SSG: Logit-Balanced Vocabulary Partitioning for LLM Watermarking

Chenxi Gu, **Xiaoning Du***, **John Grundy**

Monash University

{Chenxi.Gu, Xiaoning.Du, John.Grundy}@monash.edu

Abstract

Watermarking has emerged as a promising technique for tracing the authorship of content generated by large language models (LLMs). Among existing approaches, the KGW scheme is particularly attractive due to its simplicity, versatility, and low computational overhead. However, its effectiveness degrades significantly in low-entropy settings such as code generation and mathematical reasoning—both of which are common and practically important tasks. To explain this weakness, we introduce a novel token-level metric, Watermark Strength, which provides fine-grained insights into the behavior of KGW-scheme. Through this analysis, we identify the core limitation: the randomness of the arbitrary vocabulary partition, which can yield minimal Watermark Strength and thus ineffective watermark injection. To address this issue, we propose SSG (**S**ort-then-**S**plit by **G**roups), a method that partitions the vocabulary into two logit-balanced subsets. This design guarantees a lower bound on Watermark Strength, thereby improving the detectability of watermarks. Experiments on code generation and mathematic reasoning datasets demonstrate the effectiveness of SSG.

1 Introduction

Large language models (LLMs) can generate high-quality, human-like content, making them powerful tools for diverse generative tasks. Yet these capabilities also pose risks: LLM outputs may be misused to infringe on proprietary interests, circumvent usage policies, or spread harmful misinformation. For example, most proprietary providers such as OpenAI, Anthropic, and Google Gemini prohibit the use of their outputs for training other models (Houman, 2025; Google, 2025; Anthropic, 2024), but proving such violations is difficult when generated text cannot be reliably distinguished from human-written or non-watermarked model outputs. Similarly, malicious actors may employ LLMs to generate convincing fake news (Liu et al., 2024), prompting regulatory bodies in the US, EU, China, and G7 countries to call for robust mechanisms for content authentication and provenance (Union, 2024). These challenges highlight the urgent need for reliable watermarking techniques.

Among existing approaches, the KGW watermarking scheme has attracted attention for its simplicity, efficiency, and effectiveness in natural language generation (Kirchenbauer et al., 2024; Liu et al., 2024). KGW partitions the vocabulary into two subsets (green and red) based on a context-dependent hash, and biases the logits of the green subset to embed a hidden statistical signal. A standard statistical test, such as a z-score test, can then detect this signal by measuring the overrepresentation of green tokens.

However, KGW watermarking degrades in low-entropy settings such as code generation (Lee et al., 2024; Lu et al., 2024; Huang et al., 2025), where the next-token probability distribution is sharply peaked at few tokens. While recent works have proposed how to improve detection performance (Lee et al., 2024; Lu et al., 2024; Huang et al., 2025), these methods mainly focus only at the detection stage and leave the more fundamental weakness of the injection stage unresolved.

To better understand the limitation of KGW-scheme, we introduce a novel token-level metric, *Watermark Strength*, which quantifies the normalized probability shift toward the green set at each token. Analysis using this metric reveals that KGW can yield extremely weak watermark signals when random splits place most high logit tokens in one subset—a situation that arises non-negligibly in low-entropy tasks.

To overcome this, we propose **SSG (Sort-then-Split by Groups)**, a novel algorithm that focuses on how to avoid the case in KGW where all high-logit tokens fall into the same subset. SSG sorts

*Corresponding author.

tokens by their logits, groups them, and then applies a hash-based split such that the green and red sets are approximately logit-balanced. By doing so, we prove that, SSG guarantees a lower bound on Watermark Strength and thereby enhance the watermark detectability. Besides, to maintain the efficiency of SSG, we only conduct SSG on top-k logit tokens. Overall, with SSG, watermark detectability is strengthened even under low-entropy conditions without comprising text quality.

Our main contributions are as follows:

- We analyze the limitations of KGW watermarking in low-entropy settings and show why random vocabulary partitioning leads to weak watermark injection.
- We introduce *Watermark Strength*, a token-level metric that provides fine-grained insight into watermark injection.
- We propose **SSG**, a logit-balanced partitioning scheme that improves watermark detectability.
- We conduct comprehensive evaluations on code generation and mathematical reasoning datasets, demonstrating that SSG achieves superior detectability over baselines without degrading text quality.

2 Related Works

Large language models (LLMs) such as GPT (Radford and Narasimhan, 2018), LLaMA (Touvron et al., 2023), and GPT-4 (OpenAI et al., 2024) have advanced rapidly in recent years, achieving state-of-the-art performance across diverse tasks including machine translation (OpenAI et al., 2024), code generation (Li et al., 2023), and many others (Touvron et al., 2023). Alongside these successes, however, their widespread adoption raises concerns over potential misuse and intellectual property protection, like unauthorized dataset usage(Sun et al., 2023). To address these challenges, watermarking has emerged as a promising approach for attributing and detecting LLM-generated content.

Watermarking methods can broadly be categorized into three classes: logit-based (Kirchenbauer et al., 2024; Lee et al., 2024; Lu et al., 2024; Chen et al., 2024), sampling-based (Christ et al., 2023; Kuditipudi et al., 2024; Hou et al., 2024), and training-based (Sun et al., 2022, 2023; Gu et al., 2024). Among these, logit-based approaches are particularly attractive due to their simplicity, versatility, and low computational cost (Liu et al., 2024).

