WaterSeeker: Pioneering Efficient Detection of Watermarked Segments in Large Documents

Anonymous ACL submission

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, WaterSeeker's localization ability supports the development of interpretable AI detection systems. This work pioneers a new direction in watermarked segment detection, facilitating more reliable AI-generated content identification.

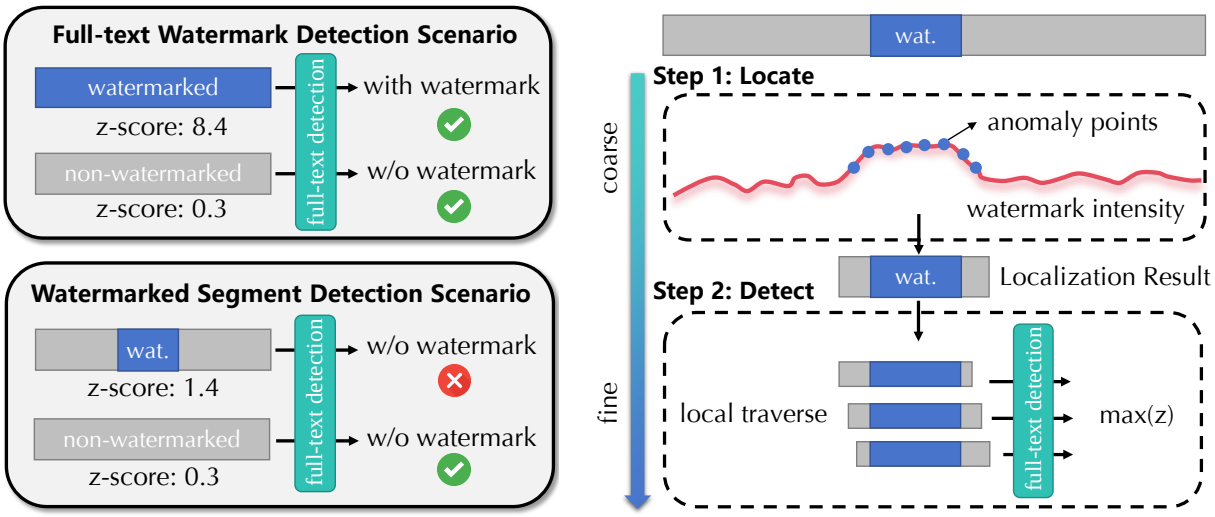
1 Introduction

As large language models (LLMs) generate high-quality text, they address practical challenges but also raise concerns such as misinformation (Liu et al., 2023b; 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. Existing watermarking methods primarily focus on distinguishing fully watermarked text from non-watermarked content (Kirchenbauer et al., 2023; Zhao et al., 2024; Liu et al., 2024b; Lu et al., 2024; Aaronson and Kirchner, 2022; Kuditipudi et al., 2024). However, in real-world scenarios, LLMs often generate only brief segments within longer documents, making it essential to detect these watermarked sections among large amounts of non-watermarked text.

Most previous algorithms relied on computing statistics across the entire document for detection, which we refer to as full-text detection methods (detailed in Section 2). However, these full-text detection methods struggle when small watermarked sections are mixed with large volumes of natural text due to dilution effects, as illustrated in Figure 1a. To the best of our knowledge, the only existing work addressing this scenario is the WinMax (Kirchenbauer et al., 2024) algorithm, which examines all possible window sizes and traverses the full text for each size. However, this method suffers from high time complexity. Additionally, our self-constructed baseline, the Fixed-Length Sliding Window (FLSW) method, faces the issue of watermark strength dilution due to inflexible window sizes, which lowers detection effectiveness.

To address these issues, we propose a novel watermark detection method called WaterSeeker. WaterSeeker follows a "first locate, then detect" approach, as shown in Figure 1b. 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



(a) Full-text detection methods often struggle in watermarked segment detection scenarios due to the dilution effect.

(b) An illustration of the WaterSeeker algorithm, using "first locate, then detect" strategy to balance time complexity and detection performance.

Figure 1: An overview of watermark detection scenarios and the proposed WaterSeeker algorithm.

in large documents. WaterSeeker significantly outperformed the baselines in balancing time complexity and detection performance. We also assessed its adaptability to different watermark strengths and segment lengths, and evaluated its two-stage utility through an ablation study. 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.
- WaterSeeker not only provides accurate detection results but also pinpoints watermark locations precisely, helping to create more explainable AI detection systems (detailed in Appendix H).

2 Related Work

Currently, mainstream LLM watermarking methods involve modifying the inference phase by altering logits or influencing token sampling (Liu et al., 2023a; Pan et al., 2024; Liu et al., 2024c). The KGW family (Kirchenbauer et al., 2023; Zhao et al., 2024; Hu et al., 2024; Liu et al., 2024b; He et al., 2024; Lu et al., 2024; Liu et al., 2024a) 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. Detection involves calculating the z-score of green tokens among the entire document; exceeding a threshold indicates watermarking.

On the other hand, the Aar family (Aaronson and Kirchner, 2022; Christ et al., 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. For detection, it also employs global statistics: it sums the correlation values of each token with the pseudo-random vector and then performs gamma transformation to derive the detection confidence. Details of the KGW and Aar watermarking algorithms can be found in Appendix A.

Despite the high accuracy of watermarking algorithms for detecting LLM-generated text, previous studies have primarily focused on distinguishing between fully watermarked and non-watermarked text, overlooking the possibility that LLMs may only generate small segments within large docu-

ments. In such cases, watermark detection algorithms based on full-text statistics can fail due to the dilution effect. A few studies have mentioned copy-paste attack (Kirchenbauer et al., 2024; Yoo et al., 2023; Wang et al., 2024), which involves mixing a portion of watermarked text with non-watermarked content, similar to our scenario. Yoo et al. (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.

In existing studies, WinMax (Kirchenbauer et al., 2024) is the only watermark detection algorithm specifically designed for this scenario. It examines all possible window sizes and traverses the entire text for each size to identify the maximum local score for threshold comparison. While this method shows some effectiveness, its high time complexity renders it impractical for real-world applications. Therefore, to achieve effective detection of watermarked segments inserted into large documents while maintaining controllable time complexity, we propose WaterSeeker, a novel and general method that uses "first locate then detect" strategy.

3 Problem Formulation

3.1 Definition

The problem of detecting watermarked segments in a long document is defined as follows: Let N be a long text with a randomly inserted watermarked segment of length L. We denote the starting index of the watermarked segment in N as s and the ending index as e, such that: $L = e - s + 1$. The objective is to determine the presence and location of a watermarked segment in N. This can be framed as a binary classification problem with additional localization, where the detection algorithm outputs:

$$\begin{aligned}\text{output} = \{&\text{'has_watermark'} : \text{boolean},\\ &\text{'indices'} : \text{list of pairs } (s', e')\}\end{aligned}$$

Here, 'has_watermark' is a boolean value indicating the presence of a watermark, and 'indices' is a list of pairs (s', e') representing the start and end indices of detected watermarked segments.

3.2 Evaluation

A watermark is considered successfully detected if: **(1)** output.has_watermark = True. **(2)** The overall Intersection over Union (IoU) between the detected segments $(s'_i, e'_i)|_{i=1}^{n}$ and the ground truth segment (s, e) exceeds a predefined threshold θ:

$$\text{IoU} = \frac{L_{\text{intersection}}}{L_{\text{union}}} > \theta.$$

Based on this criteria, we will report the following metrics of the binary classification result: False Positive Rate (FPR), False Negative Rate (FNR), and F1 Score. Moreover, we will include the average IoU between the detected segments and ground truth segments as a supplementary metric, further demonstrating the accuracy of the localization.

