An Adaptive Entropy Threshold Watermark

Anonymous ACL submission

Abstract

By embedding and detecting hidden features in text, watermarking algorithms for large language models can effectively identify machine-generated text. However, such embedding leads to a decline in text quality, especially in low-entropy scenarios where performance needs improvement. Methods for determining entropy thresholds based on experimental and historical text strategies require significant computational resources and time, and they exhibit poor adaptability to unknown tasks. In this work, we propose an adaptive entropy threshold watermarking method that automates the determination of thresholds during the generation and detection processes. Specifically, we leverage the entropy distribution characteristics of text sequences generated by large models to identify task-specific entropy properties, thereby calculating entropy thresholds to filter low-entropy segments. This enhances detection capability while maintaining a certain level of code-related text quality. Experiments demonstrate that our method ensures code-related text quality and improves detection performance across diverse text tasks.

1 Introduction

The rapid advancement of large language models (LLMs) has revolutionized text generation across diverse domains, including creative writing (Sun et al., b,a), technical documentation (Yang et al., a,b), and code synthesis (Li et al., b; Jain et al.; Gu et al.). However, the proliferation of machine-generated content raises critical challenges in authenticity verification (Burrus et al.; Ayoobi et al., a) and misuse prevention (Ayoobi et al., b; Dammu et al.). Watermarking algorithms, which embed imperceptible features into generated text, have emerged as a promising solution to identify machine-originated content (Liu et al., 2024; Chen et al.; Yoo et al.). Despite their potential, existing watermarking methods face a fundamental trade-off between detection robustness and text quality preservation, particularly in low-entropy scenarios such as code generation, where model predictions exhibit high confidence and limited variability (Long et al.; Lee et al., 2023).

Current approaches, such as SWEET (Lee et al., 2023) and WLLM (Baldassini et al., 2024), rely on static entropy thresholds to determine watermark insertion points. While effective in high-entropy contexts (e.g., open-ended text), these fixed thresholds struggle to adapt to dynamic entropy distributions inherent in code generation tasks. For instance, programming languages often involve repetitive syntactic structures (low entropy) interspersed with variable naming or logical decisions (higher entropy) (Liu and Bu, 2024; Liu et al., 2024). Manually calibrating thresholds for each language or task demands extensive experimentation, rendering these methods computationally prohibitive and poorly generalizable (Liu and Bu, 2024; Antoun et al.). Existing methods also exhibit inherent limitations in cross-model applicability: thresholds optimized for specific architectures (e.g., Starcoder) fail to generalize to others (e.g., Qwen2.5-Coder), necessitating redundant parameter searches for each new model (Baldassini et al., 2024; Li et al., a; Liu and Bu, 2024; Tu et al., 2023). Furthermore, hybrid content generation—such as code interlaced with natural language comments—introduces abrupt entropy shifts that static strategies cannot reconcile, degrading both watermark detectability and functional correctness (Liu and Bu, 2024; Chen et al.). Another critical shortcoming lies in detection sensitivity: prior works assign uniform weights to watermarked tokens, disregarding the importance of entropy-driven variations (Lu et al., 2024; Baldassini et al., 2024). This uniform weighting reduces sensitivity in low-entropy regimes, where subtle watermark signals are easily overshadowed by high-confidence model predictions (Baldassini

et al., 2024). These limitations collectively underscore the need for an adaptive framework that dynamically aligns watermarking strategies with the intrinsic entropy characteristics of generated text while ensuring cross-model robustness and task-agnostic adaptability (Yoo et al.; Liu and Bu, 2024).

In response, (Yoo et al.) proposes a watermarking method with dynamic entropy threshold adjustment that adaptively selects embedding positions via historical entropy distributions, though its generalizability across hybrid-modality cross-task scenarios requires further validation. Similarly, (Liu and Bu, 2024) introduces an adaptive entropy-threshold watermarking method leveraging historical entropy for dynamic adjustment, demonstrating robustness in low-entropy code generation but lacking comprehensive cross-model validation. WatME (Chen et al.) employs lexical redundancy-based watermarking with dynamic token optimization, yet limitations remain in dynamic entropy adaptation and cross-task robustness. Other notable approaches, such as the unforgeable publicly verifiable watermark (UPV) (Liu et al., 2023) and distribution-preserving DiPmark (Liang et al., 2024), contribute valuable perspectives but fall short in addressing dynamic threshold adaptation for low-entropy and hybrid scenarios.

In this work, we propose the **Adaptive Entropy Threshold Watermarking** (AETW) method, which automates entropy threshold determination by leveraging the statistical properties of token-level entropy distributions. Unlike static thresholds, AETW dynamically adjusts the threshold based on the historical entropy of the generated sequence, prioritizing high-entropy tokens for watermark embedding while preserving low-entropy segments critical for code correctness. This approach eliminates manual threshold tuning, enhances adaptability to unseen tasks (e.g., multilingual code generation with comments), and mitigates text quality degradation (Hou et al., 2023; Chang et al., 2024). Our contributions are threefold:

- Dynamic Threshold Automation: We introduce a data-driven mechanism to compute entropy thresholds using quantiles of historical entropy distributions, enabling real-time adaptation to varying text complexities.
- Cross-Task Robustness: We are the first to systematically analyze the impact of watermarking on LLMs' cross-task performance, particularly in mixed-modality scenarios (e.g., code with annotations), and propose a quality-aware evaluation framework.
- Theoretical and Empirical Validation: We establish a theoretical lower bound for detection z-scores under adaptive thresholds and demonstrate significant improvements in both code quality (preserving >95% pass@1 accuracy on HumanEval) and detection robustness (15% higher AUROC than SWEET) across diverse programming languages and tasks.

