WaterSeeker: Pioneering Efficient Detection of Watermarked Segments in Large Documents

Leyi Pan[1*], Aiwei Liu[1], Yijian Lu[1], Zitian Gao[2], Yichen Di[1], Shiyu Huang[3], Lijie Wen[1†], Irwin King[4], Philip S. Yu[5]

[1]Tsinghua University [2]The University of Sydney [3]Zhipu AI
[4]The Chinese University of Hong Kong [5]University of Illinois at Chicago
panly24@mails.tsinghua.edu.cn, liuaw20@mails.tsinghua.edu.cn, wenlj@tsinghua.edu.cn

Abstract

Watermarking algorithms for large language models (LLMs) have attained high accuracy in detecting LLM-generated text. However, existing methods primarily focus on distinguishing fully watermarked text from non-watermarked text, overlooking real-world scenarios where LLMs generate only small sections within large documents. In this scenario, balancing time complexity and detection performance poses significant challenges. This paper presents WaterSeeker, a novel approach to efficiently detect and locate watermarked segments amid extensive natural text. It first applies an efficient anomaly extraction method to preliminarily locate suspicious watermarked regions. Following this, it conducts a local traversal and performs full-text detection for more precise verification. Theoretical analysis and experimental results demonstrate that WaterSeeker achieves a superior balance between detection accuracy and computational efficiency. Moreover, its localization capability lays the foundation for building interpretable AI detection systems.

1 Introduction

As large language models (LLMs) generate high-quality text, they address practical challenges but also raise concerns such as misinformation (Megías et al. 2022; Chen and Shu 2024) and copyright infringement (Rillig et al. 2023). LLM watermarking technology has emerged to tackle these issues by embedding specific information (watermarks) during text generation, allowing for accurate detection through specialized algorithms. Current watermark detection methods (Kirchenbauer et al. 2023; Zhao et al. 2024; Liu et al. 2024; Aaronson and Kirchner 2022; Lu et al. 2024; Lee et al. 2023; Hu et al. 2024; Wu et al. 2023) first calculate watermark scores for individual tokens through single token detection, then compute statistics across the entire document for classification. While these full-text detection methods effectively distinguish between fully watermarked and non-watermarked texts, they fail in real-world scenarios where LLMs generate only brief segments within longer documents, due to dilution effects, as shown in Figure 1. To the best of our knowledge, the Win-Max algorithm (Kirchenbauer et al. 2024) is the only work addressing this limitation by examining all possible window sizes and selecting the maximum statistical score across all windows as the final detection result, but suffers from high time complexity.

To address these issues, we propose a novel and general watermark detection method called WaterSeeker. WaterSeeker follows a "first locate, then detect" approach, as shown in Figure 1. It initially employs a low-complexity anomaly points extraction algorithm to identify suspected watermark regions, narrowing the detection target from a long text to a small segment encompassing the ground truth segment. Next, a local traversal is performed on the localization result, conducting full-text watermark detection within each window and comparing the highest confidence result to a threshold for the final determination. Theoretical analysis suggests that this coarse-to-fine process has the potential to achieve optimal detection performance while maintaining the lowest possible complexity for solving this problem.

In the experiment, we compared the effectiveness and time complexity of WaterSeeker with baseline methods for detecting watermarked segments in large documents. WaterSeeker significantly outperformed the baselines in balancing time complexity and detection performance. Moreover, it demonstrates good adaptability to different watermark strengths, segment lengths, and document lengths, while achieving robust performance against text edit attacks. In summary, the contributions of this work are as follows:

- We comprehensively define a new scenario: detecting watermarked segments in large documents. This includes specifying algorithm inputs/outputs, evaluation metrics, and how to create test datasets.
- We propose WaterSeeker, a general watermark detection method that effectively identifies watermarked segments in large documents, tackling the issues caused by dilution effects.
- WaterSeeker outperforms baselines in achieving a superior balance between time complexity and detection effectiveness.
- Further experiments demonstrate that WaterSeeker exhibits strong adaptability across various watermark strengths, segment lengths, and document lengths, while also being robust against text editing attacks.

[*]Work was done during the intern at Zhipu AI.
[†]Corresponding author

Copyright © 2025, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.

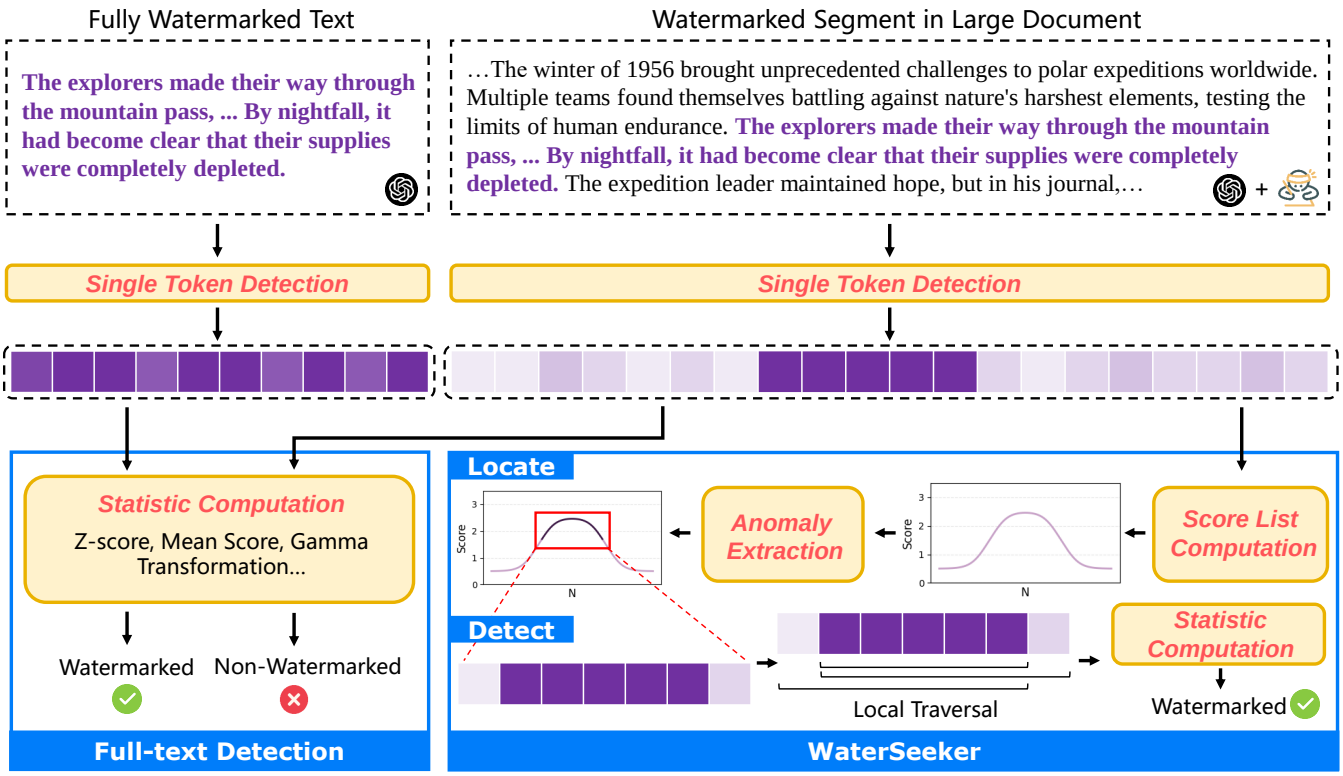


Figure 1: While full-text detection methods effectively differentiate between fully watermarked and non-watermarked texts, they often struggle with watermarked segment detection due to the dilution effect. To address this, WaterSeeker employs a "first locate, then detect" strategy, which narrows the detection range before conducting local traversal for further verification.

2 Related Work

2.1 Watermarking Algorithms

Currently, mainstream LLM watermarking methods involve modifying the inference phase by altering logits or influencing token sampling (Liu et al. 2023; Pan et al. 2024).

