A Distortion-minimization Watermarking Framework for Large Language Models: Larger Capacity, Stronger Robustness and Higher Quality

Liming Zhai†, Xuezhou Shang†, Liyun Zhang, and Po Hu*
Hubei Provincial Key Laboratory of Artificial Intelligence and Smart Learning, School of Computer Science, National Language Resources Monitoring and Research Center for Network Media, Central China Normal University
limingzhai@ccnu.edu.cn, {shangxz, liyun_zhang}@mails.ccnu.edu.cn, phu@mail.ccnu.edu.cn

Abstract

Large language model (LLM) watermarking provides verifiable source identification for generated text, and its practical deployment requires large watermark capacity, strong robustness against attacks, and high text quality. However, existing methods often struggle to balance all these criteria, typically addressing them with separate designs. To overcome this, we propose a distortion-minimization watermarking (DMW) framework that unifies capacity, robustness and quality within a single optimization paradigm. This framework models robustness and quality as distortion costs for text modifications, minimizing the total distortion for a given watermark length to achieve an optimal trade-off. Specifically, we design several distortion costs: a robustness cost leveraging semantic invariance to resist attacks, and two quality costs guiding modifications toward low-cohesion, high-variability regions to reduce perceptual impact. We then propose periodically optimized syndrome-trellis codes (PO-STCs), formulating overall distortion minimization as a periodic shortest-path problem. This enables real-time optimization for sequential generation with flexible capacity control. Extensive experiments across diverse datasets and LLMs demonstrate DMW's superiority, outperforming state-of-the-art methods across all criteria. Notably, under severe paraphrasing attacks, DMW achieves a match rate up to 46.35% higher than the best baseline, while maintaining superior text quality.

1 Introduction

Large language models (LLMs) have experienced unprecedented advancements [2, 6, 39, 47], now generating text indistinguishable from human-written content in many scenarios, such as educational tutoring, programming assistance, and multilingual translation. Although the LLMs offer immense benefits, their growing accessibility and power have also raised serious concerns [21], including the mass production of undetectable misinformation, plagiarism of copyrighted material, and generation of malicious content (*e.g.*, phishing emails or fraudulent academic papers). Such risks necessitate effective countermeasures to ensure responsible and ethical deployment of LLM technologies [4].

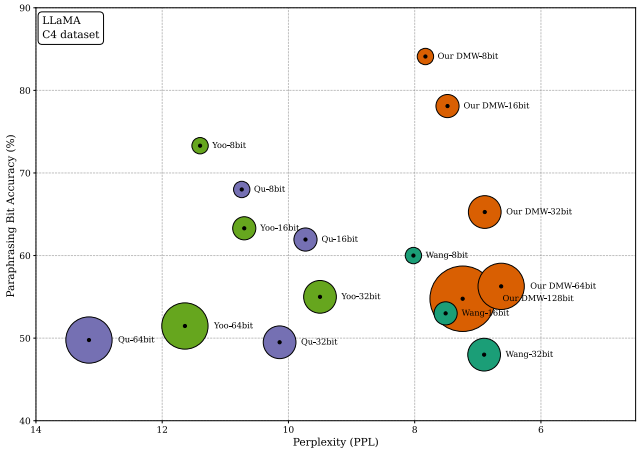


Figure 1: Comparison of capacity, robustness, and quality among Yoo *et al*. [45], Wang *et al*. [40], Qu *et al*. [33] and our DMW. Robustness is measured by bit accuracy against paraphrasing attack, and quality is measured by perplexity. Positions closer to the top-right indicate better performance.

Digital watermarking emerges as a promising technical solution to mitigate the misuse of LLMs. It enables authenticity verification, copyright protection, content traceability, and accountability enforcement for LLM outputs. An effective watermarking scheme for LLMs should simultaneously satisfy three critical criteria:

- **Large capacity:** Embeds sufficient watermark information (*e.g.*, user IDs) within the generated text.
- **Strong robustness:** Maintains the detectability or identifiability of watermarks even under various attacks.
- **High quality:** Preserves the naturalness and fluency of the watermarked text.

†Equal contribution. *Corresponding author.

Despite significant progress, existing watermarking methods often fail to simultaneously satisfy all three objectives. Most approaches employ disparate design strategies, optimizing for one or two criteria while compromising others. Early zero-bit watermarking focuses solely on detecting watermark presence without payload encoding to identify machine-generated text [22, 23, 49]. While these methods can achieve high robustness and preserve perceptual quality, their watermark capacity is inherently limited. Recent multi-bit watermarking significantly improves watermark capacity by encoding multiple bits into generated text, yet often underperform on other critical metrics [40, 46]. For instance, some multi-bit methods maintain text quality but produce fragile watermarks vulnerable to paraphrasing attacks [7, 40]. Conversely, others prioritize robustness at the expense of text quality [46]. This dilemma urgently demands a unified solution to balance the trade-offs among competing criteria.

In this paper, we propose a distortion-minimization watermarking (DMW) framework that unifies the trade-offs among capacity, robustness, and text quality for LLMs. Within this framework, robustness and quality are formulated as quantifiable distortion costs that capture the impact of text modifications on these performance metrics. Textual units that are semantically stable or exhibit lower cohesion are assigned lower costs, indicating that their modification is less likely to degrade robustness or perceptual quality. For a given watermark length, the DMW framework seeks to minimize the total costs across the entire text. Since these costs directly quantify the robustness and quality, minimizing the total costs inherently guarantees the watermark integrity and the perceptual quality of text. Formally, this transforms watermarking into a constrained optimization problem with tunable parameters for distortion costs and watermark length. Users can flexibly prioritize specific metrics (*e.g.*, robustness vs. quality) by adjusting these parameters, offering an interpretable and adaptable watermarking paradigm.

Our DMW also challenges the concept of unbiased or distortion-free LLM watermarking [7, 19, 25], which aims to preserve identical token distributions between original and watermarked texts to avoid detectable distortions. However, the claim of being distortion-free has been questioned by recent work [28], and another study [42] also indicates that truly distortion-free watermarks cannot be achieved in practice. Moreover, established theories [32] suggest that embedding watermarks into host data without incurring any cost is theoretically impossible. Therefore, we argue that rather than pursuing the unattainable goal of being distortion-free, a more pragmatic strategy is to focus on distortion minimization.

The DMW framework introduces several key technical innovations. To encode watermark into text, we propose a semantic projection approach that converts sentence-level semantics into specific bit sequences, facilitating subsequent embedding. To ensure that distortion costs accurately represent robustness and text quality, we design two distinct categories of cost functions. The first category is a robustness cost function, which leverages the invariance of sentence-level semantics. It prioritizes modifying sentences that are semantically stable, thus increasing resistance to meaning-preserving attacks. The second are two quality cost functions based on perplexity and entropy. It favors modifications to sentences with lower cohesion and greater variability, thereby maximally preserving overall text quality. To implement distortion minimization effectively, we develop periodically optimized syndrome-trellis codes (PO-STCs), which formulate the overall distortion minimization as a periodic shortest-path problem, enabling real-time watermark embedding for sequentially generated text.

The advantages of our DMW framework are summarized as follows (also see the comparison in Figure 1):

- It provides a unified solution to the capacity, robustness, and quality trade-offs inherent in LLM watermarking.
- It achieves large-capacity embedding, enabling sufficiently long watermarks proportional to sentence count.
- It exhibits strong resistance against a wide range of attacks, particularly paraphrasing attacks.
- It enhances the perceptual quality of the watermarked text, ensuring it remains fluent, diverse and natural.
- It is highly extensible, allowing for the integration of custom cost functions to meet various requirements.

2 Related Work

2.1 Zero-bit Watermarking

Zero-bit watermarking verifies whether a given text was generated by a specific model, signaling presence or absence of generation without embedding explicit messages. These techniques can be broadly classified into three categories. First, *distribution-based methods* inject statistical biases by modifying the original token probability distribution during generation, such as through vocabulary partitioning and selective token promotion [22, 29, 49]. Second, *sampling-based methods* maintain the original distribution but impose constraints during the sampling stage, employing techniques like restricted sampling or rejection sampling to guide the sampled tokens [9, 20, 25]. Third, *learning-based methods* avoid direct interference with generation, instead utilizing separately trained post-hoc detectors to identify machine-generated text through learned statistical patterns [1, 14, 43].

2.2 Multi-bit Watermarking

Multi-bit watermarking extends beyond provenance detection by embedding identifiable messages (typically bit strings)

into generated text, enabling applications like user attribution and content tracing. Current approaches fall into two categories. The first category is *message enumeration-based methods* [12, 40]. These methods use the entire message as an indivisible unit to generate the watermark signal, but requiring exhaustive search over all 2^b possible messages to find the best match during extraction (b is the number of message bits). While accurate, the exponential complexity limits practical deployment. To overcome this efficiency bottleneck, a second category, known as *bit assignment-based methods*, has emerged. These methods split the message into smaller segments and pseudo-randomly assign them to individual tokens, enabling highly efficient extraction. However, this strategy may suffer from reduced accuracy, as the imbalanced frequency of natural language can lead to an uneven allocation of tokens for different bit strings. The MPAC method proposed in [46] exemplifies this category. Subsequent efforts have focused on mitigating this imbalance, such as the balanced segment assignment algorithm [33], and the logit-derived dynamic segmentation in DERMARK [27].

Existing LLM watermarking methods struggle to achieve optimal balance among watermark capacity, robustness, and text quality. As a contrast, our proposed DMW framework addresses these three aspects simultaneously in a unified fashion, and offers flexible prioritization of desired objectives according to application requirements.

2.3 Distortion-minimization Steganography

Distortion-minimization steganography (DMS), also known as content-adaptive steganography, embeds secret data into host media (*e.g.*, images) while minimizing detectable distortions. The core idea is to selectively modify complex or noisy elements of the host media, where changes are perceptually and statistically less noticeable. This process defines a cost function to quantify modification impact, then employs a coding algorithm to minimize the overall cost during embedding.

The cost function is directly linked to steganographic security and is typically computed using heuristic or learned rules. Taking images as an example, heuristic rules may consider factors such as texture complexity [8,16,26], distribution distance from image models [36], and image side-information [11]. Conversely, learned rules employ machine learning to automatically derive distortion costs from data, using techniques include generative adversarial networks [44], reinforcement learning [31, 37], and adversarial examples [5, 38].

The coding algorithm is fundamental to DMS, with syndrome-trellis codes (STCs) [13] as the most representative example. Developed from convolutional codes, STC models embedding as a search for an optimal path within a trellis graph, where nodes represent possible carrier states and edges denote potential paths weighted by specific distortion costs. By identifying the minimum-cost path, STC embeds secret data while meeting data length constraints and minimizing overall distortion.

However, DMS relies on pre-existing media content to compute distortion costs, and STC requires all content and costs to be predetermined for embedding. This makes them unsuitable for watermarking in LLMs, where tokens are generated sequentially and the final output length is unknown in advance. In contrast, our DMW framework dynamically calculates distortion costs during text generation. Moreover, we develop periodically optimized syndrome-trellis codes (PO-STCs), enabling real-time streaming embedding for texts of varying lengths.

3 Problem Formulation

Given an LLM M, a secret key K, a prompt token sequence $\mathbf{P} = (P_0, P_1, \cdots, P_{p-1})$ of length p, and a binary watermark message $\mathbf{m} \in \{0,1\}^m$, we denote the generated watermarked token sequence at time step t as $\mathbf{S}_{0:t} = (S_0, S_1, \cdots, S_t)$, where S_t is the t-th generated token. The embedding and extraction processes of existing watermarking can be formulated as

$$S_t = \mathrm{Emb}(M, K, \mathbf{P}, \mathbf{S}_{0:t-1}, \mathbf{m}),\ t = 0, 1, \cdots, T, \tag{1}$$

and

$$\mathbf{m} = \mathrm{Ext}(K, \mathbf{S}_{0:T}), \tag{2}$$

where $\mathrm{Emb}(\cdot)$ and $\mathrm{Ext}(\cdot)$ represent the embedding and extraction functions, respectively, and T is the total number of generated tokens in the text.

In Eq. (1), watermarked tokens are generated sequentially one by one with only local context, preventing global optimization of token selections and modifications over the entire token sequence. A common alternative is to perform post-hoc modifications through overall consideration after the entire sequence is generated [45]. However, this approach cannot utilize all tokens for watermark embedding (since not all generated tokens can be modified), and thus struggles to maintain text quality comparable to in-generation watermarking.