4 Baseline Methods

4.1 Full-text Detection

Full-text Detection involves directly applying the watermark detection algorithm to the whole document. For the KGW (Kirchenbauer et al., 2023) method, $z = \frac{|s|_G - \gamma N}{\sqrt{\gamma(1-\gamma)N}}$, where $|s|_G$ represents the total count of green tokens in the entire text of length N, and γ is the expected green token proportion. For the Aar (Aaronson and Kirchner, 2022) method, the p-value is calculated by applying a gamma transformation to the sum of correlation values of all the tokens: $\text{p-value} = \Gamma\left(\sum_{i=1}^{N} \log\left(\frac{1}{1-u_i}\right), N, \text{loc} = 0, \text{scale} = 1\right)$. Assuming the i-th token is t, u_i represents the value at the t position of the corresponding pseudo-random vector.

4.2 WinMax

WinMax (Kirchenbauer et al., 2024) involves iterating through all possible window sizes, and for each window size, the entire text is traversed to calculate the z-score for each local window, taking the maximum z-score and comparing it against a specified threshold. In practical applications, a minimum window size W_{min} and a maximum window size W_{max} are typically set to control the number of iterations during the traversal. The detection process can be described by the following formula:

$$z_{\text{win-max}} = \max_{w \in [W_{min}, W_{max}]} \left(\max_i \left(\frac{|s|_{G,i} - \gamma w}{\sqrt{\gamma(1-\gamma)w}} \right) \right),$$

where w is the length of the local window, and $|s|_{G,i}$ represents the count of green tokens within the i-th local window of length w. The time complexity of WinMax is $O((W_{max} - W_{min}) \times N)$, where N is the total length of the text. Due to

the uncertain length of the watermarked segment inserted into the document, if there is a large difference between W_{max} and W_{min}, the worst-case complexity can reach $O(N^2)$.

4.3 Fix-Length Sliding Window

Fix-Length Sliding Window (FLSW) is a self-constructed method that uses a fixed-length window to traverse the text, calculating statistics within each local window. The text is flagged as watermarked if any statistic score exceeds the threshold. For localization, the method records each pair of indices (s, e) meeting the condition and uses a contiguous segment merging method with tolerance. While straightforward, FLSW struggles with varying watermark lengths due to its fixed-length nature, leading to dilution effects. Section 5.1 analyzes this effect using KGW (Kirchenbauer et al., 2023) and Aar (Aaronson and Kirchner, 2022) as examples, providing theoretical support for WaterSeeker.

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 use KGW and Aar as examples, since most watermarking algorithms are slight modifications of these two methods. Thus, analyzing these approaches offers broad applicability.

For the KGW method, assuming $\gamma_1 > \gamma$ is the proportion of green tokens in the watermarked part. Let's analyze the effect of window size W on this statistic: (1) When $W < L$ (window size smaller than watermark length):

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

(2) When $W > L$ (window size larger than watermark length):

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

From this, we can conclude that **when $W = L$, the z-score reaches its maximum**.

During detection, we aim for a higher z-score for positive cases while setting an appropriate threshold to balance the false positive rate. Next, we will analyze the constraints on the z-threshold z^* when the false positive rate within the specified window is set to be lower than a target value α.

We start with the binomial distribution $B(W, \gamma)$ that describes the number of green tokens in a window of size W. For large sample sizes (typically $W > 50$), we can approximate this as $|s|_G \sim N(W\gamma, W\gamma(1-\gamma))$. Therefore, we have $z \sim N(0, 1)$. The false positive rate α represents the area in the right tail of this distribution beyond z^*. Mathematically, this is expressed as $\alpha = 1 - \Phi(z^*)$. Solving for z^*, we get: $z^* = \Phi^{(-1)}(1-\alpha)$, which is a constant value for different W.

For the Aar method, the p-value is calculated by applying a gamma transformation to the sum of correlation values of all the tokens:

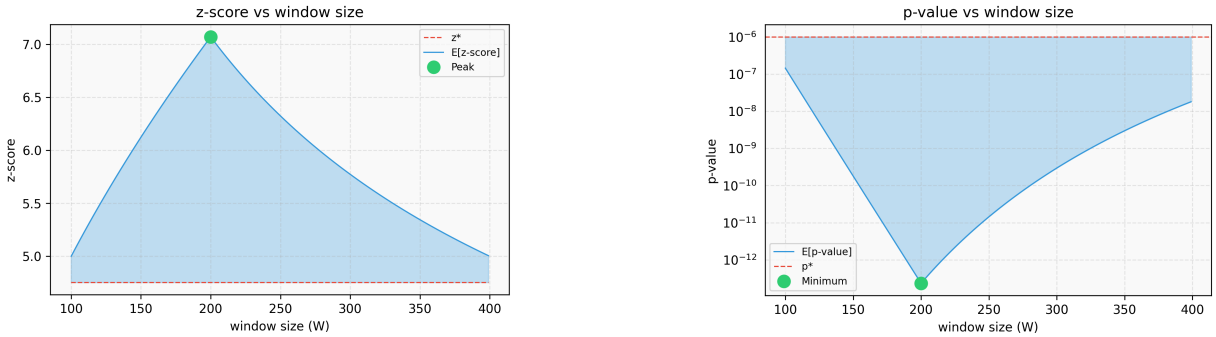
$$\text{p-value} = \Gamma\left(\sum_{i=1}^{N} \log\left(\frac{1}{1-u_i}\right), N, \text{loc} = 0, \text{scale} = 1\right).$$

We denote $\mu = E[\log(\frac{1}{1-u_i})]$ as the expected correction value for a single token. Similar to KGW, we assume $\mu_1 > \mu$ represents the expected correlation value for the watermarked portion. Due to the complexity of the Gamma-Transformation, the detailed proof can be found in Appendix C. Based on the above proof, we simulated the expected score of the statistic and the corresponding threshold as they vary with W for $L = 200$ using real data, as shown in Figure 2. It demonstrates that when the window size corresponds to the gold index, the highest detection rate can be achieved while maintaining an acceptable FPR. Therefore, the **watermark dilution effect** refers to the situation where the window used for detection is not optimal, resulting in the computed statistics not reaching their peak or minimum, which in turn affects detection performance.

5.2 Suspicious Region Localization

Based on the theoretical analysis presented in Section 5.1, the design concept of WaterSeeker employs a coarse-to-fine process to gradually approximate the gold index, thereby achieving high detection performance. In the first step (coarse step), WaterSeeker uses a localization algorithm to identify suspicious watermarked regions. This process narrows the detection target to a small segment that encompasses the ground truth segment, while ensuring relatively small deviation from the gold index. Below is the detailed process of the localization algorithm:

(1) Score List Computation: A small sliding window (i.e., $W = 50$) is used to traverse the entire text. For each window position, a statistical measure is computed to reflect the average watermark



(a) KGW, $\alpha = 10^{-6}, \gamma = 0.5, \gamma_1 = 0.75$

(b) Aar, $\alpha = 10^{-6}, \mu = 1.0, \mu_1 = 1.6$

Figure 2: Expected score of the statistics and the corresponding threshold across various W.

intensity. In the KGW method, this statistic is $|s|_G/W$, representing the density of green tokens within the window. In the Aar method, it is calculated as $\frac{\sum \log(1/(1-u))}{W}$, representing the average correlation value between tokens and the pseudo-random sequence within the window. This results in a score list s of length $N - W + 1$, where s_i represents the average watermark intensity from n_i to $n_i + W$.