The KGW family (Kirchenbauer et al. 2023; Zhao et al. 2024; Hu et al. 2024; Liu et al. 2024; Wu et al. 2023) categorizes vocabulary into green and red lists, biasing towards green tokens during generation. The bias value is typically determined by the parameter δ, which reflects the watermark strength. For these methods, single token detection determines whether each token belongs to the green list, while full-text detection involves calculating the z-score of green tokens across the entire document; exceeding a threshold indicates watermarking. Details of the KGW algorithm are as follows:

(1) Watermarking. In watermarked text generation, the process for the t-th token begins by hashing preceding tokens with a secret key, creating a red-green vocabulary partition where green tokens comprise a fraction γ. Green token logits are then incrementally increased by δ, which can be expressed as follows:

$$l'_t(y) = \begin{cases} l_t(y), & y \in R_t \\ l_t(y) + \delta, & y \in G_t \end{cases} \tag{1}$$

This subtle modification results in watermarked text exhibiting a higher frequency of green tokens compared to non-watermarked text.

(2) Detection. Detecting a KGW watermark entails computing red-green partitions for each position using preceding tokens and the hash function, then calculating the green token proportion using the z-score:

$$z = \frac{|s|_G - \gamma N}{\sqrt{\gamma(1-\gamma)N}} \tag{2}$$

, where $|s|_G$ represents the total count of green tokens in the whole text of length N.

The Aar family(Aaronson and Kirchner 2022; Christ, Gunn, and Zamir 2024; Kuditipudi et al. 2024) uses pseudo-random sequences to guide token sampling. It generates a pseudo-random vector $u \sim \text{Uniform}([0,1])^{|V|}$ based on previous tokens and selects the token i maximizing u_i^{1/p_i}, where p is the LLM's probability vector. Watermark strength is controlled by sampling temperature. In these methods, single token detection calculates the correlation value between each token and u, while full-text detection applies gamma transformation to derive the detection confidence. Details of the Aar algorithm are as follows:

(1) Watermarking. When generating the t-th token, it first involves hashing the preceding tokens using a secret key to

obtain a pseudo vector $u_t \sim \text{Uniform}([0,1])^{|V|}$. The t-th token is determined by

$$\arg\max_y \; u_t(y)^{1/p_t(y)}, \tag{3}$$

where p is the probability vector produced by LLM at the t-th step. Let's perform equivalent transformations on it:

$$\begin{aligned}
\mathbf{y} &= \arg\max_y \; u_t(y)^{1/p_t(y)} && (4)\\
&= \arg\max_y \; \frac{1}{p_t(y)} \log u_t(y) && (5)\\
&= \arg\min_y \; \frac{1}{p_t(y)} \log \frac{1}{u_t(y)} && (6)\\
&= \arg\min_y \; \log \frac{1}{p_t(y)} + \log\log \frac{1}{u_t(y)} && (7)\\
&= \arg\max_y \; \log p_t(y) - \log\log \frac{1}{u_t(y)} && (8)
\end{aligned}$$

Given that the probabilistic output p_t of an LLM is derived from the logits l_t through a softmax transformation, and when we additionally consider the sampling temperature T, Equation 8 becomes equivalent to:

$$\arg\max_y \; \frac{l_t(y)}{T} + G_t(y), \tag{9}$$

where l_t is the logits produced by the LLM, and G_t is the Gumbel noise: $G_t(y) \sim Gumbel(0,1)$. The $Gumbel(0,1)$ distribution is defined as follows: if $u \sim \text{Uniform}(0,1)$, then $-\log(-\log(u))$ follows a $Gumbel(0,1)$ distribution.

It is evident that the temperature T can be utilized to exert control over the watermark strength. As the value of T increases, the influence of Gumbel noise on the sampling process becomes more pronounced, consequently resulting in a stronger watermark.

(2) Detection. Detecting an Aar watermark involves calculating the correlation value between the pseudo vector u_t and the corresponding token y_t in the text to be examined. The correlation value can be expressed as:

$$\log \frac{1}{1 - u_t(y_t)}. \tag{10}$$

For the entire text, the statistic value can be expressed as:

$$\text{p-value} = \Gamma\left(\sum_{t=1}^{N} \log\left(\frac{1}{1-u_t(y_t)}\right), N, \text{loc} = 0, \text{scale} = 1\right), \tag{11}$$

where Γ is the Gamma Transformation function that converts the sum of correlation values to a p-value.

2.2 Limitations of Full-text Detection Methods.

Despite the high accuracy of watermarking algorithms for distinguishing between fully watermarked and non-watermarked text, their performance fall sharply when detecting watermarked segments within large documents. A few studies have mentioned copy-paste attack (Kirchenbauer et al. 2024; Yoo, Ahn, and Kwak 2023; Wang et al. 2024), which involves mixing a portion of watermarked text with non-watermarked content, similar to our scenario. Yoo, Ahn, and Kwak (2023) and Wang et al. (2024) evaluated their methods' robustness against copy-paste attacks by combining 10% to 50% watermarked text with non-watermarked text. However, as they did not develop specific detection mechanisms for this situation, their findings showed that their methods were not robust against this type of attack.

Among existing studies, only WinMax (Kirchenbauer et al. 2024) specifically addresses watermarked segment detection by examining all possible window sizes to find the highest local statistics. However, its high time complexity limits practical application. To address this limitation, we propose WaterSeeker, a novel method that employs a "first locate then detect" strategy to achieve efficient detection of watermarked segments in large documents.

3 Problem Formulation

Definition. Given a text of length N containing m watermarked segments at position $[(s_1, e_1), ...(s_m, e_m)]$, the objective is to determine the presence and location of the watermarked segment. The detection algorithm outputs: {'has_watermark': True/False, 'indices': $[(s'_1, e'_1), ..., (s'_{m'}, e'_{m'})]$}.

Evaluation. A watermark is considered successfully detected if: **(1)** output.has_watermark = True. **(2)** The overall Intersection over Union (IoU) between the detected segments $(s'_i, e'_i)|_{i=1}^{m'}$ and the ground truth segments $(s, e)|_{i=1}^{m}$ is positive, indicating no complete false detection:

$$\text{IoU} = \frac{L_{\text{intersection}}}{L_{\text{union}}} > 0. \tag{12}$$

Based on these criteria, we evaluate the classification performance using False Positive Rate (FPR), False Negative Rate (FNR) and F1 Score, as well as the localization performance using average IoU between detected and ground truth segments.

4 Baseline Methods

Full-text Detection. As explained in Section 1, involves calculate watermark scores for individual tokens and compute statistics across the entire documents for classification.

WinMax (Kirchenbauer et al. 2024) involves iterating through all possible window sizes, and for each window size, the entire text is traversed to compute statistics for each local window, taking the maximum score for final results. The detection process can be described by the following formula:

$$\text{score} = \max_{1 \le w \le N} \; \max_{1 \le i \le N-w+1} F(x_{i:i+w-1}), \tag{13}$$

where w is the length of the local window, x represents the text tokens, and F is the statistical function. The time complexity is evidently $O(N^2)$. We also introduce a WinMax variant where window size increases by intervals > 1, reducing complexity to $O(N^2/\text{interval})$.