A representative example is KGW (Kirchenbauer et al., 2024), which partitions the vocabulary into "green" and "red" sets using a hash of the preceding context, and then adds a bias to the logits of green tokens. This encourages the model to generate more green tokens, enabling detection through a z-score statistical test.

While effective for high-entropy text, KGW performance degrades substantially in low-entropy settings such as code generation, where the next-token distribution is concentrated on a small set of candidates (Lee et al., 2024; Lu et al., 2024). In such cases, perturbing logits produces only a negligible distributional shift, weakening the watermark signal and reducing detection accuracy. To mitigate this, Lee et al. (2024) propose restricting watermark injection to high-entropy tokens, while Lu et al. (2024) introduce entropy-weighted scoring to prioritize high-entropy tokens during detection.

3 Preliminaries

Language Model: A language model $\mathcal{M}$ is fundamentally a function for prediction the next token in a sequence. $\mathcal{M}$ takes an input sequence $\boldsymbol{x} = [x_0, ..., x_{n-1}]$ and produces the logit L_n, which will be transformed into a probability distribution P_n over a vocabulary set $\mathcal{V}$. Formally, we have $P_n = softmax(L_n) = softmax(\mathcal{M}(\boldsymbol{x_{0:n-1}}))$. The next token y_n is selected based on P_n by sampling. This generation process is auto-regressive until a whole sequence $\boldsymbol{y}$ is finished.

Watermarking Algorithm aims to embed hidden pattern into LLMs' output to help people identify its authorship. It comprises two stages: Watermark Injection and Watermark Detection. During Watermark Injection, the probability distribution output will be altered from P_n to $\tilde{P}_n$, which is determined by a secret key s. As a result, the output sequence is changed to $\tilde{\boldsymbol{y}}$ instead of $\boldsymbol{y}$. At Watermark Detection stage, the detection algorithm takes a suspected sequence $\boldsymbol{x}^*$ as input, and determines if it is generated by $\mathcal{M}$.

KGW scheme Watermarking Algorithm: To embed watermark when predict y_n, KGW scheme watermarking methods first divide vocabulary $\mathcal{V}$ into green token set $\mathcal{V}^g$ with size of $\gamma * |\mathcal{V}|$ where γ is green set ratio, and red token set $\mathcal{V}^r$, via a hash function determined by secret key s and context $\boldsymbol{x}_{-h:}$ Then, the logits of token v_i in $\mathcal{V}^g$, denoted as

$L^g_{n,i}$, will be increased by logit bias term δ, to get watermarked logit $\tilde{L}_n = [L^g_n + \delta, L^r_n]$. At last, next token $\tilde{y}_n$ will be sampled according to $\tilde{P}_n$. During watermark detection stage, for a suspected text $\boldsymbol{x}$, a statistical test is performed. The null hypothesis $\mathcal{H}_0$ is :`The text is not watermarked`. If the z-score of statistical test is above a predefined threshold τ, $\mathcal{H}_0$ could be rejected and one can conclude that $\boldsymbol{x}$ is watermarked.

Low-entropy Generation Tasks: Low-entropy generation tasks refer to scenarios in which the language model's next-token distribution is highly concentrated on only a few candidates. Formally, if the next-token distribution is denoted as $P = \{p_i\}$, the entropy is given by $H(P) = -\sum_i p_i \log p_i$. A low-entropy setting arises when $H(P)$ is small, meaning that most of the probability is assigned to a small subset of tokens, while the rest receive negligible likelihood. Typical examples include code generation, mathematical reasoning, structured formats generation(e.g., JSON, SQL, or HTML).

4 Method

This section provides a detailed description of SSG. In Section 4.1, we first explain why random vocabulary partition, which is adopted in prior KGW-scheme, induces the ineffectiveness in low-entropy settings. Then the whole pipeline of SSG watermark injection stage is described in Section 4.2. Section 4.3 details the process of watermark detection. Finally, we analyze SSG in Section 4.4 to show why it could achieve more reliable watermark injection.

4.1 Motivation

We begin by formalizing the Watermarking Strength, which is a token-level metric that highly-related to watermark detectability of KGW-scheme and then explain why KGW-scheme could have poor Watermarking Strength on low-entropy tasks. Let L_i be the model logit for token $v_i \in \mathcal{V}$. The corresponding unnormalized weight after softmax is

$$w_i = e^{L_i}. \tag{1}$$

Given a partition of $\mathcal{V}$ into a green set $\mathcal{V}^g$ and a red set $\mathcal{V}^r$, the total probability assigned to the green set before watermarking is

$$p_g = \sum_{v_i \in \mathcal{V}^g} \frac{w_i}{\sum_{v_j \in \mathcal{V}} w_j}. \tag{2}$$

In the KGW scheme, a fixed bias $\delta > 0$ is added to the logits of all green tokens, yielding modified logits

$$\tilde{L}_i = \begin{cases} L_i + \delta, & v_i \in \mathcal{V}^g, \\ L_i, & v_i \in \mathcal{V}^r. \end{cases} \tag{3}$$

The total probability assigned to the green set after watermarking becomes

$$\tilde{p}_g = \frac{e^\delta p_g}{e^\delta p_g + (1 - p_g)}. \tag{4}$$

We define the *Watermarking Strength* as the normalized increase in the green total probability due to watermarking:

$$f_{ws}(\delta, p_g) = \frac{\tilde{p}_g - p_g}{\sqrt{p_g(1 - p_g)}}. \tag{5}$$

which could be simplified as:

$$f_{ws}(\delta, p_g) = \frac{(e^\delta - 1)\sqrt{p_g(1 - p_g)}}{1 + (e^\delta - 1)p_g}. \tag{6}$$

Intuitively, f_{ws} measures how much the bias δ alters the model's next-token distribution toward the green set. Thus, f_{ws}, which is a concave function about δ and p_g depicted in Appendix B , serves as a quantitative measure of the effectiveness of watermark injection at each next-token generation step. The cumulative strength over a sequence determines the detectability of the watermark.

