T. Du, B. Li, and T. Wang, "Textbugger: Generating adversarial text against real-world applications," in *Proceedings 2019 Network and Distributed System Security Symposium*. Internet Society, 2019.

[71] AI@Meta, "Llama 3 model card," 2024. [Online]. Available: https://github.com/meta-llama/llama3/blob/main/MODEL_CARD.md

[72] I. J. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and harnessing adversarial examples," *arXiv preprint arXiv:1412.6572*, 2014.

[73] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Advances in neural information processing systems*, vol. 25, 2012.

[74] S. R. Hallinan, A. Liu, Y. Choi, and M. Sap, "Detoxifying text with marco: Controllable revision with experts and anti-experts," in *The 61st Annual Meeting Of The Association For Computational Linguistics*, 2023.

[75] M. Dehghan, D. Kumar, and L. Golab, "Grs: Combining generation and revision in unsupervised sentence simplification," *Findings of the Association for Computational Linguistics: ACL 2022*, 2022.

[76] M. Mita, K. Sakaguchi, M. Hagiwara, T. Mizumoto, J. Suzuki, and K. Inui, "Towards automated document revision: Grammatical error correction, fluency edits, and beyond," *arXiv preprint arXiv:2205.11484*, 2022.

[77] J. Dwivedi-Yu, T. Schick, Z. Jiang, M. Lomeli, P. Lewis, G. Izacard, E. Grave, S. Riedel, and F. Petroni, "Editeval: An instruction-based benchmark for text improvements," *arXiv preprint arXiv:2209.13331*, 2022.

[78] Z. Yang, Z. Hu, C. Dyer, E. P. Xing, and T. Berg-Kirkpatrick, "Unsupervised text style transfer using language models as discriminators," *Advances in Neural Information Processing Systems*, vol. 31, 2018.

[79] J. He, X. Wang, G. Neubig, and T. Berg-Kirkpatrick, "A probabilistic formulation of unsupervised text style transfer," in *International Conference on Learning Representations*, 2019.

[80] S. Chakraborty, Y. Ding, M. Allamanis, and B. Ray, "Codit: Code editing with tree-based neural models," *IEEE Transactions on Software Engineering*, vol. 48, no. 4, pp. 1385–1399, 2020.

[81] R. Liu, C. Gao, C. Jia, G. Xu, and S. Vosoughi, "Non-parallel text style transfer with self-parallel supervision," in *International Conference on Learning Representations*, 2021.

[82] G. Zeng, F. Qi, Q. Zhou, T. Zhang, Z. Ma, B. Hou, Y. Zang, Z. Liu, and M. Sun, "Openattack: An open-source textual adversarial attack toolkit," in *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing: System Demonstrations*,

2021, pp. 363–371.

[83] N. Boucher, I. Shumailov, R. Anderson, and N. Papernot, "Bad characters: Imperceptible nlp attacks," in *2022 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2022, pp. 1987–2004.

[84] M. T. Ribeiro, S. Singh, and C. Guestrin, "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)*, 2018, pp. 856–865.

[85] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017.

[86] Y. Bai, S. Kadavath, S. Kundu, A. Askell, J. Kernion, A. Jones, A. Chen, A. Goldie, A. Mirhoseini, C. McKinnon *et al.*, "Constitutional ai: Harmlessness from ai feedback," *arXiv preprint arXiv:2212.08073*, 2022.

[87] H. Lee, S. Phatale, H. Mansoor, K. Lu, T. Mesnard, C. Bishop, V. Carbune, and A. Rastogi, "Rlaif: Scaling reinforcement learning from human feedback with ai feedback," *arXiv preprint arXiv:2309.00267*, 2023.

[88] J.-C. Pang, P. Wang, K. Li, X.-H. Chen, J. Xu, Z. Zhang, and Y. Yu, "Language model self-improvement by reinforcement learning contemplation," *arXiv preprint arXiv:2305.14483*, 2023.

[89] R. Rafailov, A. Sharma, E. Mitchell, C. D. Manning, S. Ermon, and C. Finn, "Direct preference optimization: Your language model is secretly a reward model," *Advances in Neural Information Processing Systems*, vol. 36, 2024.

[90] T. Wu, B. Zhu, R. Zhang, Z. Wen, K. Ramchandran, and J. Jiao, "Pairwise proximal policy optimization: Harnessing relative feedback for llm alignment," *arXiv preprint arXiv:2310.00212*, 2023.

[91] R. Zheng, S. Dou, S. Gao, Y. Hua, W. Shen, B. Wang, Y. Liu, S. Jin, Q. Liu, Y. Zhou *et al.*, "Secrets of rlhf in large language models part i: Ppo," *arXiv preprint arXiv:2307.04964*, 2023.

[92] L. Engstrom, A. Ilyas, S. Santurkar, D. Tsipras, F. Janoos, L. Rudolph, and A. Madry, "Implementation matters in deep policy gradients: A case study on ppo and trpo," in *International Conference on Learning Representations*, 2020.

[93] K. Ethayarajh, W. Xu, N. Muennighoff, D. Jurafsky, and D. Kiela, "Kto: Model alignment as prospect theoretic optimization," *arXiv preprint arXiv:2402.01306*, 2024.

[94] J. Hong, N. Lee, and J. Thorne, "Reference-free monolithic preference optimization with odds ratio," *arXiv preprint arXiv:2403.07691*, 2024.

[95] F. Song, B. Yu, M. Li, H. Yu, F. Huang, Y. Li, and H. Wang, "Preference ranking optimization for human alignment," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 38, no. 17, 2024, pp. 18 990–18 998.

[96] Z. Yuan, H. Yuan, C. Tan, W. Wang, S. Huang, and F. Huang, "Rrhf: Rank responses to align language models with human feedback without tears," *arXiv preprint arXiv:2304.05302*, 2023.

[97] C. Slocum, Y. Zhang, E. Shayegani, P. Zaree, N. Abu-Ghazaleh, and J. Chen, "That doesn't go there: Attacks on shared state in {Multi-User} augmented reality applications," in *33rd USENIX Security Symposium (USENIX Security 24)*, 2024, pp. 2761–2778.

[98] Z. Fang, T. Wang, L. Zhao, S. Zhang, B. Li, Y. Ge, Q. Li, C. Shen, and Q. Wang, "Zero-query adversarial attack on black-box automatic speech recognition systems," in *Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security*, 2024, pp. 630–644.

[99] W. Liang, M. Yuksekgonul, Y. Mao, E. Wu, and J. Zou, "Gpt detectors are biased against non-native english writers," *Patterns*, vol. 4, no. 7, 2023.

[100] X. Zhang, H. Hong, Y. Hong, P. Huang, B. Wang, Z. Ba, and K. Ren, "Text-crs: A generalized certified robustness framework against textual adversarial attacks," in *2024 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2024, pp. 2920–2938.

[101] "Common crawl - open repository of web crawl data," 2023. [Online]. Available: https://commoncrawl.org/

[102] C.-Y. Lin, "ROUGE: A package for automatic evaluation of summaries," in *Text Summarization Branches Out*. Barcelona, Spain: Association for Computational Linguistics, Jul. 2004, pp. 74–81. [Online]. Available: https://www.aclweb.org/anthology/W04-1013

[103] D. Cer, Y. Yang, S.-y. Kong, N. Hua, N. Limtiaco, R. S. John, N. Constant, M. Guajardo-Cespedes, S. Yuan, C. Tar *et al.*, "Universal sentence encoder," *arXiv preprint arXiv:1803.11175*, 2018.

[104] W. Zhu and S. Bhat, "Gruen for evaluating linguistic quality of generated text," in *Findings of the Association for Computational Linguistics: EMNLP 2020*, 2020, pp. 94–108.

[105] P. Damodaran, "Parrot: Paraphrase generation for nlu." 2021.

Appendix

A Detailed Experimental Setup

Datasets. We use the following four AIGT datasets for evaluation. To ensure diversity, we consider both RLHF-trained and non-RLHF-trained LLMs. Additionally, we consider three tasks: text completion, question-answer, and instruct-writing.