(2) Anomaly Extraction: In this step, we design an anomaly extraction algorithm to identify outliers from the score list. We calculate the mean of the score list, the top-k mean, and the standard deviation, denoted as s_{mean}, $s_{\text{top-k-mean}}$, and s_{std}, respectively. The extracted outliers are the points that satisfy the following condition, where θ_1 and θ_2 are hyperparameters:

$$\text{score} > s_{\text{mean}} + \max((s_{\text{top-k-mean}} - s_{\text{mean}}) \cdot \theta_1, s_{\text{std}} \cdot \theta_2).$$

In samples containing a watermarked segment, the first term in the max expression becomes significant. Introducing the top-k mean allows for adaptive adjustment of the deviation from the mean based on different watermark strengths. Conversely, in non-watermarked samples, the difference between the top-k mean and the mean is relatively small. In this case, to reduce false positives, a point must deviate from the mean by at least $s_{\text{std}} \cdot \theta_2$ to be considered an anomaly.

(3) Fragment Connection: In this step, we use a connecting method with a certain tolerance to link nearby outliers, then filter out segments that are too short, returning a list of indices.

The scores within watermarked segments are relatively stable and close to the top-k mean, while scores outside the watermark remain stable and close to the overall mean due to the dilution effect of large natural text. When using $(s_{\text{top-k-mean}} - s_{\text{mean}}) \cdot \theta_1 \quad (\theta_1 < 1)$ as a distinction criterion, the extracted abnormal segment's start and end points (s' and e') generally satisfy $s' \in (s - W, s)$ and $e' \in (e, e + W)$. This ensures that the extracted suspicious watermarked regions likely cover the actual segments, with starting and ending deviations within a window size.

5.3 Local Traverse Detection

After obtaining the localization results from the first step, the second phase (fine step), local traverse detection, conducts a more detailed verification. For each (s', e') pair in the localization results, traverse inwards through the segments where the start point falls within $[s', s'+W)$ and the end point within $(e' - W, e']$. Based on the previous analysis of the ranges of s' and e', this traversal has a high probability of reaching the gold index. Perform full-text detection sequentially on these segments, and select the statistic with the most significance to compare with the threshold. The pseudocode for the entire WaterSeeker algorithm can be found in Algorithm 3.

5.4 Time Complexity Analysis

Time Complexity of WaterSeeker: Suspicious Region Localization has a time complexity of $O(N)$. This step involves score list computation, anomaly detection, and fragment connection, each with a time complexity of $O(N)$, where N is the text length. Local Traverse Detection has a time complexity of $O(W^2)$. For each suspicious region, this step examines W^2 windows, where W is the window size. The total time complexity of WaterSeeker 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 E) suffices for a smooth and low-noise representation of the surrounding watermark intensity. Thus, the

Table 1: We compared the detection performance of WaterSeeker with other methods, including Full-text Detection, WinMax (Kirchenbauer et al., 2024), and FLSW. The reported metrics include FPR, FNR, F1 score, and the average IoU between detected segments and ground truth segments in positive samples.

Model	Method	KGW				Aar			
		FPR ↓	FNR ↓	F1 ↑	IoU ↑	FPR ↓	FNR ↓	F1 ↑	IoU ↑
Llama-2-7b	Full-text Detection	0.000	1.000 [0.987]	0.000 [0.026]	0.000	0.000	1.000 [0.967]	0.000 [0.065]	0.001
	WinMax	0.017	0.273	0.834	0.661	0.017	0.410	0.734	0.537
	FLSW-100	0.003	0.473	0.688	0.448	0.003	0.547	0.622	0.363
	FLSW-200	0.003	0.440	0.716	0.417	0.000	0.577	0.595	0.338
	FLSW-300	0.007	0.683	0.479	0.313	0.000	0.783	0.356	0.265
	FLSW-400	0.003	0.897	0.187	0.230	0.003	0.933	0.125	0.191
	WaterSeeker(Ours)	0.017	0.313	0.806	0.624	0.010	0.440	0.713	0.507
Mistral-7b	Full-text Detection	0.000	1.000 [0.990]	0.000 [0.020]	0.000	0.000	1.000 [0.973]	0.000 [0.052]	0.001
	WinMax	0.010	0.277	0.835	0.656	0.013	0.370	0.767	0.562
	FLSW-100	0.000	0.500	0.667	0.433	0.000	0.497	0.670	0.411
	FLSW-200	0.000	0.410	0.742	0.452	0.003	0.560	0.610	0.359
	FLSW-300	0.003	0.577	0.593	0.341	0.003	0.827	0.295	0.254
	FLSW-400	0.003	0.810	0.318	0.244	0.007	0.943	0.107	0.195
	WaterSeeker(Ours)	0.007	0.290	0.827	0.641	0.010	0.390	0.753	0.542

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 (0.3=strong, 0.2=medium, 0.1=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). 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. The specific distributions of watermark strengths and lengths are detailed in Appendix F.

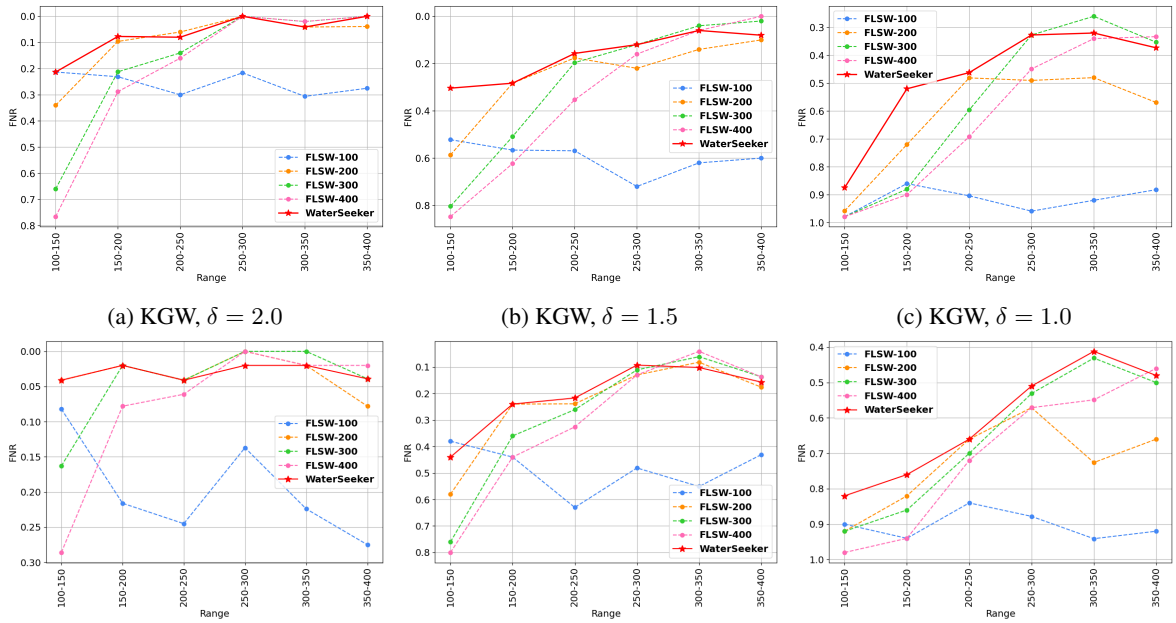
Baselines: As introduced in Section 4, Full-text Detection and WinMax (Kirchenbauer et al., 2024) are chosen, as well as Fix-Length Sliding Window method with W of 100, 200, 300 and 400.

