Adversarial Attacks on AI-Generated Text Detection Models: A Token Probability-Based Approach Using Embeddings

Ahmed Khalid & Lei Jiao
Department of ICT
University of Agder
Grimstad, Norway
{ahmed.k.kadhim, lei.jiao}@uia.no

Rishad Shafik
School of Engineering
Newcastle University
Newcastle upon Tyne, UK
rishad.shafik@newcastle.ac.uk

Ole-Christoffer Granmo
Department of ICT
University of Agder
Grimstad, Norway
ole.granmo@uia.no

arXiv:2501.18998v2 [cs.CL] 10 Apr 2025

Abstract

In recent years, text generation tools utilizing Artificial Intelligence (AI) have occasionally been misused across various domains, such as generating student reports or creative writings. This issue prompts plagiarism detection services to enhance their capabilities in identifying AI-generated content. Adversarial attacks are often used to test the robustness of AI-text generated detectors. This work proposes a novel textual adversarial attack on the detection models such as Fast-DetectGPT. The method employs embedding models for data perturbation, aiming at reconstructing the AI generated texts to reduce the likelihood of detection of the true origin of the texts. Specifically, we employ different embedding techniques, including the Tsetlin Machine (TM), an interpretable approach in machine learning for this purpose. By combining synonyms and embedding similarity vectors, we demonstrates the state-of-the-art reduction in detection scores against Fast-DetectGPT. Particularly, in the XSum dataset, the detection score decreased from 0.4431 to 0.2744 AUROC, and in the SQuAD dataset, it dropped from 0.5068 to 0.3532 AUROC.

1 Introduction

The responsibility of integrating into the scientific research and higher education community entails adhering to numerous behaviors and principles essential to safeguarding the integrity of educational and scientific progress. Consequently, the utilization of tools such as text editors or language enhancement applications must align with sound practices and uphold the ethical standards of scientific research and education (Lund et al., 2023; Foltynek et al., 2023). Despite the substantial advancements in AI models, particularly within Natural Language Processing (NLP) applications, there remains ongoing debate about the appropriate use of these tools for text generation (Leidner & Plachouras, 2017; Šuster et al., 2017). This issue could significantly impact the integrity of the academic domain (Tauginienė et al., 2018). Large Language Model (LLM) such as Generative Pre-trained Transformer (GPT) (Brown et al., 2020; OpenAI, 2022; 2023), BERT (Devlin et al., 2018), and

XLNet (Yang et al., 2020) have gained widespread acceptance among users, demonstrating such high efficiency that distinguishing between human-generated and machine-generated content has become increasingly challenging (Shahid et al., 2022; Ippolito et al., 2020).

In response to this surge in the use of AI techniques for text generation, various detection methods have been developed to ascertain the origin of text (Solaiman et al., 2019; Fagni et al., 2021; Mitrović et al., 2023; Gehrmann et al., 2019; Mitchell et al., 2023; Su et al., 2023). These methods are categorized as supervised and unsupervised. Unsupervised methods are notable for their versatility in text detection, as they are capable of handling diverse text domains due to the nature of their pretraining (Gehrmann et al., 2019; Mitchell et al., 2023). These tools calculate probabilities and distribute them throughout the text. Under the zero-shot framework, it is presumed that AI-generated text exhibits a higher degree of probability variation compared to human-authored text, which can demonstrate both stability and variation. To assess such changes, additional text generation is required, significantly increasing execution time. The latest approach, FastDetect (Bao et al., 2024), hypothesizes that AI-generated words are produced based on decisions made by the generative model, adhering to specific generation probabilities distinct from human word choices. For humans, word selection involves multiple influencing factors, rendering the output more personal and less statistically general.

The majority of generated texts are fundamentally based on embedding principles in the initial stages of any Large Language Models (LLMs), where a dense vector in space is extracted for each word to encapsulate the contextual information present in the training dataset (Goldberg & Levy, 2014; Pennington et al., 2014). The embedding maps words or tokens into a high-dimensional continuous space, which the model employs to facilitate various stages of text generation. This embedding, coupled with additional layers and algorithms used in LLMs (Vaswani et al., 2017), informs the model's predictions for subsequent words. Understanding this generative pattern enables the identification of AI-suggested content versus natural, human-like predictions, which are not strictly bound by patterns.

Building on this foundation, in this paper, we propose an adversarial attack approach in which the embeddings are reverse engineered to exploit detection models by assigning low probability rates to predicted subsequent words, thereby lowering the overall text score as assessed by the detection systems.

Among the various embedding schemes considered in this work, we place particular emphasis on the TM-based approach. The TM is an emerging machine learning technique that has demonstrated notable success in various applications (Granmo, 2018; Yadav et al., 2021; Berge et al., 2019; Abeyrathna et al., 2021; Maheshwari et al., 2023), including NLP and computer vision. One architecture employed within the TM framework is the Tsetlin Machine Auto-Encoder (TM-AE), which is used in NLP tasks to generate word embeddings (Bhattarai et al., 2024). The TM is distinguished by its interpretability and the transparency of its output (Granmo, 2018; Sharma et al., 2023; Abeyrathna et al., 2023; Yadav et al., 2021). TM-AE can produce word embeddings that encapsulate the contextual information conveyed by words derived from the training dataset (Kadhim et al., 2024). Consequently, TM can be leveraged to modify texts under examination, using its contextual insights to influence AI-generated text evaluation tools, thereby creating an interpretable adversarial attack that compromises their detection capabilities.

This work makes two primary innovations:

1. Proposing a novel adversarial attack on AI-origin detection systems, reducing detection accuracy from 0.4431 to 0.2744 AUROC on the XSum dataset and from 0.5068 to 0.3532 AUROC on the SQuAD dataset, through data perturbation with embedding models leveraging similar word probability vectors.

2. Employing the interpretable TM model to gain deeper insights into the adversarial attack mechanism and its effects on text origin detection systems.

2 AI-Text Detection Exploration

To understand the process of creating an adversarial attack on AI-text detection tools, it is essential first to comprehend the underlying mechanisms for detection. This section will explore advanced models like DetectGPT (Mitchell et al., 2023) and Fast-DetectGPT (Bao et al., 2024).

In **Detect-GPT**, the scoring process relies on perturbations introduced by making minor modifications, such as replacing or masking specific words. These perturbations are typically generated using GPT-based models, such as T5. That method engages the model's execution logic through a sequence of operations, generally involving 100 perturbations per input, with the model processing each input independently. As a result, that approach increases the execution time for detection compared to, e.g., Fast-DetectGPT. Additionally, the scoring procedure processes each input and its corresponding perturbations sequentially, further contributing to the computational burden. Summary of Detect-GPT Features:

- Perturbation-Based: Generates perturbed versions of the input text by making slight alterations using a masked language model (e.g., T5).
- Probability Curvature: Compares the log-probabilities of the original text with those of its perturbations.

The **Fast-DetectGPT** method, in contrast, is designed to expedite the identification of text origins by utilizing conditional probabilities. This approach mitigates the additional complexities introduced by Detect-GPT for handling original texts. Instead of generating perturbations, the model creates alternatives—typically around 10,000—for each input token. The conditional likelihood function for each token is then evaluated, thereby eliminating the need for perturbations. This method obviates the sequential invocation of generation models, thereby achieving faster performance. Furthermore, the generation of 10,000 samples is streamlined, requiring only a single pass through the scorer. Discrepancy is determined by comparing the log-likelihood of the original token with the mean expected likelihood of the sampled alternatives. Discrepancy Formula:

$$\text{Discrepancy} = \frac{\text{Log-likelihood (original)} - \text{Mean (alternatives)}}{\sqrt{\text{Variance (alternatives)}}}.$$

Summary of Fast-DetectGPT Features:

- Sampling-Based: Generates alternative tokens for each position in the input text based on the model's probability distribution.
- Single Forward Pass: Samples from a categorical distribution derived from the model's logits to streamline execution.

From the above summary of the AI-text detectors, it can be inferred that determining the origin of a text fundamentally relies on assessing the discrepancy between model-suggested probabilities and the natural probabilities characteristic of human decision-making. This principle is utilized in industrial tools, such as **Turnitin**, to evaluate text originality. According to their website (Turnitin, 2025), Turnitin employs an AI model grounded in the text-generation methodology of LLMs. The text is segmented into groups of five to ten sentences, with overlapping segments to analyze contextual inclusivity. Each segment is then processed by an AI checking model, which assigns a score between 0 and 1, indicating whether the text was human-written (0) or AI-generated (1). The overall average of these scores represents the final percentage of AI-generated content in the text.