To address these limitations, we propose a segment-wise embedding strategy, which balances real-time generation and global optimization. Specifically, we divide the text generation process into multiple periodic stages, with each stage containing a token segment of length s. For each token segment, we comprehensively consider the characteristics of all s tokens to select appropriate tokens for modification to embed the watermark. Therefore, Eq. (1) can be rewritten as

$$\mathbf{S}_{si:si+s-1} = \mathrm{Emb}(M, K, \mathbf{P}, \mathbf{S}_{0:si-1}, \mathbf{m}),\ i = 0, 1, \cdots, N, \tag{3}$$

where i and N denote the segment index and the total number of segments, respectively. For notational simplicity, we use $\mathbf{S}_i$ to replace $\mathbf{S}_{si:si+s-1}$ in the following text. Previous methods correspond to the special cases of $i = t, s = 1$ (online, local) and $i = 0, s = T$ (offline, global), while our method represents a compromise between these two extremes.

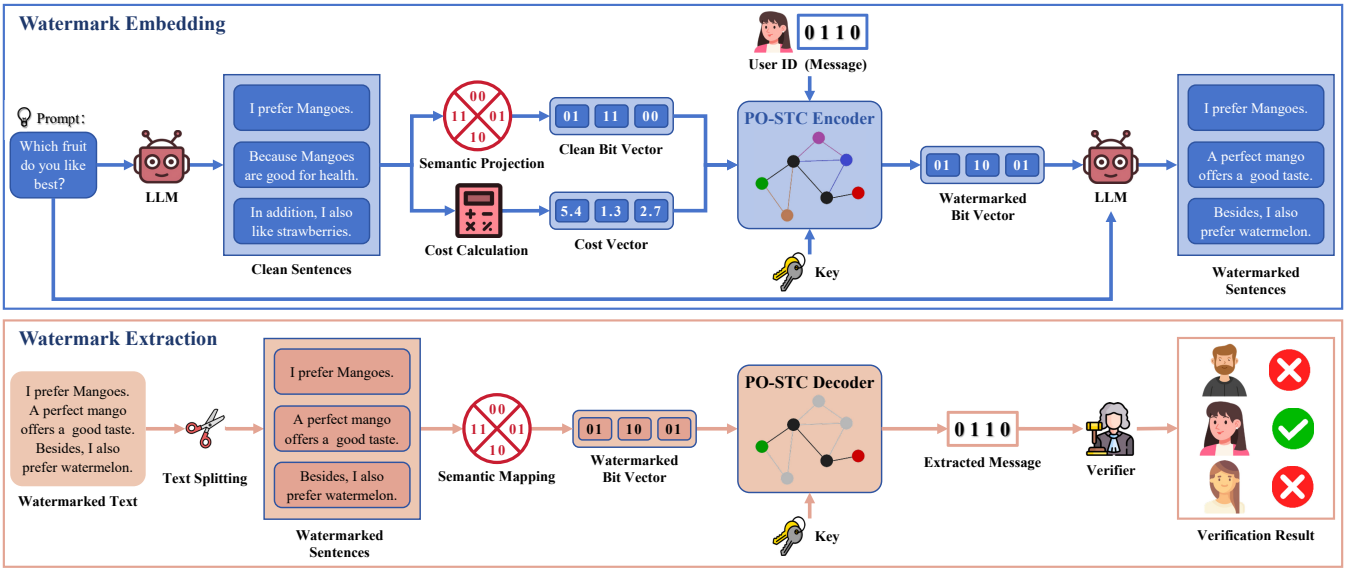


Figure 2: The workflow of distortion-minimization watermarking (DMW) framework. The cost vector in the figure is actually $[5.4, 5.4, 1.3, 1.3, 2.7, 2.7]$, since we omit duplicate cost values to save space. For conciseness, the text-to-sentence and sentence-to-text processes are not explicitly shown in the watermark embedding phase.

To accurately distinguish the token modifications within $\mathbf{S}_i$, we further denote the original (clean) and watermarked versions of the i-th token segment as $\mathbf{X}_i = (X_{i,0}, X_{i,1}, \cdots, X_{i,s-1})$ and $\mathbf{Y}_i = (Y_{i,0}, Y_{i,1}, \cdots, Y_{i,s-1})$, respectively. For each $j \in \{0, 1, \cdots, s-1\}$, if $X_{i,j} \neq Y_{i,j}$, the token S_{si+j} is modified; otherwise, it remains unchanged. Following the distortion-minimization principle, the distortion for the entire text is defined as

$$D(\mathbf{X}, \mathbf{Y}) = \sum_{i=0}^{N-1} D(\mathbf{X}_i, \mathbf{Y}_i) = \sum_{i=0}^{N-1} \sum_{j=0}^{s-1} \rho_{i,j} \cdot [X_{i,j} \neq Y_{i,j}], \quad (4)$$

where $\rho_{i,j} > 0$ represents the cost of modifying $X_{i,j}$, and $[\cdot]$ denotes the Iverson bracket defined to be 1 if the enclosed logical statement is true and 0 otherwise.

Since watermark messages are binary, we embed and extract them using binary coding schemes. Let $\mathcal{P} : \mathbf{X}_i \mapsto \{0,1\}^v$ be a projection function that maps a token segment to an v-bit sequence[1]. We employ a binary linear code $\mathcal{C}$ to implement the specific embedding and extraction functions as

$$\mathrm{Emb}(\mathbf{X}_i, \mathbf{m}_i) = \arg \min_{\mathcal{P}(\mathbf{Y}_i) \in \mathcal{C}(\mathbf{m}_i)} D(\mathbf{X}_i, \mathbf{Y}_i), i = 0, 1, \cdots, N-1, \quad (5)$$

$$\mathrm{Ext}(\mathbf{Y}) = \mathbb{H}\mathcal{P}(\mathbf{Y}) \quad (6)$$

where $\mathbf{m}_i$ denotes the u-bit sequence segment in message $\mathbf{m}$, $\mathbb{H} \in \{0,1\}^{m \times n}$ is the parity-check matrix of $\mathcal{C}$, $\mathbb{H}_i \in \{0,1\}^{u \times v}$ is a segment-based sub-matrix in $\mathbb{H}$ (see Figure 5), $\mathcal{C}(\mathbf{m}_i) = \{\mathbf{Z}_i \in \{0,1\}^v \mid \mathbb{H}_i \mathbf{Z}_i = \mathbf{m}_i\}$ is the coset of $\mathbf{m}_i$, and all operations in the equations are binary operations. Note that the embedding in Eq. (5) operates in a segment-wise manner, while the extraction in Eq. (6) is performed in a single, one-off step. In addition, we omit the symbols $M, K, \mathbf{P}$ in $\mathrm{Emb}(\cdot,\cdot)$ and $\mathrm{Ext}(\cdot)$ for notational simplicity.

4 Methodology

4.1 Overview

Our proposed distortion-minimization watermarking (DMW) framework comprises two phases: watermark embedding and watermark extraction, as illustrated in Figure 2. This framework enables high-capacity watermark payload while preserving text quality and ensuring robust extraction.

During the watermark embedding phase, the user first inputs a prompt into a large language model (LLM) to generate text, which is then split into sentences. This text splitting enables subsequent sentence-level watermark embedding, thereby benefiting watermark robustness. To embed watermarks using binary linear codes, each sentence undergoes a semantic projection to yield a fixed-length bit string. These individual bit strings from multiple sentences are concatenated to form a unwatermarked clean bit vector. The details of semantic projection are presented in Section 4.2.

In parallel with the semantic projection, a distortion cost is computed for each sentence, quantifying its suitability for modification. Two categories of cost functions are designed for cost calculation: a robustness cost function and two quality cost functions. The bits projected from the same sentence

[1] For a given token segment, the bit segment length v may be greater or less than the token segment length s, depending on the design of $\mathcal{P}$. See Section 4.2 for more details of $\mathcal{P}$.

shares the same distortion cost, and similarly, the distortion costs of all sentences are concatenated to form a cost vector, which is the same length as the bit vector. The definitions and descriptions of cost functions are provided in Section 4.3.

After preparing all the watermarking inputs, the parity-check matrix of periodically optimized syndrome-trellis code (PO-STC) is constructed using a shared secret key, and then the clean bit vector, cost vector, and binary watermark message are fed into the PO-STC Encoder to produce a watermarked bit vector. The construction and optimization of PO-STC are elaborated in Section 4.4.

Although the watermarked bit vector contains the watermark message, it must be projected back into natural language sentences. To achieve this, the watermarked bit vector is used to guide the LLM in generating a set of candidate sentences, from which the one that projects back to the exact watermarked bit vector is selected as the watermarked sentence. Finally, all such watermarked sentences are concatenated to form the complete watermarked text.

During the watermark extraction phase, the total watermarked text is similarly split into sentences, with each sentence semantically projected to a watermarked bit string. Then all the watermarked bit strings for the text are concatenated into a full watermarked bit vector prepared for watermark extraction. The parity-check matrix of the PO-STC is reconstructed using the same shared secret key, and the watermarked bit vector is fed into the PO-STC decoder to extract the binary watermark message. This PO-STC decoder is actually the implementation of Eq. (6). Finally, watermark verification is performed by comparing the extracted watermark against originally registered information.

4.2 Semantic Projection

This subsection concretizes the projection function $\mathcal{P}$ mentioned in Section 3. To embed binary watermarks into natural language sentences through coding schemes, sentences should first be converted into binary format. Additionally, attacks on LLM watermarks, such as word insertion, substitution, and paraphrasing, do not alter the original meaning of sentences. To counter such attacks, semantically similar sentences should be converted to identical bit strings. We address both bit conversion and semantic consistency through a technique called semantic projection.

The semantic projection approach partitions the semantic space of sentences into k non-overlapping regions such that all semantically similar sentences fall into the same region. Each region is then assigned a unique bit string, ensuring that all semantically similar sentences are represented by the same binary sequence.

Specifically, we first utilize an off-the-shelf sentence encoder to extract semantic feature vectors from sentences[2], implementing $f : \mathbf{X}_i \mapsto \mathbb{R}^d$ for the i-th clean sentence, where f and d are the sentence encoder and feature dimension, respectively. This sentence encoder follows a SBERT architecture [35] and is fine-tuned using contrastive learning, as did in [41] (see Appendix E for details). Next, we apply k-means clustering to partition the feature space such that sentences with similar semantics are grouped into the same cluster while dissimilar ones are separated into distinct clusters. This results in a set of cluster centroids $\{\mu_0, \mu_1, \cdots, \mu_{k-1}\}$. Finally, we assign each cluster a unique bit string of length $\lceil \log_2 k \rceil$, denoted by $\mathrm{bin}(i)$, where $\mathrm{bin}(\cdot)$ is a binarization function that converts the cluster index $i \in \{0, 1, \cdots, k-1\}$ to its binary representation. For example, when $k = 16$, all feature vectors (sentences) in cluster 6 are mapped to the 4-bit "0110".

The critical step in semantic projection is determining the correct cluster assignment for each generated sentence. Given the i-th sentence $\mathbf{X}_i$, its cluster index c is determined by finding the nearest cluster centroid with minimum distance:

$$c = \arg \min_{i \in \{0,1,\cdots,k-1\}} \mathrm{dist}\left(f(\mathbf{X}_i), \mu_i\right), \tag{7}$$

where $\mathrm{dist}(\cdot,\cdot)$ denotes the cosine distance between two feature vectors.

To further enhance semantic robustness by preventing sentences from shifting between clusters due to minor perturbations, we select only sentences that are close to cluster centroids (semantically-invariant) while distant from cluster boundaries (semantically-variant). Therefore, we use a margin to constrain sentences' distance from cluster centroids:

$$\min_{j \neq c} \mathrm{dist}\left(f(\mathbf{X}_i), \mu_j\right) - \mathrm{dist}\left(f(\mathbf{X}_i), \mu_c\right) > \gamma, \tag{8}$$

where γ is a predefined margin threshold. Only sentences satisfying both Eq. (7) and Eq. (8) are used for watermarking.

In summary, the semantic projection $\mathcal{P}(\mathbf{X}_i)$ is a process that involves: first partitioning the semantic space into clusters, then determining the cluster index c for $\mathbf{X}_i$ using Eq. (7) and (8), and finally applying a binarization function $\mathrm{bin}(c)$ to obtain the corresponding bit string.

Our semantic projection draws inspiration from the approach in [18], but differs in two key aspects:

- **Principle:** [18] randomly divides sentence semantic clusters into two categories—valid and blocked—to indicate the presence or absence of watermarks, functionally equivalent to the green and red lists in [22]. Rather than categorizing sentence semantic clusters, our method adds a bit string assignment step after clustering, establishing a direct mapping from semantics to bits and thus enabling richer semantic encoding.

- **Application:** The approach in [18] is tailored for zero-bit watermarking, whereas our approach is specifically designed for multi-bit watermarking, supporting more complex applications.

[2] With a slight abuse of notation, the symbol $\mathbf{X}_i$, originally denoting a token sequence, is used here to represent a sentence.