Hyper-parameters: The parameters related to WaterSeeker are as follows: $W = 50, k = 20$, with a tolerance for fragment connection set to 100. The parameter θ_1 is set to 0.5, and θ_2 is set to 1.5 for both algorithms. The threshold selection within the specified window is detailed in Appendix D. Notably, careful threshold selection is crucial for maintaining an acceptable false positive rate, as traversing long texts is prone to accumulating false positives. Evaluation-related parameters: θ mentioned in Section 3.2 is set to 0.5.

6.2 Detection Performance Analysis

In Table 1, we present the detection performance of WaterSeeker alongside various baseline algorithms, evaluated on four datasets. Given that full-text detection lacks localization capabilities, its FNR is necessarily 1 under the evaluation rule requiring IoU $> \theta$ for successful detection. Therefore, we additionally report the FNR and corresponding F1 score for Full-text Detection without the IoU $> \theta$ constraint, highlighted in red. Additional performance results for WaterSeeker under various mixing ratios are presented in Appendix G.

The results demonstrate that WaterSeeker achieves lower FNR and higher F1 score, significantly surpassing other watermark detection algorithms, including Full-text Detection, FLSW-100,



(a) KGW, $\delta = 2.0$ (b) KGW, $\delta = 1.5$ (c) KGW, $\delta = 1.0$

(d) Aar, temp=0.3 (e) Aar, temp=0.2 (f) Aar, temp=0.1

Figure 3: This figure compares the detection performance of WaterSeeker and FLSW under varying watermark lengths and strengths, using Llama-2-7b as the generation model.

Table 2: Average detection time per sample for various detection methods ($N \approx 10,000$, Unit: s).

	Full-text Detection	WinMax	FLSW-100	FLSW-200	FLSW-300	FLSW-400	WaterSeeker
KGW	1.70	14.16	1.76	1.76	1.79	1.80	1.75
Aar	0.54	2733.78	1.62	1.65	1.66	1.64	1.68

FLSW-200, FLSW-300, and FLSW-400, while being comparable to WinMax. Based on the principle that WinMax is guaranteed to reach the gold index by evaluating all possible windows, it represents the upper bound of detection performance. However, subsequent experiments show that WinMax has extremely high time complexity (Section 6.4). Furthermore, the IoU results in the table reveal that WaterSeeker possesses a significantly higher localization capability compared to the baselines, again comparable to WinMax.

6.3 Adaptability Analysis

While the main experiment used four datasets with a mix of watermarked segments of varying strengths and lengths, this section compares the detection capabilities of WaterSeeker and the FLSW method across specific watermark strengths and lengths. Figure 3 shows that the performance of different detection algorithms follows a consistent trend: watermarked segments with higher strengths and longer lengths are more easily detected. FLSW performs well within a length range close to its window size but shows poor adaptability in other ranges due to its fixed-length nature, which leads to dilution effects when detecting longer or shorter watermarked segments. In contrast, WaterSeeker exhibits good adaptability across different length ranges by using anomaly extraction techniques and incorporates a top-k score mechanism to adapt to varying watermark strengths.

6.4 Time Complexity Analysis

Table 2 shows the average time spent per sample for WaterSeeker and other baseline algorithms during detection. WaterSeeker's time is comparable to full-text detection and FLSW, and significantly lower than WinMax. Sections 4.2 and 5.4 analyze the theoretical time complexity of WinMax and WaterSeeker, respectively. This section offers a more comprehensive analysis based on experimental results, considering constant factors and investigating the significant disparity in WinMax's execution time between KGW and Aar.

Let t_1 be the time to compute a score in the score list, and t_2 be the time to compute the watermark detection statistic within a local window. WinMax's time expenditure to detect a sample is

Table 3: This table presents the contributions of the first step of WaterSeeker: Suspicious Segment Localization. It lists the average coverage of localization results relative to the ground truth segments under various watermark algorithms and watermark strengths, as well as the average offsets of the detected start and end indices.

Metrics	KGW			Aar		
	$\delta = 2.0$	$\delta = 1.5$	$\delta = 1.0$	temp=0.3	temp=0.2	temp=0.1
Average Coverage	0.992	0.984	0.953	0.996	0.991	0.977
Average Offset (Start)	20.799	30.057	13.992	22.559	30.546	20.624
Average Offset (End)	25.985	25.299	28.030	29.459	33.059	30.304

Table 4: Comparison of detection performance with and without Local Traverse Detection across four datasets.

Model	Setting	KGW				Aar			
		FPR ↓	FNR ↓	F1 ↑	IoU ↑	FPR ↓	FNR ↓	F1 ↑	IoU ↑
Llama-2-7b	w. local traverse	0.017	0.313	0.806	0.624	0.010	0.440	0.713	0.507
	w/o local traverse	0.003	0.390	0.756	0.520	0.000	0.450	0.710	0.462
Mistral-7b	w. local traverse	0.007	0.290	0.827	0.641	0.010	0.390	0.753	0.542
	w/o local traverse	0.003	0.380	0.763	0.534	0.003	0.413	0.738	0.490

$N \cdot t_1 + N \cdot (W_{\max} - W_{\min}) \cdot t_2$. For WaterSeeker, the localization step consists of score list computation, anomaly detection, and fragment connection, each with complexity $O(N)$. Denoting the time for the latter two sub-steps as $t_3 \cdot N$, the total time for the first step is $N \cdot t_1 + N \cdot t_3$. The local traverse detection step requires $W^2 \cdot t_2$ time. Thus, WaterSeeker's total time expenditure is $N \cdot t_1 + N \cdot t_3 + W^2 \cdot t_2$.

For the Aar (Aaronson and Kirchner, 2022) watermarking method, calculating the watermark detection statistic involves a complex GammaTransform function, resulting in a larger t_2. This leads to a significant time difference between WinMax and WaterSeeker due to the large disparity in coefficients preceding t_2. In this experiment, with $N = 10,000, W_{\max} = 400, W_{\min} = 100, W^2 = 2,500$, the coefficient difference is a factor of 1,200. For the KGW (Kirchenbauer et al., 2023) watermarking method, calculating the z-score within a local window is relatively straightforward, reducing the time difference between WinMax and WaterSeeker.

WaterSeeker is a general watermark detection method with consistently low time expenditure across various watermarking algorithms. In contrast, WinMax exhibits extremely high time costs when calculating the watermark detection statistic is complex, making it challenging for practical use.

6.5 Ablation Study

We analyze the effectiveness of the two stages of WaterSeeker through an ablation study. The first stage, Suspicious Segment Localization, aims to achieve high coverage of the ground truth segments while keeping the start and end offsets within a specified window size. This ensures subsequent local traversals can reach the gold index. Table 3 shows that Step 1 achieves an average coverage exceeding 0.95 across various watermark algorithms and strengths, with average start and end offsets less than 50, remaining within the designated window size. Step 1 fulfills its purpose effectively.

Local Traverse Detection performs a localized traversal based on the segments from Step 1, allowing for more refined verification within the window. Table 4 shows that across different LLMs and watermarking algorithms, Local Traverse consistently enhances detection F1 score and average IoU compared to directly applying full-text detection with the localization results, making it an indispensable component of WaterSeeker.