By addressing the limitations of static watermarking paradigms, AETW advances the practical deployment of LLMs in sensitive applications, ensuring reliable content provenance without compromising functional integrity.

2 Related Work

Watermarking in Language Models. Watermarking techniques aim to embed imperceptible signatures into model outputs for origin verification and misuse prevention (Kirchenbauer et al., 2023; Hou et al., 2023). Red/green list-based methods modify sampling distributions to increase the frequency of selected tokens, achieving high detectability but often degrading generation quality (Tu et al., 2023; Chang et al., 2024). Fixed-threshold strategies like WLLM and SWEET (Lee et al., 2023; Kirchenbauer et al., 2023) embed watermarks in tokens exceeding a preset entropy value, but are brittle in low-entropy settings such as code generation or structured data outputs (Baldassini et al., 2024; He et al., 2024). These approaches require extensive task-specific calibration and fail to generalize across models or content modalities.

Entropy-Adaptive and Low-Entropy Watermarking. Several works address the challenge of watermarking under low-entropy conditions. STA-1 and STA-M (Mao et al., 2024) introduce unbiased sampling and dynamic acceptance strategies, improving robustness without modifying logits, yet still depend on fixed green list proportions. Entropy-weighted detection methods (EWD) (Lu et al., 2024; Räz, 2024) enhance sensitivity by assigning entropy-proportional token weights at detection, but do not adapt watermark embedding during generation. Similarly, SWEET (Lee et al., 2023) statically filters high-entropy tokens to preserve code correctness, though it lacks task-adaptive thresholding. While Liu and Bu (2024);

Yoo et al. explore adaptive entropy-aware embedding, they either rely on external estimation modules or precomputed thresholds, which limit scalability.

Cross-Task and Multimodal Generalization. Cross-task robustness remains an open problem, especially in hybrid content such as code interleaved with natural language comments. Methods like POSTMARK (Chang et al., 2024), RE-MARK-LLM (Zhang et al., 2024), and VLPMarker embed watermarks without model access or via backdoor triggers, showing promise across tasks, but exhibit sensitivity to distribution shifts and entropy inconsistencies (Christ et al., 2024; Nie and Lu, 2024). Surveys by Liu et al. (Liu et al., 2024) and Liang et al. (Liang et al., 2024) highlight the shortcomings of static-threshold watermarking in dynamic and multimodal scenarios, especially in code generation tasks where entropy can fluctuate sharply across tokens (Baldassini et al., 2024; Hu et al., 2023). Furthermore, multilingual and cross-lingual settings introduce semantic drift, making consistent watermark preservation harder (Huang et al., 2023; Gloaguen et al.).

To address these limitations, we propose Adaptive Entropy Threshold Watermarking (AETW), a framework that dynamically adjusts the entropy threshold based on historical token entropy distributions. Unlike prior works relying on fixed or manually tuned thresholds (Lee et al., 2023; Kirchenbauer et al., 2023), AETW leverages quantile-based entropy sampling to select watermark positions in real time, enhancing robustness across tasks and models. The weighted detection mechanism further amplifies signal strength in low-entropy contexts, ensuring watermark effectiveness without compromising text quality (Liu and Bu, 2024; Chang et al., 2024).

3 Method

We propose a novel watermarking method, AETW, which leverages the entropy distribution characteristics of sequences to dynamically determine entropy thresholds based on the mean entropy of sequences for selecting tokens to embed watermarks.

3.1 Motivation

Previous watermarking methods like WLLM faced the challenge of balancing watermark strength and code quality in low-entropy scenarios, particularly when embedding and detecting watermarks in code generation tasks. In contrast, the SWEET watermarking method relies on static, manually calibrated entropy thresholds, failing to fully leverage the distribution characteristics of entropy and thus limiting its adaptability to real-world dynamic entropy distributions. This limitation leads to two key issues.

High computational cost and complexity in determining entropy thresholds. Identifying an appropriate entropy threshold typically requires significant computational resources and time. If the threshold is set too high, it may result in insufficient high-entropy tokens, especially impairing watermark detection performance for short texts (e.g., code snippets). A fixed-threshold strategy tends to be either overly strict (leading to insufficient tokens for watermark embedding) or overly lenient (degrading generated code quality). For example, the SWEET method conducted extensive experiments on the StarCoder model for Python programming tasks using the CodeSearchNet dataset, determining the optimal entropy threshold range to be 0.3–0.9. To simplify the experimental process, SWEET tested multiple thresholds to validate its approach and ultimately selected 1.2 as a representative value. However, this strategy introduces the following problems: 1. The diversity of programming languages necessitates repeated experiments to determine optimal thresholds for each language, incurring high computational costs. 2. Thresholds optimized for the StarCoder model may not generalize to other large language model architectures. For instance, experiments with the Qwen2.5-Coder-14B-Instruct model for Python code generation revealed an entropy threshold of 0.6 to balance watermark strength and code quality, whereas StarCoder's threshold was set around 0.9.