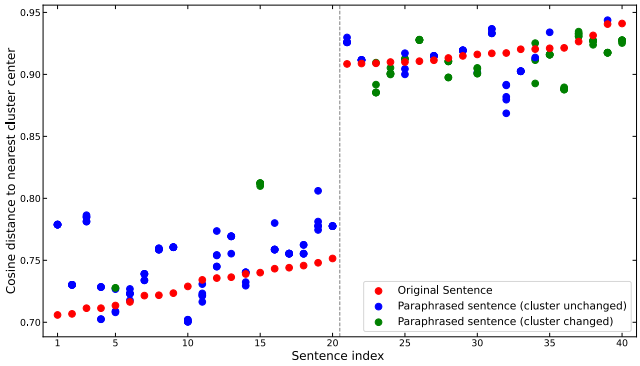


Figure 3: The locality property of sentences under semantic-preserving attacks.

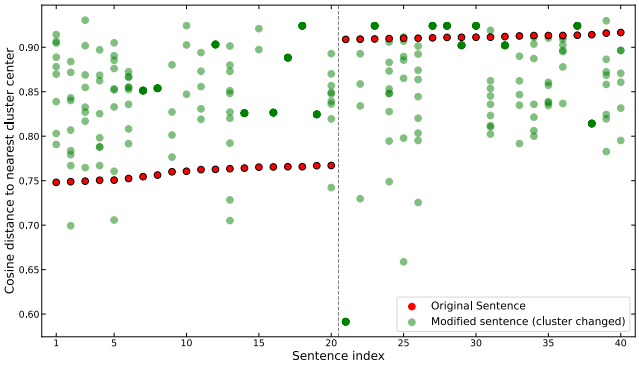


Figure 4: The distributional randomness of semantically modified sentences.

4.3 Distortion Cost

The distortion cost evaluates the suitability of modifying a sentence for watermark embedding. We propose two categories of sentence-level cost functions: robustness cost and quality cost, with the latter further encompassing fluency cost and diversity cost. Subsequently, we define the overall cost function as a combination of these components.

4.3.1 Robustness Cost

The robustness of watermarked text is reflected in the ability to resist semantic-preserving attacks. This indicates that as long as the text semantics remain unchanged, the tighter the binding between watermark and text semantics, the higher the probability of correct watermark extraction. In other words, watermark stability is determined by semantic stability. Thus, the key to designing a robustness cost function lies in quantifying the semantic stability of a sentence.

As presented in the Section 4.2, the semantic space of sentences is partitioned into clusters, where sentences near cluster boundaries tend to be semantically ambiguous and are filtered out by a margin. Building upon this, we further analyze the refined sentences based on their distance to cluster centroids. Since the bit string assigned to a sentence via semantic projection depends on its cluster index (also its cluster centroid), sentences closer to the centroid exhibit greater semantic stability. This motivates us to use the distance between a sentence and its cluster centroid as a measure of robustness: the closer the sentence is to the centroid, the more resistant it is to semantic-preserving attacks, and thus the more suitable it is for watermark modification. Accordingly, the robustness cost is defined as proportional to this distance:

$$\rho_{rob}(\mathbf{X}_i) = \mathrm{dist}\left(f(\mathbf{X}_i), \mu_c\right), \tag{9}$$

where $\mathrm{dist}\left(f(\mathbf{X}_i), \mu_c\right)$ denotes the cosine distance between sentence $\mathbf{X}_i$ and its cluster centroid μ_c (see Eq. (7)). Eq. (9) indicates that sentences closer to their centroids incur lower costs, while those farther away incur higher costs.

We conduct two experiments to validate the rationality of our robustness cost function. The first experiment verifies the locality preservation of semantic-preserving attacks. We randomly select one cluster and further sample 20 sentences located near its centroid and another 20 sentences near its boundary. These sentences are ordered ascendingly by their distances to the centroid, and each sentence is paraphrased three times. We then compare their distances to the centroid before and after paraphrasing, as illustrated in Figure 3. We observe that paraphrased sentences (blue and green dots) consistently remain close to their original sentences (red dots) in semantic space, regardless of the original distance to the centroid. However, a clear distinction emerges: sentences initially close to the centroid largely remain within the same cluster after paraphrasing (see blue dots in the left part), whereas nearly half of the boundary sentences shift into other clusters (see green dots in the right part). This confirms that centroid-proximal sentences are more semantically stable and should be prioritized for modification during watermark embedding.

The robustness cost may raise a concern: whether extensively modifying sentences near cluster centroids would bias the distribution of watermarked sentences in semantic space. Our second experiment investigates the distributional randomness of semantically modified sentences. The setup is similar to the first experiment, but the sentences are modified such that their cluster assignments change. Each sentence undergoes 10 modifications to better observe distributional patterns, as shown in Figure 4. The results reveal that regardless of the original sentence's position, the semantically modified sentences are randomly distributed within their new clusters, spanning the entire semantic space. This indicates that the distribution of watermarked sentences remains unaffected by their initial locations, ensuring that the robustness cost does not introduce predictable bias in semantic space.

4.3.2 Quality Cost

Modifying tokens or sentences to embed watermarks inevitably alters the original style of the generated text, degrad-

ing its quality primarily in fluency and diversity. To mitigate this, our DMW prioritizes modifications to sentences that have minimal impact on overall text quality.

Fluency Cost. The fluency of generated text refers to its grammatical correctness, coherence, and naturalness. Existing methods assess text fluency at the token level, such as preserving token distributions [7, 19, 25] and avoiding modifications to high-entropy tokens [22, 29, 40]. However, they ignore the hierarchical structure and overall coherence of language, failing to capture semantic logic and discourse relationships of higher-level linguistic units like sentences and paragraphs, resulting in locally reasonable but globally unnatural text.

Instead, we design a sentence-level fluency cost based on contextual dependency. Specifically, we employ a separate LLM to evaluate the fluency between the current sentence and its preceding sentences. If the current sentence exhibits weak cohesion with the preceding context, it is considered originally detrimental to overall text fluency, and modifying such a sentence causes relatively minor degradation to text quality (actually, its watermarked counterpart may even improve text quality). We quantify this impact on text fluency by using the perplexity of the combined context and current sentence. Higher perplexity indicates lower fluency, warranting a smaller cost to encourage modification of such sentences. To stabilize the fluency cost as the generated text grows, we adopt a fixed-size sliding window for perplexity calculation. Therefore, the fluency cost of the i-th sentence $(\mathbf{X}_i)$ is defined as inversely proportional to its window based perplexity:

$$\rho_{flu}(\mathbf{X}_i) = \frac{1}{\mathrm{PPL}\left(M_\rho, S_{i-r+1:i}\right) + \varepsilon}, \tag{10}$$

where $\mathrm{PPL}\left(M_\rho, S_{i-r+1:i}\right)$ denotes the perplexity of a window of r sentences $S_{i-r+1:i}$ via an LLM M_ρ, and ε is a small constant to prevent division by zero. We use the compact OPT-2.7B model as M_ρ to efficiently compute perplexity.

Diversity Cost. Text diversity refers to the variability in lexical choice, syntactic structure, and conceptual expression, making generated content more engaging and human-like. Despite its importance, only a few studies [10, 17, 18] have empirically evaluated the diversity of their watermarked outputs, and none have explicitly addressed how to technically ensure the text diversity in LLM watermarking.

Text diversity is directly influenced by generative flexibility, which reflects how many semantically valid alternatives an LLM can generate for a given sentence (*i.e.*, the "space" of acceptable semantic variations). Sentences with higher generative flexibility naturally support greater potential diversity, since the LLM has more freedom to produce different valid realizations.

Embedding watermarks involves modifying text. If this process is applied to sentences with low generative flexibility (where semantic meaning is tightly constrained), the modification can restrict expression options, leading to more repetitive or forced patterns and thus reducing overall text diversity. Conversely, embedding in sentences with high generative flexibility allows the LLM to "absorb" modifications without narrowing the expressive space, thereby preserving or even enhancing diversity. Therefore, following the principle of distortion minimization, we prioritize modifying high-flexibility sentences (assigned low costs) while avoiding modifications in low-flexibility sentences (assigned high costs).

To quantify this, we introduce an entropy-based diversity cost, which encourages modifications in sentences with higher flexibility and penalizes rigid sentence patterns. Specifically, for the i-th sentence $\mathbf{X}_i$ to be generated during watermark embedding, we generate its L candidate sentences given its context [3], and then compute the entropy $H(\mathbf{X}_i)$ over their semantic bit strings obtained from semantic projection. A higher entropy indicates greater semantic flexibility, whereas a lower entropy indicates limited diversity. To map this measure into a distortion cost, we define the diversity cost as

$$\rho_{div}(\mathbf{X}_i) = \begin{cases} 1 - \frac{H(\mathbf{X}_i)}{\log_2(L)} & \text{if } H(\mathbf{X}_i) > 0 \\ \infty & \text{if } H(\mathbf{X}_i) = 0 \end{cases} \tag{11}$$

where $\log_2(L)$ is a normalization constant ensuring that $\frac{H(\mathbf{X}_i)}{\log_2(L)} \in [0,1]$. In Eq. (11), when $H(\mathbf{X}_i) = 0$, we set the diversity cost $\rho_{div}(\mathbf{X}_i)$ to infinity rather than 1 to enforce the avoidance of modifying rigid sentence structures. Since there are k categories of sentences, we set $L = k$ (see the meaning of k in Section 4.2).

4.3.3 Overall Distortion Cost

Finally, taking the above costs into account, the overall distortion cost for the i-th sentence $\mathbf{X}_i$ is formulated as the weighted sum of three costs:

$$\rho_i = \lambda_{rob} \cdot \rho_{rob}(\mathbf{X}_i) + \lambda_{flu} \cdot \rho_{flu}(\mathbf{X}_i) + \lambda_{div} \cdot \rho_{div}(\mathbf{X}_i), \tag{12}$$

where $\lambda_{rob}, \lambda_{flu}$, and λ_{div} are the weights of the corresponding cost terms, respectively. By adjusting these weights, our DMW framework can be tailored to prioritize different performance aspects, making it adaptable to a wide range of application scenarios. The weight settings and cost effects are discussed in Section 5.1 and 5.3.

4.4 Periodically Optimized Syndrome-trellis Codes

Our proposed PO-STC is specifically designed for watermarking in streaming text data. To aid understanding, we first describe its core principles in a non-streaming setting (where

[3] L is a common hyper-parameter in LLM implementations; for example, L corresponds to the variable *num_return_sequences* in the LLaMA model.

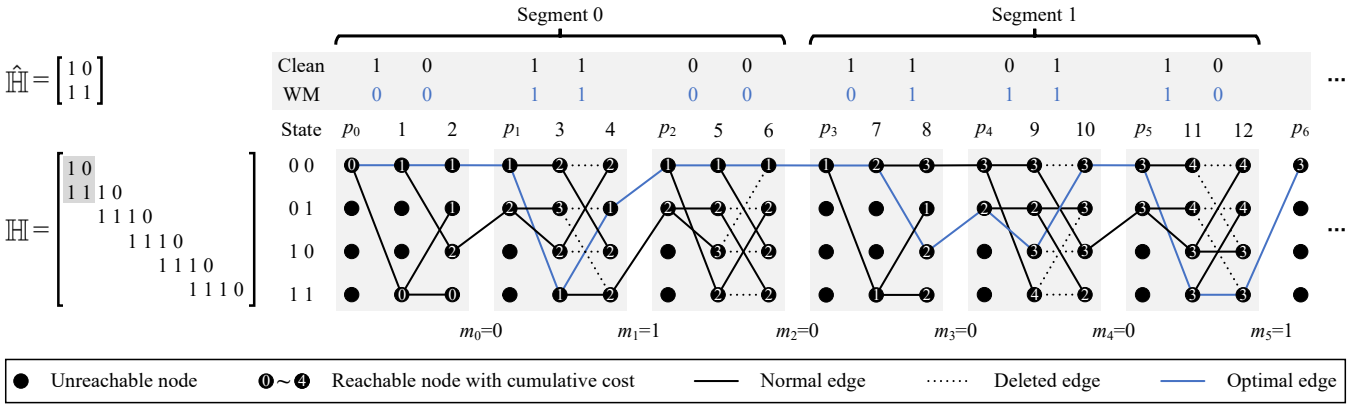


Figure 5: Example of periodically optimized syndrome-trellis code (PO-STC). The "Clean" and "WM" mean clean bit vector and watermarked bit vector, respectively.

data is pre-available), and then elaborate on the modifications required for streaming scenarios.

The PO-STC is closely related to coding theory. In classical coding, a message $\mathbf{m}$ is fed into an encoder g_{code} to produce a codeword $\mathbf{y} = g_{code}(\mathbf{m})$, and the decoder detects or corrects errors by computing the syndrome $\mathbf{s} = \mathbb{H}\mathbf{y}^T$. In contrast, our watermarking framework follows a similar but reversed logic: the message $\mathbf{m}$ is embedded into host data $\mathbf{x}$ through an encoder g_{wm}, yielding watermarked data $\mathbf{y} = g_{wm}(\mathbf{x}, \mathbf{m})$, and the syndrome $\mathbf{m} = \mathbb{H}\mathbf{y}^T$ represents the extracted message. Here, the $\mathbf{x}$ represents the clean bit vector obtained by $\mathcal{P}(\mathbf{X})$, where $\mathbf{X}$ denotes the clean token sequence mentioned in Section 3.