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, such as threshold settings, is crucial and can be challenging in different environments. Currently, parameters are manually tuned based on observations, so developing adaptive tuning methods, potentially using machine learning, would enhance WaterSeeker's robustness and practicality across diverse scenarios.

References

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

Canyu Chen and Kai Shu. 2024. Can LLM-generated misinformation be detected? In *The Twelfth International Conference on Learning Representations*.

Miranda Christ, Sam Gunn, and Or Zamir. 2024. Undetectable watermarks for language models. In *The Thirty Seventh Annual Conference on Learning Theory*, pages 1125–1139. PMLR.

Wikimedia Foundation. Wikimedia downloads.

Zhiwei He, Binglin Zhou, Hongkun Hao, Aiwei Liu, Xing Wang, Zhaopeng Tu, Zhuosheng Zhang, and Rui Wang. 2024. Can watermarks survive translation? on the cross-lingual consistency of text watermark for large language models. *arXiv preprint arXiv:2402.14007*.

Zhengmian Hu, Lichang Chen, Xidong Wu, Yihan Wu, Hongyang Zhang, and Heng Huang. 2024. Unbiased watermark for large language models. In *The Twelfth International Conference on Learning Representations*.

Albert Q Jiang, Alexandre Sablayrolles, Antoine Roux, Arthur Mensch, Blanche Savary, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Emma Bou Hanna, Florian Bressand, et al. 2024. Mixtral of experts. *arXiv preprint arXiv:2401.04088*.

John Kirchenbauer, Jonas Geiping, Yuxin Wen, Jonathan Katz, Ian Miers, and Tom Goldstein. 2023. A watermark for large language models. In *International Conference on Machine Learning, ICML 2023, 23-29 July 2023, Honolulu, Hawaii, USA*, volume 202 of *Proceedings of Machine Learning Research*, pages 17061–17084. PMLR.

John Kirchenbauer, Jonas Geiping, Yuxin Wen, Manli Shu, Khalid Saifullah, Kezhi Kong, Kasun Fernando, Aniruddha Saha, Micah Goldblum, and Tom Goldstein. 2024. On the reliability of watermarks for large language models. In *The Twelfth International Conference on Learning Representations*.

Rohith Kuditipudi, John Thickstun, Tatsunori Hashimoto, and Percy Liang. 2024. Robust distortion-free watermarks for language models. *Transactions on Machine Learning Research*.

Aiwei Liu, Leyi Pan, Xuming Hu, Shuang Li, Lijie Wen, Irwin King, and Philip S. Yu. 2024a. An unforgeable publicly verifiable watermark for large language models. In *The Twelfth International Conference on Learning Representations*.

Aiwei Liu, Leyi Pan, Xuming Hu, Shiao Meng, and Lijie Wen. 2024b. A semantic invariant robust watermark for large language models. In *The Twelfth International Conference on Learning Representations*.

Aiwei Liu, Leyi Pan, Yijian Lu, Jingjing Li, Xuming Hu, Lijie Wen, Irwin King, and Philip S Yu. 2023a. A survey of text watermarking in the era of large language models. *arXiv preprint arXiv:2312.07913*.

Aiwei Liu, Qiang Sheng, and Xuming Hu. 2024c. Preventing and detecting misinformation generated by large language models. In *Proceedings of the 47th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '24, page 3001–3004, New York, NY, USA. Association for Computing Machinery.

Yang Liu, Yuanshun Yao, Jean-Francois Ton, Xiaoying Zhang, Ruocheng Guo, Hao Cheng, Yegor Klochkov, Muhammad Faaiz Taufiq, and Hang Li. 2023b. Trustworthy llms: A survey and guideline for evaluating large language models' alignment. *arXiv preprint arXiv:2308.05374*.

Yijian Lu, Aiwei Liu, Dianzhi Yu, Jingjing Li, and Irwin King. 2024. An entropy-based text watermarking detection method. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 11724–11735, Bangkok, Thailand. Association for Computational Linguistics.

Leyi Pan, Aiwei Liu, Zhiwei He, Zitian Gao, Xuandong Zhao, Yijian Lu, Binglin Zhou, Shuliang Liu, Xuming Hu, Lijie Wen, et al. 2024. Markllm: An open-source toolkit for llm watermarking. *arXiv preprint arXiv:2405.10051*.

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

Matthias C Rillig, Marlene Ågerstrand, Mohan Bi, Kenneth A Gould, and Uli Sauerland. 2023. Risks and benefits of large language models for the environment. *Environmental Science & Technology*, 57(9):3464–3466.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. 2023. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*.

Lean Wang, Wenkai Yang, Deli Chen, Hao Zhou, Yankai Lin, Fandong Meng, Jie Zhou, and Xu Sun. 2024. Towards codable watermarking for injecting multi-bits information to LLMs. In *The Twelfth International Conference on Learning Representations*.

KiYoon Yoo, Wonhyuk Ahn, and Nojun Kwak. 2023. Advancing beyond identification: Multi-bit watermark for language models. *arXiv preprint arXiv:2308.00221*.

Xuandong Zhao, Prabhanjan Vijendra Ananth, Lei Li, and Yu-Xiang Wang. 2024. Provable robust watermarking for AI-generated text. In *The Twelfth International Conference on Learning Representations*.

A Details of Representative Watermarking Algorithms

A.1 KGW

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.

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.

A.2 Aar

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.

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.

B Pseudocode of WaterSeeker and Detection Baselines

Pseudocode of WaterSeeker, WinMax and FLSW could be found in Algorithm 3, 1 and 2, respectively.

Algorithm 1 WinMax Algorithm

```
procedure WINMAXDETECTION(tokens, W_min, W_max, threshold)
    hasWatermark ← False, indices ← [ ]
    maxStat ← - ∞, bestIndex ← None
    for W ∈ [W_min, W_max] do
        for i in 0 to len(tokens) − W do
            stat ← CalculateStatistics(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 2 FLSW Algorithm

```
procedure FLSWDETECTION(tokens, W, threshold)
    hasWatermark ← False
    indices ← [ ]
    for i in 0 to len(tokens) − W do
        stat ← CalculateStatistics(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
```

Algorithm 3 WaterSeeker Algorithm

