Character-Level Perturbations Disrupt LLM Watermarks

Zhaoxi Zhang*†
University of Technology Sydney
zhaoxi.zhang-1@student.uts.edu.au

Xiaomei Zhang†
Griffith University
xiaomei.zhang@griffithuni.edu.au

Yanjun Zhang
University of Technology Sydney
yanjun.zhang@uts.edu.au

He Zhang
RMIT University
he.zhang@rmit.edu.au

Shirui Pan
Griffith University
s.pan@griffith.edu.au

Bo Liu
University of Technology Sydney
bo.liu@uts.edu.au

Asif Gill
University of Technology Sydney
asif.gill@uts.edu.au

Leo Yu Zhang ✉
Griffith University
leo.zhang@griffith.edu.au

***Abstract*—Large Language Model (LLM) watermark has emerged as a promising technique for copyright protection, misuse prevention, and machine-generated content detection. It injects detectable signals during the LLM generation process, allowing for later identification by a corresponding detector. To assess the robustness of watermark schemes, existing studies typically adopt watermark removal attacks, which aim to erase embedded signals by modifying the watermarked text. However, we reveal that existing watermark removal attacks are suboptimal, which leads to the misconception that effective watermark removal requires either a large perturbation budget or a strong adversary's capabilities, such as unlimited queries to the victim LLM or its watermark detector. A systematic scrutinization of removal attack capabilities as well as the development of more sophisticated techniques remains largely underexplored. As a result, the robustness of existing watermarking schemes may be overestimated.**

To bridge the gap, we first formalize the system model for LLM watermark, and characterize two realistic threat models constrained on limited access to the watermark detector. We then analyze how different types of perturbation vary in their attack range, i.e., the number of tokens they can affect with a single edit. We observe that character-level perturbations (e.g., typos, swaps, deletions, homoglyphs) can influence multiple tokens simultaneously by disrupting the tokenization process. We demonstrate that character-level perturbations are significantly more effective for watermark removal compared to token-level or sentence-level approaches under the most restrictive threat model. We further propose guided removal attacks based on the Genetic Algorithm (GA) that uses a reference detector for optimization. Under a practical threat model with limited black-box queries to the watermark detector, our method demonstrates strong removal performance. Experiments across five representative watermarking schemes and two widely-used LLMs consistently confirm the superiority of character-level perturbations and the effectiveness of the reference-detector-guided GA in removing watermarks under realistic constraints. Additionally, we argue there is an adversarial dilemma when considering potential defenses: any fixed defense can be bypassed by a suitable perturbation strategy. Motivated by this principle, we propose an adaptive compound character-level attack. Experimental results show that this approach can effectively defeat the defenses. Our findings highlight significant vulnerabilities in existing LLM watermark schemes and underline the urgency for the development of new robust mechanisms.

* Work done during the visit at Griffith University.
† Equal contribution.
✉ Corresponding author.

Network and Distributed System Security (NDSS) Symposium 2026
23-27 February 2026, San Diego, CA, USA
ISBN 979-8-9919276-8-0
https://dx.doi.org/10.14722/ndss.2026.230138
www.ndss-symposium.org

I. Introduction

Large Language Models (LLMs) have become foundational components in modern AI systems [1]–[3], powering a wide range of consumer-facing applications and critical decision-making systems, such as chat assistants [4], [5], educational tools [6], [7], and content generation platforms [8], [9], etc. However, as LLMs are increasingly used to generate high-quality content indistinguishable from human-written text, growing concerns have emerged about their potential misuse [10]–[15], including misinformation generation [16], automated phishing [17], and academic fraud [18], etc. In response, there is a pressing demand for reliable mechanisms to attribute machine-generated text and distinguish it from human-written content. Among existing attribution techniques, the LLM watermark has emerged as one of the most promising approaches [19]–[26]. LLM watermarks work by subtly adjusting the model's generation process, embedding statistically detectable signals into the output text. These changes are invisible to users but can be identified by a watermark detector, which typically uses statistical tests to distinguish watermarked text from non-watermarked text.

However, the practical utility of such watermarks hinges on their robustness, i.e., the ability to remain detectable after adversarial edits. To assess robustness, researchers commonly conduct watermark removal attacks [19], [20], [24], [27], which attempt to erase embedded signals by modifying the watermarked text, typically via token-level (e.g., synonym replacement) or sentence-level perturbations (e.g., paraphrasing). However, existing approaches face several notable limitations. First, prior work often relies on unrealistic assumptions about the adversary's capabilities, such as full knowledge of the watermarking scheme, unlimited queries to the victim LLM or its watermark detector, or the availability of a similar surrogate LLM [28]–[31]. Second, these methods primarily rely on token-level or sentence-level perturbations [20], [32], [33], which we have found to be suboptimal for efficient watermark removal (see experiments in Section IV-D1 for detailed results). Third, due to the lack of guidance, these methods fail to prioritize tokens most critical to watermark removal, leading to inefficient perturbations [33], [34]. These limitations have led to a misleading perception that successful watermark removal requires either a large perturbation budget or strong adversary's capabilities. As a result, the robustness of existing watermarking schemes may be overestimated. This highlights the urgent need for a systematic investigation of watermark removal capabilities and the development of more sophisticated attack techniques.

To this end, we firstly conduct a systematic study on LLM watermark removal under realistic constraints. We assume a black-box setting where the adversary can only access the output texts of the victim LLM, with no access to its architecture, parameters, or internal states (e.g., logits). Additionally, the adversary has no knowledge of the underlying watermarking scheme and is restricted to a limited number of queries per input. Based on the adversary's access to the original watermark detector of the victim LLM, we consider two distinct threat models: (1) the adversary has no access to the detector; (2) the adversary can query the detector under a limited budget per input. Based on our categorization of adversary knowledge and capabilities, we classify existing watermark removal methods accordingly (refer to Table I). Existing approaches are not applicable under the two practical threat models we consider.

To obtain the optimal perturbation operation, we begin with evaluating the removal attack performance by using different perturbation types, where the adversary has no access to the original watermark detector. In this challenging setting, the adversary can only apply random perturbations to the watermarked text. As a result, the effectiveness of the attack is largely determined by the attack range, which refers to the number of tokens affected by a single edit. Through analysis, we observe that character-level perturbations, such as typos, deletions, swaps, insertions of zero-width or whitespace characters, and homoglyph substitutions, can disrupt the tokenization process. This disruption often splits a single token into multiple subword units, allowing each edit to affect more than one token[1]. Motivated by this insight, we design a character-level watermark removal attack and evaluate its effectiveness through extensive experiments. Results show that character-level perturbations consistently outperform token-level and sentence-level methods, enabling watermark removal with a smaller perturbation budget.

To handle the lack of direct guidance, we propose using a reference detector to improve the effectiveness of watermark removal attacks, where the adversary is allowed only a limited number of black-box queries to the original watermark detector. We first train a lightweight reference detector using data collected within the allowed query budget. This reference detector mimics the behavior of the original detector and helps guide our following Genetic Algorithm (GA) method in selecting impactful token positions. With the reference detector in place, the GA iteratively finds tokens most removal-relevant, enabling more focused and efficient perturbations while overcoming the constraint of limited queries to the original detector.

Our contributions are summarized as follows.

- **Systematic formulation of LLM watermark removal.** We provide the first comprehensive analysis of watermark removal attacks against LLMs. We formally characterize threat models under different system settings, offering a structured framework for evaluating attack effectiveness.
- **Character-level perturbations and attack range analysis.** We identify the attack range, the impact scope of a single perturbation, as a key factor affecting the effectiveness of watermark removal. We show that character-level perturbations offer the widest attack range by disrupting tokenization.
- **Reference-detector-guided removal attack and adaptive strategy.** Under the setting of limited access to the original watermark detector, we propose a GA-based removal attack guided by a reference detector, which is trained to approximate the behavior of the original detector using limited queries. We also highlight a key limitation of the reference detector: the mismatch between the reference and original detectors makes gradient-based optimization unreliable. Furthermore, we highlight an adversarial dilemma for potential defenses: for any fixed defense, there always exists an effective perturbation strategy that can bypass it. Guided by this insight, we propose an adaptive compound character-level attack. Our method is evaluated across diverse watermarking schemes and consistently achieves strong performance.[2]

II. Background

A. LLM Generation

Large Language Models (LLMs) generate text autoregressively, producing one token at each step conditioned on the given prompt (and the previously generated tokens), until an end-of-sequence token is generated or a predefined maximum length is reached. The generation process at each step t involves three stages: computing the logits, deriving a probability distribution, and sampling the next token [1]–[3]. First, the model takes the prompt and previously generated tokens $x_{<t} = \{x_1, x_2, \ldots, x_{t-1}\}$ as input and produces a logit vector

[1]This phenomenon contrasts with the effects of character-level perturbations on adversarial robustness in traditional NLP, where small input changes are intended to cause significant changes in a model's internal embeddings. From this perspective, token-level and character-level perturbations follow similar principles [35]–[39]. In contrast, the principles and effects of token-level and character-level attacks on LLM watermarks are fundamentally different.

[2]We release the source code at https://github.com/plll4zzx/CharacterRemoval4WM

TABLE I: Comparison of our study with existing typical watermark removal methods. **Adversary's capabilities & knowledge** (CK): CK1: Limited query budget to victim LLM; CK2: No access to original detector; CK3: Limited query budget to original detector; CK4: No knowledge of victim LLM; CK5: No knowledge of watermark scheme. **Attack perturbation type & strategies** (PS): PS1: Character-level perturbation; PS2: Token-level perturbation; PS3: Sentence-level perturbation; PS4: Gradient-free guidance. Here, ● denotes "Fully Considered", and ○ denotes "Not Considered".

		CK1+CK2	CK1+CK3	CK4	CK5	PS1	PS2	PS3	PS4
Stealing &Removal	[40]	●	●	●	○	○	●	○	●
	[41]	●	●	●	○	○	●	○	●
	[28]	○	○	●	○	○	●	○	●
	[30]	●	●	●	○	○	●	○	●
Removal	[42]	●	○	●	●	○	○	●	○
	[32]	●	○	●	●	○	○	●	○
	[43]	●	○	●	●	●	●	●	○
	[44]	●	○	●	●	●	○	●	○
	[34]	●	○	●	●	●	●	●	○
	[45]	●	○	●	●	○	●	○	○
	[46]	●	●	●	●	●	●	●	○
	[31]	○	○	○	●	○	●	○	○
	Ours	●	●	●	●	●	●	●	●

$\ell_t \in \mathbb{R}^{|V|}$, where $|V|$ is the vocabulary size. Second, the logits are transformed via a softmax function into a probability distribution over the vocabulary. Third, the next token x_t is sampled from this distribution and then appended to the sequence.

B. Injecting Watermark

LLM watermark aims to inject detectable signals into LLM-generated content. In this study, we focus on inference-time watermark, which injects watermark signals during text generation without modifying the model's parameters [19]–[25]. Compared to training-time watermark methods, such as trigger-based approaches [47], [48], inference-time methods are more flexible and cost-effective, as they do not require access to the model's weights or retraining. Formally, at generation step t, the LLM computes a key k_t using a hash function $\text{hash}(\cdot)$ over the previous context $c_t = \{x_{t-h}, \cdots, x_{t-1}\}$, where h is the length of context. Beyond the typical range, there exists a special case where $h = 0$, in which all tokens generated by the same LLM share the same key. This key is then used to subtly alter the LLM's generation behavior. Existing watermarking techniques fall into three categories based on the generation stage where the watermark is introduced:

- **Watermark during logits generation:** These methods aim to shift the generation process by modifying the LLM's logits. At each generation step t, a pseudorandom function and a key k_t are used to partition the vocabulary V into two subsets: a green list and a red list. A fixed bias δ is added to the logits of green-list tokens, increasing their likelihood of being sampled. So, generated texts are biased toward green-list tokens and can be detected as watermarked, such as KGW [19], Unigram [20].
- **Watermark during token probability distribution generation:** These methods aim to modify the token probability distribution. At each step t, a pseudorandom function and a key k_t are used to reorder the vocabulary tokens. Tokens whose cumulative probability exceeds a threshold γ are selected and reweighted to increase their sampling probability, resulting in watermark insertion, such as Unbias [21], DIP [22].
- **Watermark during sampling:** These methods aim to embed watermarks by modifying the sampling process. Two main strategies are commonly used. (1) A set of candidate tokens is first sampled from the original token probability distribution. Then, each candidate is assigned a pseudorandom score derived from a key k_t, and the token with the highest score is selected as x_t, such as SynthID [23]. (2) Alternatively, a pseudorandom score is derived from k_t and used in inverse transform sampling to deterministically select the next token [24], [25].

C. Detecting Watermark

Watermark detection is inherently tied to the watermark injecting strategy, as each approach modifies the token generation process differently. The goal is to evaluate whether a given text X exhibits statistical traces of watermarking, typically via a global watermark score $S_w(X)$. A higher score indicates a stronger alignment with the watermark. Formally, the input text is first tokenized into a sequence $X = \{x_1, x_2, \cdots, x_t, \cdots, x_m\}$. For each position t, a key k_t is derived from a hash function over the prior context $c_t = \{x_{t-h}, \cdots, x_{t-1}\}$, i.e., $k_t = \text{hash}(c_t)$. Depending on the specific watermark injection method (see Section II-B), a watermark score s_t for each token x_t might be computed using the token itself and its associated key k_t. If such scores are used, they are typically aggregated across the sequence to obtain the global score $S_w(X)$; otherwise, detection relies on aggregate statistics such as token counts. If $S_w(X)$ exceeds a predefined threshold τ_d, the sequence X is labelled as watermarked. Detection methods for each category of watermark scheme are detailed below:

- **Watermark during logits generation:** The detector uses k_t to determine the green list at position t and checks whether x_t belongs to it. The global watermark score is typically computed using a one-sided z-test: $S_w(X) = \frac{|X|_G - \gamma|X|}{\sqrt{\gamma(1-\gamma)|X|}}$, where $|X|_G$ is the number of green-list tokens in X, γ is the ratio of the green list to the entire vocabulary [19], [20].
- **Watermark during token probability generation:** These watermark methods modify the token probability distribution during generation, without altering the logits directly. To detect such watermarks, the detector compares how likely each token is under the watermarked LLM M_w versus the original LLM M. Specifically, for each token x_t, the token-level watermark score is computed as the log-likelihood ratio: $s_t = \log \frac{M_w(x_t|x_{<t}, k_t)}{M(x_t|x_{<t})}$, where M_w generates a key-conditioned distribution using k_t. The global watermark score is then aggregated as: $S_w(X) = \sum_{t=1}^{m} s_t$. Watermarked texts tend to follow the token probability distribution of M_w, resulting in a higher $S_w(X)$ than non-watermarked texts [21], [22].

- **Watermark during sampling:** To detect the watermark, the detector recovers the pseudorandom score assigned to x_t by using the key k_t, and computes the token-level watermark score s_t by checking whether x_t aligns with the pseudorandom score. The global watermark score of X is then calculated by summing all s_t: $S_w(X) = \sum_{t=1}^{m} s_t$. If x_t is from watermarked text, it tends to get higher s_t, and watermarked text tends to get higher $S_w(X)$ [23]–[25].

D. Watermark Removal Attacks

We study watermark removal attacks, which aim to erase the watermark embedded in LLM-generated text while preserving its original semantics. Formally, given a watermarked text $X = \{x_1, x_2, \cdots, x_t, \cdots, x_m\}$, the adversary applies an attack $\mathcal{A}$ to generate a perturbed text $\tilde{X} = \mathcal{A}_{\tilde{P}}(X)$, where $\tilde{P} \subset \{1, 2, \cdots, m\}$ indicates the positions selected for perturbation. After perturbed, the global watermark score $S_w(\tilde{X})$ is below the threshold, thereby evading watermark detection. Existing watermark removal methods can be categorized based on the level of perturbation:

- Character-level perturbations includes typos, character deletion, character swapping, insertion zero-width character or withe space, and homoglyph substitution etc., denoted as $\tilde{X} = \mathcal{A}^{C}_{\tilde{P}}(X)$, where $\tilde{P}$ is randomly selected from $\{1, 2, \cdots, m\}$ [34], [43], [44], [46].
- Token-level perturbations include word deletion, reordering, and synonym replacement etc., denoted as $\tilde{X} = \mathcal{A}^{T}_{\tilde{P}}(X)$, with $\tilde{P}$ also selected randomly [19], [20], [24].
- Sentence-level perturbations apply high-level transformations such as paraphrasing or translation, denoted as $\tilde{X} = \mathcal{A}^{S}_{\tilde{P}}(X)$, where $\tilde{P}$ is selected by a paraphraser or translator [32], [42].