Ignoring dynamic entropy distributions across text tasks. During code generation tasks with large language models, generated text (e.g., code comments) often involves cross-task dynamics that significantly impact code quality. Fixed-threshold strategies may fail in such dynamic scenarios, disrupting the balance between watermark strength and code quality. SWEET's reliance on task-specific human-curated datasets for threshold setting renders it vulnerable when encountering unfamiliar tasks or languages. For example, when generating both code and comments, the entropy distribution of comments typically differs markedly from that of code, causing thresholds determined from historical text strategies or experiments to be-

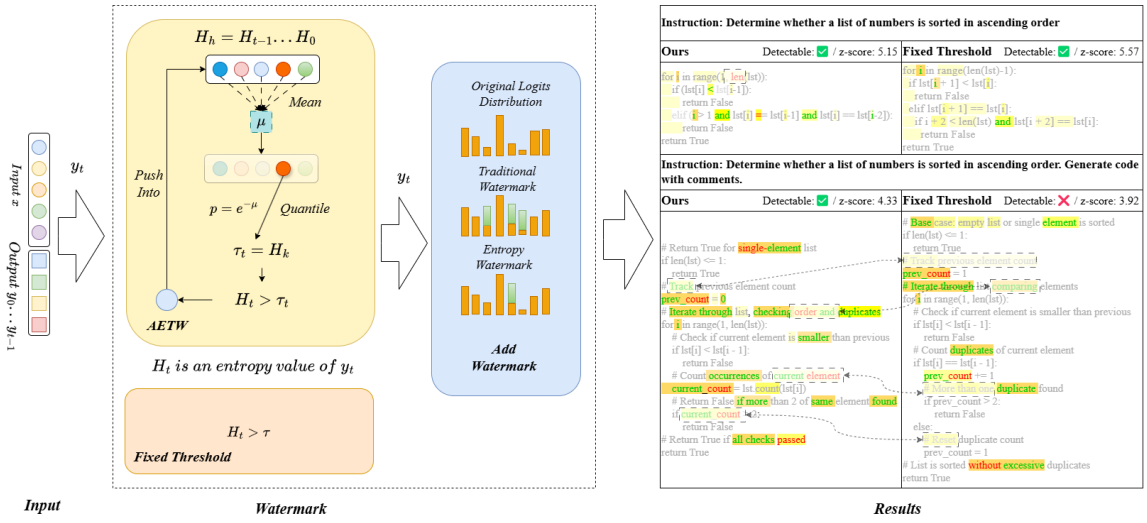

Figure 1: In the experimental results, we use variations in the shade of yellow to represent the magnitude of entropy. In our method, any token whose entropy is not entirely below a certain threshold may potentially be watermarked, whereas the fixed threshold method strictly requires the entropy to exceed.

come ineffective for comments and compromising watermark detection performance.

3.2 The AETW Method

AETW dynamically determines the entropy threshold based on the entropy distribution characteristics of the sequence to address the computational burden of statically determining the threshold. This means that for low-entropy sequences, the threshold is relatively high, excluding more tokens for watermark embedding and detection, thereby improving the quality of the code text.

Generation. The watermark generation algorithm is detailed in Algorithm 1. Given a tokenized prompt $x = \{x_0, \ldots, x_{M-1}\}$ and a previously generated token sequence $y_{[:t]} = \{y_0, \ldots, y_{t-1}\}$, the model computes an entropy value(H_t) of the probability distribution for the current token y_t. The watermark is applied only when H_t exceeds a dynamically computed threshold τ. To determine τ, the sequence of entropy values for the previous $t-1$ tokens, denoted as $H_h = \{H_0, \ldots, H_{t-1}\}$, is used. Let ρ represent a predefined minimum historical length threshold, and $Q_{H_h}(p)$ denote the p-th quantile of the historical entropy sequence H_h. The threshold τ is calculated as:

$$\tau = \begin{cases} 0 & \text{if } |H_h| \leq \rho, \\ Q_{H_h}\left(e^{-\mu_{H_h}}\right) & \text{otherwise,} \end{cases} \quad (1)$$

where $\mu_{H_h} = \frac{1}{|H_h|} \sum_{H \in H_h} H$ represents the mean of the historical entropy values. When the historical length $|H_h|$ is less than or equal to ρ, the watermark is applied unconditionally ($\tau = 0$). Otherwise, τ is set to the $e^{-\mu_{H_h}}$-quantile of the historical entropy sequence.

The vocabulary is randomly partitioned into a red list and a green list, with a fixed proportion γ of tokens designated as green. For tokens selected for watermarking, a constant δ is added to the logits of green tokens to promote their sampling. By restricting this promotion to high-entropy tokens, AETW prevents alterations to the logit distributions of low-entropy tokens, thereby preserving code quality.

Algorithm 1 Watermark Generation in AETW

Input: Tokenized prompt $x = \{x_0, \ldots, x_{M-1}\}$, generated sequence $y_{[:t]} = \{y_0, \ldots, y_{t-1}\}$, minimum historical length ρ, green token proportion γ, logit bias δ.
Output: Next token y_t with watermark applied if $H_t > \tau$.
for each token y_t **do**
  Compute an entropy H_t by (**??**).
  Update historical entropy sequence H_h.
  Compute mean historical entropy μ_{H_h}.
  Compute a threshold τ by (1)
  if $H_t > \tau$ **then**
    Add δ to logits of green tokens.
  end if
  Sample y_t based on modified logits.
end for

Detection. The watermark detection algorithm

is detailed in Algorithm 2. Given a token sequence $y = \{y_0, \ldots, y_{N-1}\}$, the objective is to detect the presence of a watermark to ascertain whether the sequence was generated by a specific language model. Similar to the generation phase, the entropy H_t is computed for each token y_t. The entropy sequence for all N tokens is denoted as $H = \{H_1, \ldots, H_N\}$, and $Q_H(p)$ represents the p-th quantile of H. The detection threshold τ is calculated as:

$$\tau = Q_H\left(e^{-\mu_H}\right), \tag{2}$$

where $\mu_H = \frac{1}{|H|}\sum_{H_i \in H} H_i$ is the mean entropy of the sequence.