Both the encoding (embedding) and decoding (extraction) processes of PO-STC rely on a parity-check matrix $\mathbb{H}$, which has dimensions $m \times n$ and comprises m sub-matrices $\hat{\mathbb{H}}$ of size $h \times w$. These sub-matrices $\hat{\mathbb{H}}$ are arranged side-by-side horizontally, with each subsequent block shifted down by one row, forming a sparse and banded structure. The height h of $\hat{\mathbb{H}}$ (typically around 10) controls the trade-off between the efficiency and performance of encoding. The width w of $\hat{\mathbb{H}}$ is determined by the relative payload $\alpha = m/n$ of the watermark, and is chosen as either $w = \lfloor n/m \rfloor$ or $w = \lceil n/m \rceil$, where $\lfloor \cdot \rfloor$ and $\lceil \cdot \rceil$ denote floor and ceiling operations respectively. For a given width n of $\mathbb{H}$, the number of sub-matrices with width $\lfloor \frac{n}{m} \rfloor$ (denoted as w_1) and those with width $\lceil \frac{n}{m} \rceil$ (denoted as w_2) must satisfy $w_1 \lfloor \frac{n}{m} \rfloor + w_2 \lceil \frac{n}{m} \rceil = n$, with arbitrary arrangement order. For ease of discussion and without loss of generality, we assume n is divisible by m, so that $w = n/m$. An example of $\mathbb{H}$ and $\hat{\mathbb{H}}$ is illustrated in the left part of Figure 5.

The core of our watermarking framework is to encode an optimal watermarked bit sequence $\mathbf{y}$ that satisfies the syndrome $\mathbf{m} = \mathbb{H}\mathbf{y}^T$, which can be viewed as finding an optimal path in a syndrome trellis using the Viterbi algorithm. The syndrome trellis is a graph with m blocks, where each block contains a grid of nodes arranged in 2^h rows and $w+1$ columns. The first column of each block serves as a transitional column, indexed as $p_1, p_2, \cdots, p_m$, while the subsequent w columns correspond to one $\hat{\mathbb{H}}$ in $\mathbb{H}$, indexed as $1, 2, \cdots, n$. Each node in the trellis represents a state, expressed in binary form, ranging from 0 to $2^h - 1$ from top to bottom in each column.

Nodes in adjacent columns are connected by edges that form the trellis paths. For any two adjacent columns $l-1$ and l, each node in column $l-1$ extends two distinct edges to two nodes in column l. One is a diagonal edge, which connects to a node in a different row and corresponds to adding the l-th column of $\mathbb{H}$ to the state of the node in column $l-1$ for state transition[4]. The other is a horizontal edge, which connects to a node in the same row and represents maintaining the state without adding the l-th column of $\mathbb{H}$. These two edge types simultaneously encode the bit assignment of y_l: a diagonal edge indicates $y_l = 1$, whereas a horizontal edge indicates $y_l = 0$. Since each clean element x_l has a corresponding distortion cost ρ_l, each edge associated with x_l can also be assigned a cost based on whether x_l equals y_l, namely $\rho_l \cdot [x_l \neq y_l]$. For instance, in Figure 5, the first node (state 00) in column p_0 connects to two nodes in column 1: states 11 via a diagonal edge and state 00 via a horizontal edge. The diagonal edge represents $y_1 = 1$ with a cost of 0 (since $x_1 = 1$), while the horizontal edge represents $y_1 = 0$ with a cost of 1.

Except for column p_0, each node in other columns receives two incoming edges from nodes in the preceding column. At each node, two candidate cumulative costs are computed by summing the cumulative costs of its two preceding nodes with the respective edge costs. The smaller of the two is retained as the node's cumulative cost, and the corresponding edge is preserved, while the other edge is deleted [5]. As an example, in Figure 5, the last node (state 11) in column 4 receives two edges from two nodes (states 01 and 11) in column 3 with cumulative costs of 3 and 1, and the costs of the incoming

[4]Addition is performed in binary, with the most significant bit at the bottom of the column.

[5]Some nodes that cannot be reached are assigned infinite cumulative costs, designated as unreachable nodes in Figure 5, where their connecting edges are omitted for clarity.

edges are 0 (diagonal) and 1 (horizontal). Thus, the two new candidate cumulative costs are $3+0=3$ and $1+1=2$, where the cumulative cost of 2 is chosen, retaining the horizontal edge and discarding the diagonal one.

In the syndrome trellis construction, each block corresponds to a watermark bit. When the path extends from the last column of one block to the transitional column of the next, only nodes consistent with the associated watermark bit are preserved and connected sequentially to the first half of nodes in the subsequent transitional column. Specifically, even-state nodes correspond to watermark bit 0, while odd-state nodes correspond to watermark bit 1. For example, in Figure 5, the first block requires embedding watermark bit 0, so only the even-state nodes in the last column (column 2) are connected to the first half of nodes in column p_1, while all edges from odd-state nodes in column 2 are discarded. This expansion-pruning process continues until all blocks are processed, yielding a path with the minimum accumulated cost that satisfies Eq. (5). The forward pass of the Viterbi algorithm is thus completed, while the backward pass traces from the terminal node with the lowest cumulative cost back to the beginning node, recording the edge types (*i.e.*, y_l) along the optimal path. In this way, the optimal watermarked bit sequence $\mathbf{y}$ is determined, guaranteeing the watermarking constraint $\mathbf{m} = \mathbb{H}\mathbf{y}^T$.

However, the procedure described above cannot be directly applied to streaming data, where x_l and its cost ρ_l are generated sequentially in real-time. To address this, we propose a phased and periodic optimization strategy that adapts the Viterbi algorithm to streaming scenarios. The watermark embedding with PO-STC then proceeds in the following steps (also see Algorithm 1 in the appendix):

1) **Trellis configuration.** Pre-estimate the expected bit length n from text generation (which can be constrained by prompts), and combine it with the target watermark length m to determine the width w of the sub-matrix $\hat{\mathbb{H}}$.

2) **Trellis segmentation.** Partition the syndrome trellis into multiple segments on a block-wise basis. Since the trellis structure depends only on w and predefined h, it can be pre-constructed independently of the actual text content.

3) **Buffered synchronization.** Employ a buffering mechanism to accumulate generated bits obtained from semantic projection until their number reaches the segment length. At the same time, compute the distortion costs of the corresponding texts, thereby producing a synchronized bit sequence segment and cost segment aligned with the trellis structure.

4) **Sub-path optimization.** Within each segment, perform the forward pass of the Viterbi algorithm and identify the optimal sub-path at the end of the segment (see columns 6 and 12 in Figure 5 for examples).

5) **Text reconstruction and output.** Map this optimal sub-path back into the corresponding watermarked text segment (see Section 4.2) and output the result for immediate use.

6) **Path transition.** Extend the optimal sub-path obtained from one segment into the transitional column of the first block in the next segment, and repeat the procedure until all segments are processed.

In this procedure, if watermark embedding is completed before text generation ends, the LLM continues generating text without further watermark embedding (unless repeated watermark embedding is explicitly required). Regarding latency, the buffering mechanism in step 3) introduces only a one-segment delay between the original text and the watermarked output. Since the segment length can be set to a small value, the PO-STC achieves near real-time performance.

The final optimal path $\mathbf{y}$ is obtained by concatenating the optimal sub-paths from all segments. One might argue that $\mathbf{y}$, being segment-wise optimal, may not achieve true global optimality as in standard STC [13]. Nevertheless, our experimental results in Appendix G show that the performance loss is negligible and thus acceptable for practical applications.

In addition to coding performance, computational complexity is another critical consideration. While standard STC exhibits both time and space complexity of $O(2^h n)$, our PO-STC maintains the same time complexity but substantially reduces space complexity. Standard STC requires storing all 2^h states across every column of the trellis until the final path is determined. In contrast, PO-STC only needs to store the states within a single segment, thus reducing the space complexity to $O(2^h v)$, where v is the bit segment length. This makes PO-STC far more memory-efficient.

In summary, while PO-STC introduces a slight loss in coding performance compared with standard STC, the improvement in real-time applicability and space efficiency make it substantially more suitable for watermarking LLM outputs.

5 Experiments

In this section, we perform extensive experiments to evaluate the performance of our DMW framework against state-of-the-art watermarking methods.

5.1 Experimental Setup

LLMs. We use two public LLMs, LLaMA-2-7B [39] and Falcon-7B [3], to generate watermarked texts. Unless otherwise explicitly stated, LLaMA-2-7B is adopted as the default LLM for experiments.

Dataset. Following previous works, we utilize two famous datasets, the Colossal Common Crawl Cleaned corpus (C4) [34] and OpenGen [24], to evaluate watermarking performance. C4 is a large-scale English dataset with more than

Table 1: Comparison of watermark correctness using LLaMA across various payloads and datasets.

Dataset	Method	8 bit		16 bit		24 bit		32 bit	
		Match Rate ↑	Bit Acc ↑	Match Rate ↑	Bit Acc ↑	Match Rate ↑	Bit Acc ↑	Match Rate ↑	Bit Acc ↑
C4	Yoo *et al.* [45]	98.90	99.67	96.00	97.90	92.30	96.77	90.80	95.82
	Wang *et al.* [40]	80.00	94.25	74.00	86.00	70.00	85.50	66.00	86.04
	Qu *et al.* wo ECC [33]	93.00	96.75	85.50	94.70	74.00	89.48	65.00	83.29
	Qu *et al.* [33]	94.00	97.75	87.00	97.53	82.00	94.83	72.00	88.81
	Our DMW	**99.50**	**99.93**	**98.08**	**99.87**	**94.15**	**98.37**	**91.30**	**97.75**
OpenGen	Yoo *et al.* [45]	98.45	99.37	94.40	97.90	90.90	92.33	88.60	95.20
	Wang *et al.* [40]	88.00	92.75	66.00	85.50	54.00	72.00	45.00	68.73
	Qu *et al.* wo ECC [33]	95.50	98.12	86.00	94.32	77.00	90.36	70.00	88.28
	Qu *et al.* [33]	96.00	98.40	84.00	93.83	76.00	90.50	70.00	84.50
	Our DMW	**98.82**	**99.71**	**98.46**	**99.50**	**94.24**	**98.61**	**92.53**	**97.24**

156 billion tokens, filtered to remove low-quality and non-linguistic content. OpenGen was created by randomly drawing 3,000 text segments from the WikiText-103 [30] validation set, with each segment consisting of two sentences.

Baselines. To demonstrate the superiority of our DMW, we compare it with three representative multi-bit watermarking methods: the post-generation method by Yoo *et al.* [45], the message enumeration-based method by Wang *et al.* [40], and the bit assignment-based method by Qu *et al.* [33]. Since the watermarking method [33] employs error-correction code (ECC) to ensure robustness, we also conduct experiments using its non-ECC version for a fair comparison.

Metrics. We use diverse metrics for a comprehensive evaluation. Watermark correctness and robustness are measured by both match rate (the proportion of successfully extracted full watermark messages) and bit accuracy (the proportion of correctly identified bits within all watermark messages). Watermarked text quality is assessed via perplexity (fluency), Sem-Ent (diversity) [15], and BERTScore (fidelity) [48].

Implementation Details. For our DMW, we embed watermark messages of varying lengths (8, 16, 24, and 32 bits) with a relative payload $\alpha = 0.5$, unless otherwise specified. PO-STC is configured with a sub-matrix height $h = 6$ and a segment length $v = 8$. The semantic projection uses $k = 16$ clusters, and the fluency window size is $r = 8$. The weights of three distortion costs are set to $\rho_{rob} = 2.5$, $\rho_{flu} = 1.5$ and $\rho_{div} = 1.0$, respectively. LLM hyper-parameters include nucleus sampling with temperature 0.9, top-p 0.8, and repetition penalty 1.2. All experiments are conducted on a server with 2 Intel Xeon 8352S CPUs and 4 NVIDIA L40 48GB GPUs.

5.2 Comparison with state-of-the-art methods

Correctness Evaluation. Watermark correctness refers to the accuracy with which embedded messages can be extracted from intact watermarked text. The match rate and bit accuracy of the baselines and our DMW using LLaMA-2-7B across four payloads and two datasets are reported in Table 1, where the best result is marked in bold for each group of comparisons. Additional experimental results obtained using Falcon-7B are provided in Appendix C.

We observe that our DMW consistently outperforms the four baselines by a significant margin in all cases, indicating that the DMW can more accurately reveal the messages from watermarked text. More specifically, the accuracy of all methods decreases with the increase of payload, but our DMW can maintain the accuracy better. Even with a 32-bit payload, the match rate and bit accuracy of DMW are only reduced by about 8% and 2%, respectively, while those of baselines decreased significantly.