Fix-Length Sliding Window (FLSW) is a self-constructed method that uses a fixed-length window to traverse the text. The text is flagged as watermarked if any statistic score within

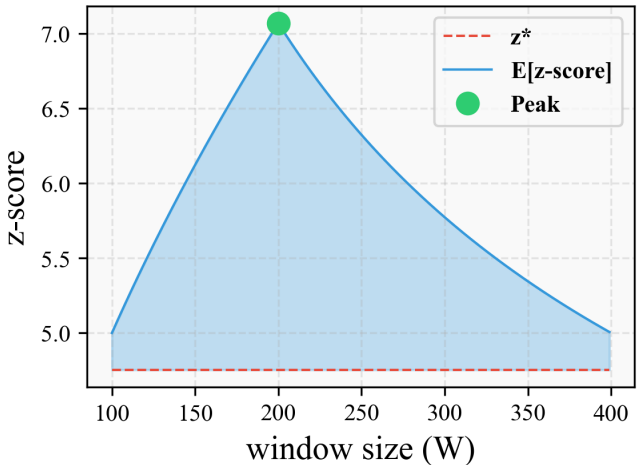


Figure 2: Expected z-score and the corresponding threshold z^* across various W, $\alpha = 10^{-6}, \gamma = 0.5, \gamma_1 = 0.75$.

the local windows exceeds the threshold. The pseudocode for all baselines can be found at Appendix A.

5 Proposed Method: WaterSeeker

5.1 Theoretical Basis: Gold Index is the Best

This section provides the theoretical foundation of WaterSeeker, showing that using actual start and end indices (gold index) for watermark detection achieves the highest expected detection rate. We analyze using KGW (Kirchenbauer et al. 2023) as a case study. Assuming $\gamma_1 > \gamma$ is the proportion of green tokens in the watermarked part, the watermark segment has a length L, and the statistical function F is the z-score computation. Let's analyze the effect of window size W on this statistic:

(1) When $W < L$:

$$E[z_W] = \frac{W\gamma_1 - \gamma W}{\sqrt{\gamma(1-\gamma)W}} = \sqrt{W} \cdot \frac{\gamma_1 - \gamma}{\sqrt{\gamma(1-\gamma)}}. \quad (14)$$

(2) When $W > L$:

$$E[z_W] = \frac{L\gamma_1 + (W-L)\gamma - \gamma W}{\sqrt{\gamma(1-\gamma)W}} = \frac{L(\gamma_1 - \gamma)}{\sqrt{\gamma(1-\gamma)W}}. \quad (15)$$

From this, we can conclude that **when $W = L$, the z-score reaches its maximum**. During detection, we aim to maximize positive case z-scores while maintaining false positive rate below a target threshold α. For a window of size W, the number of green tokens follows $B(W, \gamma)$, which approximates to $N(W\gamma, W\gamma(1-\gamma))$ for large W. This yields $z \sim N(0,1)$, and the threshold z^* is derived as: $z^* = \Phi^{(-1)}(1-\alpha)$, which is a constant value for different W. The simulation result using real data are shown in Figure 2.

5.2 Suspicious Region Localization

Based on the theoretical analysis, WaterSeeker employs a coarse-to-fine process to gradually approximate the gold index. In the coarse step, a localization algorithm identifies suspicious regions and narrows detection to a small segment containing the ground truth, while maintaining minimal deviation. This step involves three sub-steps:

(1) Score List Computation: Similar to existing methods, watermark scores for individual tokens are calculated through single token detection. Then, a small sliding window (i.e. $W = 50$) is used to traverse the text to compute average scores within the window, serving as a smoothing operation. This results in a score list s of length $N - W + 1$, where s_i represents the average watermark intensity from text token x_i to $x_i + W$.

(2) Anomaly Extraction: We design an anomaly extraction algorithm inspired by previous work in the field of style change detection (Zangerle et al. 2021) and intrinsic plagiarism detection (Manzoor et al. 2023). The mean score (s_{mean}) and top-k mean ($s_{\text{top-k-mean}}$) are calculated. Outliers are determined by:

$$\text{score} > s_{\text{mean}} + \frac{(s_{\text{top-k-mean}} - s_{\text{mean}})}{2}. \quad (16)$$

This is non-trivial because it ensures that the extracted suspicious watermarked regions likely cover the actual segments, with starting and ending deviations within a window size: when the sliding window falls entirely within the watermarked segment, the scores stabilize near $s_{\text{top-k-mean}}$; when the window falls completely outside, the scores stabilize below s_{mean}. Consequently, the extracted abnormal segment's start and end points (s' and e') generally satisfy $s' \in (s - W, s)$ and $e' \in (e, e + W)$. Moreover, the use of $s_{\text{top-k-mean}}$ ensures good adaptability to various watermark strengths. The corresponding experiment results can be found at Section 6.5.

(3) Fragment Connection: Adjacent outliers are connected with a predefined tolerance threshold, and segments shorter than a minimum length are filtered out, producing a list of segment indices.

5.3 Local Traverse Detection

After obtaining the coarse localization results, a fine-grained detection is performed by traversing segments within the predicted ranges. For each (s', e') pair in the localization results, the algorithm examines segments with start points in $[s', s' + W)$ and end points in $(e' - W, e']$. Based on the previous analysis, these ranges likely contain the true watermarked indices. Full-text detection is performed on these segments, and the most significant statistic is compared against a threshold. The complete WaterSeeker algorithm is presented in Algorithm 1.

5.4 Time Complexity Analysis

Time Complexity of WaterSeeker. WaterSeeker consists of two main components: (1) Suspicious Region Localization with $O(N)$ complexity, where N is the text length, and (2) Local Traverse Detection with $O(W^2)$ complexity, where W is the window size. The total complexity is $O(N + W^2)$. In practice, W^2 is typically kept lower than N, as a slightly larger window (i.e., $W = 50$, detailed in Appendix C) suffices for a smooth and low-noise representation of the sur-

Algorithm 1: WaterSeeker Algorithm

```
procedure LOCALIZATION(tokens)
    scores = SlidingWindow(tokens)
    threshold = s_mean + (s_top-k-mean − s_mean) / 2
    return ConnectOutliers(scores¿threshold)
end procedure
procedure DETECTION(tokens, segs)
    detected = []
    for (s', e') in segs do
        best = −∞, idx = None
        for s ∈ [s', s' + W), e ∈ (e' − W, e'] do
            score=WatermarkScore(tokens[s:e])
            if score > best then
                best = score
                idx = (s, e)
            end if
        end for
        if best > threshold then
            detected.append(idx)
        end if
    end for
    return bool(detected), detected
end procedure
procedure WATERSEEKER(tokens)
    return Detection(tokens, Localization(tokens))
end procedure
```

Table 1: Results of full-text detection methods.

Dataset	FPR↓	FNR↓	F1↑
KGW-Llama	0.000	0.983	0.033
KGW-Mistral	0.000	0.993	0.013
Aar-Llama	0.000	0.980	0.039
Aar-Mistral	0.000	0.980	0.039

rounding watermark intensity. Thus, the overall time complexity of WaterSeeker is $O(N)$.

Lower Bound Complexity for the Problem. To detect watermarked segments in a long text, any algorithm must examine each token in the text at least once. This requirement establishes a lower bound of $\Omega(N)$ for the time complexity of the problem, as at least one full pass through the text is necessary. Consequently, the WaterSeeker algorithm achieves a time complexity that matches the theoretical lower bound of the problem.

6 Experiment

6.1 Experiment Settings

Watermarking Methods and Language Models: We selected two representative watermarking algorithms, KGW (Kirchenbauer et al. 2023) and Aar (Aaronson and Kirchner 2022), each at three strength levels. KGW's strength was set by the δ parameter (2.0=strong, 1.5=medium, 1.0=weak), while Aar's strength used the temperature parameter (0.5=strong, 0.4=medium, 0.3=weak). We used Llama-2-7b (Touvron et al. 2023) and Mistral-7b (Jiang et al. 2024) as generation models.

Dataset Construction: The first 30 tokens of each entry in the C4 dataset (Raffel et al. 2020) were used for prompts. Watermarked segments of random length (100 to 400 tokens) were then generated using randomly selected watermark strengths. For positive examples, one such segment was randomly inserted into each 10,000-token Wikipedia passage (Foundation 2022). While most experiments used single-segment insertion, we also conducted experiments with multiple watermarked segments inserted into the same passage, with results reported in Table 4. Negative examples consist of unmodified 10,000-token Wikipedia corpus. Based on this procedure, four datasets were created, each containing 300 positive and 300 negative examples: KGW-Llama, KGW-Mistral, Aar-Llama, and Aar-Mistral.