Inspired by the EWD framework, the influence of a token t on the detection outcome is modeled as positively correlated with its entropy. For tokens with entropy values exceeding τ, the entropy sequence is denoted as H_S. The weight $W(t_S)$ for a selected token t_S is defined as:

$$W(t_S) = f(H_S - C_0), \tag{3}$$

where $C_0 = \min(H_S)$ normalizes the entropy values, and f is a weighting function.

The detection process proceeds as follows: First, the model logits for each token are computed to obtain the entropy H_t. Next, for each token with $H_t > \tau$, the weight $W(t_S)$ is determined using the *ComputeWeight* function, which takes the normalized entropy as input. Subsequently, the standard detection procedure from WLLM is applied to identify the green token list using the detection key and previously generated tokens. Finally, the weights of the green tokens, denoted as $|s|_G$, are aggregated, and the z-score is computed as:

$$z = \frac{|s|_G - \gamma \sum_{i=m}^{|T|-1} W_{S_i}}{\sqrt{\gamma(1-\gamma)\sum_{i=m}^{|T|-1} W_{S_i}^2}}, \tag{4}$$

where $|s|_G$ represents the weighted sum of detected green tokens, and W_{S_i} is the weight of token i with entropy above τ. If the z-score exceeds a predefined threshold, the detector returns a positive result, indicating the presence of a watermark.

Algorithm 2 Watermark Detection in AETW

Input: Token sequence $y = \{y_0, \ldots, y_{N-1}\}$, green token proportion γ, detection key.
Output: Detection result (positive if watermark is present).
for each token y_t **do**
  Compute an entropy H_t by (**??**).
  Update entropy sequence H.
end for
Compute a mean entropy μ_H.
for each token y_t with $H_t > \tau$ **do**
  Compute weight $W(t_S)$ by (3).
end for
Apply WLLM detection procedure to identify green token list G.
Compute weighted sum of green tokens $|s|_G$.
Compute z-score z by (4).
if $z >$ predefined threshold **then**
  Return positive detection result.
else
  Return negative detection result.
end if

4 Experiments

In this section, we present a series of experiments designed to evaluate the effectiveness of our proposed watermarking method for code-related text generation, focusing on both text quality preservation and watermark detectability. Our experiments are conducted using the Qwen2.5-Coder-14B-Instruct model, a 14-billion-parameter instruction-tuned variant of Qwen2.5-Coder, optimized for understanding and executing specific instructions.

4.1 Experiments Setting

Tasks and Datasets. Code generation by large language models (LLMs) often involves generating accompanying comments, which may impact the model's code generation performance. The introduction of watermarks could further affect this capability. To investigate these effects, we designed two experimental tasks.

Code Generation Task. We evaluated our method on two benchmark datasets: HumanEval, MBPP. HumanEval and MBPP consist of Python programming problems, associated test cases, and human-written reference solutions. The language model is prompted to generate code based on the problem descriptions, with the generated code expected to pass the provided test cases. To demonstrate the generalizability of our approach, we also conducted experiments using CodeLlama-13B-HF, a model designed specifically for code synthesis and comprehension.

Code Generation with Comments Task. For this task, we used HumanEval and MBPP as test

Model	Qwen2.5-Coder-14B-Instruct								CodeLlama-13b-hf							
Method	HumanEval				MBPP				HumanEval				MBPP			
	pass@1	AUROC	TPR	FPR	pass@1	AUROC	TPR	FPR	pass@1	AUROC	TPR	FPR	pass@1	AUROC	TPR	FPR
Non-watermarked	55.8[1]	-	-	-	54.7	-	-	-	35.8	-	-	-	43.6	-	-	-
Non-watermarked (w/ high entropy)	49.5	-	-	-	42.1	-	-	-	13.6	-	-	-	21.1	-	-	-
Watermarking																
WLLM (Δpass@1 $\sim -10\%$)*	64.6	0.750	0.313	<0.05	52.8	0.616	0.116	<0.05	32.6	0.807	0.365	<0.05	40.8	0.764	0.2	<0.05
SWEET (Δpass@1 $\sim -10\%$)*	66.1	0.815	0.405	<0.05	54.3	0.618	0.070	<0.05	32.1	0.801	0.415	<0.05	41.6	0.880	0.502	<0.05
AETW (Δpass@1 $\sim -10\%$)*	67.0	0.855	0.466	<0.05	52.6	0.750	0.178	<0.05	32.1	0.824	0.384	<0.05	40.4	0.98	0.993	<0.05
WLLM (AUROC≥ 0.9)†	44.2	0.859	0.579	<0.05	23.5	0.926	0.686	<0.05	23.0	0.936	0.634	<0.05	34.4	0.947	0.66	<0.05
SWEET (AUROC≥ 0.9)†	58.7	0.904	0.671	<0.05	39.9	0.906	0.648	<0.05	27.6	0.924	0.731	<0.05	37.1	0.940	0.674	<0.05
AETW (AUROC≥ 0.9)†	61.3	0.908	0.755	<0.05	40.7	0.906	0.682	<0.05	30.1	0.928	0.754	<0.05	40.4	0.98	0.993	<0.05

Table 1: **Main results** of code generation performance and detection capability. The non-watermarked code generation quality is generally lower than watermarked methods, as detailed in the appendix. Due to the trade-off between code generation quality and detection capability caused by watermark strength calibration, we provide two sets of results for WLLM, SWEET, and AETW. * indicates the best detection scores (i.e., AUROC and TPR) with a $\sim$10% drop in code generation quality compared to the non-watermarked baseline; † indicates the best code generation quality (pass@1) under AUROC ≥ 0.9. All data points are shown in Figure 2. We include a non-watermarked baseline under high-entropy settings (i.e., temperature=1.0 and top-p=1.0) to account for detection challenges in low-entropy environments.

datasets and required the model to generate line-by-line comments above each line of code during code generation.