In low-entropy settings, suppose the top m tokens have logits $L_1 \geq \cdots \geq L_m \gg \sum_{i>m} L_i$. With probability 2^{1-m}, random vocabulary split places all m top tokens into the *same* subset, which happens non-negligibly when m is small in low-entropy settings, and hence p_g is close to 0 or 1 as $\sum_{i>m} p_i \to 0$. As a result, f_{ws} could be nearly 0 in Equation 6, causing a ineffective watermark injection.

To improve on f_{ws}, one option is increase the bias term δ, however, this could induce a significant text quality drop (Kirchenbauer et al., 2024). Instead, SSG aims to avoid such phenomenon by guaranteeing the tokens can be divided into two logit-balanced sets. By doing so, SSG enhances watermark detectability without comprising text quality.

4.2 Watermark Injection

Algorithm 1 describes how SSG generate watermarked text with a language model $\mathcal{M}$. First, the

model computes the logits $L = \mathcal{M}(\boldsymbol{x})$ for all vocabulary tokens. The tokens are then sorted in descending order of their logits, and the resulting sequence is partitioned into consecutive groups of size two.

Each group is processed independently: using a pseudorandom generator seeded by the secret key s combined with the token ids in every group $\mathcal{B}$, one token from the group is assigned to the green set $\mathcal{V}^g$, and the other to the red set $\mathcal{V}^r$. After partitioning, the logits of all tokens in the green set are increased by δ, while the red tokens remain unchanged. The modified logits $\tilde{L}$ are then passed through the softmax function to produce the biased probability distribution $\tilde{P}$, from which the next token y is sampled.

Despite the strong effectiveness of SSG demonstrated in our experiments, the procedure of SSG on the whole vocabulary can be computationally expensive, as it requires sorting the logits at every token prediction step. To strike a balance between effectiveness and efficiency, we propose to conduct SSG only on top-k logits tokens. The key observation is that in low-entropy settings, only a few tokens dominate the probability. Thus, ensuring that these high-logit tokens are evenly distributed between the green and red sets is sufficient to achieve a balanced vocabulary split. Based on this insight, SSG applies only on the top-k highest-probability tokens, while randomly splitting the remaining tokens. Our experiments show that this approach substantially reduces computational cost while maintaining watermarking effectiveness.

4.3 Watermark Detection

As an algorithm for logit-balanced vocabulary partition, SSG can incorporate any KGW-scheme detection algorithm like (Kirchenbauer et al., 2024; Lee et al., 2024; Lu et al., 2024; Huang et al., 2025). In Algorithm 2, we adopt the design from (Lu et al., 2024) as an example for detection due to its reported high performance, which uses the entropy value of every token to determine its weight on final detection result.

4.4 Analysis of SSG

In this section, we present an analysis of SSG to demonstrate why it can guarantee the lower bound of watermark strength even in low-entropy settings.

Let all p_i be sorted descendingly: $p_1 \geq p_2 \geq p_3 \geq \cdots \geq p_m \gg \sum_{i>m} p_i \approx 0$ and $p_{1:m}$ are high-probability tokens. SSG forms consecutive pairs $(p_1, p_2), (p_3, p_4), \ldots$ and assigns one token from each pair to $\mathcal{V}^g$. Define

$$\alpha = \sum_j p_{2j}, \beta = \sum_j p_{2j-1}. \tag{7}$$

Then we have:

$$\alpha \geq \frac{1-p_1}{2}, \beta \leq \frac{1+p_1}{2}. \tag{8}$$

Based on this, the lower bound p_g^{lb} and upper bound p_g^{ub} of p_g are:

$$p_g^{lb} = \frac{1-p_1}{2} \leq p_g \leq \frac{1+p_1}{2} = p_g^{ub}. \tag{9}$$

Because f_{ws} is a concave function about p_g, we then have:

$$f_{ws}(p_g) \geq \min\{f_{ws}(p_g^{lb}), f_{ws}(p_g^{ub})\} > 0. \tag{10}$$

Algorithm 1 Single Step Watermark Injection of SSG

Require: LLM $\mathcal{M}$, context $\boldsymbol{x}$, vocabulary $\mathcal{V}$, secret key s, watermark window size h, logit bias δ, Green list ratio γ, Top-k hyperparameter k
Ensure: Next token y, Green set $\mathcal{V}^g$, Red Set $\mathcal{V}^r$