- *GROVER News [62].* GROVER is a large CLM trained on RealNews [62] dataset using GPT3 as the base model. RealNews dataset is a large corpus (120GB) of news articles from Common Crawl [101]. We contacted the authors to get the RealNews dataset. GROVER has not been trained using RLHF. Instead, it was trained by SFT using attributes including "domain", "data", "authors", and "title" as the prompt and news as the output. So that the user can generate news by inputting the above attributes. We use the largest version of GROVER, named GROVER-Mega, to create the machine-generated dataset. GROVER-Mega has 48 layers and 1.5 billion parameters. We sample RealNews to create the human-generated dataset.
- *HC3 [16].* The Human ChatGPT Comparison Corpus (HC3) is a question-answering dataset collected to study the characteristics of ChatGPT's responses compared to human experts. It aims to understand how close ChatGPT's responses are to those of human experts across various domains, covering open-domain, financial, medical, legal, and psychological areas. HC3 dataset collects questions from these domains and their corresponding human answers, then uses ChatGPT (powered by GPT3.5) to generate the machine-generated dataset.
- *GPA [17].* The GPABenchmark is a cross-disciplinary corpus comprising human-written, GPT-written, GPT-completed, and GPT-polished research paper abstracts. It is introduced with the aim of training and evaluating detectors for identifying LLM-generated academic writing. The authors collect the human-generated dataset from arXiv and

Table 6: Hyperparameters setting of detectors and GradEscape.

	Model	#. Human Texts	#. LLM Texts	#. Epoch	Batch Size	Learning Rate	Scheduler	Max Length	Warmup Ratio
Detector	RoBERTa	6,000	6,000	5	32	5e-5	linear	512	0
	GPT2	6,000	6,000	5	32	5e-5	linear	512	0
	BERT	6,000	6,000	5	32	5e-5	linear	512	0
	MPU	6,000	6,000	5	32	5e-5	linear	512	0
	CheckGPT	6,000	6,000	5	32	2e-4	cosine	512	0
	GLTR	6,000	6,000	-	-	-	-	512	-
Encoder-decoder	BERT-BERT	15,000	30,000	1.8	16	5e-5	linear	512	0.1
Evader	GradEscape	0	9,000	5	64	5e-5	linear	512	0

Table 7: Detector performance without attacks. ER_b represents the baseline evasion rate, which indicates how many machine-generated texts are identified as human-generated.

	RoBERTa		GPT2		BERT	
Dataset	F1	ER_b	F1	ER_b	F1	ER_b
GROVER	0.921	0.016	0.919	0.012	0.906	0.001
HC3	0.995	0.004	0.985	0.000	0.992	0.001
GPA	0.994	0.000	0.992	0.000	0.996	0.002
GPTWiki	0.983	0.000	0.972	0.002	0.987	0.003

use ChatGPT (based on GPT3.5) to constrain the machine-generated dataset. In our paper, we only focus on GPT-written abstracts since they are more harmful than GPT-completed and GPT-polished abstracts.

- *GPTWiki Intro [63].* This dataset contains Wikipedia introductions and GPT-generated introductions for 150k topics. The authors use the title of the Wikipedia page and the first 7 words from the introduction paragraph as prompts. This dataset is generated by a GPT3 model fine-tuned for instruction-following.

Metrics. We employ the following five metrics to estimate evaders' performance. The evasion rate is used to measure attack utility. Since the attacker hopes to constrain the syntactic and semantic similarity and ensure readability, we employ four extra text quality metrics: ROUGE, cos-sim, perplexity, and GRUEN.

- *Evasion Rate.* The evasion rate refers to the fraction of machine-generated texts edited by evaders that are classified as human-written. More specifically given a machine-generated text dataset $\mathcal{T}_M$,

$$\text{Evasion Rate} = \frac{\sum_{i=0}^{|\mathcal{T}_M|} \mathbf{1}\left(D_V\left(\mathcal{F}_\theta\left(\mathcal{T}_{M,i}\right)\right) = y_{\text{human}}\right)}{|\mathcal{T}_M|}, \tag{12}$$

where $\mathbf{1}\{I\}$ denotes the indicator function that is 1 when I is true and 0 when I is false.

- *ROUGE [102].* Since attack utility and text quality are in a trade-off, we need to synthesize the evasion rate with text quality metrics to evaluate evaders. ROUGE is a text quality metric measuring syntactic similarity between texts before and after being edited by evaders. It computes the fraction of text overlap. ROUGE has several variants. In this paper, we employ ROUGE-Lsum as the calculation method.
- *Cos-sim.* This metric is used to compute semantic similarities. We first use Google's *universal sentence encoder* (USE) [103] to generate embeddings for texts before and after editing, and then compute their cosine distance.
- *Perplexity.* Perplexity is a metric to measure text readability. A low perplexity indicates that the model is more confident in its predictions, and the text is more fluent and easier to read. Mathematically, the perplexity of a probabilistic model on a piece of text is defined as:

$$\text{PPL} = 2^{-\frac{1}{N_w}\sum_{i=1}^{N_w}\log_2 P(w_i|w_1,w_2,\ldots,w_{i-1})}. \tag{13}$$

Here, N_w is the number of words; $P\left(w_i \mid w_1, w_2, \ldots, w_{i-1}\right)$ is the conditional probability of word w_i given the preceding word.

- *GRUEN [104].* GRUEN is also used to measure text readability. Unlike perplexity, GRUEN normalizes its values between 0 and 1. GRUEN evaluates text quality from multiple perspectives, including "grammaticality", non-redundancy, discourse, focus, structure, and coherence.

Detectors Setup. Our evaluation includes three basic detectors (RoBERTa, GPT2, and BERT), two advanced detectors (MPU and CheckGPT), and one statistic-based detector (GLTR). The hyperparameters of detectors' training are detailed in Table 6.

- *Basic Detectors.* We fine-tune these models on each dataset using 12,000 training samples and 4,000 test samples with an equal split between LLM-generated and human-generated texts. The training uses a batch size of 32, 5 epochs, a learning rate of 5e-5, and early stopping to prevent overfitting.
- *MPU [11].* This detector is designed to improve detection accuracy on short texts by leveraging multi-scale data augmentation and an additional PU loss. We follow the original paper's configurations for data augmentation and the PU loss. For training arguments, we set the same as those of basic detectors.
- *CheckGPT [17].* This detector freezes the LM and adds a BiLSTM head to enhance training efficiency. We set the learning rate to 2e-4 and use a cosine learning rate scheduler as described in the original paper. Other training arguments are consistent with those for basic detectors.

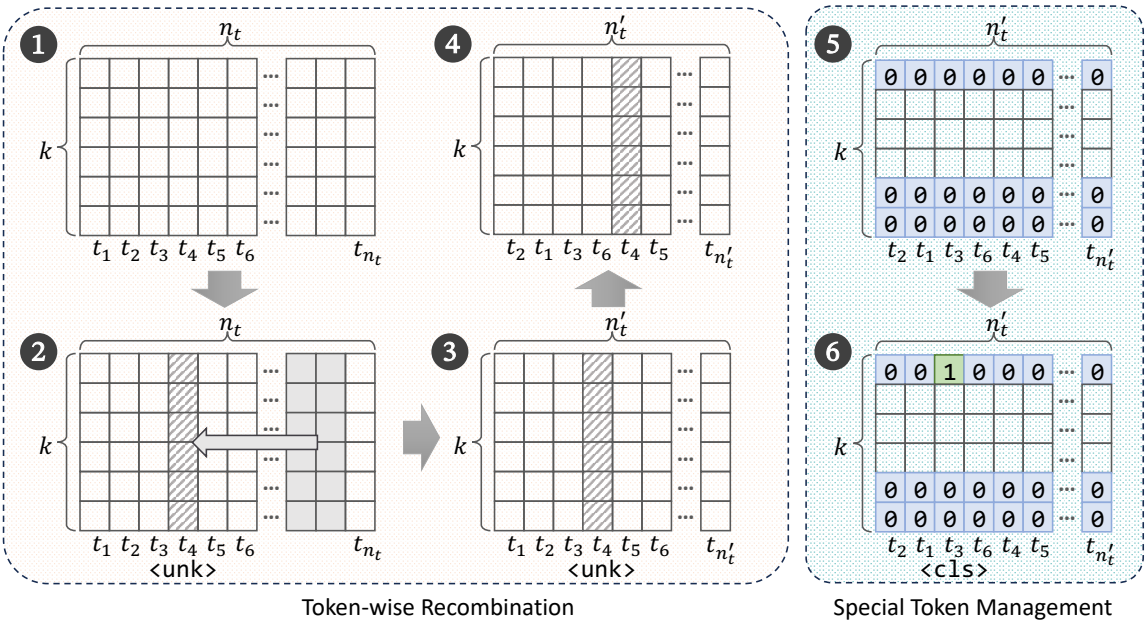


Figure 13: Detailed steps of probability matrix post-processing.