```
procedure SUSPICIOUSREGIONLOCALIZATION(tokens, W, k, θ1, θ2)
    Compute score list using sliding window
    Calculate s_mean, s_top-k-mean, and s_std
    Detect anomalies using threshold:
    s_mean + max((s_top-k-mean − s_mean) · θ1, s_std · θ2)
    Connect nearby outliers and filter short segments
    return filteredSegments
end procedure
procedure LOCALTRAVERSEDETECTION(tokens, suspiciousRegions, W, threshold)
    hasWatermark ← False
    indices ← [ ]
    for (s′, e′) in suspiciousRegions do
        maxScore = −∞, maxIndice = ()
        for s ∈ [s′, s′ + W) and e ∈ (e′ − W, e′] do
            if DetectWatermark(tokens[s : e])> maxScore then
                maxScore = DetectWatermark(tokens[s : e])
                maxIndice = (s, e)
            end if
        end for
        if maxScore > threshold then
            hasWatermark ← True
            indices.append(maxIndice)
        end if
    end for
    return hasWatermark, indices
end procedure
procedure WATERSEEKER(tokens, W, k, θ1, θ2, threshold)
    suspiciousRegions ← SuspiciousRegionLocalization(tokens, W, k, θ1, θ2)
    hasWatermark, indices ← LocalTraverseDetection(tokens, suspiciousRegions, W, threshold)
    return hasWatermark, indices
end procedure
```

C Detailed Proof for Aar

C.1 Expected p-value of watermarked text

In this sub-section, we aim to analyze how the expected p-value for watermarked text varies with changes in the detection window size W.

Recall the p-value calculation formula:

$$\text{p-value} = \Gamma(S, W, \text{loc} = 0, \text{scale} = 1), \tag{12}$$

where $S = \sum_{i=1}^{W} \log(\frac{1}{1-u_i})$, and W is the window size.

For watermarked tokens, $E[\log(\frac{1}{1-u_i})] = \mu_1$, and for non-watermarked tokens, $E[\log(\frac{1}{1-u_i})] = \mu_0$, where $\mu_1 > \mu_0$.

The expectation of S for different W values is as follows:

- When $W \leq L$: $E[S] = W\mu_1$
- When $W > L$: $E[S] = L\mu_1 + (W - L)\mu_0$

The expectation of p-value is:

$$E[\text{p-value}] = E[\Gamma(S, W, \text{loc} = 0, \text{scale} = 1)] \tag{13}$$

Since GammaTransform is non-linear, we cannot directly substitute $E[S]$. However, we can use Jensen's inequality to obtain an approximation:

$$E[\Gamma(S, W, 0, 1)] \geq \Gamma(E[S], W, 0, 1) \tag{14}$$

Therefore, we can analyze $\Gamma(E[S], W, 0, 1)$ to obtain a lower bound for the expectation of p-value. Define function $f(W) = \Gamma(E[S], W, 0, 1)$:

- When $W \leq L$: $f(W) = \Gamma(W\mu_1, W, 0, 1)$
- When $W > L$: $f(W) = \Gamma(L\mu_1 + (W - L)\mu_0, W, 0, 1)$

Analyzing the behavior of $f(W)$:

- When $W \leq L$, both $E[S]$ and the shape parameter W increase as W increases.
- When W just exceeds L, the growth rate of $E[S]$ suddenly decreases (from μ_1 to μ_0), while the shape parameter W continues to increase linearly.

Consider the behavior of $f(W)$ near $W = L$:

- When W increases from $L - \epsilon$ to L, $E[S]$ increases by $\epsilon\mu_1$, and the shape parameter increases by ϵ.
- When W increases from L to $L + \epsilon$, $E[S]$ increases by $\epsilon\mu_0$, and the shape parameter increases by ϵ.

Since $\mu_1 > \mu_0$, at the point $W = L$, the rate of decrease of $f(W)$ suddenly slows down. This suggests that $f(W)$ is likely to reach its minimum value at $W = L$, or at a point very close to L. While this analysis does not strictly prove that the expectation of p-value is minimized exactly at $W = L$, it strongly suggests that the expectation of p-value is likely to reach its minimum value at or very near $W = L$.[1]

C.2 Calculation of p-threshold to control false positive rate

Similar to the analysis for KGW method, we also need to analyze the constraints on the p-value threshold p^* when the false positive rate within the specified window is set to be lower than a target value α.

For non-watermarked text, $u_i \sim \text{Uniform}([0, 1])$. Consequently, the test statistic S follows a Gamma distribution: $S \sim \text{Gamma}(W, 1)$, where W is the shape parameter and 1 is the scale parameter. The p-value is calculated using the Gamma CDF:

$$\text{p-value} = 1 - \text{GammaCDF}(S, W, 1), \tag{15}$$

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: $P(S > \text{GammaInv}(1 - p^*, W, 1)) = \alpha$, where GammaInv is the inverse of the Gamma CDF.