Baselines: As introduced in Section 4, we selected Full-text Detection and WinMax (Kirchenbauer et al. 2024) with varying window size intervals, along with the Fixed-Length Sliding Window method using W of 100, 200, 300, and 400.

Hyper-parameters: The parameters related to WaterSeeker are as follows: $W = 50, k = 20$, with a tolerance for fragment connection set to 100. The threshold selection within the specified window is detailed in Appendix B. Notably, careful threshold selection is crucial for maintaining an acceptable false positive rate, as traversing long texts is prone to accumulating false positives.

6.2 Results of Full-text Detection

Table 1 shows that full-text detection methods perform poorly across all four datasets, with an F1 score of less than 0.1. This indicates that full-text detection methods are totally ineffective for detecting watermarked segments in large documents.

6.3 WaterSeeker Compared with WinMax

Main Experiment. From the data in Table 2 and Table 3, we can compare the detection and localization capabilities of WinMax with different window size intervals (1, 50, 100, 200) and WaterSeeker for watermarked segments, as well as the time costs for processing individual samples. WaterSeeker's detection and localization performance is only slightly behind that of WinMax-1, while achieving a time savings of 1000 times. Given that WinMax evaluates all possible windows to ensure it reaches the gold index, it represents the upper bound of detection performance. However, as the window interval sizes for WinMax increase, the processing time decreases linearly with the interval size, yet it remains higher than that of WaterSeeker, while its detection and localization performance falls below that of WaterSeeker.

Balancing Performance and Time Cost. To clearly compare the balance between detection performance and time cost for WinMax and WaterSeeker, we collected additional data points for WinMax, as shown in Figure 3(a). Points further to the bottom right in the figure indicate a superior balance. It is evident that WaterSeeker is positioned clearly below and to the right of the curve formed by the WinMax data

Table 2: We evaluated the detection performance of WaterSeeker against various methods, including Full-text Detection, WinMax (Kirchenbauer et al. 2024), and FLSW. Results in this table utilized the Llama-2-7b model. The metrics reported include false positive rate (FPR), false negative rate (FNR), F1 score, average Intersection over Union (IoU) between detected and ground truth segments, and processing time per sample. Best performances are highlighted in bold, while the second-best are underlined.

Method	**KGW**					**Aar**				
	FPR↓	FNR↓	F1↑	IoU↑	Time(s)↓	FPR↓	FNR↓	F1↑	IoU↑	Time(s)↓
WinMax-1	0.017	**0.193**	**0.885**	**0.713**	1632.11	0.017	**0.277**	**0.831**	**0.616**	3615.42
WinMax-50	0.017	0.220	0.868	0.672	34.31	0.007	0.307	0.816	0.577	72.13
WinMax-100	0.013	0.237	0.859	0.632	17.16	0.003	0.330	0.800	0.554	35.34
WinMax-200	0.010	0.273	0.834	0.547	9.12	0.003	0.363	0.776	0.486	18.38
WaterSeeker	0.017	0.213	0.872	0.675	**1.75**	0.010	0.300	0.819	0.578	**0.41**
FLSW-100	**0.003**	0.383	0.761	0.451	1.76	0.003	0.440	0.716	0.403	1.31
FLSW-200	**0.003**	0.300	0.822	0.487	1.76	**0.000**	0.380	0.765	0.411	1.29
FLSW-300	0.007	0.340	0.792	0.383	1.76	**0.000**	0.413	0.739	0.306	1.29
FLSW-400	**0.003**	0.407	0.743	0.275	**1.75**	**0.000**	0.443	0.715	0.228	1.27

Table 3: We evaluated the detection performance of WaterSeeker against various methods, including Full-text Detection, WinMax (Kirchenbauer et al. 2024), and FLSW. Results in this table utilized Mistral-7b as the generation model. The metrics reported include false positive rate (FPR), false negative rate (FNR), F1 score, average Intersection over Union (IoU) between detected and ground truth segments, and processing time per sample. Best performances are highlighted in bold, while the second-best are underlined.

Method	**KGW**					**Aar**				
	FPR↓	FNR↓	F1↑	IoU↑	Time(s)↓	FPR↓	FNR↓	F1↑	IoU↑	Time(s)↓
WinMax-1	0.010	**0.243**	**0.857**	**0.641**	1632.11	0.013	**0.297**	**0.819**	**0.588**	3615.42
WinMax-50	0.007	0.270	0.841	0.613	34.31	0.007	0.320	0.806	0.560	72.13
WinMax-100	0.003	0.283	0.833	0.588	17.16	0.007	0.333	0.797	0.531	35.34
WinMax-200	0.003	0.340	0.794	0.511	9.12	0.003	0.347	0.789	0.501	18.38
WaterSeeker	0.010	0.253	0.850	0.634	**1.75**	0.010	0.300	**0.819**	0.563	**0.41**
FLSW-100	**0.000**	0.463	0.698	0.393	1.76	**0.000**	0.473	0.690	0.367	1.31
FLSW-200	**0.000**	0.373	0.770	0.426	1.76	0.003	0.387	0.759	0.412	1.29
FLSW-300	0.003	0.437	0.719	0.331	1.76	0.003	0.387	0.759	0.334	1.29
FLSW-400	0.003	0.540	0.629	0.218	**1.75**	0.007	0.440	0.715	0.249	1.27

points, demonstrating a better balance between detection performance and time cost.

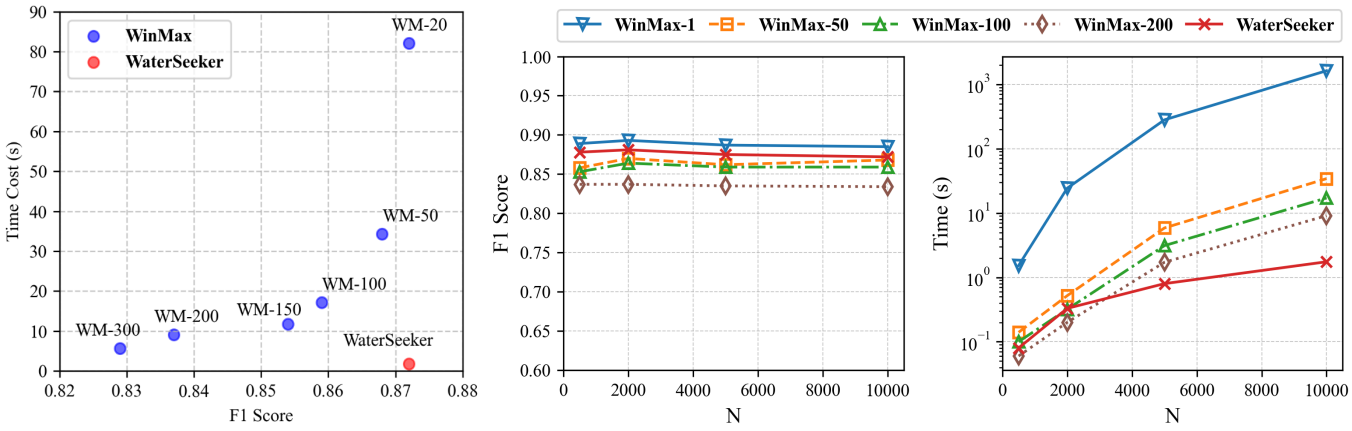
Robustness against Varying Text Lengths. To further validate the robustness of watermark detection algorithms against varying document lengths (mixing ratios), we tested WaterSeeker and WinMax at $N = 500$, 2000, 5000, and 10000, measuring the detection F1 score and time cost, as illustrated in Figure 3(b), (c). Both WaterSeeker and WinMax exhibit stable detection performance with changes in N. However, WinMax's time cost increases at a higher rate than that of WaterSeeker, indicating its impracticality for large documents.