- *GLTR [39].* GLTR collects probability ranks for each word in a text to create word-rank features. We use GPT2-XL to compute these probabilities and train a logistic regression classifier for classification. We use grid search with an L2 penalty to find the optimal model.

Evaders Setup. We select two perturbation-based evaders (RP and DFRFooler) and two paraphrase-based evaders (DIPPER and SentPara) as baselines.

- *RP [23].* This method randomly replaces words with synonyms. We vary the number of replacements in {10, 15, 20, 25, 30, 35, 40}.
- *DFTFooler [23].* DFTFooler replaces words with the highest prediction probabilities. We choose GPT2-XL to compute these probabilities. The number of replacements considered is the same as RP.
- *DIPPER [14].* This evader paraphrases target texts in an end-to-end manner. We use the fine-tuned T5-XXL model released by the authors. We evaluate four LEX-ORD settings: {L20-O0, L40-O0, L60-O0, L60-O60}, which control lexicon and order consistency.
- *SentPara [8].* SentPara uses Parrot [105] toolkit's small paraphrase models to paraphrase texts sentence-by-sentence, then concatenates the results. We use the fine-tuned T5-Base paraphrase and evaluate all five settings controlling Levenshtein distance.

B Probability Matrix Post-processing

In GradEscape, the probability matrix P output by the evader is fed into the text encoder and the detector model. We refer to both the text encoder and the detector model collectively as *downstream models*. Even though the evader model and the downstream model utilize the same type of tokenizers, there may still be structural inconsistencies between P and W_{emb}. To bridge this gap, we employ a method named probability matrix post-processing. This process comprises two phases, as illustrated in Figure 13. The first phrase is *token-wise recombination*, where we handle excess token columns and rearrange the order of other token columns. The second phase is *special token management*, where we mask the evader's special tokens and add the special tokens required by the downstream model.

Token-wise Recombination. The number and order of tokens in the tokenizer of the evader model may differ from those in the downstream model's tokenizer. To address discrepancies in token quantity, we add the values from the excess token columns in $P^{k\times n_t}$ to the `<unk>` column and remove these excess token columns, as shown in step ❷. By doing so, we transform the act of the evader model predicting excess tokens into an increased probability of predicting `<unk>`. To resolve token order inconsistency, we apply the token-remapping algorithm proposed by Bagdasaryan et al. [56] to reorder the token columns. Finally, in step ❹, we get a resized and recorded probability matrix $P^{k\times n'_t}$.

Special Token Management. In step ❺, we mask the rows in $P^{k\times n'_t}$ based on the evader's input token IDs. The input format for BART is structured as: $x =$ `<cls>`$,x_1,x_2,\ldots,x_m,$`<sep>,<pad>`. The output also adheres to this format. In this example, there are three special tokens in the sequence, which means $k = m+3$. We zero out the rows corresponding to special tokens to eliminate their influence in the downstream model's output. In step ❻, we

Table 8: Open model attack results against MPU.

Dataset	Evader	ROUGE	cos-sim	PPL↓	GRUEN	ER
GROVER	GradEscape	0.901	0.960	40.5	0.684	**0.966**
	RP	0.915	**0.986**	50.2	0.584	0.789
	DFTFooler	0.900	0.982	76.6	0.524	0.964
	DIPPER	0.875	0.969	**23.3**	**0.764**	0.963
	SentPara	0.909	0.977	40.8	0.661	0.248
HC3	GradEscape	0.923	**0.986**	19.5	0.664	**0.766**
	RP	0.896	0.977	32.8	0.614	0.171
	DFTFooler	0.895	0.975	46.4	0.589	0.230
	DIPPER	0.901	0.965	**11.1**	**0.799**	0.005
	SentPara	0.906	0.981	16.2	0.748	0.054
GPA	GradEscape	0.902	0.963	31.4	0.614	**0.404**
	RP	0.908	0.976	37.6	0.662	0.161
	DFTFooler	0.905	**0.979**	49.9	0.636	0.173
	DIPPER	0.873	0.959	**15.5**	**0.823**	0.005
	SentPara	0.922	0.979	20.7	0.779	0.012
GPTwiki	GradEscape	0.909	0.975	24.7	0.700	**0.616**
	RP	0.892	0.983	35.7	0.603	0.467
	DFTFooler	0.893	0.981	50.2	0.560	0.599
	DIPPER	0.903	0.963	**15.1**	**0.770**	0.008
	SentPara	0.906	**0.983**	30.2	0.658	0.279

Table 9: Open model attack results against CheckGPT.

Dataset	Evader	ROUGE	cos-sim	PPL↓	GRUEN	ER
GROVER	GradEscape	0.891	0.963	46.3	0.674	**0.972**
	RP	0.915	**0.986**	50.2	0.584	0.061
	DFTFooler	0.900	0.982	76.6	0.524	0.104
	DIPPER	0.875	0.969	**23.3**	**0.764**	0.901
	SentPara	0.909	0.977	40.8	0.661	0.018
HC3	GradEscape	0.903	0.977	21.2	0.687	**0.704**
	RP	0.896	0.977	32.8	0.614	0.562
	DFTFooler	0.895	0.975	46.4	0.589	0.545
	DIPPER	0.901	0.965	**11.1**	**0.799**	0.038
	SentPara	0.906	**0.981**	16.2	0.748	0.152
GPA	GradEscape	0.904	0.971	31.5	0.676	**0.858**
	RP	0.908	0.976	37.6	0.662	0.503
	DFTFooler	0.905	**0.979**	49.9	0.636	0.468
	DIPPER	0.873	0.959	**15.5**	**0.823**	0.024
	SentPara	0.922	0.979	20.7	0.779	0.093
GPTwiki	GradEscape	0.925	0.982	21.6	0.720	**0.534**
	RP	0.892	0.983	35.7	0.603	0.529
	DFTFooler	0.893	0.981	50.2	0.560	0.469
	DIPPER	0.903	0.963	**15.1**	**0.770**	0.037
	SentPara	0.906	**0.983**	30.2	0.658	0.412

set the positions for the essential special tokens required by the downstream model to one.

C Extra Open Model Attack Results

C.1 Evade Basic Detectors

Table 7 shows detectors' performance before being attacked. We can see that all detectors have low baseline evasion rates ($< 2\%$).

Figure 17 and Figure 18 show open model attack results on GPA dataset and GPTWiki dataset, respectively. Table 16 illustrates evasion examples of GradEscape. We observe that GradEscape behaves differently across various datasets. On the GROVER dataset, which is relatively easy to attack, GradEscape altered only a few words, resembling a perturbation-based evader. Conversely, on the HC3 dataset, which is more challenging to attack, GradEscape reorganized the text structure, demonstrating characteristics of a paraphrase-based evader.

C.2 Evade Advanced Detectors

In the previous experiments, the victim detectors were vanilla fine-tuned LMs. Recently, some researchers have proposed improved deep learning-based detectors for AIGT detection. MPU [11] alters the training process by incorporating PU loss, while CheckGPT [17] modifies model architecture by adding a BiLSTM head to the output side of the LM. In this section, we show that our GradEscape can handle both modifications on training loss and model architecture.

Attack Setup. We conduct attacks on MPU and CheckGPT across four datasets and compare GradEscape results with our baseline evaders. We adjust α and β to ensure that GradEscape's ROUGE score is around 0.9. For baseline evaders, we choose the experimental results where the ROUGE score is closest to 0.9.

Table 10: Opaque model attack results against MPU.

Dataset	Attack	ROUGE	cos-sim	PPL↓	GRUEN	ER
GROVER	Query	0.905	0.955	**32.4**	**0.722**	0.974
	Shadow	0.917	**0.981**	33.9	0.720	**0.996**
	Open	0.901	0.960	40.5	0.684	0.966
HC3	Query	0.920	0.960	24.0	**0.715**	0.366
	Shadow	0.902	0.967	21.4	0.695	0.300
	Open	0.923	**0.986**	**19.5**	0.664	**0.766**
GPA	Query	0.910	**0.966**	32.7	**0.643**	0.422
	Shadow	0.886	0.927	37.0	0.590	**0.430**
	Open	0.902	0.963	**31.4**	0.614	0.404
GPTwiki	Query	0.921	0.975	29.6	0.668	0.568
	Shadow	0.916	**0.980**	25.4	0.666	0.472
	Open	0.909	0.975	**24.7**	**0.700**	**0.616**

Results. The open attack results against MPU and CheckGPT are shown in Table 8 and Table 9, respectively. We find that GradEscape achieves higher evasion rates than all baselines against both MPU and CheckGPT. Although DIPPER has the highest text quality among all evaders, it exhibits a low evasion rate. Note that DIPPER is an 11B LLM; both its training and inference overhead are very high. The text quality of GradEscape is significantly better than that of perturbation-based evaders and also slightly better than SentPara, which is similar in size to GradEscape.