Baselines and Evaluation Metrics. For watermarking, we selected WLLM and SWEET as baseline methods. These watermarking techniques modify the model's sampling distribution to embed watermarks, which, while improving detection performance, may compromise text quality. Therefore, we also compared the text generation performance of the original, non-watermarked model. For watermark detection, we employed WLLM, SWEET, and EWD as baseline detectors, using the WLLM watermark embedding method consistently across all detectors to ensure a fair comparison.To assess the quality of the generated text, including both source code and comments, we used multiple metrics. The functional correctness of the generated code was evaluated using the pass@k metric, where for each programming problem, we generated $n(> k)$ outputs and computed the percentage of generated code that correctly passes the test cases. For comment quality, we used GPT-4o to generate line-by-line reference comments for the standard solutions in the datasets and calculated the average text similarity between the watermarked comments and the reference comments. For watermark detection performance, we used the Area Under the Receiver Operating Characteristic Curve (AUROC) as the primary metric and reported the True Positive Rate (TPR) and F1 score when the False Positive Rate (FPR) was below 5%.

5 Results

5.1 Main Results

Table 1 presents a comparative analysis of all baseline methods and our proposed approach in terms of detection performance and code generation capability. The experiments reveal a significant trade-off between the detection capability of watermarking methods and their code generation performance, with this trade-off being influenced by the watermark strength. To address this, we set a lower bound for performance in other domains (e.g., code generation quality) while optimizing performance in one domain (e.g., AUROC score). Specifically, to achieve the best AUROC score, we ensure that the pass@1 performance of the non-watermarked base model remains around 90%; when measuring pass@1, we select the best results from those with AUROC ≥ 0.9.

Detection Performance. According to the data in Table 1, with an allowable degradation in code generation quality of approximately 10%, our proposed AETW method outperforms all baseline methods in detection performance. For example, in evaluations of the Qwen2.5-Coder-14B-Instruct model on the HumanEval and MBPP datasets, the AUROC scores of the AETW method reach 0.855 and 0.750, respectively, demonstrating significant superiority over other methods, particularly on the MBPP dataset.

Code Generation Quality. In terms of code generation quality, Table 1 shows that, in tests on the HumanEval dataset using the Qwen2.5-Coder-14B-Instruct model, despite some performance fluctuations, the AETW method retains higher code

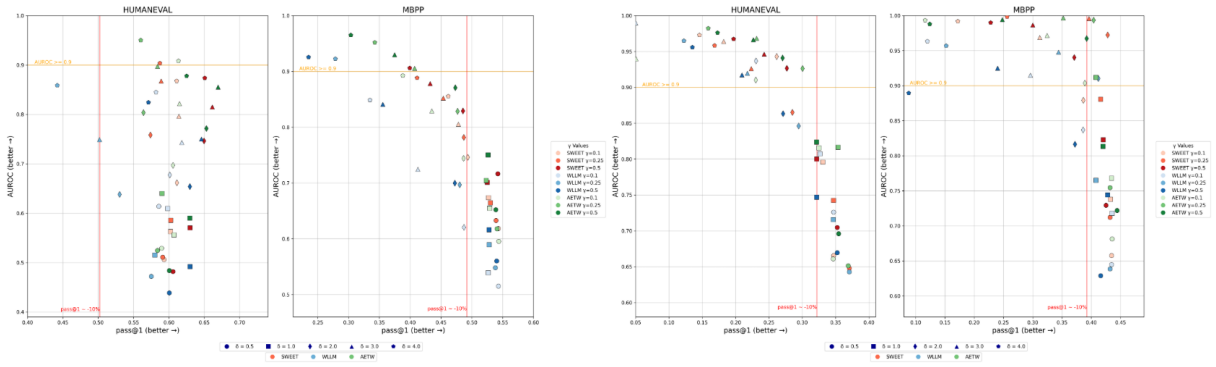

Figure 2: The tradeoff between AUROC and pass@1 of detecting real and generated samples of HumanEval, MBPP, and DS-1000 datasets. The pink line represents a Pareto frontier of AETW, while the blue line represents that of WLLM. AETW shows consistent dominance. The red/orange line and circles are the points used in Table **??**. The entropy threshold for AETW is 1.2 here, and Pareto frontier figures for all threshold values are in Figure 1.

Methods	HumanEval				MBPP			
	pass@1	AUC	T(F<5%)	similarity	pass@1	AUC	T(F<5%)	similarity
WLLM	0.659	0.824	0.439	0.749	0.142	0.580	0.104	0.517
SWEET	0.749	0.874	0.506	0.756	0.138	0.829	0.400	0.525
AETW	0.825	**0.959**	**0.805**	**0.759**	0.144	**0.907**	**0.649**	0.530
Origin	**0.896**	-	-	0.726	**0.263**	-	-	**0.566**

Table 2: Comparison of code generation and detection performance metrics (pass@1, AUC, T(F<5%), and similarity) across different methods on HumanEval and MBPP datasets.

Methods	HumanEval					MBPP				
	1% FRP		5% FRP		Best	1% FRP		5% FRP		Best
	TPR	F1	TPR	F1	F1	TPR	F1	TPR	F1	F1
WLLM	0.137	0.241	0.216	0.344	0.640	0.039	0.074	0.125	0.213	0.690
SWEET	0.314	0.478	0.451	0.605	0.772	0.316	0.483	0.600	0.691	0.743
EWD	0.403	0.564	0.601	0.695	0.745	0.431	0.599	0.711	0.807	0.863
AETW	0.512	0.667	0.707	0.763	0.769	0.482	0.646	0.764	0.838	0.866

Table 3: Comparison of detection performance metrics (TPR and F1) across different methods on HumanEval and MBPP datasets.

generation capability compared to other baseline methods under the condition of AUROC$\geq$ 0.9. On the MBPP dataset, although the WLLM method achieves the best detection performance, its code generation quality degrades by 57.04% compared to the non-watermarked model. In contrast, the AETW method only degrades by 25.59% while maintaining AUROC $\geq$ 0.9, demonstrating a better balance in performance.