$L \leftarrow \mathcal{M}(\boldsymbol{x})$
$\pi \leftarrow$ indices of $\mathcal{V}$ sorted by L_i in descending order
Partition $\pi_{0:k-1}$ into consecutive groups $\mathcal{B}_1, \mathcal{B}_2, \ldots, \mathcal{B}_{k/2}$ of size 2
$\mathcal{V}^g \leftarrow \emptyset$, $\mathcal{V}^r \leftarrow \emptyset$
for each group $\mathcal{B}$ **do**
  $(v_1, v_2) \leftarrow$ tokens in $\mathcal{B}$
  Seed RNG with $\mathrm{hash}(s, v_1, v_2, \boldsymbol{x}_{-h:})$
  $r \leftarrow$ RNG output from $(0, 1)$
  if $r <= 0.5$ **then**
    $\mathcal{V}^g \leftarrow \mathcal{V}^g \cup \{v_1\}$, $\mathcal{V}^r \leftarrow \mathcal{V}^r \cup \{v_2\}$
  else
    $\mathcal{V}^g \leftarrow \mathcal{V}^g \cup \{v_2\}$, $\mathcal{V}^r \leftarrow \mathcal{V}^r \cup \{v_1\}$
  end if
end for
$T \leftarrow \gamma * |\mathcal{V}| - k/2$
Seed RNG with $\mathrm{hash}(s, \boldsymbol{x}_{-k:})$
$\mathcal{V}^* \leftarrow$ Permutate $\mathcal{V}_{k:}$ seed by RNG
$\mathcal{V}^g \leftarrow \mathcal{V}^g \cup \{\mathcal{V}^*_{0:T-1}\}$,
$\mathcal{V}^r \leftarrow \mathcal{V}^r \cup \{\mathcal{V}^*_{T:}\}$
for each $v_i \in \mathcal{V}^g$ **do**
  $\tilde{L}_i \leftarrow L_i + \delta$
end for
$\tilde{P} \leftarrow \mathrm{softmax}(\tilde{L})$
Sample next token $y \sim \tilde{P}$
return $y, \mathcal{V}^g, \mathcal{V}^r$

which means f_{ws} has a lower bound and it is determined by p_1. In this formulation, the value of p_1 directly controls the tightness of the lower bound. When p_1 is very large, the distribution is dominated by a single high-probability token. In this extreme case, the residual probability becomes small, resulting in a looser lower bound. Intuitively, if the model is already overwhelmingly confident in predicting one token, no watermarking scheme can substantially alter the distribution without comprising text quality, and thus the strength of the watermark signal is inherently limited.

Conversely, when p_1 is moderate, the probability is more evenly distributed among the top tokens. In this case, SSG ensures that both α and β are bounded away from 0 and 1, leading to a tighter bound on f_{ws}.

Taken together, this analysis demonstrates that SSG provides a provable guarantee of watermark strength that scales with the concentration of the next-token distribution. While extremely peaked distributions (large p_1) inherently limit the effectiveness of watermarking, SSG ensures that as long as probability is shared among multiple candidate tokens, the watermark signal remains detectable.

5 Experiments

We conduct experiments across multiple datasets, models, and watermarking methods using the open-source toolkit MarkLLM (Pan et al., 2024).

Tasks and Datasets. We evaluate two representative low-entropy tasks: code generation and mathematical problem solving. For code generation, we use HumanEval (Austin et al., 2021) and MBPP (Chen et al., 2021), both consisting of Python programming problems with reference solutions. For mathematical reasoning, we sample 10% problems from GSM8K (Cobbe et al., 2021), which contains grade-school level math questions in English. For high entropy tasks, we use C4 (Raffel et al., 2023) and CNN/DailyMail (Hermann et al., 2015). Following prior work (Lee et al., 2024; Lu et al., 2024; Huang et al., 2025), we only consider generations longer than 15 tokens for watermark detection.

Models. For code generation, we employ Qwen2.5-Coder-7B(Hui et al., 2024), a model specialized in programming tasks. For mathematical reasoning, we use DeepSeekMath-7B(Shao et al., 2024), which is trained specifically for math problem solving. To assess the generalizability of SSG, we also evaluate the versatile LLaMA-3-8B (Grattafiori et al., 2024) on both code and math tasks. The max generation lengyh is set to 512 in all experiments.

Watermarking Methods. We compare three representative watermarking methods: KGW, SWEET, and EWD. KGW is the classical baseline in this line of research, while SWEET and EWD are methods designed to better handle low-entropy scenarios. Without lossing generality, we set watermark context window h as 1, δ as 2.0, green set ratio γ as 0.5 for all methods.

Baselines and Metrics. Since SSG can be incorporated into any KGW-scheme watermarking method, we report results for each method both with and without SSG, thereby isolating the effect of our contribution. Following prior works (Lu et al., 2024; Lee et al., 2024), we report true positive rates (TPR) at 1% and 5% false positive rates (FPR), along with the corresponding F1-score. For generation quality, we measure Pass@1 accuracy on HumanEval and MBPP for code generation, and answer accuracy on GSM8K for mathematical reasoning. Besides, we use perplexity to measure text quality on C4 and CNN/DailyMail. These metrics together reflect the trade-off between watermark detectability and text quality.

5.1 Main Results

Table 1 and Table 2 shows the performance of multiple watermark algorithms on code generation and mathematic reasoning tasks respectively, while Table 5 shows result on high-entropy tasks. Across all models and datasets, the introduction of watermarking leads to a drop in Pass@1 compared with the unwatermarked baseline. This indicates that watermarking inevitably perturbs the next-token distribution, and such perturbations come at the cost of text quality. In other words, watermarking strengthens attribution and traceability but reduces the probability of generating the correct solution.