Robustness Evaluation. We evaluate the robustness of different watermarking methods against three common semantic-preserving attacks: word insertion, synonym substitution, and paraphrasing. For the insertion and substitution attacks, we randomly modified 10% of the words in the watermarked text. For the paraphrasing attack, we adopt a substantially more destructive, document-level approach rather than sentence-level rewriting. We prompt GPT-3.5-turbo to rewrite entire passages as a whole, enabling global reorganization and cross-sentence restructuring. The attack process and exact prompt are provided in Appendix B. The robustness evaluation, measured by match rate and bit accuracy on the LLaMA-2-7B model using the C4 dataset, is shown in Figure 6. Additional Falcon-7B results are presented in Appendix D.

As seen from Figure 6, our DMW comprehensively surpasses the baselines in both match rate and bit accuracy. This advantage is particularly pronounced under paraphrasing attacks: in the case of 8-bit watermark, the DMW's match rate is 30% higher than the second-best method, demonstrating its strong resilience against watermarking attacks. Qu *et al.*'s method [33] ranks second overall, with its robustness stemming from the use of ECC, but its performance degrades significantly when ECC is disabled. Wang *et al.*'s method [40], which emphasizes watermark capacity and customization, is most susceptible to attack impacts.

The robustness degradation from 8-bit to 32-bit payloads reflects an inherent trade-off in multi-bit watermarking: longer watermarks require more extensive text modifications and

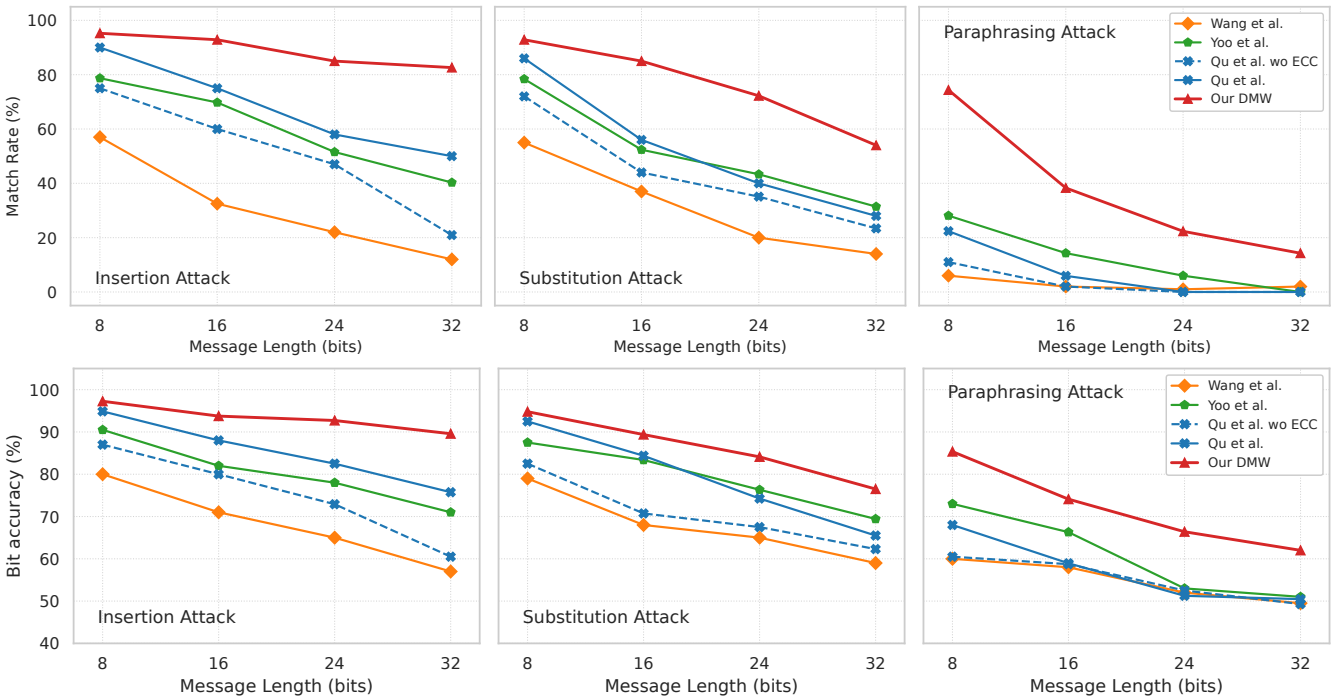


Figure 6: Comparison of watermark robustness using LLaMA on C4 dataset across various payloads.

create longer dependency chains during extraction, amplifying vulnerability to attacks. However, different bit lengths serve distinct practical purposes. The 8-bit configuration is motivated by deployments where reliability is paramount, *e.g.*, content provenance, policy enforcement, or high-stakes attribution. Its stronger robustness under all attacks demonstrates suitability for scenarios that require dependable watermark recovery over capacity. The 32-bit configuration, despite relatively lower robustness, maximizes watermark payload when richer identifiers, multi-party tracing, or fine-grained ownership metadata must be encoded. In such contexts, a higher payload is more valuable even with reduced resilience.

Fluency Evaluation. We employ perplexity to evaluate the fluency of watermarked text, referencing human-written text to assess its proximity to natural language. The perplexity distributions across different text types are presented in Figure 7, where all watermarking methods are evaluated under a 16-bit payload for simplicity.

Figure 7 shows that our DMW consistently achieves the lowest median perplexity across both models and datasets. In some cases, the median perplexity of DMW is even slightly lower than that of human text and is substantially lower than the those of Yoo *et al*. [45] and Qu *et al*. [33]. This indicates that DMW introduces minimal perceptual distortion during watermark embedding, thereby preserving text fluency. Furthermore, the interquartile range (IQR) of DMW is narrower than most other methods, suggesting greater stability in fluency performance. These advantages provide compelling evidence for the effectiveness of our proposed fluency cost.

Diversity Evaluation. We utilize semantic entropy (Sem-Ent [15]) to evaluate the diversity of both human-written text and watermarked texts under a 16-bit payload. The results are shown in Figure 8, where higher Sem-Ent scores indicate better text diversity, with the dashed line representing the semantic entropy of human text as the reference benchmark.

As illustrated, our DMW method performs the best among all watermarked texts, with a Sem-Ent score that is very close to and, in the case of the C4 dataset, slightly higher than that of human text. This confirms that the watermarked text of DMW maintains a diversity level comparable to that of human text, validating the effectiveness of our diversity cost. In contrast, other watermarking methods exhibit noticeably reduced semantic entropy relative to human text, reflecting their neglect of diversity in watermark design and the consequent inherent restriction on output variability.

Efficiency Evaluation. Computational efficiency is critical for the practical deployment of LLM watermarking. To evaluate the efficiency of different watermarking methods, we compare their embedding and extraction times under identical hardware conditions, with the results listed in Table 2.

As shown in Table 2, our DMW achieves the second-best embedding efficiency, trailing only Qu *et al*.'s method. This is because Qu *et al*. introduce a segment assignment mechanism explicitly optimized for embedding efficiency, while DMW pursues comprehensive watermarking performance, which requires computing multiple costs during embedding and consequently incurs additional time overhead.

Regarding extraction efficiency, our DMW achieves the

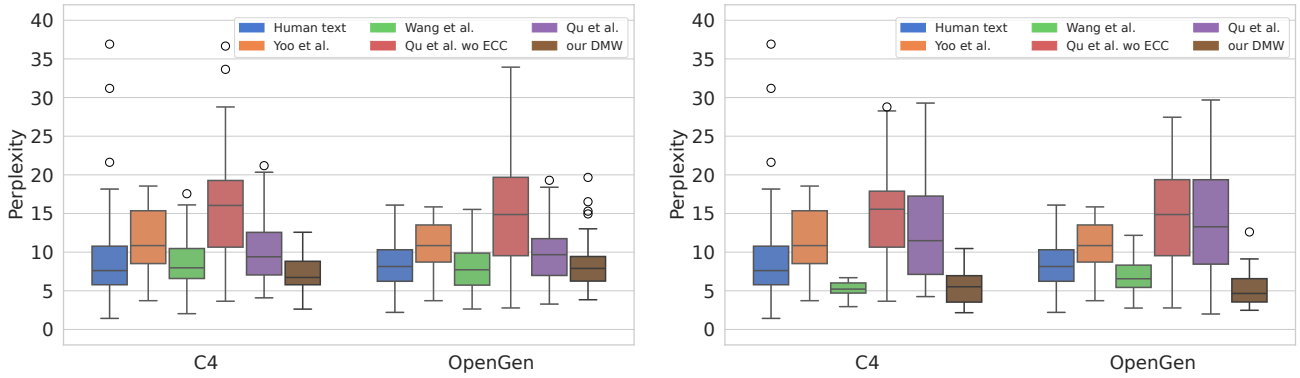


Figure 7: Comparison of fluency between human text and watermarked texts generated by LLaMA (left) and Falcon (right).

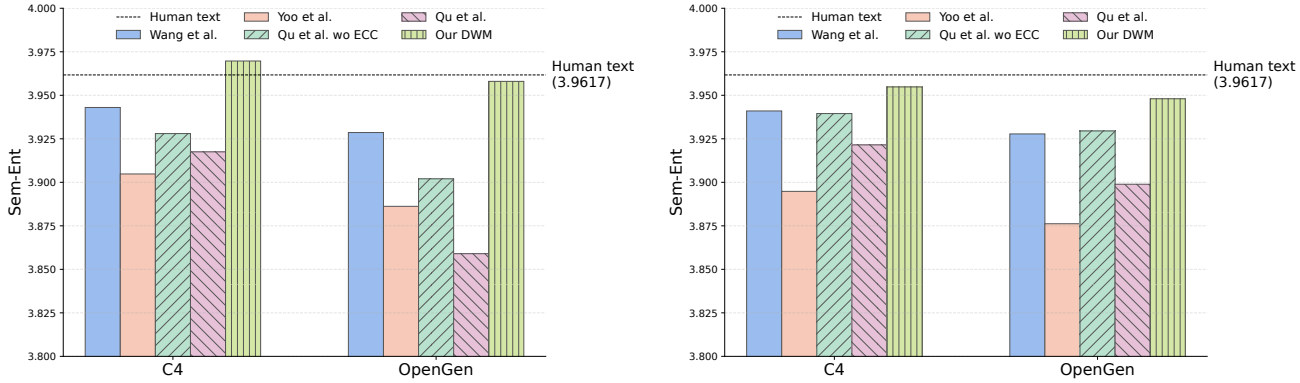


Figure 8: Comparison of diversity between human text and watermarked texts generated by LLaMA (left) and Falcon (right).

lowest extraction time, exceeding all baselines by approximately an order of magnitude. According to Eq. (5), watermark extraction in DMW only involves semantic projection and sparse matrix multiplication, which are highly efficient operations to reduc runtime overhead. In contrast, Yoo *et al*. [45] involves multiple language model calls and extensive candidate watermark validation, Wang *et al*. [40] requires exponential-time enumeration with respect to watermark bit length, and Qu *et al*. [33] introduces an additional error-correction stage. Consequently, these methods require significantly more extraction time.

5.3 Parameter Analysis

In this subsection, we analyze the effects of key parameters on our DMW's performance. We specifically discuss the parameters related to the semantic projection and the distortion cost; parameters for PO-STC are provided in Appendix H.

Semantic Projection Parameter. In semantic projection, the cluster number k determines the number of semantic categories to which a sentence can be projected, which in turn defines the bit string length, *i.e*., $\lceil \log_2 k \rceil$. To evaluate the effect of different k values on watermarking performance, we vary k and observe the changes in the text fluency (measured by perplexity), correctness (measured by the match rate), and robustness (measured by the match rate under paraphrasing attacks). We conduct experiments with a 24-bit watermark payload on LLaMA using the C4 dataset. Results are presented in Table 3, where MR and Para. MR represent match rate and paraphrasing match rate, respectively.

Table 3 shows that as k increases, perplexity, match rate, and paraphrasing match rate all rise. This indicates that a smaller k helps preserve text quality but lowers watermark

Table 2: Comparison of watermarking efficiency (in seconds) under various payloads and datasets. The best result is **boldfaced**, and the second best one is underlined.