WinMax's Limitations for Multiple Segments. As shown in Equation 13, when multiple watermarked segments are inserted within the same document, WinMax cannot function properly. In contrast, WaterSeeker is able to adapt to this situation. The detection and localization performance for multiple watermarked segments can be found in the Table 4, comparing performance of WaterSeeker and FLSW.

6.4 WaterSeeker Compared with FLSW

Main Experiment. As shown in Table 2 and Table 3, although the time cost of FLSW is comparable to that of WaterSeeker, its detection performance is significantly inferior. This difference is due to FLSW's fixed-length nature, which restricts its capability to utilize the gold index for detecting watermarked segments of varying lengths.

Multiple Segments Detection. Table 4 presents the detection and localization results for documents containing three watermarked segments. The experimental setup mirrors that of the main experiment, utilizing Llama-2-7b as the generation model. The results indicate that as the number of inserted segments increases, it becomes easier to detect a watermarked segment (all methods show improved F1 scores). However, the Intersection over Union (IoU) did not exhibit significant changes. Notably, WaterSeeker continues to outperform the FLSW algorithm across all four configurations, consistent with the main experiment that included only one segment per



(a) Balancing detection performance and time cost: WinMax vs. WaterSeeker.

(b) Detection performance comparison across different methods with varying text lengths.

(c) Time cost comparison across different methods with varying text lengths.

Figure 3: A detailed comparison of WinMax and WaterSeeker regarding their detection performance and time cost, as well as their performance across varying text lengths.

Table 4: Comparison of detection and localization performance between WaterSeeker and FLSW with multiple segment insertion (one document containing three watermarked segments), including the time cost for processing each sample.

Method	KGW					Aar				
	FPR↓	FNR↓	F1↑	IoU↑	Time(s) ↓	FPR↓	FNR↓	F1↑	IoU↑	Time(s) ↓
FLSW-100	**0.000**	0.177	0.903	0.433	1.76	0.013	0.147	0.914	0.434	1.31
FLSW-200	**0.000**	0.110	0.941	0.475	**1.75**	0.007	0.093	0.948	0.461	1.29
FLSW-300	0.003	0.130	0.929	0.399	1.76	0.007	0.120	0.933	0.369	1.29
FLSW-400	0.003	0.153	0.916	0.314	1.76	**0.003**	0.150	0.917	0.288	1.27
WaterSeeker	0.010	**0.057**	**0.966**	**0.649**	1.89	0.010	**0.057**	**0.966**	**0.542**	**0.83**

sample.

Further Analysis of Fix-length Nature. To better illustrate FLSW's inability to adapt to watermarked segments of varying lengths, we selected two types of extreme examples from the main experiment dataset. As shown in Table 5, for segments with strong watermark intensity but short length, using larger window sizes such as FLSW-300 or FLSW-400 leads to the inclusion of many non-watermarked segments, which dilutes the watermark intensity and results in a significant drop in performance. Conversely, for segments with weak watermark intensity but longer length, using smaller window sizes like FLSW-100 or FLSW-200 results in an insufficient number of watermark tokens for detection, preventing the accumulation of intensity and adversely affecting the detection results.

6.5 Ablation Study

We analyze the effectiveness of the two stages of WaterSeeker through an ablation study.

Stage 1: Suspicious Region Localization. This stage aims to narrow down the detection target from a large document to a smaller region. The goal is to achieve high coverage of the ground truth segments while maintaining the start and end offsets within a specified window size. This ensures that subsequent local traversals can access the gold index. As shown in Table 6, Step 1 achieves an average coverage exceeding 0.9, with average start and end offsets remaining below W across various watermark algorithms and strengths, demonstrating good adaptability.

Table 5: Performance of WaterSeeker and FLSW in two types of examples: segments with strong watermark intensity but short length (length ≤ 150, KGW $\delta = 2.0$), and segments with weak watermark intensity but long length (length ≥ 350, KGW $\delta = 1.0$).

	TPR	IoU
Strong but Short		
FLSW-300	0.000	0.000
FLSW-400	0.000	0.000
WaterSeeker	**0.667**	**0.572**
Weak but Long		
FLSW-100	0.375	0.150
FLSW-200	0.625	0.447
WaterSeeker	**0.813**	**0.642**

Stage 2: Local Traverse Detection. Local Traverse Detec-

Table 6: This table shows the contributions of the first stage of WaterSeeker: Suspicious Segment Localization. It lists the average coverage of localization results compared to ground truth segments for various watermark algorithms and strengths, along with the average offsets of the detected indices.

	Strength	Avg. Cov.	Avg. Off.
KGW	$\delta = 2.0$	0.989	$0.34W$
	$\delta = 1.5$	0.964	$0.35W$
	$\delta = 1.0$	0.950	$0.43W$
Aar	temp $= 0.5$	0.945	$0.13W$
	temp $= 0.4$	0.948	$0.12W$
	temp $= 0.3$	0.920	$0.10W$

Table 7: Comparison of detection performance with and without Local Traverse Detection.

Settings	KGW		Aar	
	F1	IoU	F1	IoU
w. Traversal	**0.872**	**0.675**	**0.819**	**0.578**
w/o Traversal	0.817	0.576	0.765	0.509

tion performs a localized iteration based on the segments from Stage 1, allowing for more refined verification within the window. Table 7 shows that across different watermarking algorithms, Local Traverse consistently enhances detection F1 score and average IoU compared to directly applying detection with the localization results, making it an indispensable component of WaterSeeker.

6.6 Robustness against Text Edit Attacks

In this section, we further examine WaterSeeker's robustness against text edit attacks, as watermarked segments generated by LLMs may be altered before integration into non-watermarked documents.

Figure 4 illustrates the detection robustness of WaterSeeker, WinMax and FLSW against random word deletion (ratio=0.3) and substitution attacks (ratio=0.3, utilizing WordNet (Fellbaum 1998) for synonym sets). The figure also contrasts these results with those obtained under no attack, along with the associated time costs. In this experiment, the KGW watermarking algorithm was employed, and Llama-2-7b is utilized as generation model. Since text editing can weaken the strength of the watermark, the watermark fragment intensity was set to a strong level under the "No attack" condition, specifically $\delta = 2.0$.

It can be observed from the figure that WaterSeeker achieves strong robustness against word deletion and word substitution attacks, with an F1 score exceeding 0.9. Compared with other baselines, only WinMax-1 perform slightly better than WaterSeeker, but at a significant cost in terms of time.

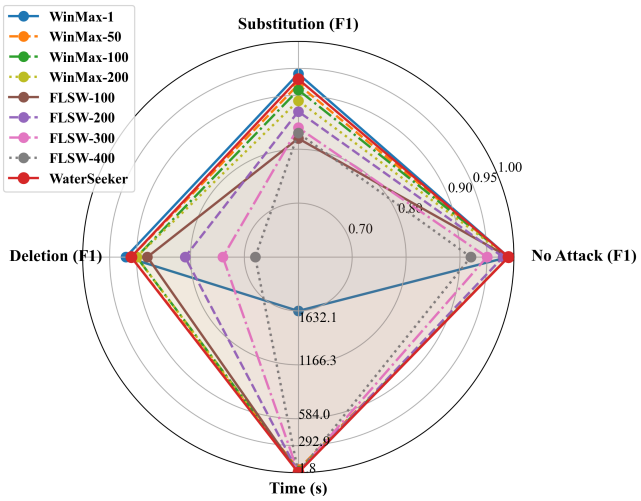


Figure 4: Robustness of WinMax, FLSW, and WaterSeeker against text edit attacks. The detection F1 score is reported for no attack, random word deletion attack (ratio = 0.3), and random word substitution attack (ratio = 0.3), along with the time cost for processing one sample.