Summary. For character-level and token-level, perturbation positions are typically chosen randomly without considering the watermark structure [19], [20], [24], [34], [43], [44], [46]. For sentence-level methods, such as paraphrasing, the position is selected according to the semantic demand [32], [42]. Consequently, we define these methods as *random strategies*, as they lack awareness of watermark-related features while removing. To remove the watermark, these methods typically rely on altering as many tokens as possible. As a result, the systematic scrutinization of removal attack capability — as well as the development of more sophisticated techniques — remains largely underexplored (refer to Appendix A-A for more discussion about related work).

Evaluation Metric. Following prior works [19]–[24], we adopt a set of metrics to assess the effectiveness of watermark removal attacks. These metrics can be categorized into two groups according to their purposes:

- Metric for perturbation budget: (1) Character editing distance ($\mathrm{ED}_C(X, \tilde{X})$): Number of modified characters. (2) Character editing rate ($\mathrm{ER}_C(X, \tilde{X})$): Ratio of modified characters, computed as $\frac{\mathrm{ED}_C(X,\tilde{X})}{|X|_C}$, where $|X|_C$ is the character number of X. (3) Token editing distance ($\mathrm{ED}_T(X, \tilde{X})$): Number of modified tokens. (4) Token editing rate ($\mathrm{ER}_T(X, \tilde{X})$): Ratio of modified tokens, $\frac{\mathrm{ED}_T(X,\tilde{X})}{|X|}$, where $|X|$ is the token number of X.
- Metric for removal performance: (1) Watermark score dropping rate ($\frac{S_w(X)-S_w(\tilde{X})}{S^w_{max}-S_{min}}$): where, S^w_{max} is defined as the maximum value of $S_w(\cdot)$ across all sentences in watermarked dataset, and S_{min} is defined as the minimum value of $S_w(\cdot)$ across all sentences in non-watermarked dataset. (2) Attack success rate (ASR): The percentage of watermarked texts where the watermark is no longer detected after the attack, ASR is computed over a dataset consisting of watermarked samples only.

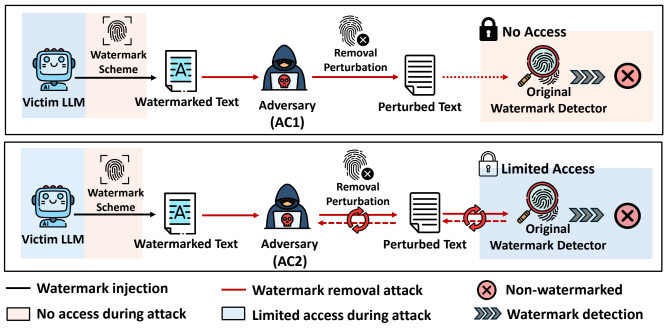

Fig. 1: Illustration of threat models in watermark removal.

Based on our analysis of the C4 dataset [49], each token contains an average of 5.18 characters. Therefore, when a single character is modified within a token, both the character-level and token-level editing distances are 1, while the character-level editing rate is approximately $\frac{1}{5.18}$ of the token-level editing rate. For a fair comparison, all editing distance (ED) and editing rate (ER) in this paper refer to the token level (ED_T and ER_T) unless otherwise specified.

III. Problem Formulation

A. System Model

We consider a LLM system where the model provider offers a black-box API of a large language model (LLM), which takes user input and returns text output embedded with a watermark [8], [9], [50]. A corresponding watermark detector API is provided to determine whether a given text contains a watermark generated by the LLM. Based on the accessibility of the watermark detector, we define two system models:

- **System Model 1:** The watermark detector is private and can only be accessed by the model provider or authorized parties.
- **System Model 2:** The watermark detector is provided as a public API. Anyone can query the detector with a text input and obtain the detection result.

Modern LLM systems typically deploy access controls to detect and block malicious behaviors, such as reverse engineering or data extraction [4], [5], [51]. Therefore, in practical scenarios, adversaries are allowed only a limited number of black-box queries to the LLM and (if applicable) the watermark detector for each input and its slightly perturbed versions.

B. Threat Model

Adversary's Goal. We consider adversaries who aim to remove watermarks from LLM-generated text using minimal imperceptible perturbation. Formally, the goal is to find a perturbed text $\tilde{X}$ such that the global watermark score $S_w(\tilde{X})$ falls below a detection threshold τ_d, while minimizing the editing rate:

$$\arg\min_{\tilde{X}} \mathrm{ER}(X, \tilde{X}), \text{ s.t. } S_w(\tilde{X}) < \tau_d. \tag{1}$$

This goal aligns with practical scenarios where adversaries seek to bypass watermark detectors while preserving the text's utility, enabling unauthorized use of the model output. Moreover, analyzing the minimum perturbation required for removal offers a precise measure of watermark robustness.

Adversary's Knowledge. According to the system model, the adversary only has access to the watermarked text generated by the victim LLM API. As a result, they have no knowledge of the underlying watermark scheme or its parameters (e.g., δ, γ, h described in Section II-B). This limitation makes watermark stealing attacks infeasible [28], [40], [41], as such attacks typically rely on knowledge of the watermark embedding mechanism. Due to the black-box nature of the LLM API, the adversary cannot access the model's architecture, weights, or internal outputs such as logits. Consequently, they are unable to perform model distillation or identify a similar open-source surrogate, which rules out attacks that approximate the token probability distribution using surrogate LLMs [29], [31], [52].

Adversary's Capabilities. Based on the system model, we define two levels of adversarial capabilities (ACs), as illustrated in Figure 1, each corresponding to different system models:

- **AC1** (System model 1): The adversary has *no access* to the original watermark detector. They can only interact with the victim LLM through a limited number of black-box queries per input.
- **AC2** (System model 2): The adversary has *limited access* to the victim LLM and its original watermark detector. They can query them with a limited number of black-box queries per input.

These capabilities reflect practical constraints imposed by modern LLM systems, which implement access control to prevent malicious behaviors [4], [5], [51].

IV. Benchmark of Removal Attack for LLM Watermark

In this section, we systematically investigate the effectiveness of watermark removal attacks under the setting of **AC1** and highlight the clear advantages of character-level perturbations. We begin by analyzing the inherent strengths of character-level attacks, particularly their broader impact on watermarking mechanisms under constrained editing budgets. Building on this insight, we introduce a simple yet effective baseline that applies character-level perturbations in the **AC1** scenario. Finally, we conduct a comprehensive experimental evaluation comparing character-level attacks with token-level and sentence-level methods, demonstrating the superior removal effectiveness and better text quality preservation achieved by character-level approaches.

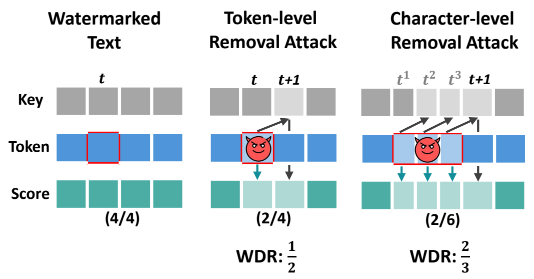

Fig. 2: Comparison of attack range between token-level and character-level perturbations in watermark removal. Under a context size of $h = 1$, modifying one token affects the token itself and the key computation for h subsequent tokens, resulting in an attack range of 2 (i.e., $h + 1$). Modifying a single character (e.g., via homoglyph substitution) will split the affected token into at least 3 subword tokens (e.g., splitting token t into sub-tokens t^1, t^2, t^3). This perturbation also influences the next h positions, yielding an attack range of 4 (i.e., $h+3$). Assuming each token watermark score $s_t \in \{0, 1\}$, with dark shading indicating $s_t = 1$, the global watermark score of the watermarked text is $S_w(X) = \frac{\sum s_t}{|X|} = \frac{4}{4}$. After applying token-level perturbations, the score drops to $S_w(\tilde{X}) = \frac{2}{4}$, yielding a watermark score dropping rate (WDR) of $\frac{1}{2}$. In contrast, character-level perturbations result in WDR $= \frac{2}{3}$.

A. Motivation of Using Character-Level Perturbation in Removal Attack

The goal of a watermark removal attack is to reduce the global watermark score $S_w(X)$ of a text X below the detection threshold. This score is computed by aggregating the individual watermark scores of all tokens in the sequence $X = \{x_1, x_2, \cdots, x_t, \cdots, x_m\}$. When a token x_t is modified, its own score s_t will be affected. In addition, since the watermark key k_{t+i} is computed from the preceding context $c_{t+i} = \{x_{t-h+i}, \cdots, x_{t-1+i}\}$ ($x_t \in c_{t+i}$, if $1 \le i \le h$), a single modification can also affect the keys of the following h tokens (i.e., $k_{t+i}, i \in [1, h]$). Consequently, this also alters the watermark scores of those subsequent tokens (i.e., $s_{t+i}, i \in [1, h]$). This means that the effectiveness of a watermark removal attack depends not only on the editing rate—how many tokens are changed—but also on the attack range, i.e., how many tokens are affected by a single edit. Under the constraint of keeping the editing rate small, it becomes critical to choose perturbations that maximize the attack range in order to reduce the global watermark score more effectively.

To analyze the attack range more intuitively, we consider that a token's score is negatively affected (watermark score decreases) if either the token itself or its key is changed. In token-level attacks, modifying one token directly affects its own score and the keys of the following h tokens, leading to an attack range of $h + 1$. In contrast, character-level attacks can have a broader impact. As shown in Figure 2, replacing a middle character of the t-th token with a homoglyph can cause the token to be split into at least three subword tokens (the subword before homoglyph, homoglyph, and the subword after homoglyph) during tokenization—specifically x_{t^1}, x_{t^2}, and x_{t^3} in the perturbed text. For example, replacing "t" in "letter" with its homoglyph "ť" (U+0165) may split the word into ["le", "ť", "ter"]. This means that a single character-level edit can directly affect three tokens. Furthermore, because the key k_{t+1} depends on the previous h tokens, the watermark score of x_{t+1} token in the perturbed text will also be affected. As a result, a single character-level modification leads to an attack range of $h+3$.

Figure 2 illustrates this effect under the common setting of $h = 1$. For simplicity, we assume that each token in the watermarked text carries watermark information, with a watermark score of 1 (shown in dark color, light color means a watermark score of 0). In this case, the attack range of a token-level attack is 2, while a character-level attack reaches 4, twice as much. In terms of reducing the global watermark score, character-level attacks are significantly more effective.

Takeaway: Maximizing the attack range is crucial for effective watermark removal under a low editing rate.

B. A Random Strategy for Character-Level Removal Attack

Based on the above analysis, we propose a simple baseline method for watermark removal based on character-level perturbation. As described in the threat model in Section III-B, in **AC1**, the adversary is not allowed to interact with a local surrogate LLM and the original watermark detector of the victim LLM. As a result, the attacker adopts a random perturbation strategy: randomly selecting positions in the text and applying character-level perturbation.

To maximize the attack range of each perturbation, we modify only a single character for each selected token, preferably one located near the center of the token. For example, the token "position" may be modified to "pošition", where "š" (U+0161 in Unicode) is a homoglyph of "s". This method can be formulated as follows:

$$\tilde{X} = \mathcal{A}^C_{\tilde{P}}(X), \text{ s.t. } \tilde{P} \xleftarrow{\text{rand}} \{1, \cdots, m\}, \quad \frac{|\tilde{P}|}{m} \leq \epsilon, \quad (2)$$

where $\mathcal{A}^C$ is a character-level attack, $\tilde{P} \subset \{1, 2, \cdots, m\}$ is sampled uniformly at random, m is the token number of X, and ϵ is upper bond of editing rate.

C. Experimental Setup

1) Watermarked Data: We follow the experimental setup used in prior work [19], [20], [33], [46]. Specifically, the prompts used to generate watermarked text are sampled from the RealNewsLike subset of the C4 dataset [49]. Watermarked text is generated by two victim LLMs: LLaMA-3-8B [2] and OPT-1.3B [1]. All experiments are conducted on NVIDIA A100 GPUs. We evaluate five representative watermark methods: KGW [19], DIP [22], SynthID [23], Unigram [20], and

TABLE II: Comparison of watermark removal performance under token-level, character-level, and sentence-level attacks across five watermarking schemes. For sentence-level attacks, we use DIPPER (†) and AuthorMist (∗). The editing rate of DIPPER can be approximately adjusted to $\text{ER} \approx 0.5$, whereas AuthorMist lacks editing rate controllability. Character-level attacks consistently achieve higher WDR and ASR under comparable editing rates.

		Token		Char		Sentence		
	ER	WDR(↑)	ASR(↑)	WDR(↑)	ASR(↑)	ER	WDR(↑)	ASR(↑)
OPT								
KGW	0.1	0.0832	0.0224	0.1274	0.2228	0.6130†	0.2947†	0.7710†
	0.5	0.3349	0.9725	0.4146	0.9949	2.5579*	0.3710*	0.9863*
DIP	0.1	0.1847	0.5917	0.2000	0.6586	0.5832†	0.3539†	0.9784†
	0.5	0.4206	0.9978	0.4226	0.9967	2.4413*	0.4194*	1.0000*
SynthID	0.1	0.1698	0.0709	0.2191	0.1368	0.5068†	0.3587†	0.6229†
	0.5	0.4259	0.9574	0.4486	0.9970	2.4469*	0.4750*	0.9933*
Unigram	0.1	0.0437	0.0520	0.0788	0.0788	0.5911†	0.1540†	0.4377†
	0.5	0.2176	0.8654	0.2663	0.9072	2.3805*	0.2077*	0.6815*
Unbias	0.1	0.1760	0.5684	0.1964	0.6728	0.5797†	0.3682†	0.9823†
	0.5	0.4030	0.9968	0.4151	0.9978	2.4596*	0.4286*	0.9965*
LLaMA								
KGW	0.1	0.0923	0.1359	0.1136	0.2261	0.5240†	0.2461†	0.7376†
	0.5	0.3472	0.9675	0.3747	0.9830	2.3925*	0.3867*	0.9714*
DIP	0.1	0.1724	0.7188	0.1977	0.8150	0.5033†	0.3355†	0.9292†
	0.5	0.4069	0.9988	0.3983	0.9988	2.3680*	0.4234*	0.9875*
SynthID	0.1	0.1499	0.1064	0.1849	0.1824	0.5112†	0.3089†	0.6801†
	0.5	0.3739	0.9686	0.3842	0.9970	2.4235*	0.4200*	0.9933*
Unigram	0.1	0.0243	0.0562	0.0529	0.1564	0.5320†	0.1775†	0.8625†
	0.5	0.1233	0.5366	0.1855	0.8290	2.3342*	0.2328*	0.8812*
Unbias	0.1	0.1812	0.7463	0.1983	0.7750	0.4804†	0.3264†	0.9625†
	0.5	0.4105	0.9938	0.3963	0.9963	2.3913*	0.4217*	0.9958*

TABLE III: Comparison of AUC scores before (BA) and after applying watermark removal attacks.