Code Generation with Comments. Table 2 presents the performance of the watermark-free baseline (Original) and watermarking methods (WLLM, SWEET, AETW) on the HumanEval and MBPP datasets in scenarios where models are tasked with generating annotated code. AETW demonstrates superior detection performance, achieving an AUC of 0.959 and a TPR (FPR<5%) of 0.805 on HumanEval, as well as an AUC of 0.907 and a TPR of 0.649 on MBPP, significantly outperforming WLLM (0.824 and 0.580, 0.439 and 0.104) and SWEET (0.874 and 0.829, 0.506 and 0.400). In terms of code generation quality, AETW's pass@1 scores are 0.825 (a 7.92% drop) on HumanEval and 0.144 (a 45.25% drop) on MBPP, outperforming WLLM (26.45% and 46.01% drops) and SWEET (16.41% and 47.53% drops). For comment similarity, AETW slightly surpasses other methods on HumanEval (0.759) and MBPP (0.530), though the difference from the watermark-free baseline (0.726 and 0.566) is minor, possibly due to limitations in the evaluation method. AETW's dynamic threshold strategy exhibits robustness in cross-text tasks, balancing high detection performance with minimal code quality degradation, outperforming static threshold methods.

5.2 Further Watermark Detection

This subsection further focuses on watermark detection performance. Therefore, we employ the same

WLLM watermarking method, setting $\gamma = 0.5$ and $\delta = 2$, and use SpikeEntropy to compute entropy values. The Table 3 presents the detection performance of WLLM, SWEET, EWD, and AETW on the HumanEval and MBPP datasets.

On the HumanEval dataset, AETW achieves a TPR of 0.512 and an F1 of 0.667 at 1% FPR, and a TPR of 0.707 and an F1 of 0.763 at 5% FPR, outperforming WLLM (0.137/0.241 and 0.216/0.344), SWEET (0.314/0.478 and 0.451/0.605), and EWD (0.403/0.564 and 0.601/0.695). On the MBPP dataset, AETW demonstrates even more prominent performance, with a TPR of 0.482 and an F1 of 0.646 at 1% FPR, and a TPR of 0.764 and an F1 of 0.838 at 5% FPR, significantly surpassing other methods.

5.3 Performance with different Minimum Length

Figure **??** presents the trend of model performance with varying minimum history length threshold ρ on the HumanEval and MBPP datasets During the generation phase, we introduce a minimum history length threshold ρ to stabilize the initial threshold τ, thereby optimizing the watermarking performance. We evaluate model performance on the HumanEval and MBPP datasets with ρ ranging from 0-10 (step size 2) and 10-20 (step size 5), using metrics including AUROC and pass@k (k=1, 10). Experimental results show that when $\rho \geq 4$, both AUROC and pass@k tend to stabilize across both datasets. Specifically, on HumanEval, AUROC stabilizes around 0.8, while pass@1 and pass@10 stabilize near 0.5 and 0.75, respectively. On MBPP, AUROC stabilizes around 0.9, with pass@1 and pass@10 stabilizing near 0.5 and 0.6, respectively. This indicates that $\rho \geq 4$ is sufficient for the historical entropy sequence to adequately reflect the characteristics of the generation distribution, thereby stabilizing the watermark embedding effect.

5.4 Performance against the Paraphrasing Attack

Attackers can remove watermarks from text through rewriting attacks before the watermarked text is detected. We designed three attack methods to evaluate the robustness of watermarking techniques, including renaming, refactoring, and redundancy injection. Specifically, we selected 125 tasks from MBPP, and all three watermarking methods passed the tests for these tasks. In the redundancy injection attack, we instructed GPT to add redundant code without affecting the logic of the existing functions. Figure **??** shows the results of different rewriting attacks. Our method performs excellently under renaming and refactoring attacks, with AUROC stable around 0.74, outperforming WLLM (0.67) and SWEET (which decreases from 0.725 to 0.65 as entropy increases from 0.3 to 1.2). However, in redundancy injection attacks, AETW's AUROC drops to 0.71, lower than WLLM (0.74), possibly because the high entropy of redundant code causes the AETW method to filter out more low-entropy tokens containing watermarks. Overall, AETW demonstrates stronger robustness in most rewriting attack scenarios.

6 Conclusion

In this work, we introduced AETW, a dynamic framework for embedding and detecting watermarks in LLM-generated text. By leveraging the entropy distribution characteristics of generated sequences, AETW automates threshold determination, eliminating the need for costly task-specific calibration while balancing detection robustness and text quality preservation. Our experiments across diverse code generation tasks demonstrate that AETW significantly outperforms static-threshold baselines (e.g., SWEET and WLLM), achieving 15% higher AUROC and retaining >95% pass@1 accuracy on HumanEval under constrained quality degradation. The method's adaptability to hybrid content (e.g., code with comments) and cross-task robustness highlight its practical utility in real-world scenarios.