The Turnitin model relies on the GPT-3 framework for scoring. GPT-3 is trained on extensive Internet content, allowing it to generate text by predicting the most likely next words based on its training data. This prediction process is primarily governed by the transformer architecture (Vaswani et al., 2017), specifically its decoder, which determines the subsequent symbol during text generation. The key components of this mechanism are:

1. Embedding Layer: Comprising two types—input embedding, which calculates contextual information for each symbol, and position embedding, which identifies each symbol's placement within a sentence.
2. Transformer Layer: Includes the self-attention mechanism, which evaluates the importance of sentence components based on individual symbols.
3. Output Layer (Softmax): Processes the preceding results across the vocabulary to select the most likely next word.

Based on the above observations, the sentence score can be lowered by replacing words with alternatives subject to the absolute probabilities within the embedding vector. This reduces the likelihood of selecting the next word, thereby decreasing the sentence's overall score. As a result, the probability of detecting the text as AI-generated is reduced.

3 Proposed Adversarial Strategy

In this section, we first introduce the framework for the adversarial attack using the embedding models and then we discuss the TM-AE architecture and elaborate on its implementation within this study.

3.1 Adversarial Framework

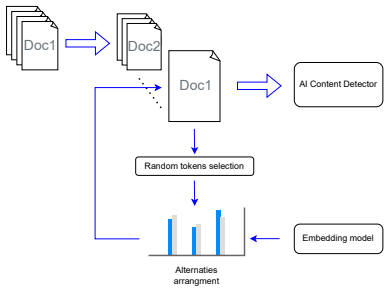

Figure 1: Proposed Adversarial Attack Framework. This figure illustrates the proposed design of an adversarial attack, where the input text (e.g., Doc1, Doc2, ...) is perturbed by selecting alternative tokens with low probability scores generated by embedding models, with the goal of misleading the detection model.

We aim to develop an adversarial attack scheme on AI-generated text detection tools to deceive these models into classifying AI-generated texts as human-written. Embedding models are employed to guide the probability distribution during the selection of alternative words for replacement. Figure 1 illustrates the proposed design of the adversarial attack scheme targeting AI-generated text detection systems. By replacing targeted words, the overall text score decreases compared to scores typically associated with AI-generated or human-written texts. Embedding models, which compute dense vectors for each token in the vocabulary, are integral to establishing relationships between tokens within a high-dimensional space. These vectors have been widely adopted in Pre-Trained Language Model (PLM) model architectures, particularly in Transformer-based models.

This work incorporates three methods for constructing probability distributions for tokens:

3.1.1 Embedding Vector of Similarity

The first method uses the original vector produced by the embedding model to identify alternatives based on their similarity to the target token. Cosine similarity is used as the similarity metric, and two primary parameters are introduced:

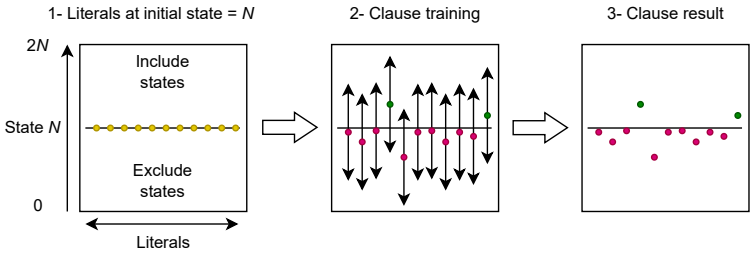

Figure 2: Clause formation in TM. The y-axis is the state index while the x-axis is the literal index. When the state of an automaton is above N, its corresponding literal is included in the clause. Before training, the states of the automata are configured as N (the yellow dots in the left figure). During training, the states are updated (move up and down shown in the middle figure) based on the learning mechanism and training samples. Once trained, the clause is expressed by ANDing the included literals (the green dots in the right figure) and ignoring the excluded literals (red dots).

- **Similarity Vector Length**: Defines the length of the similarity vector derived from the embedding model's original vector. In most cases the length is 400.
- **Similarity Threshold**: Specifies the minimum similarity score for selecting alternatives to the target token.

3.1.2 Synonym Similarity Vector

The second method utilizes grammatically correct synonyms sourced from external lexical databases, such as WordNet, which organizes English words into sets of cognitive synonyms crafted by human linguists and experts. The similarity between the target token and its synonyms is computed using the embedding vector and cosine similarity. Alternatives are ranked based on their similarity scores, with a predefined similarity degree used to select the final alternatives.

3.1.3 Hybrid Scheme with Synonym and Embedding Vectors

The third method is a hybrid approach of the above two, conducted in two stages. In the first stage, a synonym was randomly selected from WordNet. In the second stage, this synonym was replaced with a low-probability word derived from the knowledge vector generated by an embedding model. The TM-AE embedding model was chosen for this scheme due to its transparent and interpretable structure.

3.2 Implementation of Tsetlin Machine

The options among the deep-learning based approaches lack of interpretability due to their black-box nature. To address this, the TM architecture was employed to enhance synonym substitution and assess the impact of these substitutions on text origin detection tools. The TM provides interpretability by allowing detailed insights into decision-making processes, making it a suitable choice for understanding the implications of adversarial attacks. Below we explain the operational concept of TM and how it is adapted to this work.

TM (Granmo, 2018) effectively addresses complex pattern recognition tasks by leveraging propositional conjunctives, with each literal being managed by an individual automaton. Literals represent the entire vocabulary of the input, including their negations. For instance, in a sparse dataset where rows correspond to documents and columns correspond to literals (twice the size of the vocabulary), encoding assigns a value of 1 to a column if the corresponding *word* exists in a document, while its negation receives the opposite value. If the *word* does not appear in the document, the value is 0, and its negation is set to 1.

TM characterizes words through clauses, which are propositional expressions formed by included literals. For example, to describe the word *car*, its clause could include the literals for words *door*, *wheels*, and *no wing*, where *door* and *wheels* are literals in the original form, and *no wing* arises from the literal representing the negation of the word *wing*. Although *wing* does not directly contribute to the clause, its negation plays a role in forming the description.

Clauses are iteratively trained to represent a word. In more detail, the final form of a clause (C) is represented as a conjunctive propositional expression consisting of literals that surpass the threshold state N (See Figure 2). This clause, among others with a size of n clauses with their selective literals, collectively define the output class (y). See Eq. (1).

$$y = u\left(\sum_{j=1}^{n} C_j(X)\right). \tag{1}$$

Here u is a unit step thresholding, $u(v) = 1$ **if** $v \geq 0$ **else** 0, and X is the input vector. Clearly, as seen from Eq. (1), one word can have multiple corresponding clauses. For instance, the word *heart* can have one clause with *love* and *woman*, and another clause with *old* and *hospital*.

The TM-AE architecture (Bhattarai et al., 2024) employed in this design is based on an enhanced TM structure (Glimsdal & Granmo, 2021), incorporating a coalesced weight matrix W to facilitate the simultaneous training of multiple outputs and enabling nested elections among clauses. For example, for the word *heart*, the weight assigned to the clause containing *love* and *woman* may differ from the weight assigned to the clause containing *old* and *hospital*, depending on the training context. Particularly, in this work, the weights from TM-AE are utilized to compute similarity probabilities for target words.

4 Results

Section B in the appendix includes further details that offer a concise overview of the datasets, Pre-Trained Language Models (PLMs), detection models, and embedding models used. In the experiments, the Area Under the Receiver Operating Characteristic (AUROC) score was utilized to evaluate the performance of AI-generated text detection models. An AUROC score of 1.0 signifies perfect detection, indicating the model's certainty that the text was AI-generated. Conversely, an AUROC score of 0.5 represents a random detection performance. Perturbations were generated for each dataset sample using PLM source models, as outlined in Bao et al. (2024), which served as the baseline source for text samples in this study. Subsequently, an additional perturbation was applied based on the proposed approach, as detailed below.

4.1 Experiments with Embedding Vector of Similarity

The datasets were sampled first using five PLM source models (GPT-2 XL, OPT-2.7, GPT-Neo-2.7, GPT-J-6, and GPT-NeoX-20) to generate AI-text samples, and then perturbed using six embedding models (GloVe, FastText, Word2Vec, TM-AE, ELMo, and BERT). The maximum permissible word change ratio was set to 5% of each sample, with an average text length of approximately 150 words across all datasets (XSum, SQuAD, and Writing Prompts). Consequently, the number of altered words did not exceed eight, and in most cases, fewer words were changed. The process involved filtering each text to exclude non-informative tokens that do not represent valid English words. The remaining tokens were then checked for their presence in the vocabulary of the embedding model used. As the embedding models have limited vocabularies, not all words could be replaced.

Table 1 illustrates the detection performance (AUROC scores) for all datasets (XSum, SQuAD, and Writing Prompts) using the Fast-DetectGPT. Two experimental scenarios were considered. **White-Box Environment**: Here, the text-generation source model is known, and the same model is used for scoring. **Black-Box Environment**: A different model is used for scoring, specifically GPT-Neo-2.7.