		Token			Char		
	BA	ER=0.1	ER=0.3	ER=0.5	ER=0.1	ER=0.3	ER=0.5
OPT							
KGW	1.0000	0.9997	0.9838	0.8898	0.9990	0.9305	0.7043
DIP	1.0000	0.9786	0.7453	0.5700	0.9714	0.7091	0.5670
SynthID	0.9999	0.9947	0.8544	0.6337	0.9904	0.7639	0.5528
Unigram	1.0000	1.0000	0.9975	0.9625	0.9999	0.9945	0.9608
Unbias	1.0000	0.9795	0.7393	0.5896	0.9721	0.7142	0.5506
LLaMA							
KGW	1.0000	0.9997	0.9836	0.8758	0.9996	0.9710	0.8447
DIP	0.9998	0.9722	0.7061	0.5783	0.9561	0.6854	0.5702
SynthID	1.0000	0.9933	0.8358	0.6342	0.9869	0.7874	0.6077
Unigram	1.0000	1.0000	0.9982	0.9859	0.9993	0.9879	0.9374
Unbias	0.9999	0.9692	0.7124	0.5748	0.9584	0.7005	0.5651

Unbias [21], using the official implementations provided by MarkLLM [53]. For KGW, Unigram, DIP, and Unbias, we set $\gamma = 0.5$. In KGW and Unigram, γ denotes the proportion of the green list in the vocabulary. In DIP and Unbias, γ represents the cumulative probability of the selected tokens. For KGW and Unigram, we set $\delta = 2$, which indicates the logit bias added to green list tokens. For KGW, we set the context length $h = 1$. Other hyperparameters follow the default configurations in MarkLLM.

2) Implementation Details: We compare our baseline character-level attack with token-level and sentence-level approaches for watermark removal. In the token-level attack, we randomly select tokens and replace them with synonyms generated using Gensim [54]. For sentence-level attacks, we use DIPPER and AuthorMist to rewrite watermarked text. DIPPER is a paraphrasing model specifically designed for watermark removal [32], while AuthorMist [55] is trained via reinforcement learning to paraphrase AI-generated text into a more human-like style, aiming to evade detection while preserving semantic. As an aggressive paraphraser, AuthorMist extensively alters the writing style and vocabulary of the input, resulting in significantly longer outputs with most words modified, and consequently a very high ER. For the character-level attack, we consider five types of visually imperceptible perturbations: (1) **Typo**: replace a character with a nearby keyboard key (e.g., "t" $\rightarrow$ "r"). (2) **Deletion**: remove the selected character from the token. (3) **Swap**: swap a character with its adjacent one (e.g., "their" $\rightarrow$ "thier"). (4) **Insertion**: insert a zero-width character (e.g., U+200B) or a whitespace at a chosen position. (5) **Homoglyph substitution**: replace a character with a visually similar Unicode homoglyph (e.g., "g" $\rightarrow$ "ğ", U+011F). Typo, Deletion, and Insertion usually create at least 2 sub-tokens. Homoglyph substitution changes often lead to 3 or more sub-tokens. Swap typically produces at least $2 \sim 4$ sub-tokens.

TABLE IV: Comparison of text quality after watermark removal using three types of perturbations across five watermarking schemes. Evaluation is based on BLEU($\uparrow$), ROUGE-F1(RF, $\uparrow$), and PPL rate(PR, $\downarrow$). Sentence-level methods include DIPPER ($\dagger$), with editing rates roughly aligned to ER ≈ 0.5, and AuthorMist ($*$), which yields high and less controllable editing rates.

		Token			Char			Sentence			
	ER	BLEU	RF	PR	BLEU	RF	PR	ER	BLEU	RF	PR
OPT											
KGW	0.1	0.7669	0.9078	1.5886	0.7714	0.8678	1.1198	0.6130†	0.3830†	0.7685†	-0.0425†
	0.5	0.1664	0.5586	15.9911	0.1739	0.4402	1.7028	2.5579*	0.0875*	0.4544*	-0.1708*
DIP	0.1	0.7683	0.9079	1.6342	0.7713	0.8675	1.1080	0.5832†	0.3944†	0.7756†	0.0391†
	0.5	0.1615	0.5584	16.7235	0.1737	0.4395	1.8324	2.4413*	0.0890*	0.4596*	-0.0947*
SynthID	0.1	0.7680	0.9085	1.8014	0.7732	0.8688	1.3514	0.5068†	0.4551†	0.8072†	0.1647†
	0.5	0.1621	0.5572	20.8919	0.1751	0.4425	2.7043	2.4469*	0.1079*	0.4778*	0.2673*
Unigram	0.1	0.7658	0.9075	1.7040	0.7711	0.8669	1.0857	0.5911†	0.3904†	0.7676†	-0.0263†
	0.5	0.1643	0.5517	16.9014	0.1737	0.4351	1.4794	2.3805*	0.0966*	0.4596*	-0.1705*
Unbias	0.1	0.7683	0.9080	1.5725	0.7707	0.8676	1.1270	0.5797†	0.3801†	0.7733†	0.0292†
	0.5	0.1626	0.5580	16.5869	0.1740	0.4391	1.8644	2.4596*	0.0871*	0.4578*	-0.1012*
LLaMA											
KGW	0.1	0.7688	0.9088	1.7966	0.7718	0.8685	0.5560	0.5240†	0.4574†	0.8013†	0.3457†
	0.5	0.1616	0.5570	23.0072	0.1763	0.4412	0.4952	2.3925*	0.1030*	0.4767*	0.1460*
DIP	0.1	0.7680	0.9088	1.8353	0.7711	0.8676	0.5454	0.5033†	0.4800†	0.8111†	0.3818†
	0.5	0.1619	0.5581	24.4797	0.1729	0.4399	0.4720	2.3680*	0.1075*	0.4834*	0.1549*
SynthID	0.1	0.7683	0.9086	1.8186	0.7713	0.8678	0.5212	0.5112†	0.4643†	0.8074†	0.3171†
	0.5	0.1608	0.5570	23.6556	0.1759	0.4413	0.4372	2.4235*	0.1060*	0.4737*	0.0791*
Unigram	0.1	0.7691	0.9098	1.7713	0.7731	0.8699	0.5425	0.5320†	0.4385†	0.7908†	0.2600†
	0.5	0.1602	0.5571	20.2958	0.1786	0.4473	0.4701	2.3342*	0.1006*	0.4790*	0.1947*
Unbias	0.1	0.7683	0.9084	1.8765	0.7724	0.8678	0.5438	0.4804†	0.4799†	0.8112†	0.3939†
	0.5	0.1619	0.5585	24.3046	0.1755	0.4407	0.4522	2.3913*	0.1024*	0.4789*	0.1555*

D. Evaluation

In this section, we first compare the effectiveness of character-, token-, and sentence-level perturbations for watermark removal, showing the superiority of character-level attacks (Section IV-D1). We then evaluate the impact of each perturbation type on text quality (Section IV-D2) and analyze how editing rate affects removal performance (Section IV-D3). We further compare different character-level perturbation operations and find homoglyph insertion to be the most effective (Section IV-D4). In addition, we discuss the frequency-based zero-feedback attack (Section IV-D5). Appendix A-B and A-C present additional results on text length and cross-lingual generalization, respectively.

1) Comparison with Token-Level and Sentence-Level Removal Attacks: Table II compares the effectiveness of watermark removal using token-level, character-level, and sentence-level perturbations across five watermark schemes. Each method is evaluated on texts generated by two LLMs: OPT and LLaMA. For sentence-level attacks, we use the DIPPER and AuthorMist as paraphrasers. DIPPER is configured with a lexical diversity of 20 and an order diversity of 0. These settings yield an ER of approximately 0.5, ensuring a fair comparison with other methods. For token-level and character-level attacks, we report results under both low ($\mathrm{ER} = 0.1$) and high ($\mathrm{ER} = 0.5$) perturbation budgets.

Our results show that character-level attacks consistently outperform both token-level and sentence-level methods across all settings. The advantage of character-level attacks is especially notable under low editing budgets. For instance, on KGW, SynthID, and Unigram watermarks with $\mathrm{ER} = 0.1$, character-level attacks achieve an average ASR of 0.1672, nearly double that of token-level (0.0740), aligning with our analysis in Section IV-A on the benefits of broader attack ranges. For weaker watermark schemes like DIP and Unbias, token- and character-level attacks perform similarly. Both achieve high ASR and WDR across ER settings, indicating token-level perturbations are already effective, with limited gains from character-level attacks.

Following prior work [19], [32]–[34], [45], [46], our previous experiments primarily used the optimal detection thresholds provided by watermark designers to ensure the best detection performance. Further, to evaluate whether our attacks remain effective under varying detection thresholds τ_d, we compare the Area Under the ROC Curve (AUC) of the original watermark detector before and after the attack. Specifically, we treat original non-watermarked texts as negative samples, and watermarked texts and attacked watermarked texts as positive samples before and after the attack, respectively. By sliding τ_d, we compute the true positive rate and false positive rate at each point and the resulting AUC. As shown in Table III, the original detector achieves an average AUC of 1.0 before attack (BA), indicating a strong ability to distinguish watermarked from non-watermarked text. After character-level attacks, the average AUC drops to 0.9833, 0.8244, and 0.6861 at $\mathrm{ER} = 0.1, 0.3, 0.5$, respectively; for token-level attacks, the corresponding AUCs are 0.9887, 0.8556, and 0.7294. These results indicate that, at the same ER, character-level attacks consistently achieve lower AUC values, suggesting better removal performance.

2) Impact on Text Quality: Table IV evaluates the impact of watermark removal on text quality across three types of perturbations using OPT-generated watermarked text. We evaluate text quality using three widely adopted metrics: BLEU [56], ROUGE-F1 [57], and perplexity rate (PPL rate $= \frac{\mathrm{PPL}(\tilde{X}) - \mathrm{PPL}(X)}{\mathrm{PPL}(X)}$), where PPL is measured by the victim LLM. Higher BLEU and ROUGE-F1 indicate better preservation of semantic quality, while lower PPL rate reflects better fluency. Results are reported at editing rates $\mathrm{ER} = 0.1$ and 0.5, using 100 token inputs. Character-level attacks achieve comparable

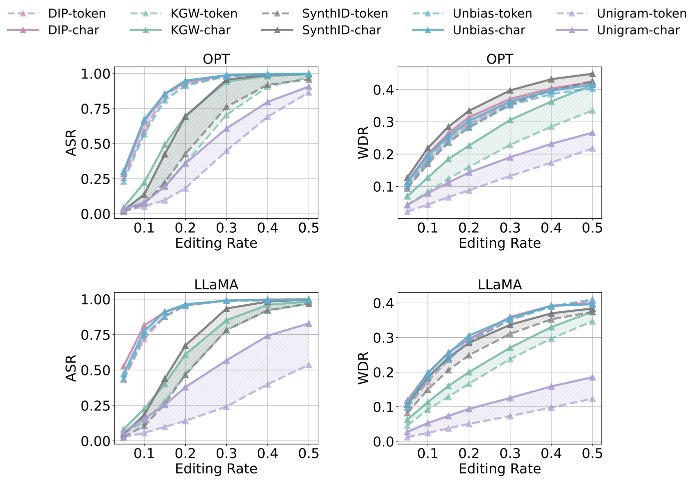

Fig. 3: ASR and WDR of token- and character-level attacks at varying $\mathrm{ER} \in [0.05, 0.5]$, evaluated on 100-token texts.

TABLE V: Performance of frequency-based zero-feedback watermark removal attacks in the **AC1** setting.

		Token (OPT)		Char (OPT)		Token (LLaMA)		Char (LLaMA)	
	ER	WDR(↑)	ASR(↑)	WDR(↑)	ASR(↑)	WDR(↑)	ASR(↑)	WDR(↑)	ASR(↑)
KGW	0.1	0.0880	0.0788	0.1542	0.3322	0.1003	0.1893	0.1340	0.2786
	0.5	0.3457	0.9623	0.4045	0.9932	0.3677	0.9643	0.3995	0.9821
DIP	0.1	0.1794	0.5827	0.2049	0.7050	0.1928	0.7167	0.2085	0.7542
	0.5	0.4239	0.9964	0.4135	0.9964	0.4287	1.0000	0.4195	0.9917
SynthID	0.1	0.1818	0.0673	0.2367	0.1481	0.1668	0.1077	0.2065	0.1919
	0.5	0.4611	0.9596	0.4824	1.0000	0.3956	0.9562	0.4120	0.9966
Unigram	0.1	0.0982	0.1336	0.1249	0.2705	0.0934	0.3716	0.0493	0.1609
	0.5	0.4747	0.9966	0.4530	0.9932	0.4548	0.9962	0.2589	0.9693
Unbias	0.1	0.1820	0.5603	0.2090	0.6773	0.1911	0.7208	0.2019	0.7792
	0.5	0.4267	0.9965	0.4113	0.9965	0.4123	0.9958	0.4117	1.0000

or superior semantic fidelity to token-level attacks, especially under higher perturbation. For OPT, at $\mathrm{ER} = 0.1$, their average BLEU/ROUGE-F1 scores (0.7715/0.8677) are close to token-level (0.7675/0.9080); at $\mathrm{ER} = 0.5$, the scores (0.1741/0.4393) remain similar to token-level (0.1634/0.5568), though ROUGE-F1 is slightly lower. This is because ROUGE is based on n-gram overlap, and character-level perturbations increase token fragmentation, thereby decreasing the proportion of overlapping tokens. Character-level attacks also yield much lower perplexity, indicating better fluency. At $\mathrm{ER} = 0.1$, the average PPL rate is 1.1584 compared to 1.6601 for token-level. The same pattern can also be observed on watermarked texts generated by LLaMA. Although sentence-level attacks produce more fluent text, they require a very high editing rate to be effective.

3) Impact of Editing Rate: Figure 3 compares the effectiveness of token-level and character-level watermark removal attacks under different ER. Overall, character-level attacks consistently outperform token-level attacks in both attack success rate (ASR) and watermark score dropping rate (WDR), across all watermark schemes and editing rates. This performance gap is especially evident for KGW, Unigram, and SynthID when $\mathrm{ER} \leq 0.2$, where character-level attacks achieve a significantly higher average ASR than token-level ones (0.2840 vs. 0.1461 for OPT; 0.2197 vs. 0.1325 for LLaMA). These findings align with our earlier analysis (Section IV-A), which highlights the broader impact range of character-level attacks. In summary, character-level attacks are more effective than token-level attacks at removing watermarks, making them a more reliable tool for evaluating watermark robustness.

4) Comparison Among Different Character Perturbations: Figure 4 presents the ASR of five character-level perturbation types: Deletion, Homoglyph substitution, Insertion, Swap, and Typo, across five watermark schemes as the editing rate increases. Overall, all perturbations show improved ASR with increasing editing rates. Notably, Homoglyph substitution consistently achieves higher ASR than other methods, especially at lower editing rates (e.g., 0.05 and 0.1). This effectiveness can be explained by the attack range analysis in Section IV-A. Homoglyph substitution tends to disrupt tokenization more severely by splitting a single token into at least three subword tokens. In contrast, other perturbations (e.g., deletion, insertion of whitespace or zero-width characters, swap, and typo) typically cause only a two-subword split. Therefore, we adopt homoglyph substitution as the primary character-level type in the following study.

5) Frequency-Based Zero-Feedback Attack: In this experiment, we aim to evaluate whether general knowledge about watermarks can enhance removal effectiveness under the **AC1**. As discussed in Section II-B, watermark schemes affect the frequency of generated tokens, we therefore use token frequency as general knowledge for removal. Specifically, we compute each token's frequency in watermarked (fq_w) and non-watermarked (fq_n) texts, and define the frequency metric as $\mathrm{fq}_w/\mathrm{fq}_n$. Tokens in each text are ranked by this metric in descending order, and perturbations are applied until the target editing rate (ER) is reached. For character-level attacks, perturbations target high-ranking tokens directly. For token-level attacks, we substitute high-ranking tokens with their lower-frequency synonyms.

Table V shows the effectiveness of this strategy. The results show that character-level attacks still outperform token-level attacks across all settings. Compared to the random strategy in Equation (2) and Table II, this frequency-based approach yields similar performance in terms of ASR and WDR for all watermarks except Unigram. When $\mathrm{ER} = 0.1$, the average ASR improvement over the random strategy is only 0.009 (OPT) and 0.0685 (LLaMA) for token-level; 0.0429 (OPT) and 0.0020 (LLaMA) for character-level attacks. When $\mathrm{ER} = 0.5$, the average improvement turns to -0.0024 (OPT) and 0.0895 (LLaMA) for token-level; -0.0001 (OPT) and 0.0272 (LLaMA) for character-level attacks, suggesting no consistent benefit from frequency. An exception is the Unigram watermark. Unlike other watermarking methods that use token-specific green lists, it applies a shared green list to all tokens to enhance robustness [20]. This causes the frequency of green tokens in Unigram watermarked text to be significantly higher than others, making the frequency-based attack more effective.