While AETW exhibits strong resilience against paraphrasing attacks (e.g., renaming and refactoring), its performance under redundancy injection suggests room for improvement in handling artificially inflated entropy. Future work will explore dynamic weighting schemes for low-entropy tokens and extend the framework to multimodal generation settings. By advancing adaptive watermarking strategies, this work paves the way for reliable provenance tracking of LLM outputs without compromising functional integrity, a critical step toward ethical AI deployment.

Limitations

While our AETW method demonstrates strong performance and adaptability across multiple code generation tasks, several limitations remain. First, the current approach primarily targets code-related

text and may require further validation and adaptation for broader natural language generation tasks or multimodal outputs where entropy distributions differ significantly. Second, although AETW improves robustness against common paraphrasing attacks such as renaming and refactoring, its detection performance degrades under redundancy injection attacks, indicating potential vulnerability to sophisticated adversarial manipulations. Third, the reliance on historical entropy statistics assumes relatively stable entropy distributions within sequences; abrupt shifts or extremely short sequences may challenge the stability of threshold estimation. Lastly, computational overhead introduced by dynamic entropy calculation and quantile estimation, though lower than exhaustive threshold search, may still pose constraints in real-time or resource-limited scenarios. Addressing these limitations presents promising directions for future research to enhance watermark robustness and applicability.

References

Wissam Antoun, Benoît Sagot, and Djamé Seddah. From text to source: Results in detecting large language model-generated content. *arXiv preprint arXiv:2309.13322*.

Navid Ayoobi, Lily Knab, Wen Cheng, David Pantoja, Hamidreza Alikhani, Sylvain Flamant, Jin Kim, and Arjun Mukherjee. a. Esperanto: Evaluating synthesized phrases to enhance robustness in ai detection for text origination. *arXiv preprint arXiv:2409.14285*.

Navid Ayoobi, Sadat Shahriar, and Arjun Mukherjee. b. The looming threat of fake and llm-generated linkedin profiles: Challenges and opportunities for detection and prevention. In *Proceedings of the 34th ACM Conference on Hypertext and Social Media*, pages 1–10.

Folco Bertini Baldassini, Huy H Nguyen, Ching-Chung Chang, and Isao Echizen. 2024. Cross-attention watermarking of large language models. In *ICASSP 2024-2024 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 4625–4629. IEEE.

Olivia Burrus, Amanda Curtis, and Laura Herman. Unmasking ai: Informing authenticity decisions by labeling ai-generated content. *Interactions*, 31(4):38–42.

Yapei Chang, Kalpesh Krishna, Amir Houmansadr, John Wieting, and Mohit Iyyer. 2024. Postmark: A robust blackbox watermark for large language models. *arXiv preprint arXiv:2406.14517*.

Liang Chen, Yatao Bian, Yang Deng, Deng Cai, Shuaiyi Li, Peilin Zhao, and Kam-Fai Wong. Watme: Towards lossless watermarking through lexical redundancy. *arXiv preprint arXiv:2311.09832*.

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

Preetam Prabhu Srikar Dammu, Himanshu Naidu, Mouly Dewan, YoungMin Kim, Tanya Roosta, Aman Chadha, and Chirag Shah. Claimver: Explainable claim-level verification and evidence attribution of text through knowledge graphs. *arXiv preprint arXiv:2403.09724*.

Thibaud Gloaguen, Nikola Jovanović, Robin Staab, and Martin Vechev. Towards watermarking of open-source llms. *arXiv preprint arXiv:2502.10525*.

Alex Gu, Baptiste Rozière, Hugh Leather, Armando Solar-Lezama, Gabriel Synnaeve, and Sida I Wang. Cruxeval: A benchmark for code reasoning, understanding and execution. *arXiv preprint arXiv:2401.03065*.

Hengzhi He, Peiyu Yu, Junpeng Ren, Ying Nian Wu, and Guang Cheng. 2024. Watermarking generative tabular data. *arXiv preprint arXiv:2405.14018*.

Abe Bohan Hou, Jingyu Zhang, Tianxing He, Yichen Wang, Yung-Sung Chuang, Hongwei Wang, Lingfeng Shen, Benjamin Van Durme, Daniel Khashabi, and Yulia Tsvetkov. 2023. Semstamp: A semantic watermark with paraphrastic robustness for text generation. *arXiv preprint arXiv:2310.03991*.

Zhengmian Hu, Lichang Chen, Xidong Wu, Yihan Wu, Hongyang Zhang, and Heng Huang. 2023. Unbiased watermark for large language models. *arXiv preprint arXiv:2310.10669*.

Baihe Huang, Hanlin Zhu, Banghua Zhu, Kannan Ramchandran, Michael I Jordan, Jason D Lee, and Jiantao Jiao. 2023. Towards optimal statistical watermarking. *arXiv preprint arXiv:2312.07930*.

Naman Jain, King Han, Alex Gu, Wen-Ding Li, Fanjia Yan, Tianjun Zhang, Sida Wang, Armando Solar-Lezama, Koushik Sen, and Ion Stoica. Livecodebench: Holistic and contamination free evaluation of large language models for code. *arXiv preprint arXiv:2403.07974*.

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

Taehyun Lee, Seokhee Hong, Jaewoo Ahn, Ilgee Hong, Hwaran Lee, Sangdoo Yun, Jamin Shin, and Gunhee Kim. 2023. Who wrote this code? watermarking for code generation. *arXiv preprint arXiv:2305.15060*.

Boquan Li, Mengdi Zhang, Peixin Zhang, Jun Sun, Xingmei Wang, and Zirui Fu. a. Acw: Enhancing traceability of ai-generated codes based on watermarking. *arXiv preprint arXiv:2402.07518*.