Source Model	Baseline	BERT	ELMo	FastText	GloVe	TM-AE	Word2Vec
The White-Box Environment							
GPT-J-6	0.9866	0.7887	0.762	0.7270	0.7428	0.7268	**0.7248**
GPT-Neo-2.7	0.9946	0.8338	0.8056	0.7741	0.7815	0.7795	**0.7700**
GPT-NeoX-20	0.9744	0.7061	0.6677	**0.6403**	0.6612	0.6464	0.6483
GPT-2 XL	0.9953	0.8240	0.7988	0.7778	**0.7620**	0.7713	0.7779
OPT-2.7	0.9918	0.774	0.7379	0.7203	0.7292	0.7263	**0.7137**
Avg.	0.9885	0.7853	0.7544	0.7279	0.7353	0.7301	**0.7269**
The Black-Box Environment							
GPT-J-6	0.9601	0.7538	0.7343	0.7068	0.7292	**0.6926**	0.7052
GPT-Neo-2.7	0.9988	0.9399	0.9307	0.9187	0.9334	**0.9163**	0.9209
GPT-NeoX-20	0.9412	0.6998	0.6686	0.6387	0.6773	**0.6302**	0.6485
GPT-2 XL	0.9847	0.8541	0.8402	0.8184	0.8475	**0.8109**	0.8240
OPT-2.7	0.9595	0.7854	0.7636	**0.7356**	0.7689	0.7387	0.7358
Avg.	0.9689	0.8066	0.7875	0.7636	0.7913	**0.7577**	0.7669

Table 1: Compare the detection performance (AUROC scores) of Fast-DetectGPT for various embedding models, evaluated in both white-box and black-box environments, highlighting the effect of embedding diversity on detection effectiveness.

Results Overview: Word2Vec demonstrated the highest effectiveness for adversarial attacks, reducing the detection score to 0.7269 on average in the white-box scenario and 0.7669 in the black-box scenario. BERT showed the least impact, with scores of 0.7853 and 0.8066 in the white-box and black-box environments, respectively, making it the most resistant embedding model. TM-AE exhibited competitive performance, achieving the lowest average detection score of 0.7577 in the black-box environment and 0.7301 in the white-box environment, closely rivaling Word2Vec. The results suggest that BERT's transformer-based architecture produces probability distributions similar to those of the PLMs used for text generation. This alignment makes it easier for detection models to identify AI-generated content, as evidenced by the relatively high scores. In contrast, embedding models like TM-AE and Word2Vec, which leverage distinct probability distributions, posed greater challenges to detection models.

Embedding Models Across Detection Methods: To further evaluate the effectiveness of embedding models, an additional experiment was conducted involving nine detection methods in the white-box environment (Fast-DetectGPT, Detect-GPT, NPR, LRR, DNA, Likelihood, Rank, LogRank, and Entropy) and four detection methods in the black-box environment (Fast-DetectGPT, Detect-GPT, NPR, LRR). Six embedding models (GloVe, FastText, Word2Vec, TM-AE, ELMo, and BERT) were tested across three datasets (XSum, SQuAD, and Writing Prompts), with five PLMs (GPT-2 XL, OPT-2.7, GPT-Neo-2.7, GPT-J-6, and GPT-NeoX-20). In the experiments involving the detectors (Fast-DetectGPT, LRR, and NPR), the PLM model GPT-Neo-2.7 was used as the scoring model. Also, for the detectors LRR and NPR, perturbation generation employed the PLM model T5-3B. Figure 3 presents a heat map summarizing the results, with embedding models repre-

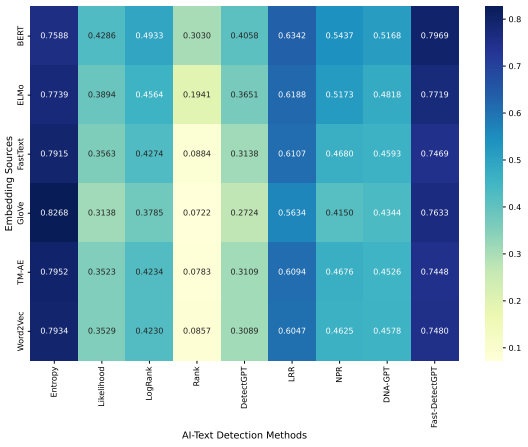

Figure 3: Heatmap illustrating the AUROC scores across various AI-text detection methods and embedding models. The x-axis represents the detection methods, while the y-axis corresponds to embedding models.

sented by rows and detection methods by columns.

Key Findings: Fast-DetectGPT and Entropy were the most resilient detection methods, maintaining AUROC scores above 0.8 for several embedding models. Rank performed poorly, with scores dropping below 0.1 for many embedding models. GloVe achieved favorable results, effectively lowering detection performance across various methods. BERT was least effective in representing adversarial attacks, consistently achieving high detection scores, corroborating earlier findings about its alignment with expected probability distributions. TM-AE and Word2Vec delivered consistently strong performance across different detection methods, demonstrating their utility for adversarial attacks. For detailed experimental results, refer to the tables provided in the appendix (Section C), which outline the performance of each method individually.

4.2 Experiments Utilizing Synonym Similarity Vectors

Previous experiments may not guarantee grammatically and semantically accurate substitutions to represent adversarial attacks, as they rely solely on embedding models and the dense vectors these models provide to generate replacements. In practical scenarios, constructing such adversarial representations would benefit from ensuring substitutions do not disrupt the context or alter the original meaning of the text. To achieve this, the following experiments employed human-curated synonym sets from WordNet to determine replacements for target words. For instance, the word car has synonyms such as *motorcar*, *railcar*, *auto*, *cablecar*, *machine*, *elevatorcar*, *automobile*, *railroadcar*, *railwaycar*, *gondola*. Replacing the target word with an appropriate synonym does not affect the overall sentence meaning. In these experiments, an embedding model was utilized to rank the likelihood of these synonyms, and the effect of varying both the number of substituted words and the nature of the substitution—whether the synonym was the most similar, intermediate, or least similar in the probability vector—was analyzed.

Impact of Disturbance Percentage on Detection Accuracy Figure 4 (left) illustrates the effect of increasing the percentage of word replacements (disturbance percentage) on the detection accuracy. As discussed previously, the percentage represents the maximum allowable substitutions rather than the exact number of replaced words. In this experiment, synonyms were selected from the middle of the synonym probability vector, with Fast-DetectGPT as the detection method and Word2Vec as the embedding model. The experiment was conducted under both white-box and black-box environments across three datasets (XSum, SQuAD, Writing Prompts) and five PLMs (GPT-2 XL, OPT-2.7, GPT-Neo-2.7, GPT-J-6, GPT-NeoX-20).

The results show a clear trend: as the disturbance percentage increased, the detection performance declined significantly. For example, the detection accuracy dropped from approximately 0.9 to 0.6 as the disturbance percentage increased from 1% to 20%. In certain cases, the detection score fell below 0.35 at a 20% disturbance (Min AUROC is 0.3428), indicating a complete failure of the detection method to identify the origin of the text.

Effect of Synonym Location on Detection Accuracy Figure 4 (right) examines the impact of varying the position of the selected synonym in the probability vector on detection accuracy, using the same experimental settings as above. The min, mid, and high labels correspond to the least similar, intermediate, and most similar synonyms, respectively, based on the probability rankings. The results demonstrate that selecting lower-probability synonyms leads to a greater decline in detection accuracy. The detection scores decreased from over 0.95 to below 0.75 on average across detection models when low-probability

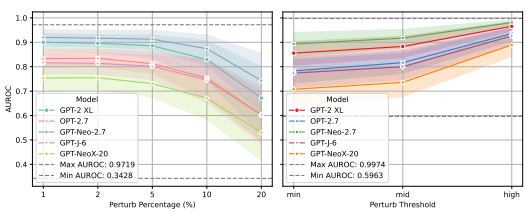

Figure 4: Impact of disturbance percentage and perturbation threshold on detection accuracy. The left panel shows the effect of increasing the percentage of replaced words, while the right panel illustrates the influence of synonym proximity (min, mid, high).

synonyms were used, highlighting the vulnerability of detection methods to such substitutions.

4.3 Hybrid Experiments Using Synonym and Embedding Similarity Vectors

In the hybrid model, the interpretability of the TM-AE embedding is particularly advantageous. Since the first stage is human-understandable (craft by human), the transparency of the second stage, which uses TM-AE, ensures that the entire two-stage process remains fully traceable. This transparency allows us to track completely the inference process of word replacement. For instance, for the target word *car*, which is eventually replaced by *engine*, we can observe how the synonym *machine* was selected in the first stage and then how *machine* is replaced by *engine* in the second stage. This human-understandable nature is critical for further analysis and debugging.