6) Human Evaluation of Text Quality under Random Character-Level Watermark Removal: We conducted a human

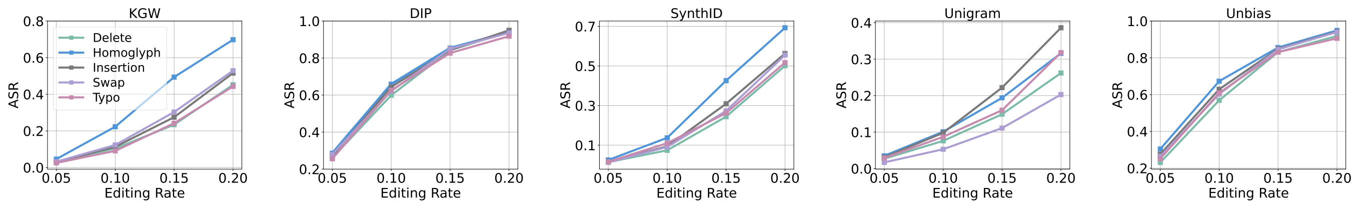

Fig. 4: ASR of five character-level perturbation types, including typos, deletions, swaps, insertions, and homoglyph substitutions, across five watermark schemes. The length of watermarked text is 100 tokens, and they are generated by OPT.

TABLE VI: Human evaluation results for watermarked text and perturbed text from the three types of attacks. The table reports the average scores across all raters for each dimension, along with the 95% confidence interval of the overall score based on the t-distribution. We use CEFR levels [58] to indicate English proficiency (ranging from A1 for beginners to C2 for proficient users), and mark native English speakers with "#".

ID	Age	Gen	CEFR	Edu	Field	WM				Sentence				Token				Char			
						Gram	**Corr**	**Flue**	**Overall**	**Gram**	**Corr**	**Flue**	**Overall**	**Gram**	**Corr**	**Flue**	**Overall**	**Gram**	**Corr**	**Flue**	**Overall**
1	25	F	B2	PhD	IT	2.758	2.636	2.788	2.73±0.16	2.322	2.610	2.593	2.51±0.14	1.298	1.404	1.404	1.37±0.16	2.255	2.294	2.333	2.29±0.19
2	24	M	C2#	Master	Nurs	2.455	2.636	2.636	2.58±0.20	2.424	2.508	2.492	2.47±0.14	1.298	1.263	1.175	1.25±0.14	2.118	2.255	2.333	2.24±0.19
3	35	F	C2#	PhD	Chem	2.576	2.727	2.788	2.70±0.17	2.525	2.441	2.661	2.54±0.14	1.316	1.333	1.298	1.32±0.16	2.118	2.216	2.490	2.27±0.19
4	24	M	C1	PhD	IT	2.727	2.788	2.848	2.79±0.15	2.542	2.508	2.678	2.58±0.14	1.404	1.333	1.316	1.35±0.16	2.157	2.353	2.431	2.31±0.19
5	27	F	C1	Master	Chem	2.424	2.545	2.667	2.55±0.18	2.407	2.390	2.525	2.44±0.15	1.158	1.298	1.439	1.30±0.16	2.157	2.157	2.275	2.20±0.19
6	27	M	C2#	PhD	IT	2.697	2.606	2.515	2.61±0.18	2.254	2.627	2.593	2.49±0.15	1.404	1.263	1.123	1.26±0.16	2.196	2.196	2.314	2.24±0.18
7	26	F	C1	PhD	IT	2.394	2.576	2.758	2.58±0.18	2.475	2.542	2.559	2.53±0.14	1.368	1.175	1.456	1.33±0.16	2.196	2.157	2.235	2.20±0.18
8	33	M	C1	Master	Nurs	2.515	2.424	2.606	2.52±0.25	2.356	2.373	2.542	2.42±0.15	1.351	1.211	1.281	1.28±0.16	1.765	2.275	2.490	2.18±0.18
9	27	F	C2#	PhD	Chem	2.182	2.818	2.818	2.61±0.18	2.441	2.441	2.593	2.49±0.14	1.211	1.193	1.228	1.21±0.15	1.980	2.196	2.412	2.20±0.19
10	30	M	C1	PhD	IT	2.545	2.576	2.788	2.64±0.17	2.610	2.492	2.576	2.56±0.14	1.316	1.211	1.211	1.25±0.14	2.216	2.176	2.373	2.25±0.19
11	21	F	C2#	Master	Nurs	2.545	2.727	2.727	2.67±0.17	2.169	2.475	2.525	2.39±0.15	1.070	1.351	1.158	1.19±0.15	2.314	2.098	2.235	2.22±0.18
12	28	M	B2	PhD	IT	2.788	2.697	2.697	2.73±0.16	2.339	2.593	2.542	2.49±0.14	1.105	1.281	1.351	1.25±0.16	2.039	2.314	2.529	2.29±0.19
13	23	F	C1	Master	IT	2.545	2.667	2.879	2.70±0.17	2.424	2.492	2.610	2.51±0.14	1.386	1.158	1.035	1.19±0.16	2.235	2.275	2.314	2.27±0.18
14	30	M	C1	PhD	Chem	2.727	2.606	2.758	2.70±0.17	2.153	2.576	2.593	2.44±0.14	1.316	1.439	1.088	1.28±0.14	2.176	2.294	2.294	2.25±0.19
15	34	F	C1	PhD	IT	2.212	2.333	2.455	2.33±0.25	2.220	2.136	2.458	2.27±0.17	1.246	1.123	1.000	1.12±0.13	2.275	2.039	2.039	2.12±0.19
Avg						**2.54±0.10**	**2.62±0.07**	**2.72±0.07**	**2.63±0.06**	**2.38±0.08**	**2.48±0.07**	**2.57±0.03**	**2.48±0.04**	**1.28±0.06**	**1.27±0.05**	**1.24±0.08**	**1.26±0.04**	**2.15±0.08**	**2.22±0.05**	**2.34±0.07**	**2.24±0.03**

evaluation to assess visual imperceptibility. Following the setup in [20], [23], we recruited 15 participants, who were selected based on factors including age (20-35), gender (8 female, 7 male), education (5 Master's, 10 PhD), language (all participants are proficient in English, including 5 native English speakers), and professional background (including IT, nursing, chemistry). Each participant rated 200 anonymized texts—including original watermarked samples and outputs from three watermark removal attacks—based on grammaticality/coherence, correctness, fluency, and overall quality, using a 0–3 scale (with higher scores indicating better quality). Table VI reports the average scores and confidence interval from each rater across the four evaluation dimensions. The results show that watermarked texts received the highest average overall score (2.626), followed by sentence-level (2.476) and character-level attacks (2.235), both of which scored significantly higher than token-level attacks (1.263). These results indicate that character-level attacks better preserve text quality and provide stronger visual imperceptibility than token-level attacks.

V. Guided Character-level Attack for LLM Watermark

As discussed in Section III-B, the objective of watermark removal is to reduce the global score $S_w(\tilde{X})$ of a modified text $\tilde{X}$ below the detection threshold. However, a closer look at the watermark injection and detection process in Section II-B and II-C shows that not all token modifications effectively lower $S_w(\tilde{X})$. For example, in KGW [19], only modifications that convert green tokens into red tokens reduce $S_w(\tilde{X})$; conversely, converting red tokens into green ones strengthens the watermark. Therefore, to achieve effective watermark removal with minimal editing rate, it is essential to identify and perturb removal-relevant tokens, i.e., whose modification is most likely to decrease the global watermark score. This requires guidance beyond the random strategy.

In this section, we introduce Genetic Algorithm (GA) to guide watermark removal under the **AC2** setting, where the adversary has limited black-box access to the original watermark detector. Since GA involves iterative evaluation of numerous perturbed candidates, directly querying the original detector throughout the optimization would be impractical under limited query budgets. To address this constraint, we train a lightweight reference detector to approximate the original detector's behavior, allowing the GA to locate removal-relevant tokens even under strict query budgets. We also evaluate the GA under an excessive setting in which the original detector is fully accessible, verifying its ability to identify tokens relevant to watermark removal (refer to Appendix A-D).

A. Genetic Algorithm-Based Attack with Limited Access to the Original Watermark Detector

1) Reference Detector: As discussed in Section II-B, watermarking alters the token selection behavior of LLMs, resulting in measurable statistical deviations between watermarked and non-watermarked outputs. These differences make it feasible to predict the watermark through learned models. We follow the setting of previous work [59], which assumes that the watermark detectors return a confidence score (e.g., the global watermark score). This setting is aligned with real-world

Algorithm 1 GA-based Removal with Reference Detector

```
Require: Watermarked text $X$, the number of token in text $m$, reference detector $D_{\text{ref}}$, iteration rounds $n$, population size $p$, parent size $p_s$, loss threshold $\delta_l$, score threshold $\tau_l$, weight for editing rate $\lambda$, maximum editing rate $\epsilon$.
1: // Filter high-gradient tokens from $\{1, \cdots, m\}$
2: Initial position set $\mathcal{P}$
3: Initial population $\mathbf{P}_1 = \{\tilde{\mathcal{P}}^{(q)}\}_{q=1}^{p}$, $\tilde{\mathcal{P}}^{(q)} \subset \mathcal{P}$, $\frac{|\tilde{\mathcal{P}}^{(q)}|}{m} \leq \epsilon$
4: for iteration $j = 1$ to $n$ do
5:     for $q = 1$ to $p$ do
6:         $\tilde{X}^{(q)} = \mathcal{A}^{C}_{\tilde{\mathcal{P}}^{(q)}}(X), \tilde{\mathcal{P}}^{(q)} \in \mathbf{P}_j$
7:         Compute reference score $w^{(q)} = D_{\text{ref}}(\tilde{X}^{(q)})$
8:         Compute editing rate $e^{(q)} = \text{ER}(\tilde{X}^{(q)}, X)$
9:         if $w^{(q)} > \tau_l$ then
10:            // Stage 1: Minimize reference score only
11:            $\mathcal{L}^{(q)} = w^{(q)}$
12:        else
13:            // Stage 2: Joint optimization
14:            $\mathcal{L}^{(q)} = w^{(q)} - \lambda \cdot (\epsilon - e^{(q)})$
15:        end if
16:    end for
17:    Select parents $Q_j = \text{top-}p_s(\mathbf{P}_j)$ in ascending $\mathcal{L}^{(q)}$
18:    Best perturbed text $\tilde{X} = \arg\min_{\tilde{X}^{(q)}} \mathcal{L}^{(q)}, q \in [1, p]$
19:    $\mathcal{L}_j = \min_{q \in [1,p]}(\mathcal{L}^{(q)})$
20:    if $|\mathcal{L}_j - \mathcal{L}_{j-1}| < \delta_l$ and $j > 1$ then
21:        $Q_j = Q_{j-1}$ // Keep previous parent
22:    end if
23:    Next population: $\mathbf{P}_{j+1} \xleftarrow[\text{crossover}]{\text{mutation}} Q_j$
24: end for
25: return Final perturbed text $\tilde{X}$
```

scenarios, where practical AI detection APIs often return soft predictions (e.g., confidence scores or watermark probabilities) rather than just binary labels [60]–[62]. Motivated by these, we design the reference detector as a regression model that predicts the global watermark score $S_w(X)$ for an input X.

However, reference detectors inherently differ from original detectors. While the original detector is rule-based and tightly coupled with a specific watermarking scheme (Section II-C), the reference detector is data-driven and typically relies on a neural network to approximate its behavior. As a result, they may be overly sensitive to high-gradient tokens, reacting strongly to local changes that have limited effect under the original detector. This mismatch poses challenges for removal strategies guided by the reference detector, particularly those relying on gradient-based optimization [35]–[37] (more analysis about the mismatch refer Appendix A-E).

Takeaway: Gradient-based optimization guided by the reference detector is unreliable for watermark removal.

2) Genetic Algorithm: Since GA is gradient-free, it is less affected by the mismatch between the reference detector and the original detector. Under the **AC2** setting, where access to the original detector is limited, the GA leverages the prediction from the reference detector to identify the removal-relevant positions. Specifically, it seeks a minimal subset of token positions whose perturbation substantially reduces the watermark score predicted by the reference detector D_{ref}. The objective function is defined as follows:

$$\underset{\tilde{P} \subset \{1,\cdots,m\}}{\arg\min} \; D_{\text{ref}}(\mathcal{A}^{C}_{\tilde{P}}(X)) + \lambda \cdot \frac{|\tilde{P}|}{m}, \tag{3}$$

where λ is the weight for editing rate, $\tilde{P} \subset \{1, \cdots, m\}$ is an individual in the GA population that represents a set of token positions of the input text, and m is the number of tokens in X. This objective is consistent with the adversary's goal in Section III-B after applying Lagrange multipliers. In each iteration of GA, it generates perturbed texts, evaluates them with reference detector, and updates the best solution if a significant improvement is observed. The best individuals are selected as parents Q, which are then used to generate the next population through crossover and mutation. The process continues until a maximum number of iterations is reached. To further improve the stability of the optimization process and mitigate the impact of the mismatch between the reference detector and the original detector, we incorporate three key components into the GA framework: filtering high-gradient tokens, a two-stage optimization objective, and a convergence threshold. We summarize the full procedure in Algorithm 1.

Filtering High-Gradient Tokens. As discussed above, the reference detector tends to assign disproportionately large gradients to a few influential tokens, which may not align with the original detector. To mitigate this mismatch, we exclude such high-gradient tokens from the initial population, preventing them from misleading the search process. Specifically, we first compute the gradient magnitude $\|\frac{\partial D_{\text{ref}}(X)}{\partial x_t}\|_2$ for each token x_t with respect to the output of the reference detector D_{ref}. We then calculate the mean μ and standard deviation σ of all gradient values in the text X. Tokens whose gradient magnitudes exceed $\mu + \alpha \cdot \sigma$ are considered high-gradient tokens and are filtered out, where α is a scaling factor controlling the sensitivity of filtering.

Two-Stage Optimization Objective. The GA is designed to jointly optimize two objectives: (i) minimizing the predicted watermark score $D_{\text{ref}}(\tilde{X})$ and (ii) minimizing the editing rate. However, these objectives typically converge at different paces. In practice, the editing rate often decreases more quickly, as reducing the number of perturbations is generally easier than effectively suppressing $D_{\text{ref}}(\tilde{X})$. This early convergence of the editing rate can hinder further reduction in $D_{\text{ref}}(\tilde{X})$, as a smaller editing rate restricts the search space. To address this issue, we adopt a two-stage optimization strategy. In the first stage, the GA focuses solely on minimizing the watermark score until it falls below a predefined threshold τ_l. Once this threshold is reached, the algorithm enters the second stage, where both $D_{\text{ref}}(\tilde{X})$ and editing rate are jointly optimized. The overall objective is formally defined as follows:

$$\mathcal{L} = \begin{cases} D_{\text{ref}}(\tilde{X}), & \text{if } D_{\text{ref}}(\tilde{X}) > \tau_l, \\ D_{\text{ref}}(\tilde{X}) - \lambda \cdot (\epsilon - \text{ER}(X, \tilde{X})), & \text{otherwise}, \end{cases} \tag{4}$$

where λ is the weight for editing rate, ϵ is the upper bond for editing rate.

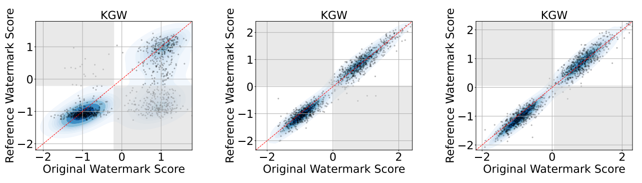

Fig. 5: Scatter plots comparing reference detector predictions (y-axis) with original detector scores (x-axis) for KGW, with Ref-0, Ref-5, and Ref-9 shown from left to right.