For all three watermarking methods (KGW, SWEET, and EWD), the corresponding SSG variants yield substantial improvements in detection performance. In particular, SSG consistently raises TPR and F1 under both the 1% and 5% FPR thresholds, in some cases by large margins. Importantly, these gains are achieved without further degrading Pass@1, demonstrating that SSG can significantly strengthen watermark detectability while preserving text quality.

Model	Method	Humaneval					MBPP				
		P@1	T@1	F1@1	T@5	F1@5	P@1	T@1	F1@1	T@5	F1@5
Qwen2.5-coder-7B	UW	31.8	0.6	1.2	3.7	6.7	42.3	1.9	3.6	5.8	10.5
	KGW	26.2	22.0	35.8	36.0	51.3	38.4	37.8	54.6	64.0	75.9
	+SSG	25.6	39.0	55.9	45.7	60.7	37.0	58.7	73.6	71.7	81.3
	SWEET	26.2	29.3	45.1	36.1	66.1	40.2	53.7	69.5	72.8	82.0
	+SSG	29.9	50.0	66.4	**78.0**	**84.2**	37.6	62.2	83.5	76.2	84.2
	EWD	26.5	32.9	49.3	52.4	66.7	37.6	57.4	75.7	74.6	83.2
	+SSG	27.8	**50.6**	**66.9**	70.7	80.6	38.1	**74.3**	**84.9**	**84.1**	**89.1**
LLaMA-3-8B	UW	34.1	8.5	15.6	15.2	25.4	56.3	13.6	18.3	29.7	10.5
	KGW	27.4	13.4	23.5	22.6	35.4	51.9	21.1	27.6	28.3	42.5
	+SSG	32.3	14.0	24.5	49.4	64.0	51.6	85.5	89.9	89.9	92.4
	SWEET	29.3	7.9	14.6	26.2	40.0	51.9	74.1	78.5	81.0	87.2
	+SSG	29.3	24.4	39.0	37.8	53.0	53.2	93.0	96.0	96.8	96.1
	EWD	34.1	26.8	42.1	54.9	68.7	51.6	87.7	91.3	94.2	94.7
	+SSG	29.9	**59.1**	**74.0**	**73.2**	**82.2**	51.6	**97.1**	**98.1**	**98.9**	**97.1**

Table 1: Evaluation of watermarking methods on **Humaneval** and **MBPP**. For every setting, we run 5 times and report the average result. Numbers are percentages (i.e., original scores $\times 100$). **UW** denotes the unwatermarked baseline. **UW** denotes the unwatermarked baseline. Metrics are reported as percentages (original scores $\times 100$): **P@1** = Pass@1, **T@1** = True Positive Rate (TPR) at 1% False Positive Rate (FPR), **F1@1** = F1-score at 1% FPR, **T@5** = TPR at 5% FPR, **F1@5** = F1-score at 5% FPR. Best results in each column are bolded.

Model	Method	P@1	T@1	F1@1	T@5	F1@5
DSMath-7B	UW	27.3	4.5	8.6	6.8	12.2
	KGW	21.7	41.7	58.5	57.6	71.0
	+SSG	25.8	90.9	94.9	91.7	93.4
	SWEET	27.3	94.7	96.9	**97.0**	**96.2**
	+SSG	25.0	94.7	96.9	**97.0**	**96.2**
	EWD	21.2	93.2	96.1	95.5	95.5
	+SSG	25.0	**96.2**	**97.7**	**97.0**	**96.2**
LLaMA-3-8B	UW	38.6	5.3	10.0	12.9	21.9
	KGW	37.1	63.6	77.4	65.2	76.8
	+SSG	34.1	89.4	94.0	91.0	95.5
	SWEET	38.6	79.5	88.2	94.7	95.1
	+SSG	34.1	**100**	**99.6**	**100**	**97.8**
	EWD	35.6	74.2	84.8	95.5	95.5
	+SSG	35.6	99.2	99.2	99.2	97.4

Table 2: Evaluation of watermarking methods on **GSM8K**.

5.2 Analysis on Watermark Strength

To further assess the effect of SSG on token-level watermark detectability, we analyze the distribution of Watermark Strength on code generation tasks with Qwen2.5-Coder-7B and LLaMA-3-8B. Figure 3 illustrates the comparison between KGW and SSG. The key difference is that KGW produces a substantially larger proportion of tokens with near-zero Watermark Strength. Such tokens contribute little to the statistical signal and ultimately lower the overall z-score of the sequence, impairing detection. By contrast, SSG reshapes the distribution by reducing the probability of near-zero values and enforcing a lower bound on Watermark Strength, thereby ensuring more consistent detectability across tokens.

5.3 The choice of Top-k

We further investigate how the choice of top-k in SSG influences watermarking performance, text quality, and decoding efficiency. Experiments are conducted across six settings: Qwen2.5-Coder-7B and LLaMA-3-8B on HumanEval and MBPP for code generation, and DSMath-7B and LLaMA-3-8B on GSM8K for mathematical reasoning. We compare the baseline EWD with its variants combined with SSG, varying $k \in \{2, 4, 8, 16\}$ to control the number of top-logit tokens used for balanced vocabulary partitioning. For decoding efficiency, we normalize the speed of EWD to 100%.