Since S follows a $\text{Gamma}(W, 1)$ distribution for non-watermarked text, we can express this as:

$$1 - \text{GammaCDF}(\text{GammaInv}(1 - p^*, W, 1), W, 1) = \alpha. \tag{16}$$

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

D 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}.

Table 5: 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

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

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 5 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.

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

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

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

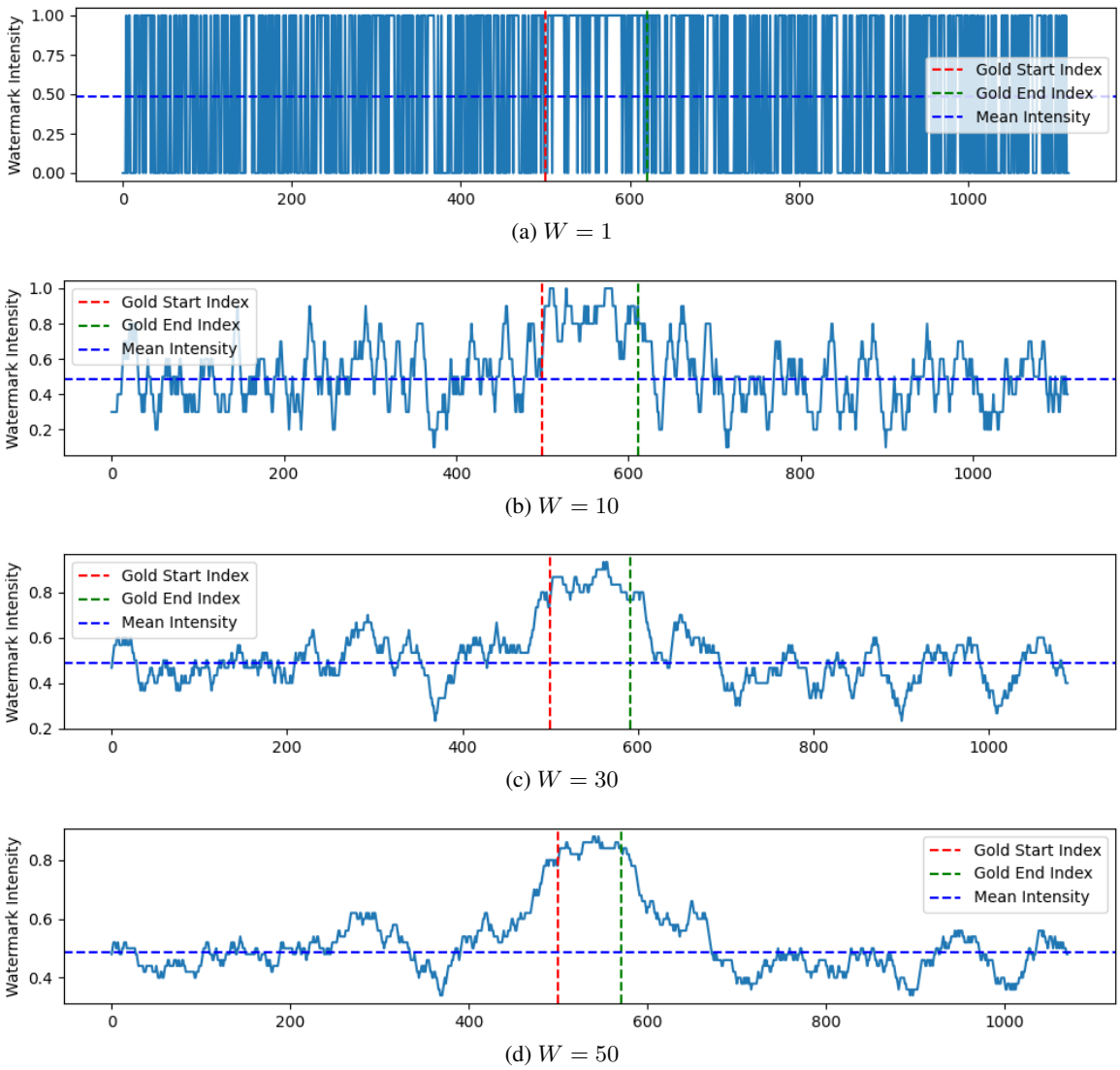
Expanding this, we have:

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

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

We can further simplify:

[1]For a more rigorous proof, a deeper mathematical analysis of the GammaTransform function or numerical simulations would be necessary.



(a) $W = 1$

(b) $W = 10$

(c) $W = 30$

(d) $W = 50$

Figure 4: Case study: Impact of varying window sizes on watermark intensity calculation in the KGW algorithm.

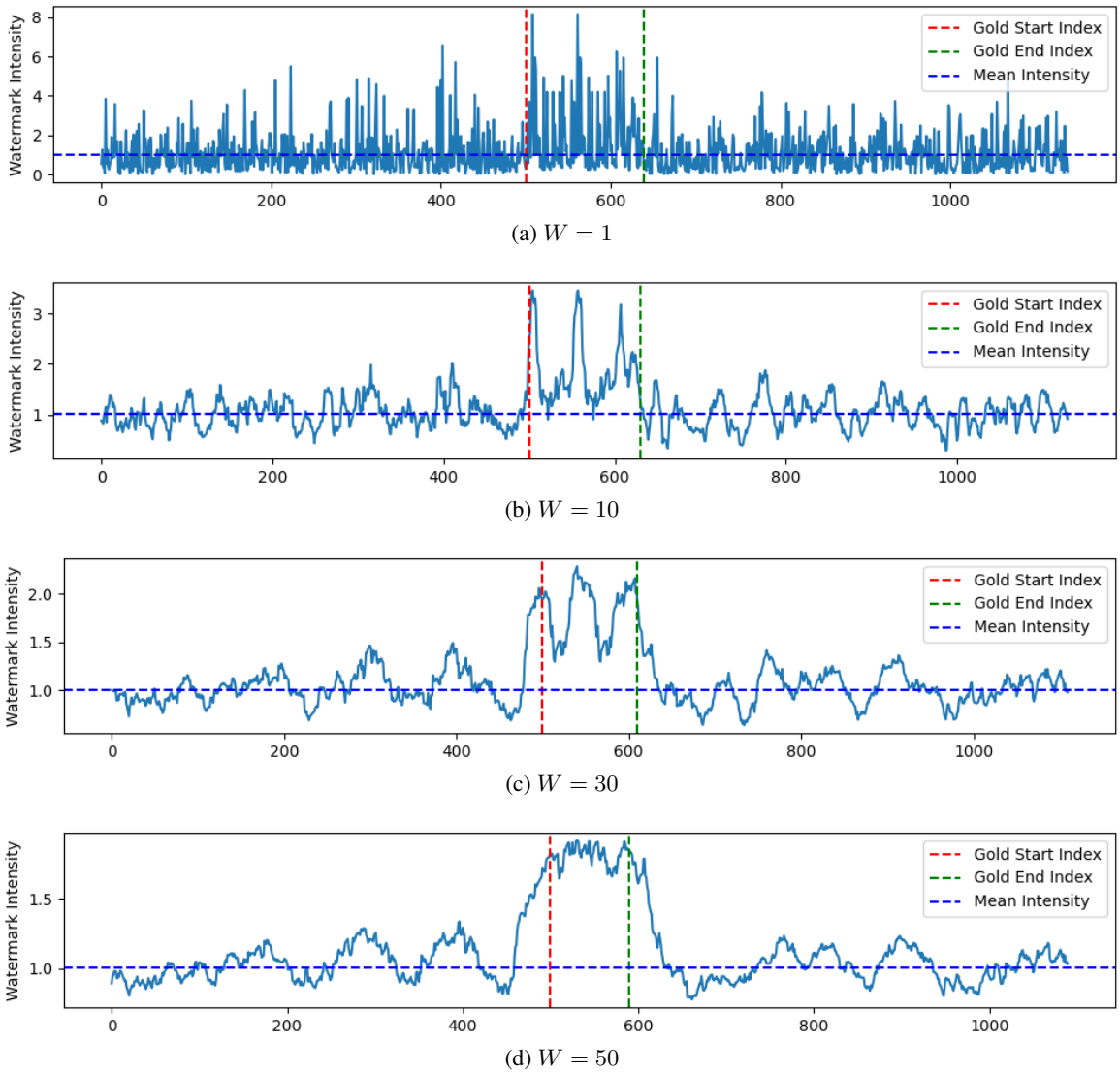
$$P(z \geq z^*) = \sum_{k=0}^{W} \binom{W}{k} \gamma^k (1-\gamma)^{W-k} \mathbb{I}\left\{k \geq \gamma W + z^* \sqrt{W(1-\gamma)}\right\}.$$

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.

For Aar, as analyzed in Appendix C, $p^* = \alpha$, therefore set $p^* = 10^{-6}$ for all lengths.

E 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.



(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 Aar algorithm.

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 4 and 5 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 5c (the ground truth segment part). Overall, $W = 50$ demonstrates the least noise. Consequently, we adopt $W = 50$ for our main experiments.

F Sample Distribution in the Main Experiment

It can be observed from Figure 6 that all four datasets include samples of watermarked segments with varying intensities and lengths, demonstrating the comprehensiveness and fairness of the dataset construction.

G Further Experimental Results under Different Mixing Ratios

In main experiment, 100-400 watermarked tokens are inserted into non-watermarked text of 10,000 tokens, resulting in watermarked text mixing ratios of 1% to 4%. This section expands the evaluation by testing the performance of various methods across a broader range of mixing ratios, achieved by adjusting the length of the non-watermarked text. We deliberately avoid adjusting the mixing ratio by inserting multiple watermark segments into the non-watermarked text, as the WinMax method can only return

(a) KGW, Llama

(b) Aar, Llama

(c) KGW, Mistral

(d) Aar, Mistral

Figure 6: Distribution of sample numbers for the four datasets involved in the main experiment.

the highest-scoring segment and is not designed for multi-segment insertion scenarios. Additionally, we do not use fixed-length watermark segments because in real-world detection scenarios, the length of the watermark segment is typically unknown. Moreover, if the watermark length were fixed and known, a simple Fixed-length Sliding Window detection method (with the window size equal to the watermark segment length) would suffice to solve the problem.

We provided further experimental results under three different settings:

1. 100-400 watermarked tokens are inserted into 5,000 non-watermarked tokens, with a mixing ratio of 2% (100 / 5100) -7.4% (400 / 5400). The results are demonstrated in Table 6.

2. 100-400 watermarked tokens are inserted into 2,000 non-watermarked tokens, with a mixing ratio of 4.8% (100 / 2100) -16.7% (400 / 2400). The results are demonstrated in Table 7.

3. 100-400 watermarked tokens are inserted into 500 non-watermarked tokens, with a mixing ratio of 16.7% (100 / 600) -44.4% (400 / 900). The results are demonstrated in Table 8.

Experimental results demonstrate that WaterSeeker exhibits high robustness across various mixing ratios, showing consistent performance trends across different settings. Specifically, WaterSeeker significantly outperforms full-text detection and FLSW methods in both detection accuracy and localization effectiveness, while achieving comparable performance to the upper bound method, WinMax.