D Extra Opaque Model Attack Results

D.1 Evade Advanced Detectors

In this section, we demonstrate the effectiveness of GradEscape against advanced detectors in opaque model at-

Table 11: Opaque model attack results against CheckGPT.

Dataset	Attack	ROUGE	cos-sim	PPL↓	GRUEN	ER
GROVER	Query	0.897	0.950	54.4	0.654	**1.000**
	Shadow	0.892	0.956	**43.6**	**0.684**	**1.000**
	Open	0.891	**0.963**	46.3	0.674	0.972
HC3	Query	0.891	0.972	18.8	**0.688**	**1.000**
	Shadow	0.900	**0.983**	**18.4**	0.649	**1.000**
	Open	0.903	0.977	21.2	0.687	0.704
GPA	Query	0.915	0.946	34.4	**0.727**	**1.000**
	Shadow	0.897	0.952	36.1	0.683	**1.000**
	Open	0.904	**0.971**	**31.5**	0.676	0.858
GPTwiki	Query	0.905	0.960	27.0	0.648	**1.000**
	Shadow	0.896	0.945	30.0	0.619	**1.000**
	Open	0.925	**0.982**	**21.6**	**0.720**	0.534

Table 12: GradEscape's query attack results against GLTR.

Dataset	ROUGE	cos-sim	PPL↓	GRUEN	ER_b	ER
GROVER	0.911	0.973	32.5	0.721	0.263	0.558
HC3	0.890	0.979	19.3	0.667	0.047	0.724
GPA	0.934	0.973	36.0	0.613	0.132	0.656
GPTWiki	0.923	0.959	34.0	0.608	0.031	0.526

tack scenarios. We utilize the same attack setup as described in Appendix C.2. We execute query and shadow dataset attacks against MPU and CheckGPT across four datasets. For the query attacks, we choose 4000 samples as the query dataset. The experimental outcomes for MPU and CheckGPT are presented in Table 10 and Table 11, respectively. We can find that opaque model attacks achieve comparable or even higher evasion rates than open model attacks with similar text quality. The evasion rates of query and shadow dataset attacks against CheckGPT are higher than those of open model attacks, primarily because CheckGPT freezes most of the model parameters, updating only the parameters of the classification head, which helps prevent overfitting of the surrogate model.

D.2 Evade Statistic-based Detectors

Apart from deep learning-based detectors, statistic-based detectors are also used for post-hoc detection. Since GradEscape updates the evader based on the detector's gradients, a concern with GradEscape is whether it can evade statistic-based detectors. In this section, we demonstrate that GradEscape can also tackle statistic-based detectors by constructing a deep learning-based surrogate model.

Attack Setup. We select GLTR [39] as the victim detector since it performs the best among existing statistic-based detectors [38]. GLTR leverages the insight that decoding strategies tend to favor tokens assigned high probabilities by the LM. Utilizing a backend LM, GLTR extracts features based on the number of tokens within the Top-10, Top-100, and Top-1000 ranks as determined by the token probability distributions. We use GPT2-XL as the backend LM. We choose 2000 samples to form the query dataset, and then fine-tune a GPT2-Base model as the surrogate model using the results of these queries. As in previous experiments, we adjust α and β to target a ROUGE score around 0.9.

Table 13: Performance of real-world detectors without attacks. Datasets in gray background are selected to attack.

Detector	Dataset	Precision	Recall	F1	Accuracy
Sapling	GROVER	0.85	0.28	0.43	0.62
	HC3	0.84	0.99	0.91	0.91
	GPA	0.61	1.00	0.76	0.70
	GPTWiki	0.72	0.94	0.82	0.78
Scribbr	GROVER	1.00	0.14	0.25	0.64
	HC3	1.00	0.97	0.99	0.98
	GPA	1.00	0.99	0.99	0.99
	GPTWiki	0.93	0.14	0.25	0.57

Results. The results, as presented in Table 12, indicate that GradEscape boosts the average evasion rate by 0.5 compared to the pre-attack baseline. Notably, even for datasets like HC3 and GPTWiki, where GLTR demonstrates strong performance, GradEscape achieves evasion rates exceeding 50%. This suggests that DNN models are capable of imitating the underlying patterns of statistic-based detectors through querying.

D.3 Evade Heterogeneous Detectors

We compare GradEscape with four baselines under scenarios where the training datasets of the evader and the detector vary. Evasion rates are measured against three distinct detectors: RobERTa, MPU, and GLTR. Figure 14 presents the experimental results. Out of 24 comparison groups, GradEscape achieves the highest evasion rate in 16 cases. Notably, when attacking the more challenging HC3 dataset GradEscape ranks first in 9 out of 12 comparisons. RP and DFTFooler are adaptive attacks for GLTR, and thus achieve high evasions against GLTR. Even so, GradEscape outperforms them when attacking GLTR in many cases.

E Extra Real-world Experimental Results

The detection performance of Sapling and Scribbr without attacks is shown in Table 13. We choose two datasets for each real-world detector to conduct our GradEscape attack. Figure 19 illustrates tokenizer inference attack observations against Sapling. Figure 20 and Figure 21 show attack examples on HC3 dataset against Sapling and Scribbr, respectively.

Table 14 presents the comparison with our baselines. For each baseline, we adjust the knob so that the ROUGE scores are similar to those of the texts modified by GradEscape. As shown, GradEscape achieves state-of-the-art evasion rates on both Sapling and Scribbr. Although SentPara shows a greater $\overline{DC}$ decrease when attacking Scribbr with GPA dataset, GradEscape's DC values are concentrated around

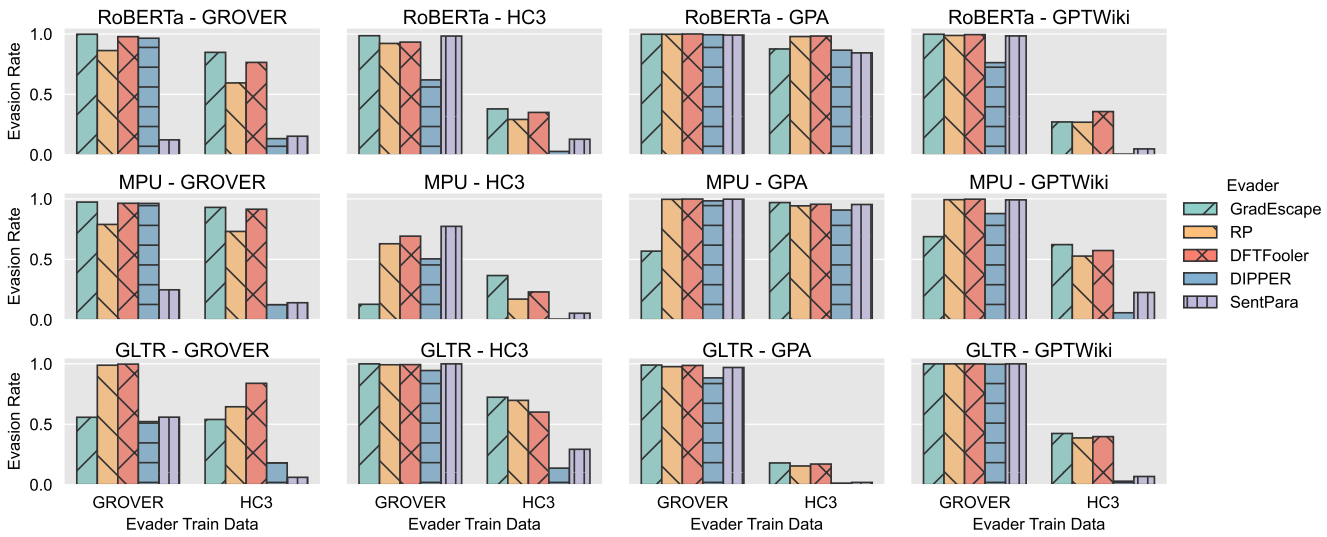


Figure 14: Transferability comparison with baselines under heterogeneous detector training. Each title "DETECTOR - DATASET" indicates that the victim model is a DETECTOR trained on the DATASET.

GPT4 Annotation Prompt

Here are two texts. The first one is the original text, and the second one is the modified text.