The results in Figure 1 show that even when SSG is applied only to the top-2 logits, it yields substantial improvements in detection performance (TPR@1, TPR@5, and F1) compared to the baseline. Increasing k beyond 2 brings only marginal gains in detection while leaving text quality (Pass@1) unchanged, but incurs additional overhead that slows down decoding. Thus, larger k values fail to provide further detectability benefits

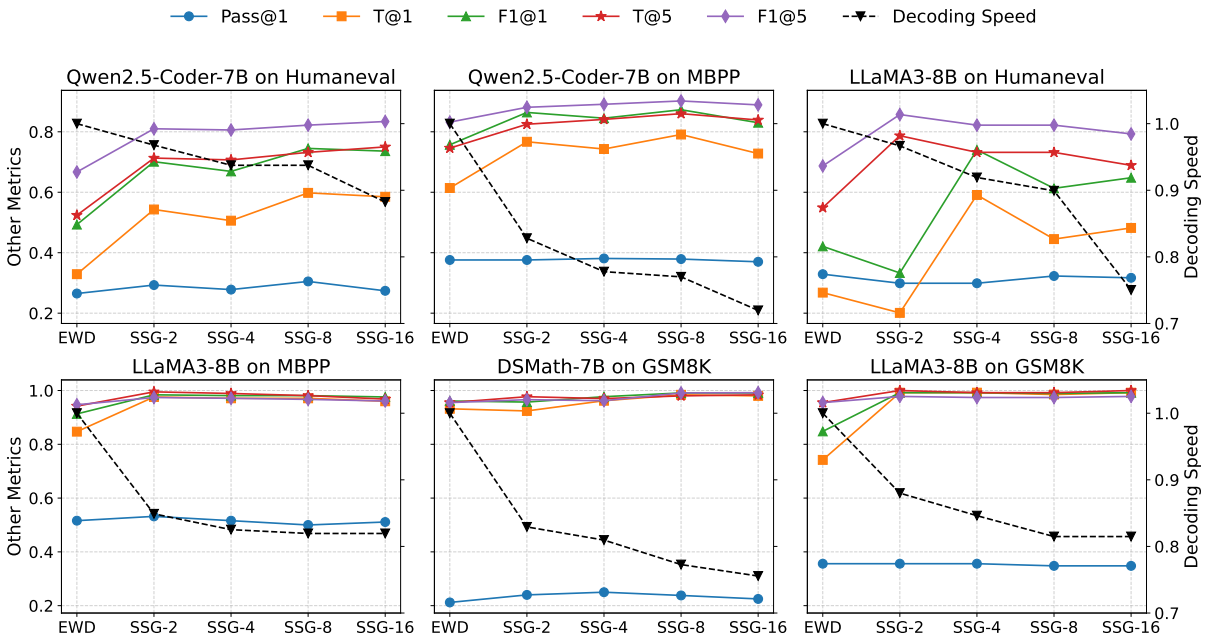

Figure 1: Influence of top-k on SSG performance.

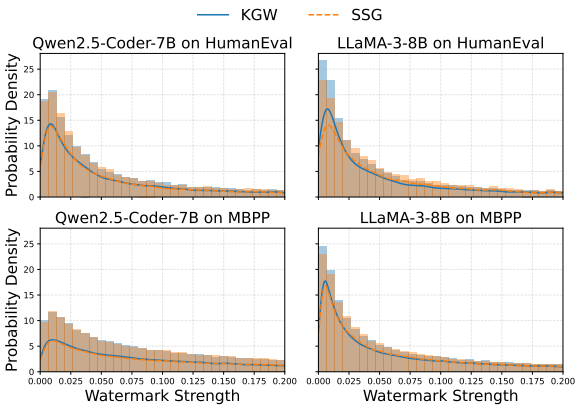

Figure 2: Examples of the watermark strength distributions on Qwen2.5-Coder-7B and LLaMA3-8B for code generation tasks.

and instead reduce efficiency.

Considering this trade-off, we recommend moderate settings of $k = 2$ or $k = 4$, which offer strong detection capability without significant efficiency loss.

5.4 Results without Original Prompts

In realistic scenarios such as code authorship detection, the original prompts used during generation may be unavailable at the time of watermark detection. To evaluate this setting, we follow prior work (Lu et al., 2024; Lee et al., 2024; Huang et al., 2025) and conduct experiments using a general prompt. Table 3 and Table 4 report the detection results under this condition for code generation and mathematical reasoning, respectively. Compared to the case where original prompts are accessible, detection performance generally decreases across all methods, except for KGW without SSG. This is expected since KGW is the only method that does not depend on the original prompt for detection. Moreover, the effect of incorporating SSG is mixed: it improves detection in some cases but degrades it in others, reflecting the inherent reliance of SSG on logit values conditioned by the original prompts.

5.5 Results on Robustness

We also evaluate the robustness of SSG when the watermarked sentence has been paraphrased by GPT-4o-mini, which is believed to be a strong **paraphrase attack** setting. The experiment results showed in Table 6 and 7 demonstrated that the robustness of SSG varies on different LLMs compared to other methods.