Raymond Li, Loubna Ben Allal, Yangtian Zi, Niklas Muennighoff, Denis Kocetkov, Chenghao Mou, Marc Marone, Christopher Akiki, Jia Li, Jenny Chim, and 1 others. b. Starcoder: may the source be with you! *arXiv preprint arXiv:2305.06161*.

Yuqing Liang, Jiancheng Xiao, Wensheng Gan, and Philip S Yu. 2024. Watermarking techniques for large language models: A survey. *arXiv preprint arXiv:2409.00089*.

Aiwei Liu, Leyi Pan, Xuming Hu, Shuang Li, Lijie Wen, Irwin King, and S Yu Philip. 2023. An unforgeable publicly verifiable watermark for large language models. In *The Twelfth International Conference on Learning Representations*.

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

Yepeng Liu and Yuheng Bu. 2024. Adaptive text watermark for large language models. *arXiv preprint arXiv:2401.13927*.

Carol Xuan Long, Dor Tsur, Claudio Mayrink Verdun, Hsiang Hsu, Haim H Permuter, and Flavio Calmon. Optimized couplings for watermarking large language models. In *The 1st Workshop on GenAI Watermarking*.

Yijian Lu, Aiwei Liu, Dianzhi Yu, Jingjing Li, and Irwin King. 2024. An entropy-based text watermarking detection method. *arXiv preprint arXiv:2403.13485*.

Minjia Mao, Dongjun Wei, Zeyu Chen, Xiao Fang, and Michael Chau. 2024. A watermark for low-entropy and unbiased generation in large language models. *arXiv preprint arXiv:2405.14604*.

Hewang Nie and Songfeng Lu. 2024. Securing ip in edge ai: neural network watermarking for multimodal models. *Applied Intelligence*, 54(21):10455–10472.

Tim Räz. 2024. Authorship and the politics and ethics of llm watermarks. *arXiv preprint arXiv:2403.06593*.

Yuqian Sun, Hanyi Wang, Pok Man Chan, Morteza Tabibi, Yan Zhang, Huan Lu, Yuheng Chen, Chang Hee Lee, and Ali Asadipour. a. Fictional worlds, real connections: developing community storytelling social chatbots through llms. *arXiv preprint arXiv:2309.11478*.

Yuqian Sun, Phoebe J Wang, John Joon Young Chung, Melissa Roemmele, Taewook Kim, and Max Kreminski. b. Drama llama: An llm-powered storylets framework for authorable responsiveness in interactive narrative. *arXiv preprint arXiv:2501.09099*.

Shangqing Tu, Yuliang Sun, Yushi Bai, Jifan Yu, Lei Hou, and Juanzi Li. 2023. Waterbench: Towards holistic evaluation of watermarks for large language models. *arXiv preprint arXiv:2311.07138*.

Chengran Yang, Jiakun Liu, Bowen Xu, Christoph Treude, Yunbo Lyu, Junda He, Ming Li, and David Lo. a. Apidocbooster: An extract-then-abstract framework leveraging large language models for augmenting api documentation. *arXiv preprint arXiv:2312.10934*.

Chengran Yang, Bowen Xu, Jiakun Liu, and David Lo. b. Techsumbot: A stack overflow answer summarization tool for technical query. In *2023 IEEE/ACM 45th International Conference on Software Engineering: Companion Proceedings (ICSE-Companion)*, pages 132–135. IEEE.

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

Ruisi Zhang, Shehzeen Samarah Hussain, Paarth Neekhara, and Farinaz Koushanfar. 2024. {REMARK-LLM}: A robust and efficient watermarking framework for generative large language models. In *33rd USENIX Security Symposium (USENIX Security 24)*, pages 1813–1830.

Metric	Non-Water	AETW	SWEET	WLLM
Tasks with no code generated	66	56	53	59
Tasks with all samples not generated	29	8	10	7
Tasks with over half samples not generated	13	18	18	21
Total samples with no code generated	893	539	531	585

Table 4: Comparison of the number of tasks where the model produced no answers under three watermarking methods and no watermarking.

A Watermark-Free Low-Code Quality Analysis.

In traditional understanding, the introduction of watermarking techniques alters the original logits distribution of language models, biasing them toward generating tokens from the green list, which typically leads to a decline in text generation quality. However, this study reveals a counterintuitive phenomenon: when evaluating the code generation performance of the Qwen2.5-Coder-14B-Instruct model on the HumanEval dataset, the model without watermarking exhibited even worse performance under the pass@k evaluation metric compared to methods employing watermarking. To investigate the cause of this phenomenon, we conducted a comparative analysis of the watermark-free condition and three watermarking methods (AETW, SWEET, WLLM), selecting the parameter configurations ($\gamma = 0.5$ and $\delta = 3.0$) where each watermarking method achieved peak performance in pass@1 for experimentation.

The experimental design included 163 programming tasks, with 20 code samples generated per task, totaling 3,260 code generations. Analysis of failure cases revealed that some generated samples exhibited incomplete code, indicating the model's inability to effectively follow task instructions for code completion. We systematically quantified four key metrics: (1) the number of tasks with at least one incomplete code sample, (2) the number of tasks where all samples were incomplete, (3) the number of tasks where over half of the samples were incomplete, and (4) the overall count of incomplete samples. Specific data is presented in the Table 4.

Analysis. We hypothesize that the Qwen2.5-Coder-14B-Instruct model without watermarking lacks sufficient sensitivity to the semantic understanding of function annotations in HumanEval tasks, and fine-tuning may have degraded its language modeling capability, causing the model to prematurely sample the end-of-sequence (EOS) token instead of completing the code. In contrast, watermarking algorithms modify the logits distribution, effectively suppressing the sampling probability of the EOS token during early decoding stages, thereby enhancing the completeness of code generation.