7 Conclusion

This work introduces a new scenario for detecting watermarked segments in large documents and establishes corresponding evaluation metrics. We identified the limitations of full-text detection methods in this context and proposed a "first locate, then detect" watermark detection algorithm that utilizes a coarse-to-fine strategy. We validated the detection performance and time complexity of our algorithm through a series of analyses and experiments, demonstrating its ability to effectively balance both aspects. Future research could explore more advanced locating methods based on this concept to potentially yield improved detection results.

Limitations

While our method has demonstrated effectiveness in detecting watermarked segments within large documents, there are still some limitations that need to be addressed in future work. First, from an evaluation perspective, due to resource constraints, we only conducted experiments on Llama-2-7B and Mistral-7B models. The effectiveness of our method on larger and more powerful models remains to be further verified. Second, WaterSeeker's performance may decrease with very short or weak watermarks. Enhancing the sensitivity of WaterSeeker to detect shorter and weaker watermarks is an area for future improvement, which may involve refining the anomaly extraction algorithms or incorporating additional contextual analysis. Lastly, parameter selection, including threshold settings to control false positives, is crucial and can be challenging in different environments. Stricter threshold controls can reduce the detection rate, necessitating adjustments based on the specific requirements of the actual settings.

Acknowledgments

This work is primarily supported by the Key Research and Development Program of China (No. 2024YFB3309702).

We would like to express our gratitude to the anonymous AAAI 2025 Workshop PDLM reviewers (Reviewer haMz, h57D, Kyvu) and Area Chair bYE9 for their valuable feedback and suggestions that helped improve this paper.

References

Aaronson, S.; and Kirchner, H. 2022. Watermarking GPT outputs. https://www.scottaaronson.com/talks/watermark.ppt.

Chen, C.; and Shu, K. 2024. Can LLM-Generated Misinformation Be Detected? In *The Twelfth International Conference on Learning Representations*.

Christ, M.; Gunn, S.; and Zamir, O. 2024. Undetectable watermarks for language models. In *The Thirty Seventh Annual Conference on Learning Theory*, 1125–1139. PMLR.

Fellbaum, C. 1998. *WordNet: An electronic lexical database*. MIT press.

Foundation, W. 2022. Wikimedia Downloads.

Hu, Z.; Chen, L.; Wu, X.; Wu, Y.; Zhang, H.; and Huang, H. 2024. Unbiased Watermark for Large Language Models. In *The Twelfth International Conference on Learning Representations*.

Jiang, A. Q.; Sablayrolles, A.; Roux, A.; Mensch, A.; Savary, B.; Bamford, C.; Chaplot, D. S.; Casas, D. d. l.; Hanna, E. B.; Bressand, F.; et al. 2024. Mixtral of experts. *arXiv preprint arXiv:2401.04088*.

Kirchenbauer, J.; Geiping, J.; Wen, Y.; Katz, J.; Miers, I.; and Goldstein, T. 2023. A Watermark for Large Language Models. In Krause, A.; Brunskill, E.; Cho, K.; Engelhardt, B.; Sabato, S.; and Scarlett, J., eds., *International Conference on Machine Learning, ICML 2023, 23-29 July 2023, Honolulu, Hawaii, USA*, volume 202 of *Proceedings of Machine Learning Research*, 17061–17084. PMLR.

Kirchenbauer, J.; Geiping, J.; Wen, Y.; Shu, M.; Saifullah, K.; Kong, K.; Fernando, K.; Saha, A.; Goldblum, M.; and Goldstein, T. 2024. On the Reliability of Watermarks for Large Language Models. In *The Twelfth International Conference on Learning Representations*.

Kuditipudi, R.; Thickstun, J.; Hashimoto, T.; and Liang, P. 2024. Robust Distortion-free Watermarks for Language Models. *Transactions on Machine Learning Research*.

Lee, T.; Hong, S.; Ahn, J.; Hong, I.; Lee, H.; Yun, S.; Shin, J.; and Kim, G. 2023. Who Wrote this Code? Watermarking for Code Generation. *arXiv preprint arXiv:2305.15060*.

Liu, A.; Pan, L.; Hu, X.; Meng, S.; and Wen, L. 2024. A Semantic Invariant Robust Watermark for Large Language Models. In *The Twelfth International Conference on Learning Representations*.

Liu, A.; Pan, L.; Lu, Y.; Li, J.; Hu, X.; Wen, L.; King, I.; and Yu, P. S. 2023. A survey of text watermarking in the era of large language models. *arXiv preprint arXiv:2312.07913*.

Lu, Y.; Liu, A.; Yu, D.; Li, J.; and King, I. 2024. An Entropy-based Text Watermarking Detection Method. In Ku, L.-W.; Martins, A.; and Srikumar, V., eds., *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 11724–11735. Bangkok, Thailand: Association for Computational Linguistics.

Manzoor, M. F.; Farooq, M. S.; Haseeb, M.; Farooq, U.; Khalid, S.; and Abid, A. 2023. Exploring the Landscape of Intrinsic Plagiarism Detection: Benchmarks, Techniques, Evolution, and Challenges. *IEEE Access*, 11: 140519–140545.

Megías, D.; Kuribayashi, M.; Rosales, A.; Cabaj, K.; and Mazurczyk, W. 2022. Architecture of a fake news detection system combining digital watermarking, signal processing, and machine learning. *Journal of Wireless Mobile Networks, Ubiquitous Computing, and Dependable Applications*, 13(1): 33–55.

Pan, L.; Liu, A.; He, Z.; Gao, Z.; Zhao, X.; Lu, Y.; Zhou, B.; Liu, S.; Hu, X.; Wen, L.; et al. 2024. MarkLLM: An Open-Source Toolkit for LLM Watermarking. *arXiv preprint arXiv:2405.10051*.

Raffel, C.; Shazeer, N.; Roberts, A.; Lee, K.; Narang, S.; Matena, M.; Zhou, Y.; Li, W.; and Liu, P. J. 2020. Exploring the limits of transfer learning with a unified text-to-text transformer. *The Journal of Machine Learning Research*, 21(1): 5485–5551.

Rillig, M. C.; Ågerstrand, M.; Bi, M.; Gould, K. A.; and Sauerland, U. 2023. Risks and benefits of large language models for the environment. *Environmental Science & Technology*, 57(9): 3464–3466.

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

Wang, L.; Yang, W.; Chen, D.; Zhou, H.; Lin, Y.; Meng, F.; Zhou, J.; and Sun, X. 2024. Towards Codable Watermarking for Injecting Multi-Bits Information to LLMs. In *The Twelfth International Conference on Learning Representations*.

Wu, Y.; Hu, Z.; Zhang, H.; and Huang, H. 2023. DiPmark: A Stealthy, Efficient and Resilient Watermark for Large Language Models. *arXiv preprint arXiv:2310.07710*.

Yoo, K.; Ahn, W.; and Kwak, N. 2023. Advancing Beyond Identification: Multi-bit Watermark for Language Models. *arXiv preprint arXiv:2308.00221*.

Zangerle, E.; Mayerl, M.; Potthast, M.; and Stein, B. 2021. Overview of the style change detection task at pan.

Zhao, X.; Ananth, P. V.; Li, L.; and Wang, Y.-X. 2024. Provable Robust Watermarking for AI-Generated Text. In *The Twelfth International Conference on Learning Representations*.

A Pseudocode of Detection Baselines

Pseudocode of WinMax and FLSW could be found in Algorithm 2 and 3, respectively.