Convergence Threshold δ_l. Due to the mismatch between the decision boundaries of the reference detector D_{ref} and the original detector D_{ori}, small decreases in $D_{\text{ref}}(\tilde{X})$ may not correspond to meaningful reductions in $D_{\text{ori}}(\tilde{X})$. Relying solely on marginal improvements in D_{ref} may mislead the optimization process, causing the GA to converge to suboptimal or ineffective perturbed text. To mitigate this issue, we introduce a convergence threshold δ_l (Line 20 in Algorithm 1). If the improvement in loss between two consecutive iterations is less than δ_l, the algorithm retains the previous best solution. This prevents the GA from overreacting to insignificant changes in $D_{\text{ref}}(\tilde{X})$, thereby improving the stability of the optimization.

Best-of-N with Reference Detector. To highlight the advantages of the GA, we introduce a simplified version as a baseline, referred to as the Best-of-N Attack. This method sets iteration rounds in Algorithm 1 as 1. In this case, the adversary generates N perturbed candidates $\{\tilde{X}^{(q)}\}_{q=1}^{N}$, and evaluates them using the reference detector D_{ref}. The candidate with the lowest watermark score $D_{\text{ref}}(\tilde{X}^{(q)})$ is selected and submitted to the original detector as a transfer attack.

B. Experimental Setup

1) Reference Detector: To support guided attacks, we train a separate reference detector for each watermark scheme. For each watermark scheme, we construct a dataset containing 5000 watermarked and non-watermarked text samples. The reference detector is implemented by fine-tuning a BERT [63] regression model to predict the normalized global watermark score. To improve the reliability and generalization of the reference detector, we apply light data augmentation by using token-level and character-level perturbations. Given the limited access under the **AC2** setting, we restrict the augmentation to a small number of variants per sample. Specifically, we consider three configurations for each watermark scheme: Ref-0 (no augmentation), Ref-5 (five augmented variants per sample), and Ref-9 (nine augmented variants per sample).

2) Baseline Method: We adapt the watermark removal method introduced in [45] as a baseline for our study, referring to it as "Sand". In their original design, perturbations are added to the text incrementally over N rounds. At each round, the victim model is used to verify whether the perturbation degrades text quality. If not, the perturbation is accepted. In contrast, our work focuses on evaluating the robustness of watermark schemes under the **AC2** setting, rather than preserving text quality. To accommodate these constraints, we modify their strategy: we still apply perturbations incrementally over N rounds, but at each round, we evaluate whether $D_{\text{ref}}(\tilde{X})$ decreases, if so, the perturbation is accepted.

C. Evaluation

In this section, we present a comprehensive evaluation of our GA-based watermark removal method guided by a reference detector. We begin by assessing its effectiveness across multiple watermarking schemes (Section V-C1), followed by an analysis of the reference detector's quality and its impact on removal performance (Sections V-C2, V-C3, and Appendix A-F). We also evaluate gradient-based attacks using the reference detector (Section V-C4) and discuss the computational complexity and resource overhead of our methods (Section V-C5). Additional ablations are provided in Appendix, including the effect of GA iterations (Appendix A-F1), the choice of N in Best-of-N (Appendix A-F2), and high-gradient token filtering (Appendix A-F3).

1) Performance Comparison of Guided Attack: Table VII presents the effectiveness of the GA-based watermark removal attack, Best-of-N attack and Sand attack [45]. For GA, we set the iteration rounds $n = 15$ and population size $p = 100$, resulting in 1500 queries to the reference detector. For the Best-of-N strategy, we consider two settings: (1) $N = 10$, and (2) $N = 1500$, which uses the same query budget as GA's query budget. All experiments are conducted with text length set to 100, Ref-9 is chosen as the reference detector. The editing rate of all methods is constrained by an upper bound $\epsilon = 0.1$, meaning that each attack can modify at most 10 tokens per text. The results show that GA consistently outperforms both Best-of-N ($N = 10$) and Sand in terms of ASR and WDR. For example, under character-level perturbations, GA achieves ASR scores of 0.6514 on OPT and 0.6615 on LLaMA, significantly higher than Best-of-N (0.4414 / 0.4466) and Sand (0.2622 / 0.3099). Under the same query budget ($N = 1500$), Best-of-N achieves ASR scores of 0.4555 (OPT) and 0.4468 (LLaMA) with character-level perturbations, similar to its performance at $N = 10$. This is due to the lack of optimization and potential mismatch with the reference detector, which limits the benefit of additional queries. In contrast, GA still outperforms Best-of-N under the same query budget, suggesting that GA can more effectively utilize the guidance from reference detectors and mitigate mismatch issues between the reference and original detectors. Additionally, Sand performs worse than Best-of-N under both perturbation types. This is likely due to its incremental design: at each step, Sand applies a small perturbation and accepts it only if the reference detector score decreases. However, due to the mismatch between the reference and original detectors, small perturbations often fail to provide reliable feedback. Moreover, character-level attacks consistently outperform token-level attacks, reaffirming their superior effectiveness in watermark removal.

2) Performance of Reference Detector: Figure 5 visualizes the predicted watermark scores from the reference detectors versus the original detectors for KGW (refer to Appendix A-F4

TABLE VII: Comparison of watermark removal performance across three guided strategies (Best-of-N, Genetic Algorithm, and Sand) using both token-level and character-level perturbations.

		Best-of-N (10) Token		Best-of-N (10) Char		Best-of-N (1500) Token		Best-of-N (1500) Char		GA Token		GA Char		Sand Token		Sand Char	
		WDR(↑)	ASR(↑)	WDR(↑)	ASR(↑)	WDR(↑)	ASR(↑)	WDR(↑)	ASR(↑)	WDR(↑)	ASR(↑)	WDR(↑)	ASR(↑)	WDR(↑)	ASR(↑)	WDR(↑)	ASR(↑)
OPT	KGW	0.0985	0.0842	0.1591	0.2626	0.0956	0.1164	0.1490	0.3082	0.1128	0.1233	0.2110	0.4966	0.0305	0.0103	0.0763	0.0685
	DIP	0.1833	0.5791	0.2033	0.6871	0.1797	0.6151	0.2057	0.6727	0.1864	0.6187	0.2433	0.8453	0.0686	0.0935	0.1624	0.5000
	SynthID	0.1896	0.0606	0.2536	0.1650	0.1886	0.0774	0.2611	0.2121	0.2262	0.1145	0.3093	0.4209	0.1091	0.0168	0.2160	0.0976
	Unigram	0.0639	0.0741	0.1271	0.2907	0.0916	0.1370	0.1508	0.4110	0.1629	0.4829	0.1950	0.6575	0.0382	0.0274	0.0915	0.1164
	Unbias	0.1787	0.5532	0.2067	0.6667	0.1781	0.5496	0.2006	0.6738	0.1864	0.6187	0.2530	0.8369	0.0609	0.1489	0.1717	0.5284
LLaMA	KGW	0.0980	0.1667	0.1216	0.2558	0.0950	0.1750	0.1295	0.2643	0.1176	0.2321	0.1693	0.4214	0.0384	0.0357	0.0849	0.1179
	DIP	0.1827	0.6833	0.2056	0.8175	0.1869	0.7042	0.2116	0.8000	0.1805	0.6917	0.2580	0.9042	0.0889	0.3458	0.1720	0.6708
	SynthID	0.1604	0.1010	0.1915	0.1996	0.1586	0.1010	0.2054	0.2054	0.1779	0.1448	0.2447	0.3603	0.0369	0.0034	0.1601	0.1010
	Unigram	0.0296	0.0651	0.0596	0.1724	0.0292	0.0575	0.0557	0.1686	0.0694	0.1686	0.1888	0.7356	0.0127	0.0345	0.0148	0.0307
	Unbias	0.1843	0.7167	0.2074	0.7875	0.1804	0.6750	0.2055	0.7958	0.1841	0.7208	0.2379	0.8667	0.0749	0.2833	0.1614	0.6292

TABLE VIII: Effect of reference detector quality on watermark removal. We compare $\mathrm{ASR}(\uparrow)$ of GA, Best-of-N, and Sand attacks using three D_{ref}: Ref-0, Ref-5, and Ref-9.

		KGW		DIP		SynthID		Unigram		Unbias	
	Ref	Token	Char	Token	Char	Token	Char	Token	Char	Token	Char
GA	0	0.3112	0.3596	0.3714	0.5504	0.0456	0.1145	0.4169	0.6007	0.4432	0.6418
	5	0.3270	0.3801	0.5276	0.7842	0.0633	0.4209	0.4268	0.6096	0.5431	0.7730
	9	0.3744	0.4966	0.5622	0.8453	0.0535	0.3603	0.4343	0.6575	0.5795	0.8369
Best-of-N	0	0.0924	0.2464	0.5931	0.6623	0.0487	0.1379	0.0915	0.2927	0.5622	0.6395
	5	0.1170	0.2567	0.5714	0.6883	0.0751	0.1866	0.0894	0.2886	0.5451	0.6395
	9	0.0842	0.2626	0.5791	0.6871	0.0606	0.1650	0.0741	0.2907	0.5922	0.7021
Sand	0	0.0068	0.0240	0.0540	0.0288	0.0404	0.0000	0.0274	0.0822	0.1277	0.0461
	5	0.0137	0.0822	0.0683	0.4568	0.0101	0.1178	0.0308	0.1336	0.0745	0.4078
	9	0.0103	0.0685	0.0935	0.5000	0.0168	0.0976	0.0274	0.1164	0.1489	0.5284

TABLE IX: ASR and WDR of gradient-based transfer attacks using TextBugger and DeepWordBug.

	Textbugger		DeepWordBug	
	WDR(↑)	ASR(↑)	WDR(↑)	ASR(↑)
KGW	0.0450	0.0095	0.0422	0.0095
DIP	0.0239	0.0288	0.0293	0.0360
SynthID	0.0748	0.0067	0.0729	0.0067
Unigram	0.0273	0.0189	0.0092	0.0034
Unbias	0.0560	0.0490	0.0293	0.0390

for additional results on other watermark schemes). The x-axis shows the global watermark scores computed by the original detector, while the y-axis shows the predictions of the reference model. The red dashed line ($y = x$) indicates perfect prediction. Shaded regions correspond to misclassified samples, while white regions in the top-right and bottom-left represent true positives and true negatives, respectively. As data augmentation increases, the reference detector's predictions become more aligned with the original scores, and the number of misclassified samples decreases.

3) Impact of Reference Detector in Removal Attack: Table VIII evaluates the impact of reference detector quality, improved through data augmentation, on the effectiveness of watermark removal. We report ASR results on five watermarking schemes (generated by OPT) using three removal methods: GA, Best-of-N, and Sand, each tested with three reference detectors: Ref-0, Ref-5, and Ref-9. As data augmentation increases, the quality of the reference detector improves, resulting in more effective watermark removal. This trend indicates that enhanced detector quality enables more reliable guidance during removal. GA consistently achieves higher ASR than Best-of-N and Sand under the same reference model. Sand is the most sensitive to detector quality, likely due to its incremental strategy: it adds one perturbation at a time and decides whether to retain it based on small changes in

TABLE X: ASR comparison under different adaptive settings against 4 defenses. "GA" denotes the normal GA-based attack in Algorithm 1. "Adaptive GA $(\cdot)$" represents the adaptive GA-based attacks in Equation (5). Detector type indicates whether the victim watermark detector is the original (D_{ori}) or enhanced with defense modules (such as F_{SC}).

	Detector type	KGW	DIP	SynthID	Unigram	Unbias
GA	D_{ori}	0.4214	0.9042	0.3603	0.7548	0.8667
Adaptive GA (SC)	D_{ori}	0.5429	0.9125	0.3644	0.9502	0.9125
	$D_{\text{ori}} \oplus F_{\text{SC}}$	0.4250	0.8917	0.4049	0.8697	0.8917
Adaptive GA (OCR)	D_{ori}	0.5214	0.8333	0.4122	0.5896	0.8333
	$D_{\text{ori}} \oplus F_{\text{OCR}}$	0.5036	0.9500	0.5405	0.4776	0.9333
Adaptive GA (DE)	D_{ori}	0.4507	0.8583	0.4595	0.9776	0.9000
	$D_{\text{ori}} \oplus F_{\text{DE}}$	0.3028	0.9083	0.3311	0.7612	0.8583
Adaptive GA (UN)	D_{ori}	0.4718	0.9333	0.4459	0.9776	0.8750
	$D_{\text{ori}} \oplus F_{\text{UN}}$	0.4507	0.9167	0.4324	0.7016	0.8667

$D_{\text{ref}}(\tilde{X})$. As shown in Figure 5, minor variations in D_{ref} may not accurately reflect the original detector's behavior, which severely limits Sand's effectiveness.

4) Gradient-Based Adversarial Textual Attack is Not Effective: Table IX reports the performance of gradient-based adversarial attacks that attempt to transfer from a reference detector to the original detector. Following our analysis in Section V-A, the inherent mismatch between the reference and original detectors prevents gradient-based methods from reliably identifying watermark-relevant tokens. We evaluate two representative methods: TextBugger [35] and DeepWordBug [37], both of which rank tokens by gradient magnitude and iteratively apply hybrid perturbations (token-level and character-level) to reduce the reference detector's predicted watermark score below a decision threshold using minimal edits. As shown in the table, both methods achieve an $\mathrm{ASR} < 0.1$ across all watermark schemes. These results are significantly lower than those of Best-of-N and GA-based strategies, demonstrating that gradient-based transfer attacks are largely ineffective for watermark removal in this setting.

5) Query Cost Overhead: Our attack is designed for the offline setting, where a reference detector is trained once and reused for multiple attacks. Its training data can be collected incrementally, making the cost flexible and amortizable. To train the reference detector, we collect 5000 watermarked and 5000 non-watermarked samples. We consider three detector variants (Ref-0/5/9), which require 1, 6, and 10 queries per watermarked sample, respectively—resulting in a total of 5000, 30000, and 50000 queries. In comparison, the online

Best-of-N attack uses N queries per sample; at $N = 10$, attacking 5000 samples requires 50000 queries, which is equal to or greater than the number of queries required to train the reference detectors.

In terms of runtime on an NVIDIA A100 GPU, Best-of-N attacks complete within 1s/sample. GA converges in < 20 iterations (each $1 \sim 3$s), with total time < 30s/sample.

VI. Discussion

In this section, we explore potential defenses against character-level watermark removal attacks. A natural defense strategy is to apply preprocessing techniques to reverse character-level perturbations. We consider four representative defense mechanisms: (1) Spell-checking and correction (SC), which automatically detects and corrects spelling errors in the text; (2) Optical character recognition (OCR), which renders the text as an image and re-extracts its content; (3) Unicode normalization (UN), which converts semantically equivalent characters into a standard form to ensure consistent text representation; (4) Deletion (DE) of anomalous characters, which directly removes invalid or suspicious symbols.

In these adaptive scenarios, where the attacker designs perturbations that proactively account for defense mechanisms, watermark robustness presents an adversarial dilemma. For any fixed defense, there always exists at least one perturbation strategy capable of bypassing it. To systematically study this problem, we propose an adaptive compound character-level attack based on a two-level optimization framework. The inner optimization selects, for each token position, the perturbation combination ($C^{\ddagger}$) that minimizes the edit distance ED_C after applying the defense function, indicating that it best bypasses the defense. The outer optimization searches over token subsets to find those whose modification most reduces the watermark score D_{ref}. This can be formalized as:

$$\operatorname*{arg\,min}_{\tilde{P} \subset \{1,\cdots,m\}} D_{\text{ref}}\Big(\operatorname*{arg\,min}_{C^{\ddagger}} \mathrm{ED}_C\big(\tilde{X}, F_{\text{def}}(\tilde{X})\big)\Big) + \lambda \cdot \frac{|\tilde{P}|}{m}, \quad (5)$$

where $\tilde{X} = \mathcal{A}_{\tilde{P}}^{C^{\ddagger}}(X)$, $C^{\ddagger}$ denotes compound character-level perturbations, F_{def} denotes one of OCR, SC, UN, or DE. Compound character-level perturbations are defined as the application of one or more character modifications at the same position. Such as: Swap + homoglyph substitution (e.g., “compound” $\rightarrow$ “compuŏnd”); Typo + homoglyph substitution (e.g., “compound” $\rightarrow$ “compǐund”); Zero-width character insertion + homoglyph substitution (e.g., “compound” $\rightarrow$ “compo{U+200B}ŭnd”).