Dataset	Method	Embedding/Extraction Time (s) ↓			
		8 bit	16 bit	24 bit	32 bit
C4	Yoo *et al*. [45]	14/3.04	75/3.35	151/3.64	430/5.40
	Wang *et al*. [40]	17/2.18	51/2.52	86/3.53	130/3.86
	Qu *et al*. [33]	**12**/3.32	**15**/3.17	**27**/3.17	**49**/3.30
	Our DMW	13/**0.028**	44/**0.08**	77/**0.15**	124/**0.23**
OpenGen	Yoo *et al*. [45]	11/3.13	68/3.45	172/3.50	442/6.28
	Wang *et al*. [40]	13/2.30	49/2.43	90/3.51	145/5.84
	Qu *et al*. [33]	**9**/3.31	**17**/3.22	**25**/3.21	**43**/3.23
	Our DMW	11/**0.025**	45/**0.10**	78/**0.15**	118/**0.28**

Table 3: Effect of cluster number (k) on DMW.

k	$\lceil \log_2 k \rceil$	PPL ↓	MR ↑	Para. MR ↑
4	2	**6.83**	82.50	9.56
8	3	7.03	89.50	14.62
16	4	7.25	93.17	**23.50**
32	5	7.47	**93.24**	22.56

Table 4: Effect of window size (r) on DMW.

r	PPL ↓	MR ↑	Para. MR ↑
4	7.49	90.49	14.00
6	7.41	94.37	11.50
8	**7.06**	92.19	16.50
10	7.62	91.67	19.00
12	7.34	93.15	14.00

robustness. The reasons are as follows: Smaller k yields fewer and coarser semantic partitions, so each cluster contains richer semantic variety. This makes it easier to find a semantically suitable alternative within a new cluster when a sentence is modified, which helps maintain text quality. For a given payload length and relative payload, smaller k shortens the per-sentence bit string, requiring more sentences to carry the watermark. However, the increased number of sentences makes it more difficult to fully extract the watermark after an attack, causing both match rates to decrease.

Table 3 also reveals that the change in perplexity is relatively small, whereas match rates vary substantially and approach saturation at $k = 16$. Therefore, we select a relatively large value, $k = 16$, as the default. Although larger k slightly decreases text quality, the significant gain in robustness makes this trade-off worthwhile.

Window Size. The fluency cost uses a sliding window size r, which is used to calculate the perplexity of r sentences within the window to assess their contextual dependency. We vary r to observe its effect on the fluency, correctness, and robustness of the watermarked text. The experimental settings are the same as those for the semantic projection parameter experiments, and the results are shown in Table 4.

As shown in Table 4, the window size r has a significant impact on text fluency, with perplexity initially decreasing then increasing as r grows. This is because moderately increasing r to a certain extent improves the stability of contextual dependency, better reflecting the relationship between preceding and current sentences. However, an excessively large r diminishes the relative importance of the current sentence, leading to an inaccurate calculation of the fluency cost. In contrast, the correctness and robustness of the watermark show no clear correlation with r. Therefore, we set the default window size to $r = 8$, where perplexity is at its minimum.

Distortion Costs. Our overall distortion cost function consists of the robustness cost (ρ_{rob}), fluency cost (ρ_{flu}), and diversity cost (ρ_{div}). Each distortion cost controls a different aspect of watermark performance, and in some cases, these objectives may even conflict with one another. To evaluate the effect of different distortion costs on watermarking performance, we conduct ablation experiments with various cost combinations, with results presented in Table 5.

Table 5: Effect of different distortion costs on DMW. ρ_{rob}, ρ_{flu} and ρ_{div} denote robust cost, fluency cost and diversity cost. ✔ and ✘ indicate that whether the corresponding cost is used or not. The best result is **boldfaced**; the second best is underlined.

ρ_{rob}	ρ_{flu}	ρ_{div}	MR ↑	Para. MR ↑	PPL ↓	Sem-Ent ↑
✔	✘	✘	98.95	67.51	8.16	3.90
✘	✔	✘	98.86	38.18	**7.18**	3.81
✘	✘	✔	97.51	34.85	8.43	**3.93**
✔	✔	✘	98.71	55.70	8.18	3.85
✔	✘	✔	98.75	62.13	8.94	3.89
✘	✔	✔	98.30	39.13	7.83	3.91
✔	✔	✔	**100.0**	**73.50**	7.30	3.89

Table 5 demonstrates that using a single cost provides the greatest improvement to its corresponding performance metric. For example, when using the costs ρ_{rob}, ρ_{flu} and ρ_{div} individually, the corresponding match rates (98.95 and 67.51), perplexity (7.18), and semantic entropy (3.93) reach their most optimal values, respectively. Besides, when combining different costs, watermarking performance shifts toward corresponding directions, indicating that appropriate cost selection (or cost weight adjustment) can adapt the watermarking method to different application scenarios. Finally, when all three costs are incorporated, we tune their weights to achieve an optimal overall balance. In our case, we put more emphasis on robustness, since its improvement is substantial, while the impact on text quality remains relatively minor.

6 Conclusion

This paper presents a distortion-minimization watermarking (DMW) framework that addresses the fundamental challenge of simultaneously optimizing capacity, robustness, and quality in LLM watermarking. The core idea is to quantify robustness and quality as measurable distortion costs, transforming watermarking into an optimization problem that minimizes total distortion under a given watermark length. For this, we first semantically project sentences into bit sequences. We then design a robustness cost and two quality costs to measure the suitability of textual modifications. We also develop periodically optimized syndrome-trellis codes that enable real-time optimization and flexible capacity control for watermark embedding. Together, these innovations allow DMW to achieve a superior balance among multiple competing objectives. This has been confirmed by extensive experiments, which demonstrate consistent improvements over existing methods.

Acknowledgments

This work was supported in part by the National Natural Science Foundation of China (No. 62476108), in part by the Hubei Provincial Natural Science Foundation of China (No. 2024AFB932), and in part by the CCF-NSFOCUS Kun-Peng Scientific Research Fund (No. CCF-NSFOCUS 2025009).

Ethical Considerations

Our work aims to strengthen the auditing and accountability of large language models (LLMs) through a watermarking framework, enabling reliable tracing of generated content and helping to deter misuse and support responsible deployment. All experiments were conducted on publicly available datasets and locally executed models; no personal data, user accounts, or interactions with deployed production systems were used. All evaluated attacks (e.g., paraphrasing, word substitution) were applied only to public datasets and did not target real users or live platforms. Below we analyze potential impacts on relevant stakeholder groups.

LLM Providers. Watermarks enable reliable tracing of generated text to a specific model provider without the need to store or search large volumes of user prompts and outputs, which would otherwise incur significant storage and computational costs. However, If providers adopt watermarking in a secret-key-only scheme (verifiable only by themselves), misuse of verification keys or incorrect attribution could harm developers or expose them to legal or reputational risks.

Downstream Users (platforms, educators, regulators). These stakeholders gain improved transparency and tools for identifying AI-generated content, which can help combat misinformation or academic dishonesty. Risks include unequal access to verification if only certain entities can validate watermarks, or misplaced trust if watermarks are bypassed or improperly implemented.

End Users or Content Creators. Users may benefit from clear disclosure practices and reduced ambiguity regarding authorship. However, privacy concerns could arise if watermarking encodes information related to user prompts or usage patterns. To mitigate this, watermark signals should remain independent of identifiable user attributes, and their deployment should include clear disclosures and accessible verification methods.

Open Science

Our research artifacts for this work, including source code, datasets, training and fine-tuning scripts, and evaluation instructions, are publicly available on the Zenodo and GitHub repositories.

References

[1] Sahar Abdelnabi and Mario Fritz. Adversarial watermarking transformer: Towards tracing text provenance with data hiding. In *2021 IEEE Symposium on Security and Privacy (SP)*, pages 121–140. IEEE, 2021.

[2] Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. GPT-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023.

[3] Ebtesam Almazrouei, Hamza Alobeidli, Abdulaziz Alshamsi, Alessandro Cappelli, Ruxandra Cojocaru, Mérouane Debbah, Étienne Goffinet, Daniel Hesslow, Julien Launay, Quentin Malartic, et al. The Falcon series of open language models. *arXiv preprint arXiv:2311.16867*, 2023.

[4] Leonard F Bereska. Mechanistic interpretability for AI safety—a review. In *Proceedings of The 1st Conference on Lifelong Learning Agents*, 2022.

[5] Solène Bernard, Patrick Bas, John Klein, and Tomas Pevny. Explicit optimization of min max steganographic game. *IEEE Transactions on Information Forensics and Security*, 16:812–823, 2020.

[6] Xiao Bi, Deli Chen, Guanting Chen, Shanhuang Chen, Damai Dai, et al. DeepSeek LLM: Scaling open-source language models with longtermism. *CoRR*, 2024.

[7] Massieh Kordi Boroujeny, Ya Jiang, Kai Zeng, and Brian L Mark. Multi-bit distortion-free watermarking for large language models. *arXiv preprint arXiv:2402.16578*, 2024.

[8] Kejiang Chen, Hang Zhou, Wenbo Zhou, Weiming Zhang, and Nenghai Yu. Defining cost functions for adaptive JPEG steganography at the microscale. *IEEE Transactions on Information Forensics and Security*, 14(4):1052–1066, 2018.

[9] Miranda Christ, Sam Gunn, and Or Zamir. Undetectable watermarks for language models. In *37th Annual Conference on Learning Theory*, pages 1125–1139, 2024.

[10] Amirhossein Dabiriaghdam and Lele Wang. SimMark: A robust sentence-level similarity-based watermarking algorithm for large language models. *arXiv preprint arXiv:2502.02787*, 2025.

[11] Tomáš Denemark and Jessica Fridrich. Steganography with multiple JPEG images of the same scene. *IEEE Transactions on Information Forensics and Security*, 12(10):2308–2319, 2017.

[12] Pierre Fernandez, Antoine Chaffin, Karim Tit, Vivien Chappelier, and Teddy Furon. Three bricks to consolidate watermarks for large language models. In *IEEE international workshop on information forensics and security (WIFS)*, pages 1–6. IEEE, 2023.

[13] Tomáš Filler, Jan Judas, and Jessica Fridrich. Minimizing additive distortion in steganography using syndrome-trellis codes. *IEEE Transactions on Information Forensics and Security*, 6(3):920–935, 2011.

[14] Chenchen Gu, Xiang Lisa Li, Percy Liang, and Tatsunori Hashimoto. On the learnability of watermarks for language models. In *ICLR*, 2023.

[15] Seungju Han, Beomsu Kim, and Buru Chang. Measuring and improving semantic diversity of dialogue generation. In *Findings of the Association for Computational Linguistics: EMNLP*, pages 934–950, 2022.

[16] Vojtěch Holub, Jessica Fridrich, and Tomáš Denemark. Universal distortion function for steganography in an arbitrary domain. *EURASIP Journal on Information Security*, 2014(1):1, 2014.

[17] Abe Hou, Jingyu Zhang, Tianxing He, Yichen Wang, Yung-Sung Chuang, Hongwei Wang, Lingfeng Shen, Benjamin Van Durme, Daniel Khashabi, and Yulia Tsvetkov. SemStamp: A semantic watermark with paraphrastic robustness for text generation. In *NAACL*, pages 4067–4082, 2024.

[18] Abe Bohan Hou, Jingyu Zhang, Yichen Wang, Daniel Khashabi, and Tianxing He. k-semstamp: A clustering-based semantic watermark for detection of machine-generated text. In *ACL (Findings)*, 2024.

[19] Zhengmian Hu, Lichang Chen, Xidong Wu, Yihan Wu, Hongyang Zhang, and Heng Huang. Unbiased watermark for large language models. In *ICLR*, 2024.

[20] Kaan Efe Keleş, Ömer Kaan Gürbüz, and Mucahid Kutlu. I know you did not write that! a sampling based watermarking method for identifying machine generated text. In *Proceedings of the 1st Workshop on GenAI Content Detection (GenAIDetect)*, pages 140–149, 2025.

[21] Graham Kendall and Jaime A Teixeira da Silva. Risks of abuse of large language models, like chatgpt, in scientific publishing: authorship, predatory publishing, and paper mills. *Learned Publishing*, 37(1), 2024.

[22] John Kirchenbauer, Jonas Geiping, Yuxin Wen, Jonathan Katz, Ian Miers, and Tom Goldstein. A watermark for large language models. In *Proceedings of the 40th International Conference on Machine Learning*, pages 17061–17084. PMLR, 2023.

[23] John Kirchenbauer, Jonas Geiping, Yuxin Wen, Manli Shu, Khalid Saifullah, Kezhi Kong, Kasun Fernando, Aniruddha Saha, Micah Goldblum, and Tom Goldstein. On the reliability of watermarks for large language models. In *ICLR*, 2024.

[24] Kalpesh Krishna, Yixiao Song, Marzena Karpinska, John Wieting, and Mohit Iyyer. Paraphrasing evades detectors of AI-generated text, but retrieval is an effective defense. *Advances in Neural Information Processing Systems*, 36:27469–27500, 2023.

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

[26] Bin Li, Ming Wang, Jiwu Huang, and Xiaolong Li. A new cost function for spatial image steganography. In *IEEE International conference on image processing (ICIP)*, pages 4206–4210, 2014.

[27] Qihao Lin, Chen Tang, Xiangyang Li, et al. Dermark: A dynamic, efficient and robust multi-bit watermark for large language models. *arXiv preprint arXiv:2502.05213*, 2025.