Algorithm 2: WinMax Algorithm

```
procedure WINMAXDETECTION(tokens, interval, threshold)
    hasWatermark ← False, indices ← [ ]
    maxStat ← - ∞, bestIndex ← None
    for W ∈ [1, len(tokens)], step=interval do
        for i in 0 to len(tokens) − W do
            stat ← WatermarkScore(tokens[i : i + W])
            if stat > maxStat then
                maxStat ← stat
                bestIndex ← (i, i + W)
            end if
        end for
    end for
    if maxStat > threshold then
        hasWatermark ← True
        indices.append(bestIndex)
    end if
    return hasWatermark, indices
end procedure
```

Algorithm 3: FLSW Algorithm

```
procedure FLSWDETECTION(tokens, W, threshold)
    hasWatermark ← False
    indices ← [ ]
    for i in 0 to len(tokens) − W do
        stat ← WatermarkScore(tokens[i : i + W])
        if stat > threshold then
            hasWatermark ← True
            indices.append((i, i + W))
        end if
    end for
    indices ← ConnectFragments(indices)
    return hasWatermark, indices
end procedure
```

B Detail of Threshold Selection Within the Specified Window

A key role of threshold selection is to control the false positive rate. In this context, the task involves detecting watermark fragments within long texts, which requires traversing extensive content and can lead to an accumulation of false positives. Therefore, managing the false positive rate within the detection window is crucial in this scenario. In the experiment, we set the target false positive rate α within the detection window to 10^{-6}.

B.1 Rationale for setting α to 10^{-6}

Table 8: Simulated FPR of WaterSeeker using 10,000 samples for each watermarking method. The targeted false positive rate within the detection window is set to 10^{-6}.

Watermarking Method	Simulated FPR
KGW	0.0054
Aar	0.0042

WaterSeeker, WinMax, and FLSW all involve employing sliding windows for text traversal and conduct full-text detection within each window. As these windows overlap, they cannot be treated as independent, making it challenging to derive a theoretical upper bound for the document-level FPR from the target FPR within each window. Given this, we utilize large-scale data simulation to demonstrate that, with a target false positive rate of 10^{-6} within each window, our proposed method WaterSeeker maintains an acceptable false positive rate.

For the KGW method, we set $\gamma = 0.5$ in our experiments, meaning each token in non-watermarked text has a 0.5 probability of being green and 0.5 probability of being red. In the simulation, we generate 10,000 samples, each containing 10,000 tokens, with each token having a 0.5 probability of being 1 and 0.5 probability of being 0. For the Aar method, each token in non-watermarked text corresponds to $u_i \sim \text{Uniform}[0, 1]$. In the simulation, we again generate 10,000 samples, each containing 10,000 tokens, with each token randomly assigned a floating-point number from $[0, 1]$.

We then apply WaterSeeker to detect watermarked segments within these samples, setting the target false positive rate within the detection window to 10^{-6}. The large-scale simulation results in Table 8 demonstrate that WaterSeeker maintains a false positive rate of approximately 0.005, which is considered acceptable. For scenarios requiring more stringent FPR control, the target false positive rate can be adjusted downward. However, this inevitably compromises the detection rate, highlighting a key challenge in watermarked segment detection within large documents.

B.2 Setting the threshold to achieve a target false positive rate α

KGW. For KGW, as analyzed in Section 5.1, when the window size is large, we can approximate using the Central Limit Theorem, resulting in $z^* = \Phi^{-1}(1 - \alpha)$. When $\alpha = 10^{-6}$, this gives $z \approx 4.75$. However, when the window size W is small, the approximation to a normal distribution using the Central Limit Theorem may lead to significant deviations. Therefore, we will use the binomial distribution for precise calculations. $x \sim B(W, \gamma)$ describes the number of green tokens in a window of size W follows a binomial distribution, therefore:

$$z = \frac{x - \gamma W}{\sqrt{W\gamma(1-\gamma)}}.$$

To find $P(z \geq z^*)$:

$$P(z \geq z^*) = P\left(\frac{x - \gamma W}{\sqrt{W\gamma(1-\gamma)}} \geq z^*\right).$$

Expanding this, we have:

$$P(z \geq z^*) = \sum_{k=0}^{W} \binom{W}{k} \gamma^k (1-\gamma)^{W-k} \mathbb{I}\left\{\frac{k - \gamma W}{\sqrt{W\gamma(1-\gamma)}} \geq z^*\right\}.$$

This is the exact expression for $P(z \geq z^*)$ without any approximations.

We can further simplify:

$$P(z \geq z^*) = \sum_{k=0}^{W} \binom{W}{k} \gamma^k (1-\gamma)^{W-k} \mathbb{I}\left\{k \geq \gamma W + z^* \sqrt{W(1-\gamma)}\right\}.$$

We need to find an appropriate z^* such that $P(z \geq z^*) < \alpha$. This function does not have a direct analytical solution, so we can increment z^* in steps of 0.01 until the probability exceeds α. The final value of z^* is dependent on W, and we pre-compute these values during experiments and store them in a dictionary. In experiments, for detected segments with a length of 200 or more, we directly apply the Central Limit Theorem approximation, setting $z = 4.75$. For segments shorter than 200, we use the binomial distribution and retrieve the corresponding threshold from the pre-computed dictionary.

Aar. For Aar, recall the p-value calculation formula:

$$\text{p-value} = \Gamma(S, W, \text{loc} = 0, \text{scale} = 1), \tag{17}$$

where $S = \sum_{i=1}^{W} \log(\frac{1}{1-u_i})$, and W is the window size. For non-watermarked text, $u_i \sim \text{Uniform}([0, 1])$. Consequently, S follows a Gamma distribution: $S \sim \text{Gamma}(W, 1)$, where W is the shape parameter and 1 is the scale parameter. Equation 17 is equivalent to:

$$\text{p-value} = 1 - \text{GammaCDF}(S, W, 1), \tag{18}$$

where GammaCDF is the cumulative distribution function of the Gamma distribution with shape parameter W and scale parameter 1. To achieve a false positive rate of α, we need to set a threshold p^* such that: $P(\text{p-value} < p^*) = \alpha$. Given the definition of p-value, this is equivalent to: $P(1 - \text{GammaCDF}(S, W, 1) < p^*) = \alpha$, which can be rewritten as: $P(S >$

GammaInv$(1-p^*, W, 1)) = \alpha$, where GammaInv is the inverse of the Gamma CDF. Since S follows a Gamma$(W, 1)$ distribution for non-watermarked text, we can express this as:

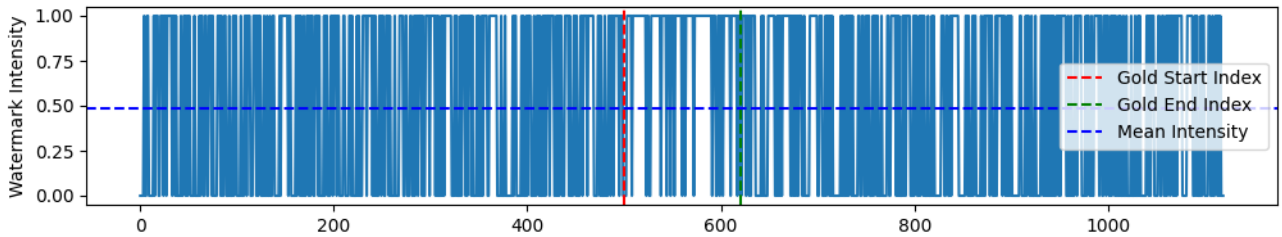
$$1 - \text{GammaCDF}(\text{GammaInv}(1 - p^*, W, 1), W, 1) = \alpha. \tag{19}$$

Solving this equation for p^*, we get $p^* = \alpha$, which is also a constant value for different W.

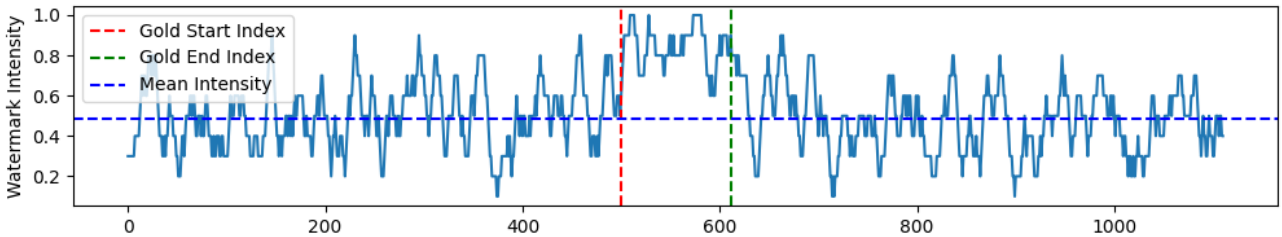
C Impact of Window Size on Watermark Intensity Calculation

The first step in WaterSeeker is score list computation. In this step, selecting an appropriate window size W for calculating mean scores is crucial. A small W introduces excessive noise, while a large W reduces granularity and increases computational time due to the need to examine W^2 windows during local traversal. Therefore, we aim to determine an appropriate window size that is relatively small while still providing a sufficiently smooth representation of watermark intensity throughout the text.