Original text: {`ORIGINAL_TEXT`}

Modified text: {`MODIFIED_TEXT`}

Please tell me whether the modification changes the meaning of the original text or would confuse a reader. Please answer with "major," "minor," or "no change." "Major" means the modification causes key information to be missing, which could confuse readers; "minor" means the expression may be changed but can be correctly understood; and "no change" means the modification has little impact on the semantics.

Figure 15: The GPT4 annotation prompt for evaluating semantic consistency.

0.4, while SentPara's DC values are concentrated around 0.1. GradEscape affects more examples and attains a higher evasion rate.

F Semantic Consistency

We assess the semantic consistency between modified and original texts by using both GPT-4-based annotation and human evaluation methods.

GPT4 Annotation. In particular, we employ `gpt-4o-2024-08-06` to determine whether the modifications made by evaders result in major semantic changes. The annotation prompt is shown in Figure 15, and the percentage of major changes is reported in Table 15. For each evader on each dataset, we randomly sample 300 text pairs for annotation. GradEscape achieves semantic consistency levels between those of perturbation-based and paraphrase-based methods. DIPPER, which utilizes an LLM for end-to-end paraphrasing, maintains high semantic consistency. In contrast, perturbation-based methods are assessed as causing more major changes, especially on shorter datasets such as HC3 and GPTWiki.

Human Evaluation. We also develop a dedicated text semantic consistency rating platform (illustrated in Figure 22) to facilitate human evaluation. We hired 7 crowdworkers to rate semantic changes using this website. Annotators are presented with pairs of original and modified texts and are asked to assign a score from 1 to 5, where a higher score indicates a greater semantic change. We set a mandatory reading time of 25 seconds per rating, and annotators can choose to skip uncertain pairs. For each evader on each dataset, we randomly choose 150 text pairs for human evaluation. We pay each annotation USD 0.21 (XE rate as of 2025/05/19), with an estimated average hourly wage of USD 8.4 (40 pairs per hour), exceeding the minimum hourly wage in Hangzhou, China (USD 3.34, XE rate as of 2025/05/19). This project was thoroughly reviewed and approved by the Science and Technology Ethics Committee of Zhejiang University, serving as our IRB. To ensure rating consistency, we provided annotators with sample text pairs and reference scores from 1 to 5 annotated by authors. The experimental results are shown in Figure 16. Human evaluation further confirms that GradEscape's semantic consistency lies between perturbation-based and paraphrase-based methods. SentPara performs worse in human evaluation, possibly because paraphrasing individual sentences may cause the combined text to become incoherent. Interestingly, human annotators are generally more tolerant of perturbation-

Table 14: Real-world comparison with baselines.

Detector	Train Data	Evasion Rate					$\overline{\text{DC}}\downarrow$				
		RP	DFTFooler	DIPPER	SentPara	GradEscape	RP	DFTFooler	DIPPER	SentPara	GradEscape
Sapling	HC3	0.080	0.121	0.397	0.251	**0.448**	0.913	0.878	0.609	0.745	**0.550**
	GPTWiki	0.392	0.462	0.332	0.266	**0.578**	0.602	0.535	0.669	0.729	**0.422**
Scribbr	HC3	0.669	0.419	0.495	0.616	**0.785**	0.467	0.622	0.559	0.481	**0.378**
	GPA	0.520	0.354	0.520	0.564	**0.658**	0.548	0.681	0.506	**0.471**	0.496

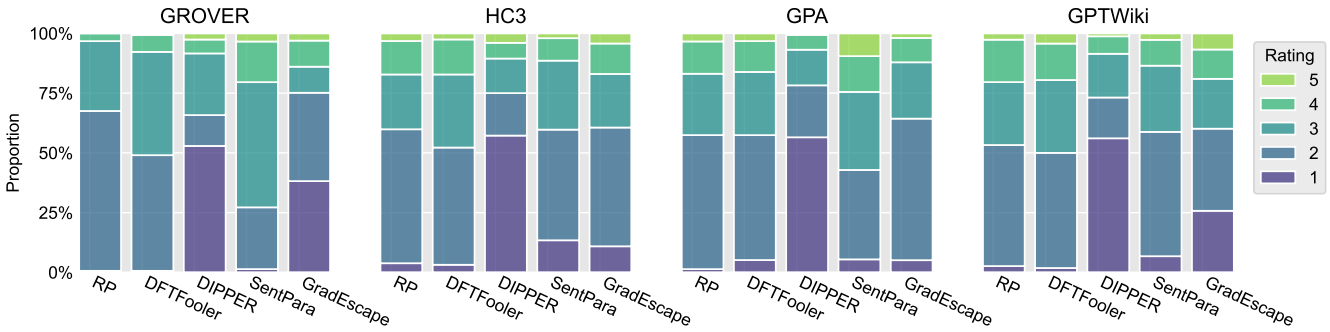


Figure 16: Human ratings of semantic changes. Higher scores indicate greater semantic distortion.

Table 15: Percentage of GPT4-annotated major semantic changes. Lower is better.

Evader	**GROVER**	**HC3**	**GPA**	**GPTWiki**
RP	13%	30%	31%	41%
DFTFooler	17%	47%	52%	44%
DIPPER	14%	11%	12%	12%
SentPara	25%	16%	12%	13%
GradEscape	7%	29%	33%	21%

based evaders, suggesting that humans are less sensitive to changes involving the replacement of individual words.

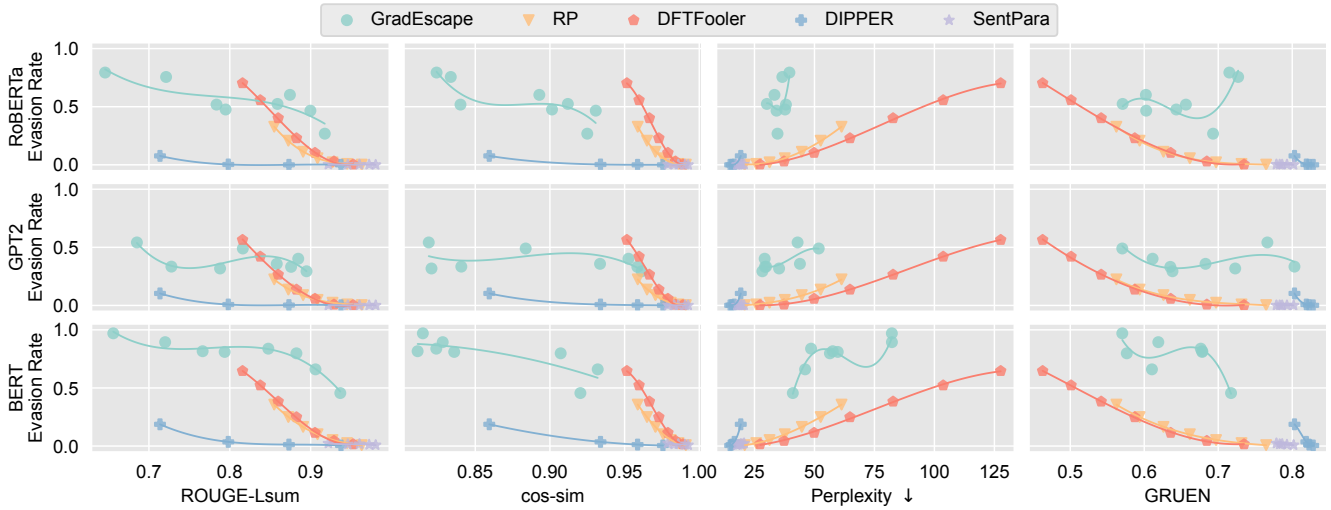


Figure 17: Evasion rates on GPA dataset.