[28] Aiwei Liu, Leyi Pan, Yijian Lu, Jingjing Li, Xuming Hu, Xi Zhang, Lijie Wen, Irwin King, Hui Xiong, and Philip Yu. A survey of text watermarking in the era of large language models. *ACM Computing Surveys*, 57(2):1–36, 2024.

[29] Yepeng Liu and Yuheng Bu. Adaptive text watermark for large language models. In *Proceedings of the 41st International Conference on Machine Learning*, pages 30718–30737, 2024.

[30] Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. Pointer sentinel mixture models. In *ICLR*, 2017.

[31] Xianbo Mo, Shunquan Tan, Weixuan Tang, Bin Li, and Jiwu Huang. ReLOAD: Using reinforcement learning to optimize asymmetric distortion for additive steganography. *IEEE Transactions on Information Forensics and Security*, 18:1524–1538, 2023.

[32] Pierre Moulin and Joseph A O'Sullivan. Information-theoretic analysis of information hiding. *IEEE Transactions on information theory*, 49(3):563–593, 2003.

[33] Wenjie Qu, Wengrui Zheng, Tianyang Tao, Dong Yin, Yanze Jiang, Zhihua Tian, Wei Zou, Jinyuan Jia, and Jiaheng Zhang. Provably robust multi-bit watermarking for AI-generated text. In *Proceedings of the 34th USENIX Conference on Security Symposium*, 2025.

[34] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of machine learning research*, 21(140):1–67, 2020.

[35] Nils Reimers and Iryna Gurevych. Sentence-BERT: Sentence embeddings using siamese BERT-networks. In *EMNLP-IJCNLP*, pages 3982–3992, 2019.

[36] Vahid Sedighi, Rémi Cogranne, and Jessica Fridrich. Content-adaptive steganography by minimizing statistical detectability. *IEEE Transactions on Information Forensics and Security*, 11(2):221–234, 2015.

[37] Weixuan Tang, Bin Li, Mauro Barni, Jin Li, and Jiwu Huang. An automatic cost learning framework for image steganography using deep reinforcement learning. *IEEE Transactions on Information Forensics and Security*, 16:952–967, 2020.

[38] Weixuan Tang, Bin Li, Shunquan Tan, Mauro Barni, and Jiwu Huang. Cnn-based adversarial embedding for image steganography. *IEEE Transactions on Information Forensics and Security*, 14(8):2074–2087, 2019.

[39] Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. LLaMA 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*, 2023.

[40] Lean Wang, Wenkai Yang, Deli Chen, Hao Zhou, Yankai Lin, Fandong Meng, Jie Zhou, and Xu Sun. Towards codable watermarking for injecting multi-bits information to LLMs. In *ICLR*, 2024.

[41] John Wieting, Kevin Gimpel, Graham Neubig, and Taylor Berg-Kirkpatrick. Paraphrastic representations at scale. In *EMNLP*, pages 379–388, 2022.

[42] Yihan Wu, Ruibo Chen, Zhengmian Hu, Yanshuo Chen, Junfeng Guo, Hongyang Zhang, and Heng Huang. Distortion-free watermarks are not truly distortion-free under watermark key collisions. *arXiv preprint arXiv:2406.02603*, 2024.

[43] Xiaojun Xu, Yuanshun Yao, and Yang Liu. Learning to watermark LLM-generated text via reinforcement learning. In *Workshop on GenAI Watermarking*, 2024.

[44] Jianhua Yang, Danyang Ruan, Jiwu Huang, Xiangui Kang, and Yun-Qing Shi. An embedding cost learning framework using gan. *IEEE Transactions on Information Forensics and Security*, 15:839–851, 2019.

[45] KiYoon Yoo, Wonhyuk Ahn, Jiho Jang, and Nojun Kwak. Robust multi-bit natural language watermarking through invariant features. In *ACL*, pages 2092–2115, 2023.

[46] KiYoon Yoo, Wonhyuk Ahn, and Nojun Kwak. Advancing beyond identification: Multi-bit watermark for large language models. In *NAACL*, pages 4031–4055, 2024.

[47] Susan Zhang, Stephen Roller, Naman Goyal, Mikel Artetxe, Moya Chen, Shuohui Chen, Christopher Dewan, Mona Diab, Xian Li, Xi Victoria Lin, et al. OPT: Open pre-trained transformer language models. *arXiv preprint arXiv:2205.01068*, 2022.

[48] Tianyi Zhang, Varsha Kishore, Felix Wu, Kilian Q Weinberger, and Yoav Artzi. BERTScore: Evaluating text generation with BERT. In *ICLR*, 2020.

[49] Xuandong Zhao, Prabhanjan Vijendra Ananth, Lei Li, and Yu-Xiang Wang. Provable robust watermarking for AI-generated text. In *ICLR*, 2024.

Appendix

A Notions

To enhance readability and ensure notational clarity, we provide a comprehensive summary of the key notations employed throughout this paper, as shown in Table 6. Serving as a quick reference, this table is designed to facilitate the reader's comprehension of our proposed methodology and the corresponding equations. The notations are grouped thematically, accompanied by brief descriptions for disambiguation.

B Paraphrasing Attack Prompt

To perform a strong paraphrasing attack, we adopt the document-level rewriting prompt recommended in [23], which is shown to be highly effective at inducing substantial semantic rewrites while preserving meaning and length.

> *As an expert copy-editor, please rewrite the following text in your own voice while ensuring that the final output contains the same information as the original text and has roughly the same length. Please paraphrase all sentences and do not omit any crucial details. ...*

We apply this prompt at the document level by feeding each entire passage to GPT-3.5-turbo in a single request, allowing the model to reorganize and rewrite the text holistically.

C Correctness Evaluation using Falcon

To validate the generalizability of our proposed DMW, this section presents supplementary results for watermark correctness evaluation on the Falcon-7B [3] model using the C4 and

Table 6: Notations and their descriptions

Notation	Description
M	Large language model
$\mathbf{P}$, p	Prompt token sequence, prompt length
K	Secret key
$\mathbf{m}$, m	Watermark message, message bit length
$\mathbf{m}_i$, u	Message segment, message segment length
t, T	Time step index, total time steps
S_t, $\mathbf{S}_{0:t}$	The t-th token, token sequence at time t
$\mathbf{S}_{si:si+s-1}$, $\mathbf{S}_i$	The i-th token segment
s, N	Token segment length, number of segments
$\mathbf{X}_i$, $\mathbf{Y}_i$	The i-th clean/watermarked token segment
$X_{i,j}$, $Y_{i,j}$	The j-th clean/watermarked token in $\mathbf{X}_i$/$\mathbf{Y}_i$
$\rho_{i,j}$	Distortion cost of $X_{i,j}$
v	length of bit segment projected from $\mathbf{X}_i$
$\mathcal{C}$	Binary linear code
$\mathbb{H}$	Parity-check matrix of $\mathcal{C}$
$\hat{\mathbb{H}}$	Base sub-matrix of $\mathbb{H}$
$\mathbb{H}_i$	The i-th segment-based sub-matrix of $\mathbb{H}$
n	Clean/watermarked bit vector length
w, h	Weight/height of sub-matrix $\hat{\mathbb{H}}$
α	Relative payload of watermarks
$\mathbf{s}$	syndrome
$\mathcal{P}(\cdot)$	Semantic projection function
$f(\cdot)$, d	Sentence encoder, feature dimension
k, μ_c	Number of clusters, nearest cluster centroid
γ	Cluster margin threshold
r	Window size for fluency cost
$\text{bin}(\cdot)$	Binarization function
$\text{dist}(\cdot,\cdot)$	Cosine distance function
ρ_{rob}, ρ_{flu}, ρ_{div}	Robustness/fluency/diversity cost
λ_{rob}, λ_{flu}, λ_{div}	Cost weight
L	Number of candidate sentences for ρ_{div}

OpenGen datasets. The experimental setup is identical to that in the main text, and the results are listed in Table 7.

The results demonstrate that, on the Falcon-7B model, our DMW significantly outperforms all baselines in both match rate and bit accuracy across all payloads and datasets. This trend is consistent with the observations on the LLaMA model in Table 1, confirming that the DMW is not dependent on a specific model and possesses strong generalization ability.

D Robustness Evaluation using Falcon

This section further provides the robustness evaluation results of the DMW on the Falcon-7B model. As illustrated in Figure 8, we tested the performance of different watermarking methods on the C4 dataset against three typical attacks under various watermark payloads.

The results clearly show that our DMW comprehensively surpasses the baselines in both match rate and bit accuracy across all attack types and payload settings, which is consistent with the observations in Figure 6.

E Details of the Clustering Module

We construct the semantic clustering module using the WikiText-2 dataset [30], which contains roughly 2 million sentences extracted through standard preprocessing. WikiText-2 provides broad topical coverage and linguistic diversity, ensuring that the resulting clusters generalize well to downstream watermarking tasks.

Following prior work on contrastive sentence representation learning [17], we fine-tune a SBERT encoder [35] (the pretrained model name is `bert-base-uncased`) using a contrastive objective. Positive pairs are generated via stochastic augmentations of the same sentence, and negative samples are drawn from within the batch. Training is performed with AdamW for five epochs. The resulting 768-dimensional feature vectors are then used for semantic clustering.

F Adaptive Attacks on Semantic Projection

In practical deployments, an informed adversary may exploit knowledge of the watermarking mechanism to launch targeted attacks against specific components. In our DMW framework, watermark bits are assigned through semantic-space clustering, and a margin is used to select sentences far from cluster boundaries to improve stability. An attacker could combine this prior knowledge with rewriting to design adaptive attacks that disrupt the sentence-to-bit mapping. We examine two threat models: (1) the attacker knows the sentence-to-bit mapping but not the margin, and (2) the attacker knows both the mapping and the margin. Both attacks attempt to alter the assigned bit sequence by inducing cluster changes.

Because sentence stability varies across the semantic space, and some sentences near cluster centers remain semantically invariant even after rewriting, we enhance attack efficiency by modifying only half the sentences with 10 rewriting attempts each. Attacker 1 rewrites randomly selected sentences, whereas Attacker 2 rewrites only boundary-region sentences. Note that these adaptive attacks operate at the sentence level rather than document level due to targeted sentence-specific modifications. Table 9 compares these two attacks against the vanilla paraphrasing on LLaMA-2-7B using the C4 dataset.

As shown in Table 9, Attack 1 achieves higher match rates than vanilla paraphrasing under longer payloads, demonstrating that knowledge of sentence-to-bit mapping alone fails to alter sentence categories through limited rewrites, confirming semantic projection stability. Attack 2 yields lower match rates across all payloads because boundary sentences are more semantically vulnerable. This limitation can be mitigated by strengthening the robustness cost. For example, replacing the linear cosine-distance term in Eq. (9) with a power-law form to bias watermark embedding toward cluster centers.

Table 7: Comparison of watermark correctness using Falcon across various payloads and datasets.

Dataset	Method	8 bit		16 bit		24 bit		32 bit	
		Match Rate ↑	Bit Acc ↑	Match Rate ↑	Bit Acc ↑	Match Rate ↑	Bit Acc ↑	Match Rate ↑	Bit Acc ↑
C4	Yoo *et al.* [45]	99.00	99.50	97.00	98.93	94.00	97.10	90.50	96.87
	Wang *et al.* [40]	92.50	98.45	75.00	90.25	62.50	82.50	45.00	78.20
	Qu *et al.* wo ECC [33]	90.00	96.25	83.00	91.05	78.00	89.94	70.00	85.99
	Qu *et al.* [33]	95.00	98.81	90.50	96.10	84.00	92.21	75.00	87.81
	Our DMW	**99.50**	**99.57**	**99.00**	**99.81**	**96.50**	**98.76**	**93.00**	**97.97**
OpenGen	Yoo *et al.* [45]	98.00	99.32	97.50	98.77	92.50	97.19	89.00	96.25
	Wang *et al.* [40]	85.00	94.75	67.50	81.28	55.00	79.67	50.00	77.09
	Qu *et al.* wo ECC [33]	92.00	96.50	85.00	91.25	79.00	88.79	74.50	85.91
	Qu *et al.* [33]	97.50	99.21	90.00	96.25	80.00	92.57	74.00	86.72
	Our DMW	**99.00**	**99.45**	**98.50**	**99.12**	**95.00**	**98.27**	**92.50**	**97.25**