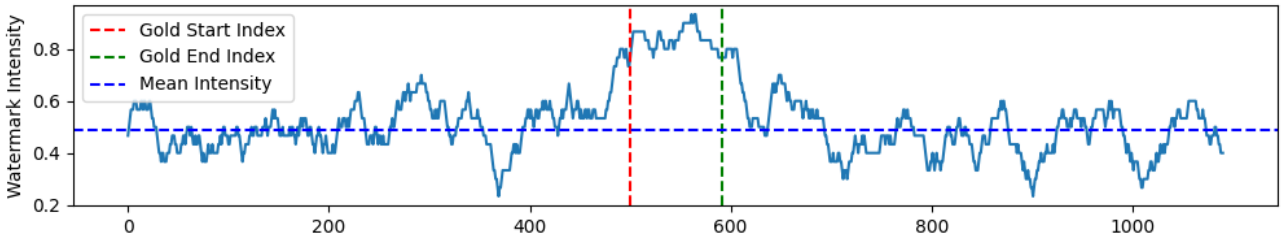
We present a case study comparing watermark intensity calculations using window sizes $W = 1, 10, 30,$ and 50. The analysis encompasses the ground truth segment and 500 tokens on either side. Figures 5 and 6 illustrate the results for the KGW and Aar algorithms, respectively. The intensity curves reveal that small window sizes, particularly $W \leq 10$, introduce significant fluctuations. While $W = 30$ exhibits reduced noise, it still presents instabilities, as shown in Figure 6c (the ground truth segment part). Overall, $W = 50$ demonstrates the least noise. Consequently, we adopt $W = 50$ for our main experiments.



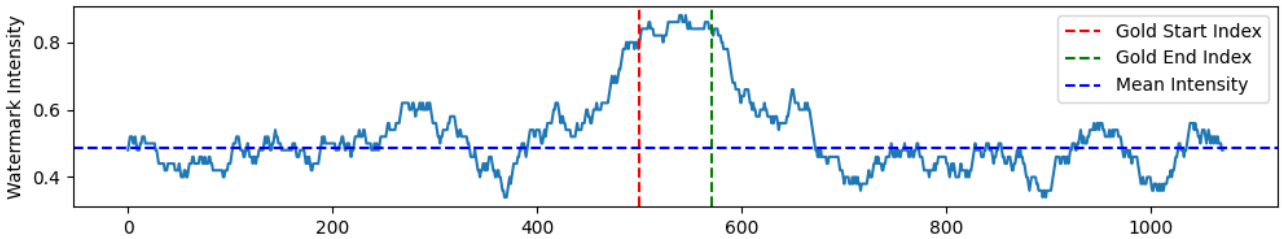
(a) $W = 1$



(b) $W = 10$

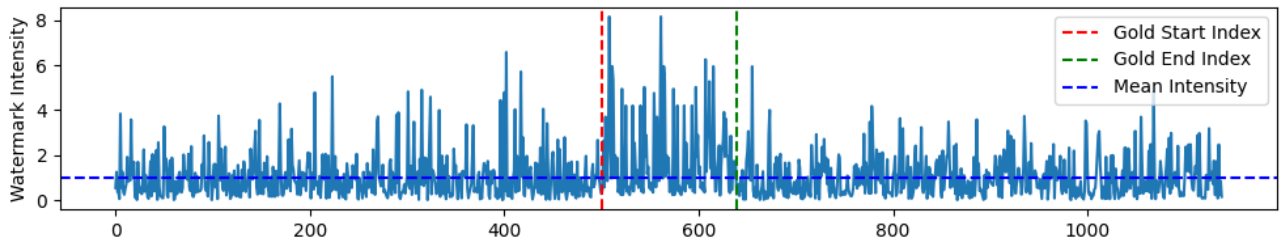


(c) $W = 30$

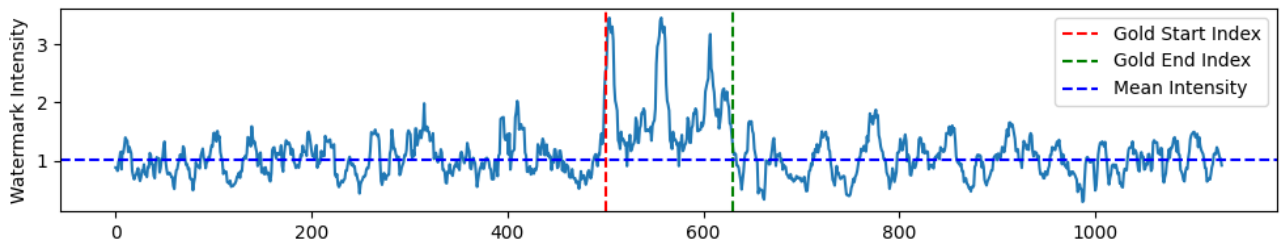


(d) $W = 50$

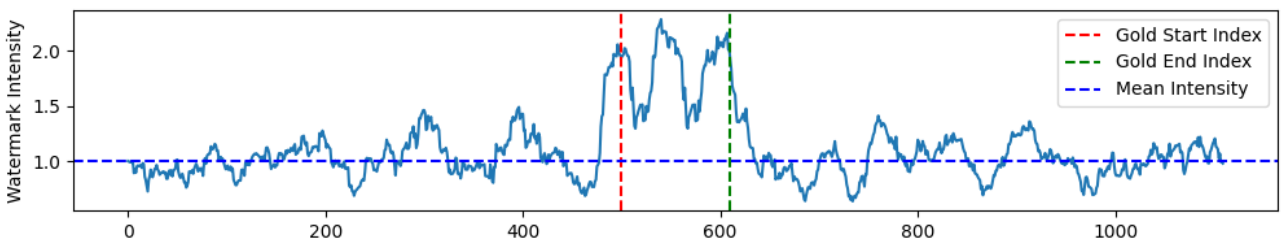
Figure 5: Case study: Impact of varying window sizes on watermark intensity calculation in the KGW algorithm.



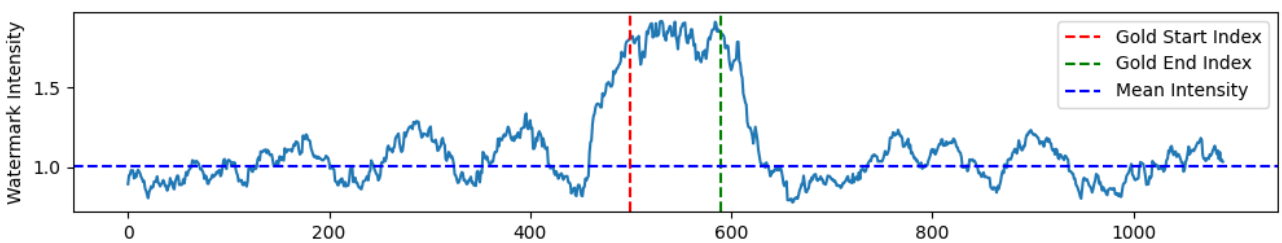
(a) $W = 1$



(b) $W = 10$



(c) $W = 30$



(d) $W = 50$

Figure 6: Case study: Impact of varying window sizes on watermark intensity calculation in the Aar algorithm.