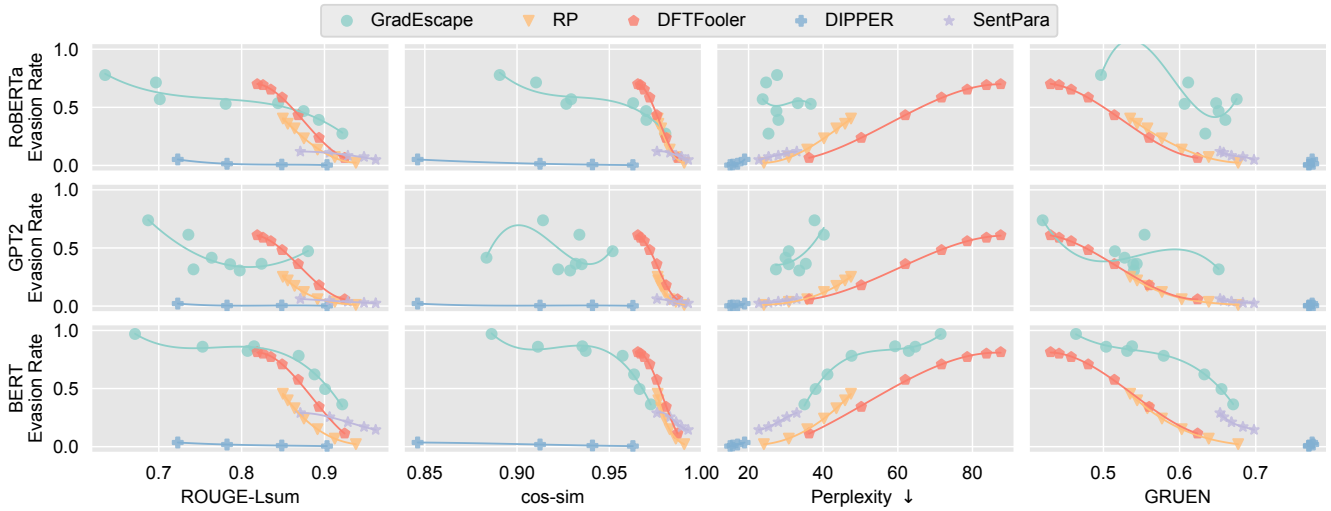
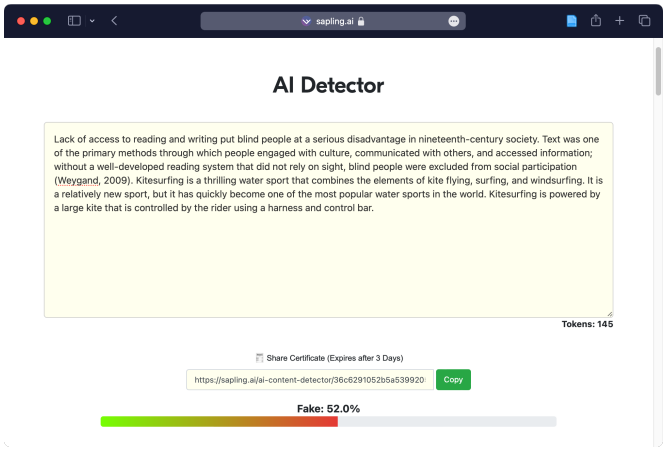


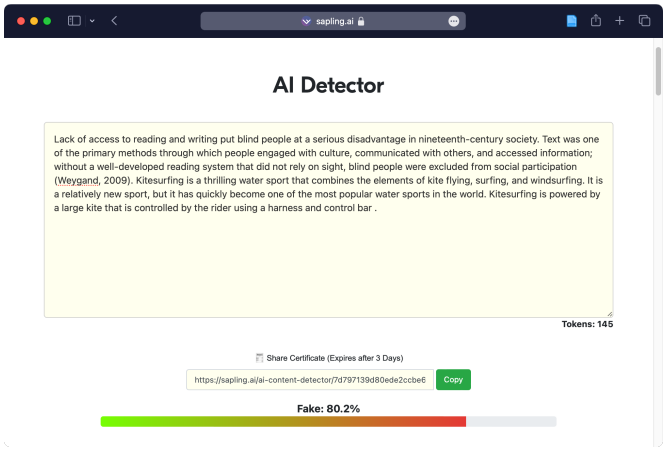
Figure 18: Evasion rates on GPTWiki dataset.

Table 16: Example outputs produced by GradEscape. The differences are highlighted in red. DC = Detection Confidence, which indicates the probability of being AI-generated predicted by the detector.

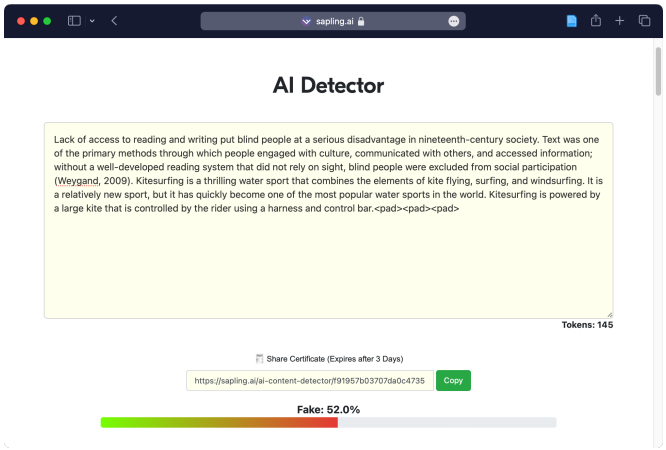
Dataset	Input	Output
GROVER	Print Article TOWN OF BELOIT - Two Beloit Memorial swimming teams dominated the 52nd annual Rock County meet Saturday at Ravenswood Pool, winning the boys' division and the girls' division. In the girls' competition, Beloit Memorial won all five events. The Cadets were led by senior Meredith Schultz, who won the 100 free and 100 breaststroke events. Marissa Cook (200 free) and Madison Wilson (200 individual medley and 100 backstroke) also were winners for Beloit Memorial. Emma Sharkey finished first in the 200 free and won the 500 free. The Beloit boys finished with 596 points to win the team title. Logan Borowicz was a double winner for the Cadets. He won the 200 individual medley and 100 fly. John Ure won the 50 free and 100 free. Michael Aigner finished first in the 100 free. Boys swimming Rock County meet - Beloit Memorial 596, Arrowhead 336, Rockford Christian 294, Whitewater 202, Independence 137, Hartford 107, Calumet 50, Beloit West 36 Boys relay - 1. Beloit Christian, 3:47.08; 2. Arrowhead, 3:47.28; 3. Rockford Christian, 3:50.73; 4. Rockford Christian, 3:51.28	Print Article TOWN OF BELOIT - Two Beloit Memorial swimming teams dominated the 52nd annual Rock County meet Saturday at Ravenswood Pool, winning the boys' division and the girls' division. In the girls competition, Beloit ® Memorial won all five events. The Cadets were led by senior Meredith Schultz, who won the 100 free and 100 breaststroke events. Marissa Cook (200 free) and Madison Wilson (200 individual medley and 100 backstroke) also were winners for Beloin Memorial. Emma Sharkey finished first in the 200 free and won the 500 free. The Beloit boys finished with 596 points to win the team title. Logan Borowicz was a double winner for the Cadets. He won the 200 individual medLEY and 100 free. John Ure won the 50 free and 100 free. Michael Aigner finished first IN the 100 return. Boys swimming Rock County meet - Beloit memorial 596, Arrowhead 336, Rockford Christian 294, Whitewater 202, Independence 137, Hartford 107, Calumet 50, Beloin West 36 Boys relay - 1. Beloit Christian, 3:47.08; 2. Arrowhead, 3.13.73; 3. Rockford Central, 3::50.73? 4. Rockf Christian, 4:51.28
	DC: 0.842, PPL: 32.1, GRUEN: 0.475	*ROUGE: 0.948, cos-sim: 0.996, DC: 0.003, PPL: 56.2, GRUEN: 0.599*
HC3	When you crack your knuckles or other joints, you are essentially releasing gas bubbles that have built up in the fluid that surrounds your joints. This can create a popping sound. Cracking your knuckles on a regular basis is not likely to cause any long-term complications or damage to your joints. However, it is possible to overdo it and cause temporary discomfort or swelling. So, it's probably a good idea to limit the amount of knuckle cracking you do.	When you crack your knuckles or other joints, you are essentially releasing gas bubbles. Apparently, this can create popping sound. Cracking your knuckle cracking on a regular basis is not likely to cause long-term complications. However, damage to your joints is possible to cause temporary discomfort. Thankfully, this effect is probably a good idea to limit the amount of knuckles cracking.
	DC: 0.999, PPL: 13.8, GRUEN: 0.865	*ROUGE: 0.741, cos-sim: 0.967, DC: 0.009, PPL: 31.8, GRUEN: 0.773*
GPA	This paper examines the politics of adversarial machine learning, which is the use of machine learning techniques to attack and defend against other machine learning algorithms. We argue that the political implications of adversarial machine learning are important and underappreciated. Adversarial attacks can be used to undermine the performance of machine learning algorithms, potentially leading to harmful or biased decisions. Adversarial defenses, in turn, can be used to protect against such attacks, but may also reinforce existing biases or inequalities. We explore the ways in which adversarial machine learning interacts with broader political and social dynamics, including the distribution of power, the framing of security and risk, and the regulation of emerging technologies. We conclude by discussing the implications of our analysis for future research and policy initiatives.	This is a paper examines the politics of adversarial machine learning, which is the use of machine learning techniques to attack and defend against other machine learning algorithms. To this paper, we argue that the political implications and underappreciated. Adversarial attacks can be used to undermine the performance of machinelearning algorithms, potentially leading to harmful or biased decisions. As a defense, in turn, to protect against such attacks, but may also reinforce existing biases or inequalities. As part of our analysis for future research and policy initiatives. To explore the ways in which adversarial Machine learning interacts with broader political and social dynamics, including the distribution of power, the framing of security and risk, and the regulation of emerging technologies. To that end, we conclude by discussing the implications of our analyses.
	DC: 1.000, PPL: 12.5, GRUEN: 0.886	*ROUGE: 0.892, cos-sim: 0.958, DC: 0.390, PPL: 24.9, GRUEN: 0.795*
GPTWiki	Radu Rosetti (Francized Rodolphe Rosetti; September 14, 1881 - November 25, 1957) was a French painter and printmaker. He was associated with the Fauves and Cubists movements. Born in Bordeaux, France, to a family of wine merchants, Rosetti studied at the Ecole des Beaux-Arts in Paris. In 1905 he exhibited his first painting at the Salon d'Automne. The following year, he joined the Section d'Or group, whose members were associated with the Fauves movement. In 1907 he exhibited with the group at the Salon des Independants. In 1908, Rosetti exhibited at the Salon des Realites Nouvelles. He joined the Societe Nationale de Peinture et de Sculpture in 1909, and became a member of the Academie des Beaux-Arts in 1917. He lived in Montmartre from 1933 until his death.	Radu Rosetti (Francized Rodolphe Rosetti; September 14, 1881 - November 25, 1957) was a French painter and printmaker, and he was associated with the Fauves and Cubists movements, and of course, Rosetti studied at the Ecole des Beaux-Arts in Paris, and became a member of the Salon d'Automne at 1905. The following year, he joined the Section d'Or group, whose members were associated with both the fauves movement. In 1907 he exhibited with the group at the Salon des Independants, and in 1908 he exhibited at Salon des Realites Nouvelles, and joined the Societe Nationale de Peinture et de Sculpture in 1909. He joined the Academie Des Beaux Arts in 1917. He lived in Montmartre from 1933 until his death.
	DC: 0.999, PPL: 14.6, GRUEN: 0.761	*ROUGE: 0.901, cos-sim: 0.982, DC: 0.266, PPL: 22.8, GRUEN: 0.749*