Table 6: Comparison of WaterSeeker with other baselines under the mixing ratio of 2%-7.4%.

Model	Method	KGW				Aar			
		FPR ↓	FNR ↓	F1 ↑	IoU ↑	FPR ↓	FNR ↓	F1 ↑	IoU ↑
Llama-2-7b	Full-text Detection	0.000	1.000 [0.990]	0.000 [0.020]	0.001	0.000	1.000 [0.957]	0.000 [0.083]	0.002
	WinMax	0.003	0.317	0.810	0.621	0.010	0.380	0.761	0.565
	FLSW-100	0.000	0.520	0.649	0.413	0.003	0.523	0.644	0.375
	FLSW-200	0.000	0.457	0.704	0.404	0.000	0.560	0.611	0.350
	FLSW-300	0.003	0.663	0.502	0.297	0.000	0.767	0.378	0.283
	FLSW-400	0.003	0.867	0.235	0.230	0.000	0.920	0.148	0.205
	WaterSeeker(Ours)	0.003	0.357	0.781	0.570	0.010	0.407	0.740	0.536
Mistral-7b	Full-text Detection	0.000	1.000 [0.990]	0.000 [0.020]	0.000	0.000	1.000 [0.937]	0.000 [0.119]	0.004
	WinMax	0.003	0.227	0.871	0.700	0.010	0.370	0.768	0.570
	FLSW-100	0.000	0.503	0.664	0.443	0.000	0.543	0.627	0.364
	FLSW-200	0.000	0.350	0.788	0.497	0.003	0.540	0.629	0.362
	FLSW-300	0.000	0.527	0.643	0.392	0.003	0.763	0.382	0.269
	FLSW-400	0.000	0.783	0.357	0.281	0.003	0.910	0.165	0.203
	WaterSeeker(Ours)	0.000	0.243	0.861	0.676	0.010	0.383	0.758	0.548

Table 7: Comparison of WaterSeeker with other baselines under the mixing ratio of 4.8%-16.7%.

Model	Method	KGW				Aar			
		FPR ↓	FNR ↓	F1 ↑	IoU ↑	FPR ↓	FNR ↓	F1 ↑	IoU ↑
Llama-2-7b	Full-text Detection	0.000	1.000 [0.967]	0.000 [0.065]	0.005	0.000	1.000 [0.873]	0.000 [0.225]	0.018
	WinMax	0.003	0.257	0.851	0.669	0.003	0.413	0.738	0.533
	FLSW-100	0.000	0.473	0.690	0.440	0.003	0.570	0.600	0.350
	FLSW-200	0.000	0.420	0.734	0.446	0.000	0.607	0.565	0.327
	FLSW-300	0.003	0.630	0.539	0.339	0.000	0.777	0.365	0.257
	FLSW-400	0.003	0.850	0.260	0.246	0.000	0.883	0.209	0.200
	WaterSeeker(Ours)	0.003	0.287	0.831	0.642	0.003	0.443	0.714	0.501
Mistral-7b	Full-text Detection	0.000	1.000 [0.980]	0.000 [0.039]	0.003	0.000	1.000 [0.870]	0.000 [0.230]	0.017
	WinMax	0.000	0.220	0.876	0.710	0.003	0.353	0.783	0.590
	FLSW-100	0.000	0.470	0.693	0.447	0.000	0.487	0.678	0.415
	FLSW-200	0.000	0.360	0.780	0.497	0.003	0.500	0.665	0.371
	FLSW-300	0.000	0.533	0.636	0.386	0.003	0.760	0.386	0.285
	FLSW-400	0.000	0.747	0.404	0.288	0.003	0.887	0.203	0.212
	WaterSeeker(Ours)	0.000	0.227	0.872	0.701	0.003	0.363	0.776	0.574

H Further Application Scenario

WaterSeeker not only provides accurate detection results, but its localization capabilities also contribute to building a more transparent and interpretable AI detection system. As illustrated in Figure 7, the AI detection system powered by WaterSeeker receives documents for analysis and outputs three components: **(1) Detection Result**, which indicates whether AI assistance was utilized; **(2) Suspicious AI-generated Segments**, which highlights segments identified as potentially AI-generated; and **(3) AI Ratio**, which displays the proportion of content attributed to AI. For example, in the context of academic integrity, highlighting suspicious segments provides actionable evidence for assessments and appeals. Additionally, reporting the AI ratio allows for more flexible establishment of academic misconduct standards.

Table 8: Comparison of WaterSeeker with other baselines under the mixing ratio of 16.7%-44.4%.

Model	Method	KGW				Aar			
		FPR ↓	FNR ↓	F1 ↑	IoU ↑	FPR ↓	FNR ↓	F1 ↑	IoU ↑
Llama-2-7b	Full-text Detection	0.000	1.000 [0.787]	0.000 [0.352]	0.081	0.000	1.000 [0.693]	0.000 [0.469]	0.109
	WinMax	0.000	0.247	0.859	0.685	0.000	0.373	0.770	0.568
	FLSW-100	0.000	0.470	0.693	0.445	0.000	0.557	0.614	0.380
	FLSW-200	0.000	0.383	0.763	0.461	0.000	0.553	0.618	0.340
	FLSW-300	0.000	0.553	0.618	0.360	0.000	0.690	0.473	0.298
	FLSW-400	0.000	0.740	0.413	0.265	0.000	0.837	0.281	0.231
	WaterSeeker(Ours)	0.000	0.277	0.839	0.647	0.000	0.410	0.742	0.522
Mistral-7b	Full-text Detection	0.000	1.000 [0.847]	0.000 [0.266]	0.061	0.000	1.000 [0.697]	0.000 [0.465]	0.112
	WinMax	0.000	0.270	0.844	0.652	0.000	0.350	0.788	0.588
	FLSW-100	0.000	0.510	0.658	0.432	0.000	0.530	0.639	0.393
	FLSW-200	0.000	0.407	0.745	0.442	0.000	0.510	0.658	0.369
	FLSW-300	0.000	0.570	0.601	0.352	0.000	0.690	0.473	0.294
	FLSW-400	0.000	0.760	0.387	0.243	0.000	0.827	0.295	0.246
	WaterSeeker(Ours)	0.000	0.303	0.821	0.612	0.000	0.407	0.745	0.543

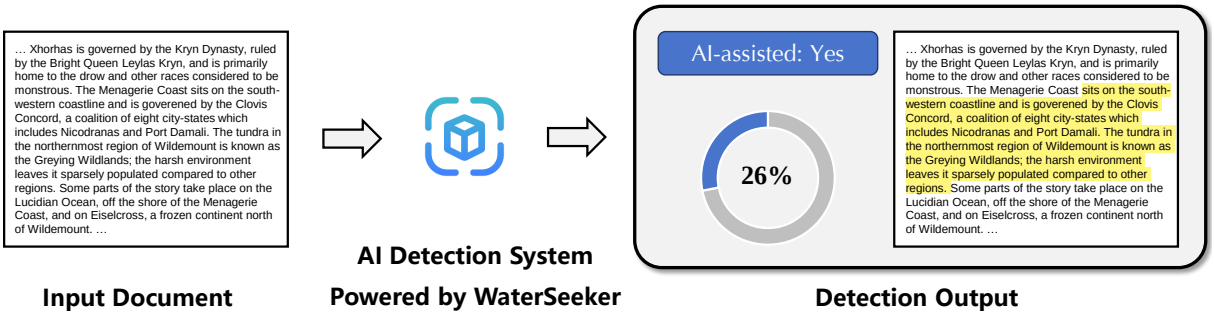


Figure 7: An illustration of WaterSeeker applied in AI detection system.