6 Conclusion

In this paper, we introduced a token-level metric, *Watermark Strength*, to evaluate watermark detectability at the granularity of individual tokens. Using this metric, we revealed why the KGW scheme suffers in low-entropy settings such as code generation and mathematical reasoning. To address this limitation, we proposed **SSG** (Sort-

Model	Method	Humaneval					MBPP				
		P@1	T@1	F1@1	T@5	F1@5	P@1	T@1	F1@1	T@5	F1@5
Qwen2.5-coder-7B	UW	31.8	0.6	1.2	3.7	6.7	42.3	1.9	3.6	5.8	10.5
	KGW	26.2	22.0	35.8	36.0	51.3	38.4	37.8	54.6	64.0	75.9
	+SSG	25.6	22.6	36.6	31.7	46.4	37.0	55.6	71.1	67.5	78.3
	SWEET	26.2	32.3	48.6	53.0	67.2	40.2	57.1	72.4	72.2	81.6
	+SSG	29.9	30.5	46.5	54.3	68.2	37.6	**72.2**	**80.2**	**75.7**	**83.7**
	EWD	26.5	**43.3**	**60.2**	**59.1**	**72.1**	37.6	67.5	**80.2**	75.4	**83.7**
	+SSG	27.8	28.0	43.6	42.1	57.3	38.1	68.3	71.7	68.3	78.9
LLaMA-3-8B	UW	32.9	8.5	15.6	15.2	25.4	56.3	13.6	18.3	29.7	29.7
	KGW	27.4	13.4	23.5	22.6	35.4	51.9	22.1	27.6	28.3	42.5
	+SSG	32.3	6.7	12.5	23.2	36.2	51.6	60.8	80.0	82.5	88.1
	SWEET	29.3	8.5	15.6	18.3	29.7	51.9	41.8	58.6	60.1	72.9
	+SSG	29.3	10.4	18.7	14.6	24.5	53.2	**94.5**	**94.5**	**91.3**	**93.1**
	EWD	34.1	18.9	31.6	**42.7**	**57.9**	51.6	82.9	82.9	83.1	88.5
	+SSG	29.9	**19.5**	**32.5**	34.8	49.8	51.6	85.8	85.8	87.3	90.9

Table 3: Evaluation of watermarking methods on **Humaneval** and **MBPP** without original prompts. All numbers are percentages; bold indicates the best per column.

Model	Method	P@1	T@1	F1@1	T@5	F1@5
DSMath-7B	UW	27.3	4.5	8.6	6.8	12.2
	KGW	21.7	41.7	58.5	57.6	71.0
	+SSG	25.8	81.8	89.6	85.6	90.0
	SWEET	**27.3**	**94.7**	**96.9**	**96.2**	**95.8**
	+SSG	25.0	78.0	87.3	82.6	88.3
	EWD	21.2	92.4	95.7	95.5	95.5
	+SSG	25.0	86.4	92.3	88.6	91.8
LLaMA-3-8B	UW	35.6	5.3	10.0	12.9	21.9
	KGW	35.6	63.6	77.4	65.2	76.8
	+SSG	34.1	71.2	82.8	78.8	86.0
	SWEET	**38.6**	60.6	75.1	84.1	89.2
	+SSG	34.1	**97.7**	**98.5**	**97.7**	**96.6**
	EWD	35.6	70.5	82.3	92.4	93.8
	+SSG	35.6	68.2	80.7	75.8	84.0

Table 4: Evaluation of watermarking methods on **GSM8K**. All numbers are percentages; bold indicates the best per column.

then-Split by Groups), a logit-balanced partitioning algorithm that guarantees a lower bound on Watermark Strength and thereby aims to mitigate ineffective watermark injections. Extensive experiments on multiple large language models across code and mathematic reasoning tasks demonstrate that SSG achieves consistently stronger detection performance while maintaining text quality.

Limitations

Although SSG proves effective in our evaluations, the experimental scope remains limited. Future work should expand to additional low-entropy domains and diverse datasets to further validate its performance. Moreover, while SSG improves watermark injection reliability, it still depends on the availability of the original prompt for optimal detection, which restricts its applicability in realistic settings. An important research direction is to design prompt-free watermarking algorithms that retain effectiveness even in low-entropy tasks.

References

Anthropic. 2024. Usage policy. Blog post.

Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, and Charles Sutton. 2021. Program synthesis with large language models. *Preprint*, arXiv:2108.07732.

Liang Chen, Yatao Bian, Yang Deng, Deng Cai, Shuaiyi Li, Peilin Zhao, and Kam fai Wong. 2024. Watme: Towards lossless watermarking through lexical redundancy. *Preprint*, arXiv:2311.09832.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, and 39 others. 2021. Evaluating large language models trained on code. *Preprint*, arXiv:2107.03374.

Miranda Christ, Sam Gunn, and Or Zamir. 2023. Undetectable watermarks for language models. *Preprint*, arXiv:2306.09194.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. 2021. Training verifiers to solve math word problems. *Preprint*, arXiv:2110.14168.

Google. 2025. Google apis terms of service. Blog post.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, and et al. 2024. The llama 3 herd of models. *Preprint*, arXiv:2407.21783.

Chenchen Gu, Xiang Lisa Li, Percy Liang, and Tatsunori Hashimoto. 2024. On the learnability of watermarks for language models. *Preprint*, arXiv:2312.04469.

Karl Moritz Hermann, Tomáš Kočiský, Edward Grefenstette, Lasse Espeholt, Will Kay, Mustafa Suleyman, and Phil Blunsom. 2015. Teaching machines to read and comprehend. In *Advances in Neural Information Processing Systems*, pages 1693–1701.

Abe Bohan Hou, Jingyu Zhang, Tianxing He, Yichen Wang, Yung-Sung Chuang, Hongwei Wang, Lingfeng Shen, Benjamin Van Durme, Daniel Khashabi, and Yulia Tsvetkov. 2024. Semstamp: A semantic watermark with paraphrastic robustness for text generation. *Preprint*, arXiv:2310.03991.