In TM-AE, the knowledge associated with a word is derived from the documents within the training database. For example, the target word *car* might be represented by a clause containing *engine* and *not wing* after training. This representation can be based on a set of training documents where *car* frequently appears alongside *engine* but not with *wing*. Notably, during the preparation of training data, word frequency is disregarded. The model considers only the presence or absence of a word (a Boolean value) when updating the corresponding column in the vocabulary of the input sparse vector. Essentially, the TM operates like an electoral system, where clauses vote on the target word using weights that are iteratively updated during training, ultimately forming a detailed and interpretable description of the word. Further details on the interpretability of the TM model can be found in works such as Bhattarai et al. (2024), Yadav et al. (2021), and Yadav et al. (2022).

Hybrid Substitution Results Figure 5 shows the impact of hybrid substitutions on detection accuracy across five PLMs (GPT-2 XL, OPT-2.7, GPT-Neo-2.7, GPT-J-6, GPT-NeoX-20), comparing nine detection methods (left) and only the Fast-DetectGPT method (right). Detection accuracy was evaluated using two datasets, XSum (blue) and SQuAD (orange). For the left result, all nine detection methods were evaluated under the white-box environment (Fast-DetectGPT, Detect-GPT, NPR, LRR, DNA, Likelihood, Rank, LogRank, Entropy), and four methods under the black-box environment (Fast-DetectGPT, Detect-GPT, NPR, LRR).

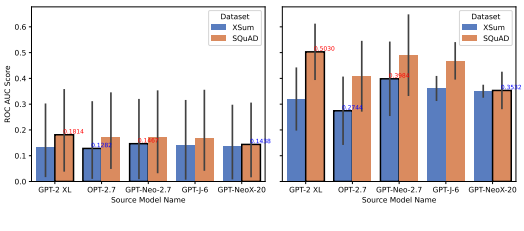

Figure 5: The impact of hybrid substitutions on detection accuracy across five PLMs, comparing all detection methods (left) and the Fast-DetectGPT method (right). Average detection scores for all methods fell to values between 0.1 and 0.2. For XSum, detection scores ranged from 0.1282 to 0.1467, while scores for SQuAD ranged from 0.1438 to 0.1814 (see blue and red numbers above dataset bars). The Fast-DetectGPT results (right) demonstrated average detection scores ranging between 0.2 and 0.5. Specifically, detection scores for XSum varied from 0.2744 to 0.3984, while scores for SQuAD ranged from 0.3532 to 0.5030.

The left results indicate a significant decline in detection accuracy, with scores falling to values between 0.1 and 0.2. Detection methods generally performed better on the SQuAD dataset compared to XSum. Moreover, as the complexity of the PLM source model increased—e.g., with GPT-NeoX-20, which contains more parameters and exhibits greater language generalization—the detection scores became more uniform, averaging below 0.15 across both datasets. This consistency suggests that highly generalized PLMs pose a greater challenge for detection methods, resulting in more uniform performance irrespective of the text's origin.

The Fast-DetectGPT results (right) show average detection scores fluctuating between 0.2 and 0.5. Specifically, detection scores for XSum varied from 0.2744 to 0.3984, while scores for SQuAD ranged from 0.3532 to 0.5030. Compared to similar work (Huang et al., 2024), these results demonstrate a substantial reduction in detection scores. For the XSum dataset, the detection score decreased from 0.4431 to 0.2744 (Percentage Reduction = 38.07%), highlighting a notable decline in the accuracy of text origin detection. Similarly, for the SQuAD dataset, the score dropped from 0.5068 to 0.3532 (Percentage Reduction = 30.30%), indicating the effectiveness of the proposed hybrid substitution method in successfully representing an adversarial attack.

5 Conclusion

This study demonstrates the efficacy of adversarial attacks leveraging embedding-based substitutions to challenge AI-text detection methods. Results reveal that embedding models such as TM-AE and Word2Vec, as well as hybrid substitution methods, significantly reduce detection accuracy, with hybrid approaches achieving detection the lowest scores, particularly in black-box settings and with complex PLMs. Conversely, BERT showed to be the least effective for adversarial attacks due to its probability distributions aligning closely with those of PLM generation models, making it easier for detection systems to identify AI-generated content. These findings underscore the vulnerabilities of detection systems and highlight the potential of interpretable embedding approaches for crafting sophisticated adversarial attacks.

Ethical Statement

This work inherently involves ethical considerations, as it explores methods that could potentially bypass systems designed to detect the origin of text. Such systems often play a critical role in evaluating academic, professional, or creative works, and misuse of these methods could lead to ethical challenges by undermining trust and accountability. The primary aim of this research is to advance cybersecurity by examining the vulnerabilities of text detection systems and identifying potential adversarial strategies that could compromise their reliability. Specifically, this study highlights the role of embedding models in crafting adversarial attacks and emphasizes the need for detection systems to incorporate robust measures against such vulnerabilities. To address this, we propose the development of hybrid detection systems that integrate embedding model-based probability distributions with those from large language models, as suggested in Huang et al. (2024), to enhance their resilience against adversarial attacks.

References

K. Darshana Abeyrathna, Ahmed A. O. Abouzeid, Bimal Bhattarai, Charul Giri, Sondre Glimsdal, Ole-Christoffer Granmo, Lei Jiao, Rupsa Saha, Jivitesh Sharma, Svein A. Tunheim, and Xuan Zhang. Building concise logical patterns by constraining Tsetlin Machine clause size. In *IJCAI*, 2023.

Kuruge Darshana Abeyrathna, Bimal Bhattarai, Morten Goodwin, Saeed Rahimi Gorji, Ole-Christoffer Granmo, Lei Jiao, Rupsa Saha, and Rohan K Yadav. Massively parallel and asynchronous Tsetlin Machine architecture supporting almost constant-time scaling. In *ICML*, 2021.

Guangsheng Bao, Yanbin Zhao, Zhiyang Teng, Linyi Yang, and Yue Zhang. Fast-detectgpt: Efficient zero-shot detection of machine-generated text via conditional probability curvature, 2024. URL https://arxiv.org/abs/2310.05130.

Geir Thore Berge, Ole-Christoffer Granmo, Tor Oddbjørn Tveit, Morten Goodwin, Lei Jiao, and Bernt Viggo Matheussen. Using the Tsetlin Machine to learn human-interpretable rules for high-accuracy text categorization with medical applications. *IEEE Access*, 7: 115134–115146, 2019. doi: 10.1109/ACCESS.2019.2935416.

Bimal Bhattarai, Ole-Christoffer Granmo, Lei Jiao, Rohan Yadav, and Jivitesh Sharma. Tsetlin Machine embedding: Representing words using logical expressions. *Findings of EACL*, pp. 1512–1522, 2024.

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners, 2020. URL https://arxiv.org/abs/2005.14165.

Ciprian Chelba, Tomas Mikolov, Mike Schuster, Qi Ge, Thorsten Brants, Phillipp Koehn, and Tony Robinson. One billion word benchmark for measuring progress in statistical language modeling. *arXiv preprint arXiv:1312.3005*, 2013.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

Tiziano Fagni, Fabrizio Falchi, Margherita Gambini, Antonio Martella, and Maurizio Tesconi. Tweepfake: About detecting deepfake tweets. *Plos one*, 16(5):e0251415, 2021.

Angela Fan, Mike Lewis, and Yann Dauphin. Hierarchical neural story generation. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Association for Computational Linguistics, 2018.

Tomas Foltynek, Sonja Bjelobaba, Irene Glendinning, Zeenath Reza Khan, Rita Santos, Pegi Pavletic, and Július Kravjar. Enai recommendations on the ethical use of artificial intelligence in education, 2023.

Sebastian Gehrmann, Hendrik Strobelt, and Alexander M Rush. Gltr: Statistical detection and visualization of generated text. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics: System Demonstrations*, pp. 111–116, 2019.

Sondre Glimsdal and Ole-Christoffer Granmo. Coalesced multi-output Tsetlin Machines with clause sharing. *CoRR*, abs/2108.07594, 2021. URL https://arxiv.org/abs/2108.07594.

Yoav Goldberg and Omer Levy. word2vec Explained: deriving Mikolov et al.'s negative-sampling word-embedding method. *arXiv preprint arXiv:1402.3722*, 2014.

Ole-Christoffer Granmo. The Tsetlin Machine - a game theoretic bandit driven approach to optimal pattern recognition with propositional logic. *arXiv preprint arXiv:1804.01508*, 2018. URL https://arxiv.org/abs/1804.01508.

Fan Huang, Haewoon Kwak, and Jisun An. Toblend: Token-level blending with an ensemble of llms to attack ai-generated text detection, 2024. URL https://arxiv.org/abs/2402.11167.