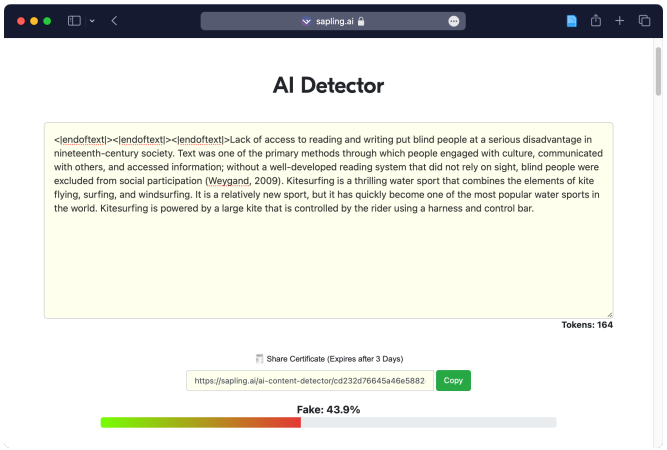
(a) Original text.



(b) Inserting a space before the last period.

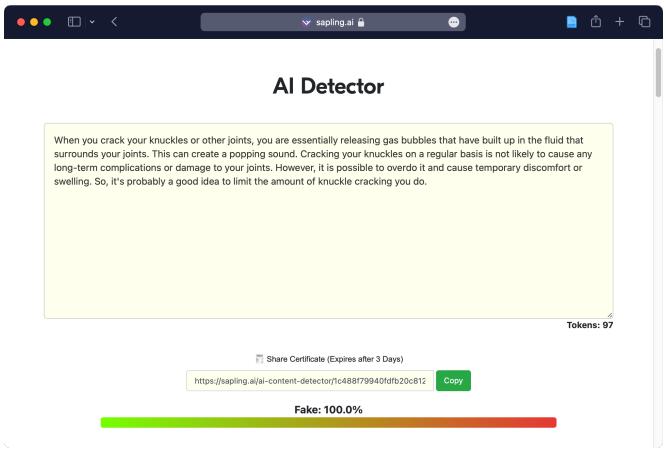


(c) Appending `<pad>`s to the text.

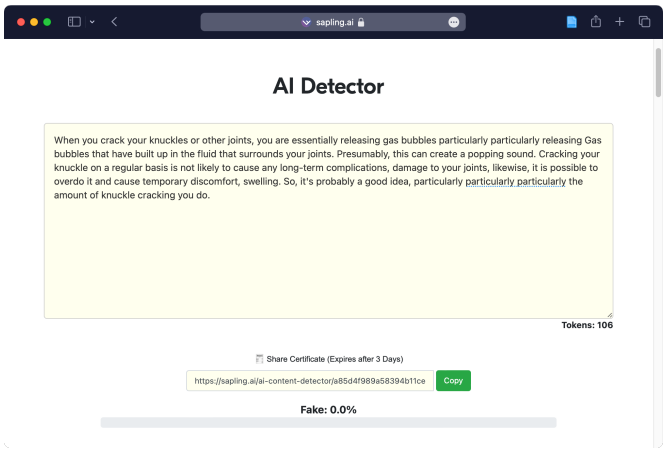


(d) Prepending `<|endoftext|>`s to the text.

Figure 19: Tokenizer inference attack against Sapling detector.

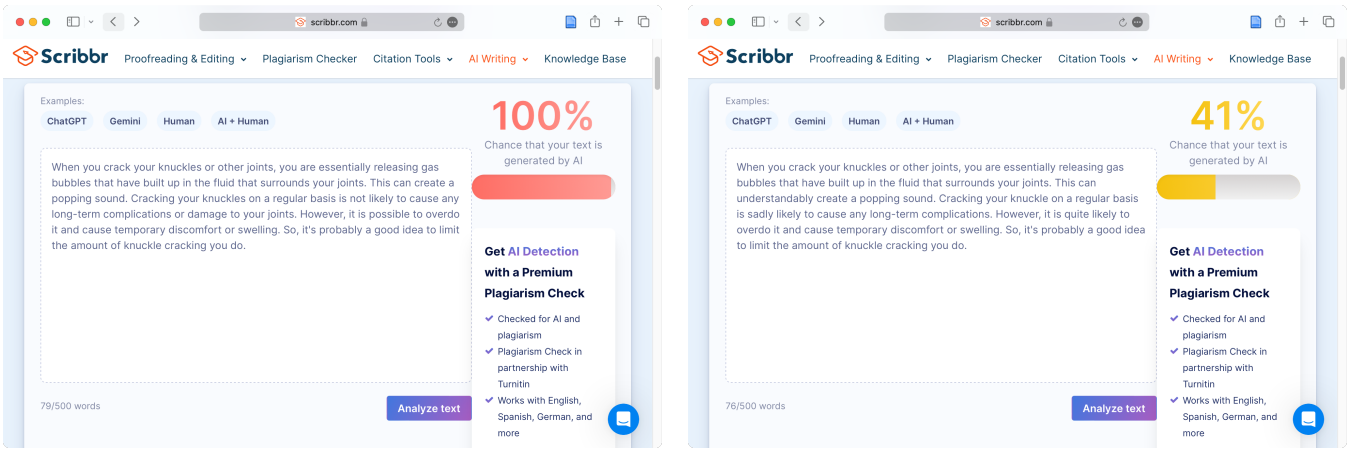


(a) Original text.



(b) Paraphrased text.

Figure 20: An attack example of HC3 dataset against Sapling detector.



(a) Original text. (b) Phraphrased text.

Figure 21: An attack example of HC3 dataset against Scribbr detector.

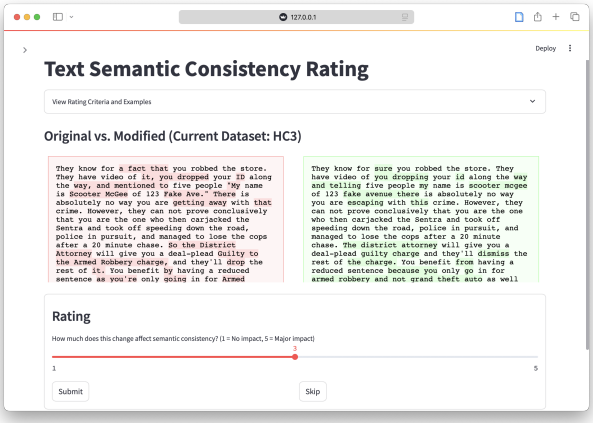


Figure 22: Webpage of text semantic consistency rating platform.