To the best of our knowledge, existing defenses lack adversarial robustness. However, even when enhanced with robustness mechanisms, they remain vulnerable, because compound character-level perturbations introduce complex distortions that hinder accurate recovery of the original token. The more complex the perturbation, the greater the ambiguity in recovery. Consequently, even if the defense removes suspicious artifacts (e.g., special characters or misspellings), it still cannot reliably recover the original token, thereby disrupting the watermark key and signal.

Evaluation. We evaluate the effectiveness of our adaptive GA-based attack under common character-level defenses. Specifically, we consider widely-used tools as modules: LanguageTool [64] for SC, Python Tesseract [65] for OCR, Python unicodedata [66] for UN.

Table X compares the ASR of two types of GA-based attacks: normal GA and adaptive GA, across five watermark schemes. Each adaptive GA is evaluated under two detector settings: the original detector D_{ori} and the defended detector $D_{\text{ori}} \oplus F_{\text{def}}$, where F_{def} represents specific defense modules (SC, OCR, UN, DE). The maximum editing rate is set to $\epsilon = 0.1$. For SC, DE, and UN, adaptive GA raises the average ASR from 0.6631 to $0.7365, 0.7293, 0.7407$, and after applying these defenses, ASR slightly drops to $0.6966, 0.6323, 0.6736$, respectively, but remains close to the normal GA. This shows that compound character-level perturbations are effective and resistant to these defenses. For OCR, adaptive GA achieves an ASR of 0.6380, slightly lower than normal GA, but applying OCR increases ASR to 0.6810, indicating that the perturbations induce OCR recognition errors that disrupt watermark detection. An exception is the Unigram watermark, where adaptive GA (with OCR) shows a notable ASR drop, as Unigram computes the key independently of context, making it harder to find effective perturbations. Overall, these results demonstrate that our adaptive GA approach can effectively enhance watermark removal while exhibiting resilience to potential character-level defenses.

VII. Conclusion

In this work, we demonstrate that character-level perturbations provide a significantly stronger watermark removal capability than token-level and sentence-level methods, due to their larger attack range. We further show that, under limited access to the original watermark detector, training a reference detector and optimizing perturbation positions using a Genetic Algorithm can offer effective guidance for watermark removal. This enables adversaries to successfully remove watermarks with a low perturbation budget. To address potential defense mechanisms such as spell checking and OCR, we propose an adaptive strategy by using compound perturbations. Overall, our findings reveal that the vulnerability of current LLM watermarking schemes has been substantially underestimated.

Our results highlight the urgent need for dedicated defenses against watermark removal attacks. One promising research direction is to enhance robustness against such attacks by improving tokenization strategies and watermark schemes. Moreover, while this work primarily focuses on watermark removal, we also see research opportunities in watermark spoofing. With improved guidance techniques, adversaries may exploit character-level perturbations to spoof watermark and falsely attribute texts to LLMs.

Acknowledgments

The first two authors are funded by the China Scholarship Council (CSC) from the Ministry of Education, China. Leo’s work is partially supported by the Australian Research Council (ARC) under grant numbers LP240100315 and DP250102634.

References

[1] S. Zhang, S. Roller, N. Goyal, M. Artetxe, M. Chen, S. Chen, and et al., "Opt: Open pre-trained transformer language models," *arXiv preprint arXiv:2205.01068*, 2022.

[2] H. Touvron, T. Lavril, G. Izacard, X. Martinet, M.-A. Lachaux, T. Lacroix, B. Rozière, N. Goyal, E. Hambro, F. Azhar, *et al.*, "Llama: Open and efficient foundation language models," *arXiv preprint arXiv:2302.13971*, 2023.

[3] Openai, "Openai chatgpt blog," 2022. https://openai.com/blog/chatgpt.

[4] Deepseek, "Deepseek terms of use," 2025. https://cdn.deepseek.com/policies/en-US/deepseek-terms-of-use.html (Last Update: January 20, 2025).

[5] Google, "Gemini api additional terms of service," 2025. https://ai.google.dev/gemini-api/terms (Effective April 3rd, 2025).

[6] Grammarly, "Grammarly." https://www.grammarly.com/ (Accessed: 2025-04-22).

[7] Chatpaper, "Chatpaper." https://chatpaper.com/chatpaper/ (Accessed: 2025-04-22).

[8] Deepmind, "Transforming the future of music creation." https://deepmind.google/discover/blog/transforming-the-future-of-music-creation/ (Accessed: 2025-04-22).

[9] Deepmind, "Quickstart: Generate and verify an image's watermark using imagen text-to-image (console)." https://cloud.google.com/vertex-ai/generative-ai/docs/image/quickstart-image-generate-console (Accessed: 2025-04-22).

[10] Z. Zhang, L. Y. Zhang, X. Zheng, J. Tian, and J. Zhou, "Self-supervised adversarial example detection by disentangled representation," in *2022 IEEE International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom)*, pp. 1000–1007, IEEE, 2022.

[11] Z. Zhang, L. Y. Zhang, X. Zheng, B. A. Hussain, and S. Hu, "Evaluating membership inference through adversarial robustness," *The Computer Journal*, vol. 65, no. 11, pp. 2969–2978, 2022.

[12] Z. Gong, Y. Zhang, L. Y. Zhang, Z. Zhang, Y. Xiang, and S. Pan, "Not all edges are equally robust: Evaluating the robustness of ranking-based federated learning," in *2025 IEEE Symposium on Security and Privacy (SP)*, pp. 2527–2545, IEEE, 2025.

[13] R. Sun, H. Hu, W. Luo, Z. Zhang, Y. Zhang, H. Yuan, and L. Y. Zhang, "When better features mean greater risks: The performance-privacy trade-off in contrastive learning," in *ASIA CCS*, 2025.

[14] H. Zhang, B. Wu, X. Yang, X. Yuan, and et al., "Dynamic graph unlearning: a general and efficient post-processing method via gradient transformation," in *Proceedings of the ACM on Web Conference*, pp. 931–944, 2025.

[15] H. Zhang, X. Yuan, and S. Pan, "Unraveling privacy risks of individual fairness in graph neural networks," in *ICDE*, pp. 1712–1725, IEEE, 2024.

[16] Y. Pan, L. Pan, W. Chen, P. Nakov, M.-Y. Kan, and W. Y. Wang, "On the risk of misinformation pollution with large language models," in *Findings of the EMNLP*, 2023.

[17] J. Hazell, "Large language models can be used to effectively scale spear phishing campaigns," *arXiv preprint arXiv:2305.06972*, 2023.

[18] E. Kasneci, K. Seßler, S. Küchemann, M. Bannert, D. Dementieva, F. Fischer, U. Gasser, G. Groh, S. Günnemann, E. Hüllermeier, *et al.*, "Chatgpt for good? on opportunities and challenges of large language models for education," *Learning and individual differences*, vol. 103, p. 102274, 2023.

[19] J. Kirchenbauer, J. Geiping, Y. Wen, J. Katz, I. Miers, and T. Goldstein, "A watermark for large language models," in *ICML*, pp. 17061–17084, PMLR, 2023.

[20] X. Zhao, P. V. Ananth, L. Li, and Y.-X. Wang, "Provable robust watermarking for AI-generated text," in *International Conference on Learning Representations*, 2024.

[21] Z. Hu, L. Chen, X. Wu, Y. Wu, H. Zhang, and H. Huang, "Unbiased watermark for large language models," in *International Conference on Learning Representations*, 2024.

[22] Y. Wu, Z. Hu, J. Guo, H. Zhang, and H. Huang, "A resilient and accessible distribution-preserving watermark for large language models," in *ICML*, 2024.

[23] S. Dathathri, A. See, S. Ghaisas, P.-S. Huang, R. McAdam, and et al., "Scalable watermarking for identifying large language model outputs," *Nature*, vol. 634, no. 8035, pp. 818–823, 2024.

[24] R. Kuditipudi, J. Thickstun, T. Hashimoto, and P. Liang, "Robust distortion-free watermarks for language models," *Transactions on Machine Learning Research*, 2024.

[25] M. Christ, S. Gunn, and O. Zamir, "Undetectable watermarks for language models," in *The Thirty Seventh Annual Conference on Learning Theory*, pp. 1125–1139, PMLR, 2024.

[26] X. Feng, H. Zhang, Y. Zhang, L. Y. Zhang, and S. Pan, "Bimark: Unbiased multilayer watermarking for large language models," in *ICML*, 2025.

[27] X. Feng, J. Liu, K. Ren, and C. Chen, "A certified robust watermark for large language models," *arXiv preprint arXiv:2409.19708*, 2024.

[28] Q. Wu and V. Chandrasekaran, "Bypassing LLM watermarks with color-aware substitutions," in *Proceedings of the 62nd ACL (Volume 1: Long Papers)*, pp. 8549–8581, ACL, Aug. 2024.

[29] B. Huang, X. Pu, and X. Wan, "b^4: A black-box scrubbing attack on LLM watermarks," in *Proceedings of the NAACL (Volume 1: Long Papers)*, pp. 9113–9126, ACL, Apr. 2025.

[30] R. Chen, Y. Wu, J. Guo, and H. Huang, "De-mark: Watermark removal in large language models," in *ICML*, 2025.

[31] H. Chang, H. Hassani, and R. Shokri, "Watermark smoothing attacks against language models," *arXiv preprint arXiv:2407.14206*, 2024.

[32] K. Krishna, Y. Song, M. Karpinska, and et al., "Paraphrasing evades detectors of ai-generated text, but retrieval is an effective defense," in *NeurIPS*, vol. 36, pp. 27469–27500, Curran Associates, Inc., 2023.

[33] J. Kirchenbauer, J. Geiping, Y. Wen, M. Shu, and et al., "On the reliability of watermarks for large language models," in *International Conference on Learning Representations*, 2024.

[34] J. Piet, C. Sitawarin, V. Fang, N. Mu, and D. Wagner, "Markmywords: Analyzing and evaluating language model watermarks," in *2025 IEEE Conference on Secure and Trustworthy Machine Learning (SaTML)*, pp. 68–91, 2025.

[35] J. Li, S. Ji, T. Du, B. Li, and T. Wang, "Textbugger: Generating adversarial text against real-world applications," in *Proceedings of the Network and Distributed System Security Symposium*, 2018.

[36] J. Morris, E. Lifland, J. Y. Yoo, J. Grigsby, D. Jin, and Y. Qi, "Textattack: A framework for adversarial attacks, data augmentation, and adversarial training in nlp," in *Proceedings of the EMNLP*, pp. 119–126, 2020.

[37] J. Gao, J. Lanchantin, M. L. Soffa, and Y. Qi, "Black-box generation of adversarial text sequences to evade deep learning classifiers," in *2018 IEEE Security and Privacy Workshops (SPW)*, pp. 50–56, IEEE, 2018.

[38] X. Zhang, Z. Zhang, Y. Zhang, X. Zheng, L. Y. Zhang, S. Hu, and S. Pan, "Exploring gradient-guided masked language model to detect textual adversarial attacks," *arXiv preprint arXiv:2504.08798*, 2025.

[39] X. Zhang, Z. Zhang, Q. Zhong, X. Zheng, Y. Zhang, S. Hu, and L. Y. Zhang, "Masked language model based textual adversarial example detection," in *ASIA CCS*, p. 925–937, 2023.

[40] N. Jovanović, R. Staab, and M. Vechev, "Watermark stealing in large language models," in *ICML*, 2024.

[41] Z. Zhang, X. Zhang, Y. Zhang, L. Y. Zhang, C. Chen, S. Hu, A. Gill, and S. Pan, "Stealing watermarks of large language models via mixed integer programming," in *2024 Annual Computer Security Applications Conference (ACSAC)*, pp. 46–60, IEEE Computer Society, Dec. 2024.

[42] Z. He, B. Zhou, H. Hao, A. Liu, and et al., "Can watermarks survive translation? on the cross-lingual consistency of text watermark for large language models," in *Proceedings of the 62nd ACL (Volume 1: Long Papers)*, pp. 4115–4129, ACL, Aug. 2024.

[43] Z. Liu, T. Cong, X. He, and Q. Li, "On evaluating the performance of watermarked machine-generated texts under adversarial attacks," *arXiv preprint arXiv:2407.04794*, 2024.

[44] A. Creo and S. Pudasaini, "Silverspeak: Evading ai-generated text detectors using homoglyphs," in *Proceedings of the 1st Workshop on GenAI Content Detection (GenAIDetect)*, pp. 1–46, 2025.

[45] H. Zhang, B. L. Edelman, D. Francati, D. Venturi, G. Ateniese, and B. Barak, "Watermarks in the sand: impossibility of strong watermarking for language models," in *ICML*, 2024.

[46] J. Liang, Z. Wang, L. Hong, S. Ji, and T. Wang, "Watermark under fire: A robustness evaluation of llm watermarking," *arXiv preprint arXiv:2411.13425*, 2024.

[47] T. Cong, X. He, and Y. Zhang, "Sslguard: A watermarking scheme for self-supervised learning pre-trained encoders," in *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications*

Security (CCS), p. 579–593, Association for Computing Machinery, 2022.
[48] H. Jia, C. A. Choquette-Choo, V. Chandrasekaran, and N. Papernot, "Entangled watermarks as a defense against model extraction," in *30th USENIX Security Symposium*, pp. 1937–1954, USENIX Association, Aug. 2021.
[49] C. Raffel, N. Shazeer, A. Roberts, K. Lee, S. Narang, M. Matena, Y. Zhou, W. Li, and P. J. Liu, "Exploring the limits of transfer learning with a unified text-to-text transformer," *Journal of Machine Learning Research*, vol. 21, no. 140, pp. 1–67, 2020.
[50] Deepmind, "Synthid." https://deepmind.google/technologies/synthid/ (Accessed: 2025-04-22).
[51] Openai, "Terms of use," 2024. https://openai.com/policies/row-terms-of-use/ (Published: December 11, 2024).
[52] A. Diaa, T. Aremu, and N. Lukas, "Optimizing adaptive attacks against content watermarks for language models," in *The 1st Workshop on GenAI Watermarking*, 2025.
[53] L. Pan, A. Liu, Z. He, Z. Gao, X. Zhao, Y. Lu, B. Zhou, S. Liu, X. Hu, L. Wen, *et al.*, "Markllm: An open-source toolkit for llm watermarking," in *Proceedings of the EMNLP: System Demonstrations*, 2024.
[54] R. Řehůřek and P. Sojka, "Software Framework for Topic Modelling with Large Corpora," in *Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks*, pp. 45–50, ELRA, May 2010. http://is.muni.cz/publication/884893/en.
[55] I. David and A. Gervais, "Authormist: Evading ai text detectors with reinforcement learning," *arXiv preprint arXiv:2503.08716*, 2025.
[56] K. Papineni, S. Roukos, T. Ward, and W.-J. Zhu, "Bleu: a method for automatic evaluation of machine translation," in *Proceedings of the 40th ACL*, pp. 311–318, 2002.
[57] C.-Y. Lin, "Rouge: A package for automatic evaluation of summaries," in *Text summarization branches out*, pp. 74–81, 2004.
[58] C. of Europe, "The cefr levels." https://www.coe.int/en/web/common-european-framework-reference-languages/level-descriptions (Accessed: 2025-07-28).
[59] Q. Pang, S. Hu, W. Zheng, and V. Smith, "No free lunch in LLM watermarking: Trade-offs in watermarking design choices," in *NeurIPS*, 2024.
[60] OpenAI, "New ai classifier for indicating ai-written tex," 2023. https://openai.com/index/new-ai-classifier-for-indicating-ai-written-text/ (Accessed: 31 Jan. 2023).
[61] Grammarly, "Ai detector by grammarly." https://www.grammarly.com/ai-detector (Accessed: 2 April. 2025).
[62] GPTZero, "Gptzero's ai detection technology." https://gptzero.me/technology#how-ai-detection-works (Accessed: 2 April. 2025).
[63] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," in *Proceedings of the NAACL (volume 1: long and short papers)*, pp. 4171–4186, 2019.
[64] LanguageTool, "Languagetool." https://github.com/jxmorris12/language_tool_python (Accessed: 2025-04-22).
[65] S. Hoffstaetter, "Python tesseract." https://github.com/madmaze/pytesseract (Accessed: 2025-04-22).
[66] Python, "Unicodedata." https://docs.python.org/3/library/unicodedata.html (Accessed: 2025-07-22).