Daphne Ippolito, Daniel Duckworth, Chris Callison-Burch, and Douglas 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*, pp. 1808–1822, 2020.

Ahmed K. Kadhim, Ole-Christoffer Granmo, Lei Jiao, and Rishad Shafik. Exploring state space and reasoning by elimination in tsetlin machines, 2024. URL https://arxiv.org/abs/2407.09162.

Jochen L Leidner and Vassilis Plachouras. Ethical by design: Ethics best practices for natural language processing. In *Proceedings of the First ACL workshop on ethics in natural language processing*, pp. 30–40, 2017.

Brady D Lund, Ting Wang, Nishith Reddy Mannuru, Bing Nie, Somipam Shimray, and Ziang Wang. Chatgpt and a new academic reality: Artificial intelligence-written research papers and the ethics of the large language models in scholarly publishing. *Journal of the Association for Information Science and Technology*, 74(5):570–581, 2023.

Sidharth Maheshwari, Tousif Rahman, Rishad Shafik, Alex Yakovlev, Ashur Rafiev, Lei Jiao, and Ole-Christoffer Granmo. REDRESS: Generating compressed models for edge inference using Tsetlin Machines. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 45(9):11152–11168, 2023. doi: 10.1109/TPAMI.2023.3268415.

Eric Mitchell, Yoonho Lee, Alexander Khazatsky, Christopher D Manning, and Chelsea Finn. Detectgpt: Zero-shot machine-generated text detection using probability curvature. *arXiv preprint arXiv:2301.11305*, 2023.

Sandra Mitrović, Davide Andreoletti, and Omran Ayoub. Chatgpt or human? detect and explain. explaining decisions of machine learning model for detecting short chatgpt-generated text. *arXiv preprint arXiv:2301.13852*, 2023.

Shashi Narayan, Shay B Cohen, and Mirella Lapata. Don't give me the details, just the summary! topic-aware convolutional neural networks for extreme summarization. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pp. 1797–1807, 2018.

OpenAI. ChatGPT. https://chat.openai.com/, December 2022.

OpenAI. GPT-4 Technical Report. *arXiv preprint arXiv:2303.08774*, 2023.

Jeffrey Pennington, Richard Socher, and Christopher D Manning. Glove: Global vectors for word representation. In *EMNLP*, pp. 1532–1543, 2014.

Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. Squad: 100,000+ questions for machine comprehension of text. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pp. 2383–2392, 2016.

Vinu Sankar Sadasivan, Aounon Kumar, Sriram Balasubramanian, Wenxiao Wang, and Soheil Feizi. Can ai-generated text be reliably detected?, 2024. URL https://arxiv.org/abs/2303.11156.

Wajiha Shahid, Yiran Li, Dakota Staples, Gulshan Amin, Saqib Hakak, and Ali Ghorbani. Are you a cyborg, bot or human?—a survey on detecting fake news spreaders. *IEEE Access*, 10:27069–27083, 2022.

Jivitesh Sharma, Rohan Yadav, Ole-Christoffer Granmo, and Lei Jiao. Drop clause: Enhancing performance, robustness and pattern recognition capabilities of the Tsetlin Machine. In *AAAI*, 2023.

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

Jinyan Su, Terry Yue Zhuo, Di Wang, and Preslav Nakov. Detectllm: Leveraging log rank information for zero-shot detection of machine-generated text. *arXiv preprint arXiv:2306.05540*, 2023.

Simon Šuster, Stéphan Tulkens, and Walter Daelemans. A short review of ethical challenges in clinical natural language processing. *arXiv preprint arXiv:1703.10090*, 2017.

Loreta Tauginienė, Inga Gaižauskaitė, Irene Glendinning, Július Kravjar, Milan Ojsteršek, Laura Ribeiro, Tatjana Odiņeca, Franca Marino, Marco Cosentino, Shiva Sivasubramaniam, and Tomáš Foltýnek. Glossary for academic integrity. Report, European Network for Academic Integrity, 2018.

Turnitin. Turnitin's ai writing detection capabilities faq: How does it work?, 2025. URL https://guides.turnitin.com/hc/en-us/articles/28477544839821-Turnitin-s-AI-writing-detection-capabilities-FAQ.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2017. URL https://arxiv.org/abs/1706.03762.

Rohan K Yadav, Lei Jiao, Ole-Christoffer Granmo, and Morten Goodwin. Human-level interpretable learning for aspect-based sentiment analysis. In *AAAI*, volume 35, pp. 14203–14212, 2021.

Rohan Kumar Yadav, Jiao Lei, Ole-Christoffer Granmo, and Morten Goodwin. Robust interpretable text classification against spurious correlations using and-rules with negation. In *IJCAI International Joint Conference on Artificial Intelligence*. International Joint Conferences on Artificial Intelligence, 2022.

Xianjun Yang, Wei Cheng, Linda Petzold, William Yang Wang, and Haifeng Chen. Dna-gpt: Divergent n-gram analysis for training-free detection of gpt-generated text. *arXiv preprint arXiv:2305.17359*, 2023.

Zhilin Yang, Zihang Dai, Yiming Yang, Jaime Carbonell, Ruslan Salakhutdinov, and Quoc V. Le. Xlnet: Generalized autoregressive pretraining for language understanding, 2020. URL https://arxiv.org/abs/1906.08237.

Appendix

A Related Work

In related studies focusing on constructing adversarial attacks on AI-generated text detection systems, two primary approaches are identified in Huang et al. (2024) and in Sadasivan et al. (2024), respectively. The approach Huang et al. (2024) involves replacing specific tokens in the text with words generated randomly by LLMs. Multiple variants of GPT models are employed to generate a probability distribution for replacement, thereby influencing the source of the scoring. However, that method significantly increases the complexity of the attack due to the inherent opacity of LLMs, which operate as black-box models, making it challenging to rationalize token selection. Furthermore, the reliance on large-scale LLMs results in extensive execution times. Additionally, Huang et al. (2024) employs the same models utilized for scoring or generating perturbations, thereby restricting the exposure of detection models to alternative types of probability distributions. This limitation arises because these detection models are inherently familiar with the probability distributions of the tokens they analyze. Conversely, approach Sadasivan et al. (2024) proposes relying on a single LLM as the source for token replacement, thereby creating a paraphrase-based attack. This approach is more constrained than the former, as it significantly limits the diversity of probability distributions considered.

B Experimental Resources

A variety of datasets, PLMs, AI-text detection models, and embedding models were utilized in this study. Except for the embedding models, the remaining datasets and models align with those used in Bao et al. (2024), aiming to evaluate the proposed approach comprehensively. This diversity ensures a broad evaluation scope, encompassing various scenarios across downstream applications. The following subsections provide a concise overview of the datasets, PLMs, detection models, and embedding models employed.

B.1 Datasets

Three English datasets from diverse domains were used in the experiments. The XSum dataset, introduced by Narayan et al. (2018), is designed for abstractive summarization and provides 500 concise summaries of news articles. The SQuAD dataset (Rajpurkar et al., 2016), based on Wikipedia contexts, includes 300 samples for training models to answer questions. The Writing Prompts dataset (Fan et al., 2018) contains 500 samples aimed at generating creative and coherent stories based on prompts. These datasets address distinct yet overlapping text generation tasks, enhancing various natural language processing capabilities.

B.2 Pre-trained Language Models (PLMs)

Several pre-trained language models (PLMs) with transformer-based architectures were employed during different stages of preparation and scoring, both during the pre-detection sample collection phase and the text perturbation process for detection. The PLMs utilized include GPT-2 XL with approximately 1.5 billion parameters, OPT-2.7B and GPT-Neo-2.7B with 2.7 billion parameters each, T5-3B with 3 billion parameters, GPT-J-6B with 6 billion parameters, and GPT-NeoX-20B with 20 billion parameters.

B.3 AI-Text Detection Models

The study utilized a set of classifiers designed for zero-shot evaluation of adversarial attacks, including: **Fast-Detect** and **Detect-GPT**: As detailed in Section 2. **Normalized Perturbation Rank (NPR)** and **Log Probability and Log Rank (LRR)**: Both leverage rank- and probability-based features to enhance accuracy while balancing computational efficiency (Su et al., 2023). **LogRank**, **Likelihood**, and **Rank**: Metrics based on token probabilities and their

ranks to evaluate the likelihood of text being AI-generated (Gehrmann et al., 2019; Solaiman et al., 2019). **Entropy** and **Divergent N-Gram Analysis (DNA)**: Techniques focusing on distributional irregularities and n-gram variations to detect machine-generated text (Ippolito et al., 2020; Yang et al., 2023).