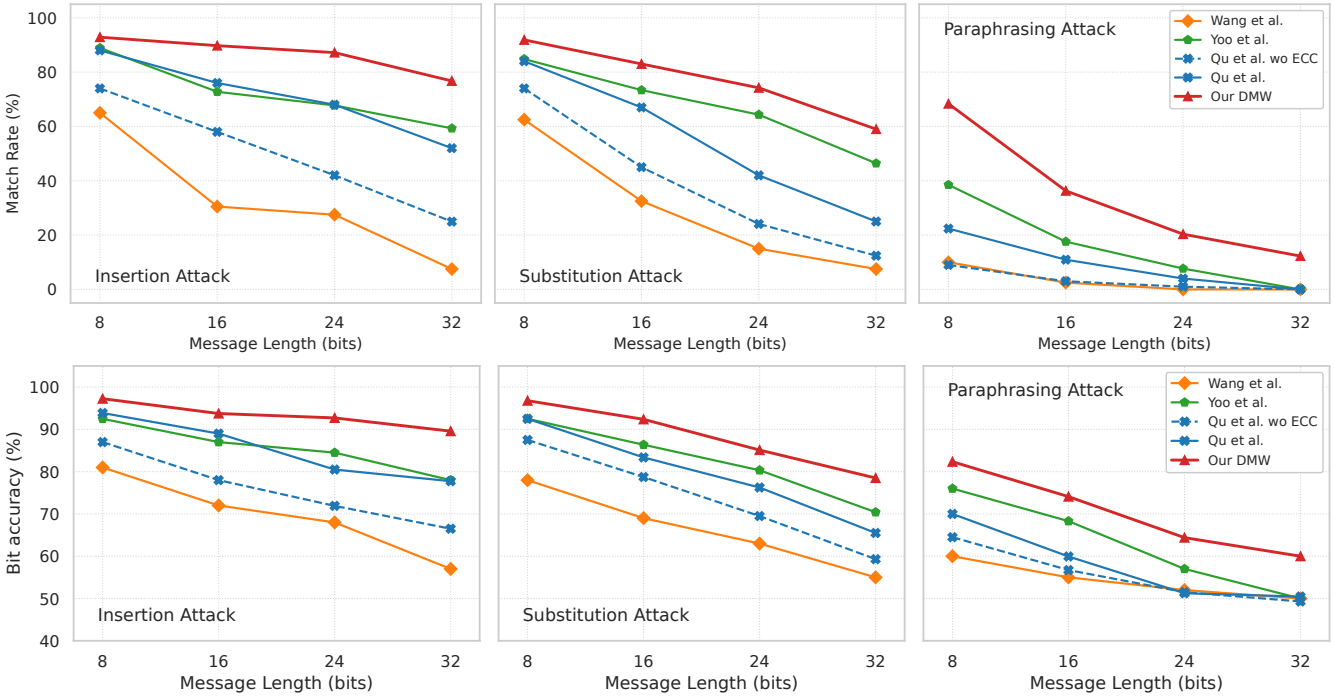


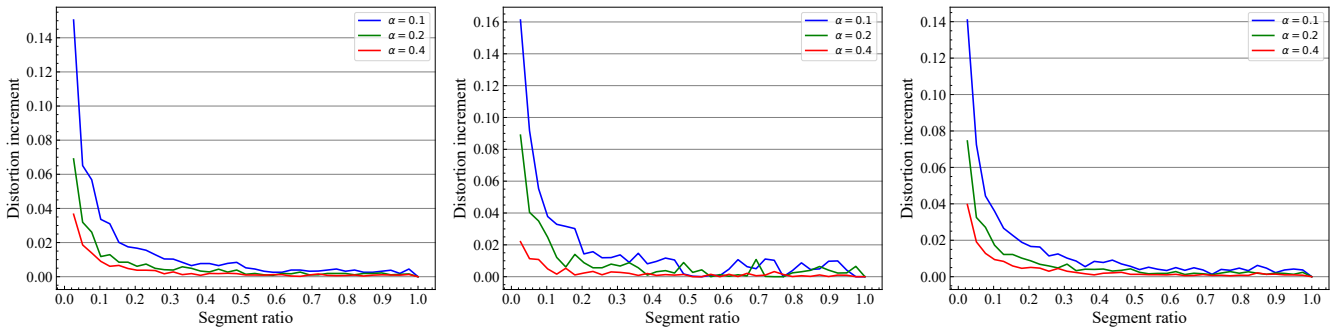
Table 8: Comparison of watermark robustness using Falcon on C4 dataset across various payloads.

G Coding Loss of PO-STC

To evaluate the potential coding loss caused by segmentation, we conduct simulation experiments comparing PO-STC with standard STC under varying segment lengths. Given an original bit vector $\mathbf{x}$ of fixed length and its associated cost vector ρ, we embed watermark message of a specified length using both PO-STC and STC, yielding watermarked bit sequences $\mathbf{y}_{postc}$ and $\mathbf{y}_{stc}$, respectively. The total distortion is then computed using Eq. (4) as $D(\mathbf{x},\mathbf{y}_{postc})$ and $D(\mathbf{x},\mathbf{y}_{stc})$. Coding loss is quantified by the distortion increment, defined as $\frac{D(\mathbf{x},\mathbf{y}_{postc})-D(\mathbf{x},\mathbf{y}_{stc})}{D(\mathbf{x},\mathbf{y}_{stc})}$, where smaller values indicate that the embedding performance of PO-STC more closely approximates that of standard STC.

We set the length of $\mathbf{x}$ to 10^4 bits and consider relative payloads of $\alpha = 0.1, 0.2, 0.4$. The cost vector ρ follows three distributions: constant cost ($\rho_i = 1$), uniform distribution ($\rho_i \sim \mathcal{U}(0,1)$), and normal distribution ($\rho_i \sim \mathcal{N}(0.5, 0.2)$). Segment length is measured by segment ratio that represents the proportion of segment length to the total length of $\mathbf{x}$, ranging from $\frac{1}{40}$ to $\frac{40}{40}$ in increments of $\frac{1}{40}$. For each condition, experiments are repeated 10 times with averaged results.

As shown in Figure 9, across all watermark payloads and cost distributions, the distortion increment consistently decreases as the segment ratio increases, converging rapidly to zero. In addition, when the relative payload α is large, the low distortion increment is more obvious. Even at the smallest segment ratio of $\alpha = 0.4$, overall distortion increases by at most 4%. These results confirm that the loss in embedding performance due to segmentation is limited.



(a) Constant cost (b) Uniform distribution cost (c) Normal distribution cost

Figure 9: Coding loss of PO-STC with different cost distributions.

Table 9: Vanilla and adaptive paraphrasing attacks on DMW.

Attack type	8-bit ↑	16-bit ↑	24-bit ↑	32-bit ↑
Vanilla attack	76.19	37.14	19.68	14.73
Adaptive attack 1	46.34	29.89	23.11	18.43
Adaptive attack 2	29.57	26.56	12.10	8.87

Table 10: Effect of segment size v on DMW.

v	PPL ↓	MR ↑	Para. MR ↑
6	6.85	96.43	22.14
8	7.09	**100**	31.54
10	5.80	**100**	24.43
12	**5.22**	98.06	**33.65**
14	6.08	97.11	27.68
16	7.12	98.29	25.86

Table 11: Effect of relative payload α on DMW.

α	PPL ↓	MR ↑	Para. MR ↑
0.125	7.15	71.00	14.50
0.25	7.28	90.85	24.00
0.5	7.52	98.79	31.00

Table 12: Effect of sub-matrix height h on DMW.

h	PPL ↓	MR ↑	Para. MR ↑
6	7.40	96.56	18.29
8	7.18	97.75	22.00
10	7.42	97.30	23.00
12	6.63	98.89	31.50
14	7.16	97.22	27.37
16	7.89	98.31	23.91

H PO-STC Parameter Analysis

This section aims to analyze the impact of several key hyper-parameters in the PO-STC on watermarking performance. These include the segment length (v), the relative payload (α), and the sub-matrix height (h). All experiments are conducted on the C4 dataset with a 24-bit watermark payload to evaluate the trade-offs among text quality (PPL), watermark correctness (MR), and robustness (Para. MR).

Segment Length. Table 10 shows the effect of different segment length v on watermarking performance. The results indicate that the choice of v significantly affects both text quality and robustness. As v increases from 6 to 12, the text's perplexity (PPL) first decreases and then increases, reaching its optimum at $v = 12$. This suggests that a moderate segment length helps maintain text coherence. Concurrently, watermark robustness (Para. MR) also peaks at $v = 12$. Overall, selecting a medium segment length (*e.g.*, 8 or 12) achieves an ideal balance among text quality, correctness, and robustness.

Relative Payload. Table 11 investigates the impact of different relative payload α on watermarking performance. The results show that as α increases, both watermark correctness (MR) and robustness (Para. MR) improve significantly. This is because a higher relative payload implies that more watermark information is embedded per unit of text, resulting in a denser watermark distribution and thus enhancing its resistance to attacks. However, this performance gain is accompanied by a slight degradation in text quality (PPL). This result reveals a direct trade-off between watermark strength and text quality, allowing users to adjust α based on the requirements of specific application scenarios.

Sub-matrix Height. Table 12 analyzes the influence of the different sub-matrix height h on PO-STC encoding. The parameter h determines the complexity and efficiency of the code. The results show that as h increases, both robustness (Para. MR) and correctness (MR) exhibit a trend of initially increasing and then fluctuating, with robustness performing best at $h = 12$. In contrast, text quality (PPL) does not show a clear monotonic trend. This indicates that h is a critical parameter that requires careful tuning. A moderate value for h (*e.g.*, 12) can maximize watermarking performance without a significant sacrifice in text quality.

Algorithm 1 PO-STC Encoder in a Python-like style

```
import numpy as np

# arrange the number and order of two types of sub-matrices
def arrange_matrices(shorter, longer, msg_length, inv_alpha):
    mat_width = []
    for i in range(1, msg_length + 1):
        mat_width.append(longer if sum(mat_width[:i]) + longer <= i * inv_alpha + 0.5 else shorter)
    mat_type = [1 if w == longer else 0 for w in mat_width]
    return mat_type, mat_width

class POSTCEncoder():
    def __init__(self, msg, alpha, mat_height, seg):
        msg_length = len(msg)
        inv_alpha = 1 / alpha
        shorter, longer = np.floor(inv_alpha).astype(int), np.ceil(inv_alpha).astype(int)
        self.columns = [get_matrix(shorter, mat_height), get_matrix(longer, mat_height)]
        mat_type, mat_width = arrange_matrices(shorter, longer, msg_length, inv_alpha)
        self.mat_type = mat_type
        self.mat_width = mat_width
        self.msg_length = msg_length

        self.state_num = 1 << mat_height
        self.state = np.arange(self.state_num, dtype=np.uint32)
        self.prices = np.full(self.state_num, np.inf, dtype=np.float32)
        self.prices[0] = 0.0
        self.path = np.zeros((longer * seg, self.state_num), dtype=np.uint8)
        self.mat_height = mat_height
        self.msg = msg
        self.seg = seg
        self.vec_idx = 0
        self.msg_idx = 0
        self.col_idx = 0

    def embed_bit(self, vector_bit, cost):
        # get two local costs: non-modification cost (c1) and modification cost (c2)
        column = self.columns[self.mat_type[self.msg_idx]][self.col_idx]
        c1, c2 = vector_bit * cost, (1 - vector_bit) * cost

        # get the global distortion and update the optimal path for each state
        alt_state = self.state ^ column
        v1, v2 = self.prices[self.state] + c1, self.prices[alt_state] + c2
        self.prices = np.minimum(v1, v2)
        self.path[self.vec_idx, :] = (self.prices == v2).astype(np.uint8)

        # adjust the preceding paths for changed state to aviod the backward process
        sel_idx = self.path[self.vec_idx, :].astype(bool)
        self.path[:self.vec_idx, sel_idx] = self.path[:self.vec_idx, alt_state[sel_idx]]
        self.vec_idx += 1
        self.col_idx += 1

        # state transition
        if self.col_idx >= self.mat_width[self.msg_idx]:
            self.prices[:self.state_num//2] = self.prices[self.msg[self.msg_idx]::2]
            self.prices[self.state_num//2:] = np.inf
            self.path[:, :self.state_num//2] = self.path[:, self.msg[self.msg_idx]::2]

            # periodically determine the optimal sub-path in a segment
            if (self.msg_idx + 1) % self.seg == 0 or self.msg_idx + 1 == self.msg_length:
                min_idx = np.argmin(self.prices[:self.state_num//2])
                stego_bits = self.path[:self.vec_idx, min_idx]
                self.prices[self.state != min_idx] = np.inf
                self.vec_idx = 0
            self.msg_idx += 1
            self.col_idx = 0

        return stego_bits if self.col_idx == 0 else np.empty(0, dtype=np.uint8)

    def is_finished(self):
        return self.msg_idx >= self.msg_length

def main():
    stego = np.empty(0, dtype=np.uint8)
    msg = np.random.randint(0, 2, size=400, dtype='uint8') # message bits to be embedded
    postc_encoder = POSTCEncoder(msg, alpha=0.4, mat_height=10, seg=10)
    while not postc_encoder.is_finished():
        cover_bit, cost = np.random.randint(0, 2, dtype='uint8'), np.float32(1.0) # simulate real-time bit/cost
        stego_bits = postc_encoder.embed_bit(cover_bit, cost)
        stego = np.append(stego, stego_bits)
```