APPENDIX A

A. Related Work

Some works have evaluated the robustness of various watermarks against character-level perturbations — such as typos, misspellings, and homoglyph substitutions — demonstrating that these attacks can be effective at degrading detection performance [19], [34], [43], [44], [46]. However, these works do not investigate the underlying reasons why character-level attacks achieve higher removal success, nor do they explore more sophisticated attacks that systematically exploit the advantages of such perturbations. Liang et al. proposed an adaptive black-box transfer attack by using a gradient-based textual adversarial attack [46]. Due to the mismatch between the decision boundary of the reference detector and the original watermark detector, their method relies on injecting substantial noise to ensure effectiveness. Stealing detailed information of watermark schemes is another option of guided removal attack [28], [30], [40], [41]. While effective in some scenarios, these attacks are computationally expensive and require strong assumptions (i.e., adversaries need to know detailed information about watermark schemes). Specifically, each stealing-based method is tailored to a narrow class of watermark schemes, limiting their general applicability in practice.

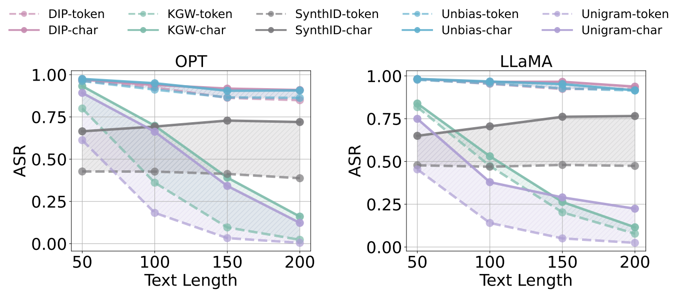

Fig. 6: ASR of token-level and character-level watermark removal attacks under varying text lengths (from 50 to 200 tokens) with $\mathrm{ER} = 0.2$. Solid lines represent character-level attacks, and dashed lines represent token-level attacks.

B. Impact of Text Length for Random Watermark Removal

Figure 6 shows how input length (ranging from 50 to 200 tokens) affects the ASR at a fixed editing rate of $\mathrm{ER} = 0.2$. For each watermarking scheme, character-level attack results are shown as solid lines, and token-level attacks as dashed lines. Character-level attacks consistently outperform token-level attacks across all input lengths and watermark schemes. This performance gap is especially evident for KGW, Unigram, and SynthID. Notably, ASR decreases with increasing input length for KGW and Unigram, but remains relatively stable, or even slightly increases for SynthID. This behavior stems from differences in detection mechanisms. As described in Section II-C, KGW and Unigram use a z-test formulation where the denominator depends on text length m, meaning longer texts require a greater proportion of green tokens to be detected as watermarked. In contrast, SynthID aggregates per-token watermark scores, making it less sensitive to text length under fixed ER. For DIP and Unbias, ASR remains consistently high across all lengths, suggesting weaker robustness.

C. Generalization to Other Languages

Beyond English, we further evaluated whether our approach generalizes to other Latin-script languages. Table XI presents results on French text generated by LLaMA and watermarked by KGW and SynthID. We compare the effectiveness of token-level and character-level attacks under the **AC1** setting. Specifically, we evaluate text lengths of 100 and 200 tokens, and for each length, we test two editing rates: 0.1 and 0.5. The results show that character-level attacks consistently achieve higher ASR than token-level attacks under the same editing

TABLE XI: Watermark removal performance on French watermarked text under the **AC1** threat model. We evaluate both token-level and character-level attacks on LLaMA-generated text embedded with KGW and SynthID watermarks. Results are reported with input lengths of 100 and 200 tokens, and ER of 0.1 and 0.5.

	ER	Token		Char	
		WDR(↑)	ASR(↑)	WDR(↑)	ASR(↑)
KGW (100)	0.1	0.0693	0.0736	0.0940	0.1207
	0.5	0.2865	0.9264	0.3472	0.9816
KGW (200)	0.1	0.0674	0.0040	0.0907	0.0080
	0.5	0.2740	0.7062	0.3304	0.9135
SynthID (100)	0.1	0.1342	0.0825	0.1719	0.1538
	0.5	0.3802	0.9537	0.3953	1.0000
SynthID (200)	0.1	0.1596	0.0584	0.1889	0.0966
	0.5	0.4359	0.9879	0.4547	1.0000

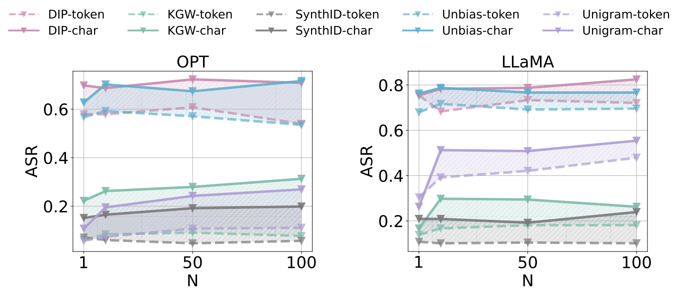

Fig. 7: ASR comparison under the Best-of-N attack with increasing $N \in \{1, 10, 50, 100\}$ for five watermark schemes. Evaluated using Ref-9 as the reference detector and editing rate ER $= 0.1$. Solid lines represent character-level attacks; dashed lines represent token-level attacks.

rate. This trend holds across both watermark schemes and text lengths. These findings are consistent with our observations on English data: character-level perturbations effectively remove watermarks with minimal edit distance by disrupting the tokenization process.

D. Genetic Algorithm-Based Attack with Unlimited Access to the Original Watermark Detector

We begin by evaluating the GA-based attack in an excessive setting, where the adversary has unrestricted access to the original watermark detector D_{ori}. Due to the use of pseudorandom functions in its detection rules (refer to Section II-C), D_{ori} is non-differentiable. So we adopt a Genetic Algorithm (GA) to optimize it. In this setting, the GA directly leverages the true detector scores to identify the most impactful perturbation positions for watermark removal. Specifically, it aims to find a minimal subset of token positions whose perturbation leads to a substantial reduction in the watermark score predicted by the original detector D_{ori}. The objective function is defined as:

$$\underset{\tilde{P} \subset \{1, \cdots, m\}}{\arg\min} \; D_{\text{ori}}(\mathcal{A}^{C}_{\tilde{P}}(X)) + \lambda \cdot \frac{|\tilde{P}|}{|X|}, \tag{6}$$

where λ is the weight for editing rate, $\tilde{P} \subset \{1, \cdots, m\}$ is an individual in the GA population that represents a set of token positions of the input text, and m is the number of tokens in X. In each iteration of GA, it generates perturbed texts, evaluates their watermark scores using D_{ori}, and updates the best solution if a significant improvement is observed. The best individuals are selected as parents Q, which are then used to generate the next population through crossover and mutation. The process continues until a maximum number of iterations is reached. We summarize the full procedure in Algorithm 2.

Algorithm 2 GA-based Removal with Original Detector

```
Require: Watermarked text $X$, the token number of text $m$, original
    detector $D_{\text{ori}}$, iteration rounds $n$, population size $p$, parent size
    $p_s$, weight for editing rate $\lambda$
 1: Initialize population $\mathbf{P}_1 = \{\tilde{\mathcal{P}}^{(q)}\}_{q=1}^{p}, \tilde{\mathcal{P}}^{(q)} \subset \{1, \cdots, m\}$
 2: for iteration $j = 1$ to $n$ do
 3:     for $q = 1$ to $p$ do
 4:         $\tilde{X}^{(q)} = \mathcal{A}^{C}_{\tilde{\mathcal{P}}^{(q)}}(X), \tilde{\mathcal{P}}^{(q)} \in \mathbf{P}_j$
 5:         Compute reference score $w^{(q)} = D_{\text{ori}}(\tilde{X}^{(q)})$
 6:         Compute editing rate $e^{(q)} = \text{ER}(\tilde{X}^{(q)}, X)$
 7:         $\mathcal{L}^{(q)} = w^{(q)} + \lambda \cdot e^{(q)}$
 8:     end for
 9:     Select parents $Q_j = \text{top-}p_s(\mathbf{P}_j)$ in ascending $\mathcal{L}^{(q)}$
10:     Best perturbed text $\tilde{X} = \arg\min_{\tilde{X}^{(q)}} \mathcal{L}^{(q)}, q \in [1, p]$
11:     Next population: $\mathbf{P}_{j+1} \xleftarrow[\text{crossover}]{\text{mutation}} Q_j$
12: end for
13: return Final perturbed text $\tilde{X}$
```

Best-of-N Attack. To highlight the advantages of the full Genetic Algorithm (GA), we introduce a simplified version as a baseline, referred to as the Best-of-N Attack. This method removes the iterative refinement of GA and instead performs a one-shot random search. Specifically, Best-of-N corresponds to a special case of the GA-based approach in Algorithm 2, where the number of optimization iterations is set to 1. In this setting, N candidate texts are generated using the random perturbation strategy defined in Equation (2), and the one with the lowest watermark score $D_{\text{ori}}(\tilde{X})$ is selected. As N increases, the likelihood of sampling removal-relevant tokens improves, leading to stronger candidates for watermark removal. The strategy is formally defined as follows:

$$\begin{gathered}\tilde{X} = \underset{\substack{\tilde{X}^{(q)} \in \mathcal{B}_{\epsilon}(X) \\ q \in [1, N]}}{\arg\min} \; D_{\text{ori}}\left(\tilde{X}^{(q)}\right), \\ \tilde{X}^{(q)} = \mathcal{A}^{C}_{\tilde{P}^{(q)}}(X), \quad \tilde{P}^{(q)} \xleftarrow{\text{rand}} \{1, \cdots, m\},\end{gathered} \tag{7}$$

where $\mathcal{A}^{C}_{\tilde{P}^{(q)}}$ denotes the character-level perturbation method introduced in Section IV-B, which applies perturbations to the central characters of tokens at randomly sampled positions. The set $\mathcal{B}$ represents the collection of candidate texts constrained by a maximum editing rate, ER $\leq \epsilon$. Although simple, this approach lacks a guidance and relies entirely on random sampling. In contrast, the full GA performs guided selection and iterative refinement, allowing it to consistently identify more effective perturbation positions at a lower editing cost.

Evaluation. Table XII compares the watermark removal effectiveness of Best-of-N and the Genetic Algorithm under the setting where the original watermark detector is fully

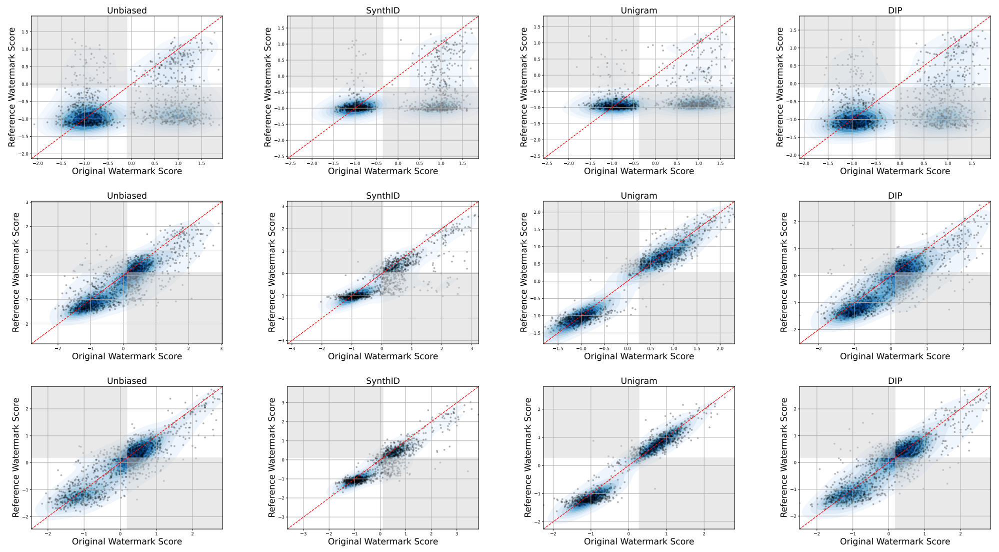

Fig. 8: Scatter plots comparing predicted watermark scores from the reference detector (y-axis) against ground-truth scores from the original detector (x-axis) for the rest watermark schemes (columns). Each row corresponds to a reference model trained with increasing levels of data augmentation: Ref-0 (top), Ref-5 (middle), and Ref-9 (bottom). The red dashed line represents perfect prediction ($y = x$). White regions indicate correct classifications (top-right and bottom-left), while shaded regions show misclassified samples.

TABLE XII: Comparison of Best-of-N and Genetic Algorithm attacks under unlimited access to the original watermark detector. The table reports WDR and ASR for each watermark scheme. Best-of-N is evaluated with different values of $N = 1, 10, 50, 500$. GA uses a population size of 100 and 5 iterations (query budget: $5 * 100$), matching the query budget of Best-of-N (500). GA consistently achieves higher performance, even when the total number of detector queries is matched.

	KGW		DIP		SynthID		Unigram		Unbias	
	WDR(↑)	ASR(↑)	WDR(↑)	ASR(↑)	WDR(↑)	ASR(↑)	WDR(↑)	ASR(↑)	WDR(↑)	ASR(↑)
Best-of-N (1)	0.1278	0.2238	0.2045	0.6883	0.2165	0.1318	0.1090	0.2053	0.1959	0.6223
Best-of-N (10)	0.1958	0.5503	0.3103	0.9805	0.2961	0.4645	0.1616	0.4797	0.3108	0.9828
Best-of-N (50)	0.2278	0.7372	0.3634	0.9978	0.3308	0.6815	0.1845	0.6280	0.3631	0.9979
Best-of-N (500)	0.2602	0.8706	0.4142	1.0000	0.3978	0.8451	0.2111	0.7534	0.4171	1.0000
GA ($5 * 100$)	0.3232	0.9144	0.4253	1.0000	0.4083	0.8788	0.2245	0.7979	0.4256	1.0000

accessible for the adversary. We evaluate Best-of-N under four different values of N (1, 10, 50, 500), and set the GA with a population size of 100 and 5 iterations, which means it needs to query the original detector 500 times. Across all five watermark schemes, GA consistently achieves higher WDR and ASR scores than Best-of-N. While the performance of Best-of-N improves with larger N, e.g., for KGW, ASR increases from 0.2238 (at $N = 1$) to 0.8706 (at $N = 500$). Note that GA outperforms Best-of-N even when both approaches make an equal number of queries (500) to the original detector. So, GA's advantage stems not only from more query budget but also from its ability to efficiently identify removal-relevant tokens through optimization.

E. Mismatch Between Reference and Original Detectors

An intuitive approach for leveraging a reference detector in watermark removal is to use its gradients to identify important tokens and apply perturbations. This idea follows the standard practice in adversarial NLP, where gradients are often used to rank token importance [35]–[37]. However, this strategy fails in our setting due to a fundamental mismatch between the reference detector and the original detector. In original detectors, such as watermark during logits generation (e.g., KGW [19]), each token is categorized as either green or red. The global watermark score is computed using token counts: $S_w(X) = \frac{(1-\gamma)|X|_G - \gamma|X|_R}{\sqrt{\gamma(1-\gamma)|X|}}$, where $|X|_G + |X|_R = |X|$. Under the common setting $\gamma = 0.5$, all tokens contribute equally to the score, regardless of their position.