B.4 Embedding Models

A diverse set of embedding models was employed to calculate token probability distributions, ensuring variety in the representation of text. The embedding models used include: **GloVe**: Pre-trained embeddings capturing word relationships based on co-occurrence statistics. **FastText**: Embeddings incorporating character-level information, effectively handling out-of-vocabulary words. **Word2Vec**: Static embeddings generated using CBOW or Skip-Gram methods for context prediction. **TM-AE**: As detailed in Section 3.2, this model uses logical expressions for word embeddings. **ELMo**: Contextual embeddings derived from bidirectional language models (BiLSTMs). **BERT**: Contextual embeddings utilizing Transformers, capturing bidirectional context. These models were trained on the One Billion Word dataset (Chelba et al., 2013), with a vocabulary size limited to 40,000 tokens.

C Detailed Experimental Results

This appendix presents detailed tables corresponding to the heatmap plot (Figure 3), summarizing the performance of each detection method and embedding model utilized in the experiments. The results include AUROC scores for various detection methods, including Fast-DetectGPT, Detect-GPT, NPR, LRR, DNA, Likelihood, Rank, LogRank, and Entropy, each provided in a separate table. The experiments were conducted across all datasets (XSum, SQuAD, and Writing Prompts), with text samples generated using different PLM models (GPT-2 XL, OPT-2.7B, GPT-Neo-2.7B, GPT-J-6B, and GPT-NeoX-20B) and perturbed using diverse embedding models (GloVe, FastText, Word2Vec, TM-AE, ELMo, and BERT).

Environment	Dataset	Model	BERT	ELMo	FastText	GloVe	TM-AE	Word2Vec
White	SQuAD	GPT-2 XL	0.6500	0.5900	0.5900	0.5400	0.5900	0.6000
		GPT-J-6	0.4800	0.4500	0.4300	0.3900	0.4000	0.4100
		GPT-Neo-2.7	0.5300	0.5000	0.4600	0.4400	0.4700	0.4600
		GPT-NeoX-20	0.3900	0.3300	0.3300	0.3100	0.3200	0.3300
		OPT-2.7	0.5800	0.5500	0.4900	0.4600	0.5000	0.4700
White	Writing Prompts	GPT-2 XL	0.6300	0.5900	0.5800	0.5400	0.5700	0.5800
		GPT-J-6	0.5900	0.5500	0.5100	0.4800	0.5000	0.5000
		GPT-Neo-2.7	0.6400	0.6000	0.5700	0.5300	0.5600	0.5700
		GPT-NeoX-20	0.5400	0.5000	0.4600	0.4400	0.4400	0.4800
		OPT-2.7	0.5300	0.5000	0.4800	0.4400	0.4600	0.4800
White	XSum	GPT-2 XL	0.4700	0.4600	0.4400	0.4200	0.4200	0.4400
		GPT-J-6	0.4100	0.3600	0.3600	0.3500	0.3500	0.3700
		GPT-Neo-2.7	0.4600	0.4500	0.4200	0.4100	0.4200	0.4100
		GPT-NeoX-20	0.3900	0.3600	0.3500	0.3400	0.3400	0.3400
		OPT-2.7	0.4900	0.4300	0.4300	0.4300	0.4400	0.4300

Table 2: Performance of the DNA-GPT detection method under white-box environment across different datasets (SQuAD, Writing Prompts, and XSum). Results are presented for various embedding models (BERT, ELMo, FastText, GloVe, TM-AE, and Word2Vec) and PLM sources (GPT-2 XL, GPT-J-6B, GPT-Neo-2.7B, GPT-NeoX-20B, and OPT-2.7B).

Environment	Dataset	Model	BERT	ELMo	FastText	GloVe	TM-AE	Word2Vec
White	SQuAD	GPT-2 XL	0.6600	0.5900	0.5400	0.4700	0.5200	0.5400
		GPT-J-6	0.3400	0.2900	0.2400	0.2100	0.2300	0.2500
		GPT-Neo-2.7	0.5100	0.4400	0.3900	0.3400	0.4100	0.3900
		GPT-NeoX-20	0.2400	0.2000	0.1900	0.1500	0.1700	0.1800
		OPT-2.7	0.5200	0.4600	0.3900	0.3500	0.4000	0.4000
White	Writing Prompts	GPT-2 XL	0.6600	0.6000	0.5000	0.4200	0.5000	0.4800
		GPT-J-6	0.4100	0.3700	0.2900	0.2300	0.2900	0.2800
		GPT-Neo-2.7	0.5800	0.5100	0.4100	0.3500	0.4100	0.4100
		GPT-NeoX-20	0.3500	0.3100	0.2400	0.2000	0.2300	0.2400
		OPT-2.7	0.4700	0.4100	0.3300	0.2800	0.3300	0.3300
White	XSum	GPT-2 XL	0.5900	0.5500	0.5200	0.4700	0.5100	0.5100
		GPT-J-6	0.4000	0.3700	0.3500	0.2900	0.3600	0.3200
		GPT-Neo-2.7	0.5700	0.5300	0.5100	0.4600	0.4800	0.4900
		GPT-NeoX-20	0.3400	0.3000	0.2800	0.2400	0.2700	0.2600
		OPT-2.7	0.4700	0.4200	0.3900	0.3600	0.4200	0.3900
Black	SQuAD	GPT-2 XL	0.2200	0.1900	0.1600	0.1300	0.1500	0.1500
		GPT-J-6	0.2600	0.2300	0.1800	0.1600	0.1800	0.1900
		GPT-Neo-2.7	0.5100	0.4400	0.3900	0.3400	0.4100	0.3900
		GPT-NeoX-20	0.2400	0.2100	0.1800	0.1500	0.1600	0.1800
		OPT-2.7	0.2200	0.1900	0.1500	0.1200	0.1500	0.1500
Black	Writing Prompts	GPT-2 XL	0.3500	0.3200	0.2300	0.2000	0.2400	0.2300
		GPT-J-6	0.3500	0.3100	0.2500	0.2000	0.2400	0.2400
		GPT-Neo-2.7	0.5800	0.5100	0.4100	0.3500	0.4100	0.4100
		GPT-NeoX-20	0.3300	0.3000	0.2200	0.1900	0.2200	0.2300
		OPT-2.7	0.3200	0.2900	0.2100	0.1800	0.2000	0.2100
Black	XSum	GPT-2 XL	0.3200	0.3000	0.2700	0.2500	0.2500	0.2600
		GPT-J-6	0.2700	0.2600	0.2300	0.1900	0.2300	0.2200
		GPT-Neo-2.7	0.5700	0.5300	0.5100	0.4600	0.4800	0.4900
		GPT-NeoX-20	0.2700	0.2500	0.2200	0.2000	0.2200	0.2200
		OPT-2.7	0.3000	0.2500	0.2400	0.2100	0.2500	0.2300

Table 3: Performance of the Detect-GPT detection method under white-box and black-box environments across different datasets (SQuAD, Writing Prompts, and XSum). Results are provided for various embedding models (BERT, ELMo, FastText, GloVe, TM-AE, and Word2Vec) and PLM sources (GPT-2 XL, GPT-J-6B, GPT-Neo-2.7B, GPT-NeoX-20B, and OPT-2.7B).

Environment	Dataset	Model	BERT	ELMo	FastText	GloVe	TM-AE	Word2Vec
White	SQuAD	GPT-2 XL	0.7500	0.7700	0.7800	0.8100	0.7900	0.7800
		GPT-J-6	0.7800	0.7900	0.8200	0.8400	0.8100	0.8200
		GPT-Neo-2.7	0.7800	0.7900	0.8100	0.8400	0.8000	0.8100
		GPT-NeoX-20	0.8000	0.8200	0.8300	0.8600	0.8500	0.8400
		OPT-2.7	0.7400	0.7400	0.7800	0.8100	0.7800	0.7800
White	Writing Prompts	GPT-2 XL	0.7100	0.7300	0.7600	0.8100	0.7500	0.7600
		GPT-J-6	0.7300	0.7500	0.7700	0.8300	0.7800	0.7700
		GPT-Neo-2.7	0.6700	0.7000	0.7200	0.7700	0.7300	0.7200
		GPT-NeoX-20	0.7600	0.7800	0.7900	0.8400	0.8000	0.8000
		OPT-2.7	0.7100	0.7300	0.7600	0.8100	0.7600	0.7500
White	XSum	GPT-2 XL	0.7600	0.7700	0.7800	0.8000	0.7800	0.7800
		GPT-J-6	0.8400	0.8500	0.8600	0.8900	0.8600	0.8600
		GPT-Neo-2.7	0.8000	0.8100	0.8200	0.8500	0.8300	0.8300
		GPT-NeoX-20	0.8200	0.8400	0.8500	0.8700	0.8500	0.8500
		OPT-2.7	0.7300	0.7400	0.7500	0.7900	0.7600	0.7600

Table 4: Performance of the Entropy detection method under white-box environment across different datasets (SQuAD, Writing Prompts, and XSum). Results are presented for various embedding models (BERT, ELMo, FastText, GloVe, TM-AE, and Word2Vec) and PLM sources (GPT-2 XL, GPT-J-6B, GPT-Neo-2.7B, GPT-NeoX-20B, and OPT-2.7B).