G Artifact Appendix

G.1 Abstract

We proposed GradEscape, the first gradient-based evader for attacking AI-generated text (AIGT) detectors. GradEscape overcomes the undifferentiable computation problem, caused by the discrete nature of text, by constructing weighted embeddings for the detector input. It then updates the evader model parameters using feedback from victim detectors, achieving high evasion rates with minimal text modifications.

This artifact is used to evaluate GradEscape against a range of detectors. It includes code for reproducing GradEscape as well as three baseline evaders. We provide a Python library that makes it easy to reproduce existing detectors and evaders. The artifact also contains trained evader models and scripts for replicating our experiments on two real-world AIGT detectors, namely Sapling and Scribbr. To help mitigate the risks posed by AIGT evaders, we showcase that our proposed active paraphrase defense can effectively reduce evasion rates.

G.2 Description & Requirements

We package the artifact into two separate files: `GradEscape.zip` and `Usenix-AE.zip`. `GradEscape.zip` contains the source code; while `Usenix-AE.zip` contains evaluation datasets and trained models. This section describes the minimal hardware and software requirements needed to run the artifact.

G.2.1 Security, privacy, and ethical concerns

Our attacks target exclusively at the target AIGT detector. Therefore, there is no security risk for evaluators. All benchmark datasets are sourced from open corpora, so there are no privacy concerns. Regarding real-world experiments, Sapling and Scribbr have updated their services with stronger models. Therefore, our artifact poses no significant risk to these two online platforms.

G.2.2 How to access

Our artifact is available through Zenodo. The artifact can be accessed at https://doi.org/10.5281/zenodo.15586856.

G.2.3 Hardware dependencies

Two Nvidia RTX A6000 GPUs are the minimum GPU requirement to run the artifact. We recommend at least a 20-core CPU, 32 GB RAM, and 256 GB of free disk space.

G.2.4 Software dependencies

- **OS:** Ubuntu 20.04+. A macOS machine is needed to open Scribbr webarchive file. If you encounter a security warning when opening the webarchive file, please go to Settings, Privacy & Security and click open anyway.
- **Package Manager:** Conda.
- **API Key:** A Sapling API key is required to run Sapling experiments. Conducting these experiments entails money costs.

G.2.5 Benchmarks

Our evaluation requires four datasets: GROVER, HC3, GPA, and GPTWiki. GROVER and GPA are provided in `Usenix-AE.zip`. Our code will automatically download and manage HC3 and GPTWiki. Some pre-trained victim detectors are also included in `Usenix-AE.zip`.

G.3 Set-up

The setup utilizes Conda for environment management.

G.3.1 Installation

Install Main Environment. Download `GradEscape.zip` and `Usenix-AE.zip`. Unzip and place them in the same directory. Then, use the following commands to create an environment:

```
conda create -n ge python=3.10
conda activate ge
cd GradEscape
./install.sh
cp src/AIGT/.config.yaml src/AIGT/config.yaml
```

Generate Word Similarity Matrix. Perturbation-based evaders rely on a word similarity matrix to select synonyms.

```
cd Usenix-AE
git clone
↪ https://github.com/nmrksic/counter-fitting.git
cd counter-fitting/word_vectors/
unzip counter-fitted-vectors.txt.zip
python ../../../GradEscape/tools/comp_cos_sim_mat.py
↪ counter-fitted-vectors.txt
```

Then edit `config.yaml` to set the correct data_dir and counter_fitting_path.

Create vLLM Environment. We need vLLM for fast paraphrasing. Since vLLM has complex dependencies, we create a new environment specifically for vLLM.

```
conda create -n vllm python=3.10
conda activate vllm
cd GradEscape
pip install -r paraphrase_requirements.txt
```

Our artifact mainly runs on `ge`; `vllm` is only used for paraphrase defense.

G.3.2 Basic Test

We provide a simple test that trains a detector using `AIGT`.

```
python basic_test.py
```

If you see "Test finished successfully!", it means the installation was successful.

G.4 Evaluation workflow

G.4.1 Major Claims

(C1): GradEscape achieves higher evasion rates than state-of-the-art evaders under the same text quality requirement. This is proven by the experiment (E1) described in Section 6.2 whose results are illustrated in Figure 3 and Figure 4.

(C2): GradEscape is effective against real-world commercial AIGT detectors. This is proven by experiment (E2) described in Section 6.6 whose results are reported in Table 4 and illustrated in Figure 20 and Figure 21.

(C3): Our proposed potential defense can reduce evasion rates below 0.2 against multiple evaders. This is proven by experiment (E3) described in Section 7 whose results are illustrated in Figure 11.

G.4.2 Experiments

(E1): *[Verify Evasion Effectiveness] [5 human-minutes + 1 compute-hour]:*
How to: Navigate to `examples` directory; activate `ge` environment; execute evader training script in `scripts`. The evasion rate and text quality metrics will be printed on the shell.
Preparation: None.
Execution: Navigate to `examples` directory and activate `ge`:

```
cd examples
conda activate ge
```

Then execute the evader training script:

```
./scripts/train_evader_roberta_grover.sh
```

Results: After the program finishes running, the evasion rate and text quality metrics (perplexity, cosine similarity, GRUEN, and ROUGE) will be printed to the terminal. Evaluators can compare these results with the first row of Figure 3. For ease of reference, we provide details of text quality metrics in Table 17.

(E2): *[Real-world Case Studies] [5 human-minutes + 30 compute-minutes]:*
How to: Set your Sapling API key environment parameter; run the two real-world case study Jupyter Notebooks. Evaluators may open `Scribbr.webarchive` to verify Scribbr results.
Preparation: A Sapling API key and a macOS machine for Scribbr verification.
Execution: Set your Sapling API key:

```
export SAPLING_API_KEY=<your_api_key>
```

Run `real_world_demo_sapling.ipynb` and `real_world_demo_scribbr.ipynb`. The execution environment is `ge`.
Results: The Sapling results will be printed in its Jupyter Notebook. Evaluators can compare the printed results with Figure 20 and Table 4. Verifying Scribbr results requires copying the output into the website rendered by `Scribbr.webarchive`. The Scribbr results should be the same as Figure 21.

Table 17: Text quality metrics summary. Expected refers to typical value ranges on GROVER; Print % indicates whether the metric is displayed as a percentage.

Metric	Range	Expected	Larger Better	Print %
Perplexity	$[1, +\infty)$	$[10, 40]$	No	No
Cos-sim	$[-1, 1]$	$[0.95, 1.00]$	Yes	No
GRUEN	$[0, 1]$	$[0.6, 0.8]$	Yes	No
ROUGE	$[0, 1]$	$[0.8, 1.0]$	Yes	Yes

(E3): *[Paraphrase Defense Experiment] [5 human-minutes + 2 compute-hours]:*
How to: Navigate to `examples` directory. First, use `vllm` environment to run `paraphrase_defense_grover.sh`. Then, activate `ge` and run `eval_paraphrase_defense_grover.sh`. It will generate a figure in the current directory named `paraphrase_defense_grover.pdf`.
Preparation: None.
Execution: Navigate to `examples`:

```
cd examples
```

Run paraphrase:

```
conda activate vllm
./scripts/paraphrase_defense_grover.sh
```

Train a new detector and evaluate the defense:

```
conda activate ge
./scripts/eval_paraphrase_defense_grover.sh
```

Results: The program will generate a figure named `paraphrase_defense_grover.pdf` in `examples/`. This figure illustrates evasion rates before and after applying our defense. Evaluators can compare the generated figure with Figure 11 in our paper.

G.5 Notes on Reusability

We implement `AIGT` in a reusable way. All configurations, including datasets, detectors, and evaders, are extracted in `arguments.py`. Followers can replicate existing AIGT detectors and evaders with little effort. The user interface of detectors and evaders is designed in a unified and HuggingFace-like way. Followers can also write their own detectors and evaders in `detectors/` and `evaders/`, respectively.

G.6 Version

Based on the LaTeX template for Artifact Evaluation V20231005. Submission, reviewing and badging methodology followed for the evaluation of this artifact can be found at https://secartifacts.github.io/usenixsec2025/.