In contrast, reference detectors are typically neural networks trained to approximate the original detector's output. Due to the nature of neural networks, their predictions are often

TABLE XIII: Effect of GA iterations (n) on watermark removal. Watermarked texts are generated by OPT. We report ASR(↑) for both token-level and character-level perturbations with $\epsilon = 0.13$ in GA. Increasing n improves ASR, with character-level perturbations showing greater gains.

	KGW		DIP		SynthID		Unigram		Unbias	
n	Token	Char	Token	Char	Token	Char	Token	Char	Token	Char
5	0.3567	0.4144	0.5578	0.8273	0.0610	0.3468	0.4106	0.5959	0.5385	0.7766
10	0.3511	0.4384	0.5624	0.8273	0.0633	0.3737	0.4279	0.6404	0.5650	0.7979
15	0.3744	0.4966	0.5622	0.8453	0.0634	0.4209	0.4343	0.6575	0.5795	0.8369

TABLE XIV: ASR(↑) of the GA on OPT-generated watermarked texts under different filtering thresholds for high-gradient tokens. Smaller α values result in more aggressive filtering. None indicates no filtering is applied. The results show that appropriate filtering improves ASR.

α	KGW	DIP	SynthID	Unigram	Unbias
1	0.4315	0.8129	0.4108	0.5788	0.7908
2	0.4555	0.8129	**0.4209**	0.6096	0.8121
3	**0.4966**	**0.8453**	0.4175	0.6267	0.8191
4	0.4555	0.8058	0.4074	**0.6575**	**0.8369**
None	0.4555	0.8237	0.3737	0.6370	0.7943

dominated by a small number of high-gradient tokens. As a result, gradient-based strategies may focus on tokens important only to the reference detector but insignificant to the original, limiting their effectiveness.

F. Experiment for Genetic Algorithm-Based Attack with Limited Access to the Original Watermark Detector

1) Impact of Iterations in GA: Table XIII shows the impact of the number of GA iteration rounds (n) on watermark removal effectiveness, with maximum editing rate $\epsilon = 0.13$. Results are reported for $n = 5$, 10, and 15, using both token-level and character-level perturbations across five watermarking schemes. As n increases, the ASR improves consistently. For instance, in the KGW scheme with character-level perturbation, ASR increases from 0.4144 (at $n = 5$) to 0.4966 (at $n = 15$), yielding a gain of 0.0822. Moreover, character-level attacks not only achieve higher ASR, but also benefit more from increased iterations. On average, ASR of token-level perturbation improves by 0.0178 from $n = 5$ to $n = 15$, while character-level improves by 0.0592. These results further highlight the advantage of character-level perturbations in GA-based watermark removal.

2) Impact of N in Best-of-N: Figure 7 illustrates the effect of increasing N in the Best-of-N strategy on attack success rate (ASR), evaluated under five watermark schemes with ER set to 0.1 and Ref-9 used as the reference detector. We compare both token-level (dashed lines) and character-level (solid lines) perturbations. Across all watermark schemes, character-level attacks consistently outperform token-level attacks. Moreover, the ASR gap between the two methods widens as N increases. For example, on OPT-generated watermarked text, at $N = 1$, the average ASR across 5 watermark schemes is 0.2670 for token-level attacks and 0.3614 for character-level attacks, yielding a gap of 0.0944. At $N = 100$, the token-level ASR remains similar (0.2642), while the character-level ASR increases to 0.4412, expanding the gap to 0.1771. This trend highlights that reference detector guidance is especially beneficial for character-level perturbations, and its advantage grows with larger search budgets. These results demonstrate that reference detectors effectively enhance watermark removal performance for both perturbation types.

3) Effect of Filtering High-Gradient Tokens: Table XIV shows the impact of filtering high-gradient tokens on the performance of the GA-based watermark removal. As introduced

TABLE XV: Performance of reference detectors with three levels of data augmentation across five watermark schemes. Each model is evaluated by the Pearson correlation between predicted and ground-truth watermark scores, and detection accuracy (ACC) when using the predicted scores for binary watermark classification.

		Ref-0		Ref-5		Ref-9	
		Pearson(↑)	ACC(↑)	Pearson(↑)	ACC(↑)	Pearson(↑)	ACC(↑)
OPT	KGW	0.5941	0.7129	0.9471	0.9003	0.9677	0.9369
	DIP	0.2980	0.5351	0.7942	0.7002	0.8777	0.8188
	SynthID	0.4832	0.4947	0.8891	0.8906	0.9308	0.9233
	Unigram	0.9527	0.9739	0.9923	0.9970	0.9933	0.9977
	Unbias	0.2743	0.5298	0.8201	0.7527	0.8850	0.8181
LLaMA	KGW	0.6611	0.7607	0.9762	0.9792	0.9806	0.9850
	DIP	0.3240	0.5871	0.8106	0.7687	0.8721	0.8225
	SynthID	0.3920	0.6261	0.8683	0.8241	0.9022	0.8312
	Unigram	0.2422	0.5516	0.9606	0.9459	0.9673	0.9557
	Unbias	0.2915	0.6057	0.8043	0.7582	0.8730	0.8069

in Section V-A, we identify high-gradient tokens based on their gradient magnitudes with respect to the reference detector, and exclude those whose gradients exceed $\mu + \alpha \cdot \sigma$, where μ and σ are the mean and standard deviation of gradient values, respectively. A smaller α results in more aggressive filtering. The table reports ASR for various α values ranging from 4 to 1, as well as the case with no filtering ("None"). We observe that filtering high-gradient tokens leads to improved ASR across most watermark schemes. Without filtering, the average ASR across the five watermark schemes is 0.6169. The best-performing filtered setting achieves an average ASR of 0.6514, corresponding to an absolute improvement of 0.0346. However, the optimal value of α varies across watermark schemes, suggesting that the effect of gradient-based token is scheme-dependent.

4) Performance of Reference Detector: Figure 8 compares the watermark scores predicted by the reference detectors with those from the original detectors for Unbiased, SynthID, Unigram, and DIP. For each row, as data augmentation increases, the reference detector's predictions become closer to the original detector's scores. However, samples near the classification threshold (around the boundaries of the shaded regions), are very sparse, which makes it difficult for the reference model to predict reliably in these areas. Even in denser regions, the reference detector often struggles to accurately capture small variations in the original watermark scores.

Table XV presents the performance of reference detectors trained to regress the watermark scores produced by the original detectors. This table reports the Pearson correlation between predicted and ground-truth watermark scores, as well as detection accuracy when using the reference model as a binary classifier. The results show that data augmentation significantly boosts performance. On OPT-generated text, Ref-5 and Ref-9 achieve average Pearson improvements of 0.3681 and 0.4104, and accuracy improvements of 0.1988 and 0.2497, respectively. On LLaMA-generated text, Ref-5 and Ref-9 improve average Pearson by 0.5018 and 0.5862, and accuracy by 0.2290 and 0.2540, respectively.

Appendix B
Artifact Appendix

A. Description & Requirements

In this artifact evaluation, we provide a scaled-down version of the experiments to validate the two primary claims presented in the paper: (1) character-level perturbations achieve superior watermark removal effectiveness compared to token-level perturbations under the same editing rate, and (2) Genetic Algorithm-based optimization can leverage the guidance of a reference detector to further improve removal attack performance. Our paper received a "Major Revision" decision, and in the revised version we are suggested to include more ablation studies and evaluation metrics to systematically investigate the effectiveness of character-level removal. These new components have already been integrated into the source code, and do not affect the artifact evaluation presented here.

1) How to access: The artifact can be accessed via GitHub (https://github.com/plll4zzx/CharacterRemoval4WM) or DOI (https://doi.org/10.5281/zenodo.15872569). We recommend downloading the code using the following command: git clone https://github.com/plll4zzx/CharacterRemoval4WM. The datasets required for artifact experiment are available through the DOI and Google Drive (https://drive.google.com/file/d/1ZRPbyv8vHs_rh4fxIPEE3TzHNa-SU54E/view?usp=sharing).

2) Hardware dependencies: A commodity desktop machine with at least 8 CPU cores and 16GB RAM. GPU with at least 10GB of memory (e.g., NVIDIA GPU with CUDA) is strongly recommended for faster execution, especially for collecting watermarked text and reference detector training.

3) Software dependencies: Ubuntu, Python 3.9, Conda, other dependencies are listed in `requirements.txt`

4) Benchmarks: We use the C4 dataset as the source of prompts to query the victim LLMs and generate watermarked text[3]. In this evaluation, we employ OPT-1.3B[4] as the victim language model to produce watermarked outputs. Our reference detectors are finetuned from BERT[5]. All model weights can be obtained from HuggingFace model repositories.

B. Artifact Installation & Configuration

The high-level configuration steps consist of two main components: dependency installation and data preparation:

1) **Dependency Installation.** Install all required dependencies by running `sh install.sh`. After installation, using `conda activate test_char` to activate the Conda environment.
2) **Data Preparation.** Download the C4 dataset and generate watermarked text using the victim LLMs. For convenience, we provide sample data via Google Drive and the DOI link. After downloading, you can extract and place the `saved_data`, `saved_attk_data`, and `saved_model` directories into the project folder.

C. Experiment Workflow

The artifact supports a 2-step experimental workflow to reproduce the main results presented in the paper:

1) **Baseline Removal Evaluation.** Execute the scripts for random attacks to validate that character-level perturbations outperform token-level perturbations in removing watermark (refer to Table II).
2) **Guided Removal Evaluation.** Train reference detectors on dataset of the watermarked data. Then, run the guided removal attacks, including Best-of-N and Genetic Algorithm (GA) -based optimization, to demonstrate their superior performance (refer to Table VII).

D. Major Claims

The artifact is designed to reproduce the following major claims of the paper:

- **C1:** Character-level perturbations have significantly higher attack success rates and watermark score reduction than token-level approaches under the same editing rate, as reported in Table II (see Appendix B-E1).
- **C2:** Guided removal attacks based on reference detector, including Best-of-N and Genetic Algorithm (GA) -based attacks, improve removal effectiveness under black-box conditions, as reported in Table VII (see Appendix B-E2).

E. Evaluation

1) Experiment (E1): In E1, we aim to evaluate whether character-level removal attacks outperform token-level approaches under the AC1 setting, which assumes no access to the original watermark detector. Consistent with the paper, we test five representative watermarking schemes: KGW, DIP, SynthID, Unigram, and Unbias.

[Preparation] To run this experiment, the C4 dataset is needed. We recommend storing it in the path `"../../dataset/c4/realnewslike"`, where `".."` refers to the parent directory relative to the current working directory. Watermarked text is generated using the OPT-1.3B model. The model weights do not require a separate download, as they are automatically retrieved during script execution. This process can be performed with the script `collect_wm_text.py`, which requires specifying the watermark name, dataset path, model name, the number of text, and the GPU device. For example:

```
python collect_wm_text.py --device 0 \
--wm_name "KGW" --dataset_name \
"../../dataset/c4/realnewslike" \
--model_name "facebook/opt-1.3b" \
--file_num 50 --file_data_num 100
```

This command generates 5000 (`file_num * file_data_num`) watermarked samples using the `facebook/opt-1.3b` model with the KGW watermark

[3]The C4 dataset is publicly available at https://huggingface.co/datasets/allenai/c4. We recommend downloading it via git for convenience. The commands are: GIT_LFS_SKIP_SMUDGE=1 git clone https://huggingface.co/datasets/allenai/c4; cd c4; git lfs pull –include "realnewslike/*"

[4]https://huggingface.co/facebook/opt-1.3b

[5]https://huggingface.co/google-bert/bert-base-uncased

on GPU device 0. To produce watermarked text with other watermarking schemes, simply modify the `--wm_name` parameter[6]. We recommend generating at least 5000 examples to support reference detector training in E2.

Although using a GPU can accelerate text generation, the process remains time-consuming. For convenience, pre-generated watermarked data are provided in the `saved_data` directory to facilitate artifact evaluation.

[Execution] The following command can be used to evaluate the effectiveness of token-level and character-level perturbations in the Random strategy removal attack:

```
python test_rand_sh.py \
--llm_name "facebook/opt-1.3b" \
--wm_name_list "['KGW']" \
--atk_style_list "['token','char']" \
--max_edit_rate_list "[0.1, 0.5]" \
--do_flag "True" --atk_times_list "[1]" \
--max_token_num_list "[100]"
```

In this example, the editing rates are set to 0.1 and 0.5, and the length of the watermarked text is fixed to 100 tokens, consistent with the settings reported in the paper. In this random strategy, the adversary performs only a single random attack per text sample, so `atk_times_list`= 1. To evaluate additional watermarking schemes, simply add their names to the `wm_name_list`.

[Results] The results of this script are saved as log files in `attack_log/Rand` and as JSON files in `saved_attk_data`. Alternatively, setting the `do_flag` parameter to `"False"` prints the results directly to the terminal. The output demonstrates that, for the same editing rate, character-level perturbations consistently achieve higher watermark score dropping rate (WDR) and attack success rates (ASR) compared to token-level perturbations (refer to Table II in the paper). Additionally, by adding more values into `max_edit_rate_list` and `max_token_num_list` reproduces the results in Figure 3&6, respectively.

2) Experiment (E2): In Experiment (E2), we aim to evaluate the effectiveness of watermark removal under the AC2 scenario, where the adversary has limited access to the original watermark detector. In this setting, a reference detector is first trained using a limited number of queries to the original detector. Subsequently, the trained reference detector is used to guide the watermark removal process.

[Preparation] Command for training the reference detector:

```
python train_ref_detector.py  --device 0 \
--wm_name "KGW" --num_epochs 15 \
--rand_char_rate 0.15 --rand_times 9 \
--llm_name "facebook/opt-1.3b" --ths 4
```

In this example, the watermarking scheme is set to KGW. Each sample is augmented 9 times, with an editing rate of 0.15 (`rand_char_rate`). The `ths` is detection threshold used to evaluate the performance of the reference detector. The number of training epochs is set to 15. Due to both data augmentation and model training are computationally intensive and time-consuming, preprocessed datasets and pre-trained reference detector are provided in the `saved_data` and `saved_model` to facilitate artifact evaluation.

[Execution] We evaluate the two proposed guided removal attacks (Best-of-N and GA) along with a baseline method (Sand). All of these attacks leverage the trained reference detector to provide guidance.

```
Best-of-N: python test_rand_sh.py \
--llm_name "facebook/opt-1.3b" \
--wm_name_list "['KGW']" \
--atk_style_list "['token','char']" \
--do_flag "True" --atk_times_list "[10]" \
--max_token_num_list "[100]" \
--max_edit_rate_list "[0.1]" \
--data_aug_list "[9]"
```

In this command, the `atk_times_list` parameter specifies the value of N in the Best-of-N strategy, determining how many perturbation candidates are sampled for each input. The `data_aug_list` is used to choose reference detector. Due to we set `rand_times` to 9 when training the reference detector, `data_aug_list` is also set to 9.

```
Sand: python test_rand_sh.py \
--llm_name "facebook/opt-1.3b" \
--wm_name_list "['KGW']" \
--atk_style_list \
 "['sand_token', 'sand_char']" \
--data_aug_list "[9]"  --do_flag "True" \
--max_edit_rate_list "[0.1]" \
--atk_times_list "[1]" \
--max_token_num_list "[100]"
```

```
GA: python test_ga_sh.py \
--llm_name "facebook/opt-1.3b" \
--wm_name "KGW" --atk_style "char" \
--num_generations 15 --do_flag "True" \
--max_edit_rate 0.13 --data_aug 9 \
--max_token_num_list "[100]"
```

[Results] As in E1, the results of these three methods are saved by default as log files in the `attack_log/Rand` and `attack_log/GA` directories and as json files in the `saved_attk_data` folder. Setting the `do_flag` parameter to `False` will print the results directly to the terminal.

The output shows that, under comparable editing rates, the GA consistently achieves higher attack success rates (ASR) than both the Best-of-N and sand attacks (refer to Table VII). Additionally, for all three methods, character-level perturbations yield higher ASR compared to token-level perturbations.

Because this evaluation process can be time-consuming, precomputed json results are provided in the `saved_attk_data` directory to facilitate artifact verification.

[6]the alias of "Unbias" is "Unbiased" in the code