Environment	Dataset	Model	BERT	ELMo	FastText	GloVe	TM-AE	Word2Vec
White	SQuAD	GPT-2 XL	0.9200	0.8800	0.8800	0.8600	0.8800	0.8800
		GPT-J-6	0.7700	0.7400	0.7100	0.7200	0.7000	0.7000
		GPT-Neo-2.7	0.8400	0.8100	0.7800	0.7800	0.7800	0.7700
		GPT-NeoX-20	0.6100	0.5600	0.5500	0.5500	0.5500	0.5500
		OPT-2.7	0.8600	0.8200	0.8000	0.7900	0.8000	0.7900
White	Writing Prompts	GPT-2 XL	0.8900	0.8700	0.8500	0.8500	0.8500	0.8500
		GPT-J-6	0.8900	0.8700	0.8300	0.8500	0.8300	0.8300
		GPT-Neo-2.7	0.8900	0.8600	0.8300	0.8400	0.8300	0.8300
		GPT-NeoX-20	0.8500	0.8200	0.7900	0.8200	0.7900	0.8000
		OPT-2.7	0.7600	0.7200	0.7100	0.7200	0.6900	0.6900
White	XSum	GPT-2 XL	0.7400	0.7200	0.6900	0.6700	0.6800	0.6900
		GPT-J-6	0.7100	0.6800	0.6400	0.6600	0.6500	0.6400
		GPT-Neo-2.7	0.7700	0.7500	0.7100	0.7200	0.7300	0.7100
		GPT-NeoX-20	0.6600	0.6200	0.5800	0.6200	0.6000	0.5900
		OPT-2.7	0.7000	0.6700	0.6500	0.6700	0.6800	0.6600
Black	SQuAD	GPT-2 XL	0.8900	0.8700	0.8600	0.8700	0.8600	0.8500
		GPT-J-6	0.7600	0.7300	0.7100	0.7200	0.6800	0.7100
		GPT-Neo-2.7	0.9500	0.9400	0.9300	0.9400	0.9300	0.9400
		GPT-NeoX-20	0.6700	0.6300	0.6200	0.6400	0.6000	0.6200
		OPT-2.7	0.8300	0.8100	0.7800	0.8000	0.7800	0.7800
Black	Writing Prompts	GPT-2 XL	0.9100	0.9000	0.8800	0.9100	0.8800	0.8900
		GPT-J-6	0.8600	0.8200	0.8000	0.8300	0.7900	0.8000
		GPT-Neo-2.7	0.9600	0.9500	0.9400	0.9500	0.9300	0.9400
		GPT-NeoX-20	0.8200	0.8000	0.7600	0.8000	0.7500	0.7700
		OPT-2.7	0.8300	0.8000	0.7800	0.8200	0.7700	0.7700
Black	XSum	GPT-2 XL	0.7700	0.7500	0.7200	0.7600	0.7000	0.7300
		GPT-J-6	0.6500	0.6500	0.6100	0.6300	0.6100	0.6100
		GPT-Neo-2.7	0.9100	0.9100	0.8900	0.9100	0.8900	0.8900
		GPT-NeoX-20	0.6100	0.5800	0.5400	0.5900	0.5500	0.5500
		OPT-2.7	0.7000	0.6800	0.6500	0.6800	0.6700	0.6600

Table 5: Performance of the Fast-DetectGPT detection method under white-box and black-box environments across different datasets (SQuAD, Writing Prompts, and XSum). Results are provided for various embedding models (BERT, ELMo, FastText, GloVe, TM-AE, and Word2Vec) and PLM sources (GPT-2 XL, GPT-J-6B, GPT-Neo-2.7B, GPT-NeoX-20B, and OPT-2.7B).

Environment	Dataset	Model	BERT	ELMo	FastText	GloVe	TM-AE	Word2Vec
White	SQuAD	GPT-2 XL	0.8100	0.7900	0.8100	0.7500	0.7800	0.7900
		GPT-J-6	0.6500	0.6300	0.6100	0.5800	0.6100	0.6100
		GPT-Neo-2.7	0.7300	0.7000	0.7000	0.6600	0.7100	0.6900
		GPT-NeoX-20	0.5300	0.5000	0.5300	0.4600	0.5000	0.5000
		OPT-2.7	0.7400	0.7400	0.7000	0.6800	0.7200	0.7100
White	Writing Prompts	GPT-2 XL	0.7800	0.7600	0.7500	0.6900	0.7500	0.7500
		GPT-J-6	0.7300	0.7100	0.7000	0.6400	0.7000	0.6900
		GPT-Neo-2.7	0.7800	0.7600	0.7400	0.6800	0.7300	0.7400
		GPT-NeoX-20	0.6800	0.6600	0.6400	0.6000	0.6400	0.6500
		OPT-2.7	0.7000	0.6700	0.6700	0.6100	0.6700	0.6800
White	XSum	GPT-2 XL	0.6600	0.6400	0.6500	0.6100	0.6400	0.6500
		GPT-J-6	0.5600	0.5600	0.5400	0.5100	0.5500	0.5400
		GPT-Neo-2.7	0.6500	0.6500	0.6500	0.5900	0.6500	0.6300
		GPT-NeoX-20	0.5300	0.5200	0.5100	0.4700	0.5100	0.5000
		OPT-2.7	0.6200	0.6000	0.5900	0.5500	0.6100	0.5800
Black	SQuAD	GPT-2 XL	0.5700	0.5600	0.5700	0.5100	0.5400	0.5500
		GPT-J-6	0.5800	0.5700	0.5400	0.5000	0.5300	0.5400
		GPT-Neo-2.7	0.7300	0.7000	0.7000	0.6600	0.7100	0.6900
		GPT-NeoX-20	0.4800	0.4600	0.4700	0.4000	0.4500	0.4500
		OPT-2.7	0.5700	0.5700	0.5400	0.5200	0.5600	0.5500
Black	Writing Prompts	GPT-2 XL	0.6500	0.6300	0.6000	0.5500	0.6100	0.6000
		GPT-J-6	0.6800	0.6700	0.6500	0.6000	0.6600	0.6500
		GPT-Neo-2.7	0.7800	0.7600	0.7400	0.6800	0.7300	0.7400
		GPT-NeoX-20	0.6100	0.6000	0.5800	0.5300	0.5800	0.5800
		OPT-2.7	0.6100	0.5900	0.5800	0.5200	0.5800	0.5900
Black	XSum	GPT-2 XL	0.4900	0.4700	0.4800	0.4500	0.4600	0.4600
		GPT-J-6	0.4900	0.4800	0.4700	0.4400	0.4800	0.4700
		GPT-Neo-2.7	0.6500	0.6500	0.6500	0.5900	0.6500	0.6300
		GPT-NeoX-20	0.4600	0.4500	0.4500	0.4100	0.4500	0.4400
		OPT-2.7	0.5300	0.5100	0.5100	0.4700	0.5200	0.5000

Table 6: Performance of the LRR detection method under white-box and black-box environments across different datasets (SQuAD, Writing Prompts, and XSum). Results are provided for various embedding models (BERT, ELMo, FastText, GloVe, TM-AE, and Word2Vec) and PLM sources (GPT-2 XL, GPT-J-6B, GPT-Neo-2.7B, GPT-NeoX-20B, and OPT-2.7B).

Environment	Dataset	Model	BERT	ELMo	FastText	GloVe	TM-AE	Word2Vec
White	SQuAD	GPT-2 XL	0.5400	0.4900	0.4700	0.4100	0.4500	0.4700
		GPT-J-6	0.3700	0.3300	0.2900	0.2600	0.2900	0.2900
		GPT-Neo-2.7	0.4300	0.3900	0.3500	0.3100	0.3600	0.3500
		GPT-NeoX-20	0.2600	0.2200	0.2000	0.1700	0.1900	0.2000
		OPT-2.7	0.4900	0.4500	0.4000	0.3600	0.4000	0.4000
White	Writing Prompts	GPT-2 XL	0.5900	0.5400	0.5000	0.4300	0.5000	0.5000
		GPT-J-6	0.5300	0.4800	0.4400	0.3700	0.4200	0.4200
		GPT-Neo-2.7	0.5800	0.5300	0.4900	0.4400	0.4800	0.4900
		GPT-NeoX-20	0.4600	0.4100	0.3700	0.3300	0.3600	0.3700
		OPT-2.7	0.4700	0.4300	0.4000	0.3400	0.3800	0.3900
White	XSum	GPT-2 XL	0.3900	0.3600	0.3300	0.3000	0.3200	0.3300
		GPT-J-6	0.2800	0.2600	0.2300	0.2000	0.2300	0.2300
		GPT-Neo-2.7	0.3700	0.3400	0.3200	0.2800	0.3000	0.3100
		GPT-NeoX-20	0.2700	0.2400	0.2100	0.1900	0.2200	0.2100
		OPT-2.7	0.4000	0.3600	0.3500	0.3100	0.3600	0.3400

Table 7: Performance of the Likelihood detection method under white-box environment across different datasets (SQuAD, Writing Prompts, and XSum). Results are presented for various embedding models (BERT, ELMo, FastText, GloVe, TM-AE, and Word2Vec) and PLM sources (GPT-2 XL, GPT-J-6B, GPT-Neo-2.7B, GPT-NeoX-20B, and OPT-2.7B).

Environment	Dataset	Model	BERT	ELMo	FastText	GloVe	TM-AE	Word2Vec
White	SQuAD	GPT-2 XL	0.6300	0.5800	0.5700	0.5000	0.5400	0.5600
		GPT-J-6	0.4300	0.3900	0.3600	0.3200	0.3600	0.3600
		GPT-Neo-2.7	0.5100	0.4700	0.4300	0.3800	0.4500	0.4300
		GPT-NeoX-20	0.3100	0.2700	0.2600	0.2200	0.2400	0.2500
		OPT-2.7	0.5600	0.5300	0.4700	0.4400	0.4900	0.4800
White	Writing Prompts	GPT-2 XL	0.6500	0.6100	0.5700	0.5000	0.5800	0.5700
		GPT-J-6	0.5900	0.5500	0.5100	0.4400	0.5000	0.5000
		GPT-Neo-2.7	0.6500	0.6000	0.5600	0.5000	0.5500	0.5600
		GPT-NeoX-20	0.5200	0.4700	0.4400	0.3900	0.4300	0.4400
		OPT-2.7	0.5400	0.4900	0.4700	0.4000	0.4600	0.4600
White	XSum	GPT-2 XL	0.4600	0.4300	0.4100	0.3700	0.4000	0.4100
		GPT-J-6	0.3400	0.3200	0.2900	0.2600	0.3000	0.2900
		GPT-Neo-2.7	0.4400	0.4200	0.4000	0.3500	0.3800	0.3800
		GPT-NeoX-20	0.3200	0.2900	0.2700	0.2400	0.2700	0.2600
		OPT-2.7	0.4600	0.4200	0.4000	0.3600	0.4100	0.4000

Table 8: Performance of the LogRank detection method under white-box environment across different datasets (SQuAD, Writing Prompts, and XSum). Results are presented for various embedding models (BERT, ELMo, FastText, GloVe, TM-AE, and Word2Vec) and PLM sources (GPT-2 XL, GPT-J-6B, GPT-Neo-2.7B, GPT-NeoX-20B, and OPT-2.7B).

Environment	Dataset	Model	BERT	ELMo	FastText	GloVe	TM-AE	Word2Vec
White	SQuAD	GPT-2 XL	0.8100	0.7800	0.7500	0.6700	0.7200	0.7300
		GPT-J-6	0.5300	0.4700	0.4300	0.3900	0.4200	0.4400
		GPT-Neo-2.7	0.6800	0.6500	0.6000	0.5400	0.6200	0.6000
		GPT-NeoX-20	0.3900	0.3600	0.3500	0.2900	0.3200	0.3300
		OPT-2.7	0.6800	0.6600	0.5800	0.5400	0.6200	0.6000
White	Writing Prompts	GPT-2 XL	0.7700	0.7400	0.6400	0.5500	0.6400	0.6200
		GPT-J-6	0.5800	0.5600	0.4700	0.3900	0.4800	0.4500
		GPT-Neo-2.7	0.7100	0.6600	0.5700	0.4900	0.5700	0.5700
		GPT-NeoX-20	0.5200	0.5000	0.4100	0.3700	0.4100	0.4300
		OPT-2.7	0.6000	0.5600	0.4900	0.4200	0.4800	0.4900
White	XSum	GPT-2 XL	0.7500	0.7100	0.7100	0.6600	0.6900	0.7100
		GPT-J-6	0.5700	0.5700	0.5400	0.4800	0.5600	0.5200
		GPT-Neo-2.7	0.7300	0.7200	0.7000	0.6500	0.7000	0.6800
		GPT-NeoX-20	0.5200	0.5100	0.4700	0.4400	0.4700	0.4600
		OPT-2.7	0.6400	0.6000	0.5800	0.5400	0.6000	0.5700
Black	SQuAD	GPT-2 XL	0.3300	0.3100	0.2900	0.2400	0.2600	0.2700
		GPT-J-6	0.3800	0.3500	0.2900	0.2700	0.2900	0.3000
		GPT-Neo-2.7	0.6800	0.6500	0.6000	0.5400	0.6200	0.6000
		GPT-NeoX-20	0.3500	0.3200	0.3000	0.2300	0.2700	0.2800
		OPT-2.7	0.3300	0.3100	0.2700	0.2200	0.2700	0.2700
Black	Writing Prompts	GPT-2 XL	0.4500	0.4300	0.3300	0.2800	0.3400	0.3300
		GPT-J-6	0.4500	0.4200	0.3500	0.2900	0.3500	0.3400
		GPT-Neo-2.7	0.7100	0.6600	0.5700	0.4900	0.5700	0.5700
		GPT-NeoX-20	0.4000	0.4000	0.3000	0.2600	0.3000	0.3100
		OPT-2.7	0.4000	0.3800	0.3000	0.2600	0.3000	0.3100
Black	XSum	GPT-2 XL	0.4400	0.4200	0.4100	0.3700	0.3800	0.3900
		GPT-J-6	0.3800	0.3700	0.3700	0.3000	0.3600	0.3400
		GPT-Neo-2.7	0.7300	0.7200	0.7000	0.6500	0.7000	0.6800
		GPT-NeoX-20	0.3700	0.3600	0.3300	0.3000	0.3300	0.3300
		OPT-2.7	0.4100	0.3700	0.3600	0.3300	0.3700	0.3500

Table 9: Performance of the NPR detection method under white-box and black-box environments across different datasets (SQuAD, Writing Prompts, and XSum). Results are provided for various embedding models (BERT, ELMo, FastText, GloVe, TM-AE, and Word2Vec) and PLM sources (GPT-2 XL, GPT-J-6B, GPT-Neo-2.7B, GPT-NeoX-20B, and OPT-2.7B).

Environment	Dataset	Model	BERT	ELMo	FastText	GloVe	TM-AE	Word2Vec
White	SQuAD	GPT-2 XL	0.3700	0.2400	0.1000	0.0700	0.0800	0.0900
		GPT-J-6	0.2200	0.1400	0.0400	0.0400	0.0400	0.0500
		GPT-Neo-2.7	0.2700	0.1900	0.0500	0.0400	0.0500	0.0700
		GPT-NeoX-20	0.1800	0.1100	0.0500	0.0200	0.0200	0.0400
		OPT-2.7	0.2800	0.2100	0.0600	0.0500	0.0500	0.0600
White	Writing Prompts	GPT-2 XL	0.4400	0.2500	0.1200	0.1000	0.1400	0.1200
		GPT-J-6	0.3800	0.2200	0.1200	0.1000	0.0900	0.1200
		GPT-Neo-2.7	0.4400	0.2700	0.1600	0.1200	0.1400	0.1300
		GPT-NeoX-20	0.3600	0.2000	0.1200	0.1100	0.1000	0.1200
		OPT-2.7	0.4200	0.2700	0.1300	0.1200	0.1200	0.1200
White	XSum	GPT-2 XL	0.2800	0.2000	0.0900	0.0700	0.0800	0.0800
		GPT-J-6	0.2100	0.1500	0.0600	0.0500	0.0700	0.0600
		GPT-Neo-2.7	0.2600	0.1700	0.0800	0.0700	0.0700	0.0800
		GPT-NeoX-20	0.2000	0.1400	0.0700	0.0600	0.0500	0.0600
		OPT-2.7	0.2400	0.1600	0.0600	0.0500	0.0700	0.0700

Table 10: Performance of the Rank detection method under white-box environment across different datasets (SQuAD, Writing Prompts, and XSum). Results are presented for various embedding models (BERT, ELMo, FastText, GloVe, TM-AE, and Word2Vec) and PLM sources (GPT-2 XL, GPT-J-6B, GPT-Neo-2.7B, GPT-NeoX-20B, and OPT-2.7B).