Houman. 2025. The openai vs deepseek knowledge distillation dispute: Technical and legal implications. Blog post.

Beining Huang, Du Su, Fei Sun, Qi Cao, Huawei Shen, and Xueqi Cheng. 2025. Low-entropy watermark detection via Bayes' rule derived detector. In *Findings of the Association for Computational Linguistics: ACL 2025*, pages 14330–14344, Vienna, Austria. Association for Computational Linguistics.

Binyuan Hui, Jian Yang, Zeyu Cui, Jiaxi Yang, Dayiheng Liu, Lei Zhang, Tianyu Liu, Jiajun Zhang, Bowen Yu, Keming Lu, Kai Dang, Yang Fan, Yichang Zhang, An Yang, Rui Men, Fei Huang, Bo Zheng, Yibo Miao, Shanghaoran Quan, and 5 others. 2024. Qwen2.5-coder technical report. *Preprint*, arXiv:2409.12186.

John Kirchenbauer, Jonas Geiping, Yuxin Wen, Jonathan Katz, Ian Miers, and Tom Goldstein. 2024. A watermark for large language models. *Preprint*, arXiv:2301.10226.

Rohith Kuditipudi, John Thickstun, Tatsunori Hashimoto, and Percy Liang. 2024. Robust distortion-free watermarks for language models. *Preprint*, arXiv:2307.15593.

Taehyun Lee, Seokhee Hong, Jaewoo Ahn, Ilgee Hong, Hwaran Lee, Sangdoo Yun, Jamin Shin, and Gunhee Kim. 2024. Who wrote this code? watermarking for code generation. *Preprint*, arXiv:2305.15060.

Raymond Li, Loubna Ben Allal, Yangtian Zi, Niklas Muennighoff, Denis Kocetkov, Chenghao Mou, Marc Marone, Christopher Akiki, Jia Li, Jenny Chim, Qian Liu, Evgenii Zheltonozhskii, Terry Yue Zhuo, Thomas Wang, Olivier Dehaene, Mishig Davaadorj, Joel Lamy-Poirier, João Monteiro, Oleh Shliazhko, and 48 others. 2023. Starcoder: may the source be with you! *Preprint*, arXiv:2305.06161.

Aiwei Liu, Leyi Pan, Yijian Lu, Jingjing Li, Xuming Hu, Xi Zhang, Lijie Wen, Irwin King, Hui Xiong, and Philip S. Yu. 2024. A survey of text watermarking in the era of large language models. *Preprint*, arXiv:2312.07913.

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

OpenAI, Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, and Ilge Akkaya et al. 2024. Gpt-4 technical report. *Preprint*, arXiv:2303.08774.

Leyi Pan, Aiwei Liu, Zhiwei He, Zitian Gao, Xuandong Zhao, Yijian Lu, Binglin Zhou, Shuliang Liu, Xuming Hu, Lijie Wen, Irwin King, and Philip S. Yu. 2024. Markllm: An open-source toolkit for llm watermarking. *Preprint*, arXiv:2405.10051.

Alec Radford and Karthik Narasimhan. 2018. Improving language understanding by generative pre-training.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. 2023. Exploring the limits of transfer learning with a unified text-to-text transformer. *Preprint*, arXiv:1910.10683.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, Y. K. Li, Y. Wu, and Daya Guo. 2024. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *Preprint*, arXiv:2402.03300.

Zhensu Sun, Xiaoning Du, Fu Song, and Li Li. 2023. Codemark: Imperceptible watermarking for code datasets against neural code completion models. In *Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, ESEC/FSE '23, page 1561–1572. ACM.

Zhensu Sun, Xiaoning Du, Fu Song, Mingze Ni, and Li Li. 2022. Coprotector: Protect open-source code against unauthorized training usage with data poisoning. *Preprint*, arXiv:2110.12925.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurelien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. 2023. Llama: Open and efficient foundation language models. *Preprint*, arXiv:2302.13971.

European Union. 2024. Regulation (eu) 2024/1689: Artificial intelligence act. Blog post.

Algorithm 2 SSG Watermark Detection

Require: LLM $\mathcal{M}$, suspected text $\boldsymbol{x}$, vocabulary $\mathcal{V}$, secret key s, threshold τ, green-list ratio γ, context window size h, Top-k hyperparameter k

Ensure: Decision on whether $\boldsymbol{x}$ is watermarked

Initialize $\boldsymbol{E} \leftarrow \mathbf{0}, \boldsymbol{g} \leftarrow \mathbf{0}$
for each token x_i in $\boldsymbol{x}_{k:}$ **do**
  $L \leftarrow \mathcal{M}(\boldsymbol{x}_{0:i-1})$
  $P \leftarrow \text{softmax}(L)$
  $E_i \leftarrow \text{Entropy}(P)$
  $(\mathcal{V}^g, \mathcal{V}^r) \leftarrow \text{SSG}(\mathcal{M}, \mathcal{V}, s, \boldsymbol{x}, h, k)$
  if $x_i \in \mathcal{V}^g$ **then**
    $\boldsymbol{g}_i \leftarrow 1$
  end if
end for
for each E_i in $\boldsymbol{E}$ **do**
  $W_i \leftarrow E_i - \min(\boldsymbol{E})$
end for
$|s|_G \leftarrow \boldsymbol{W} \cdot \boldsymbol{g}$
$z \leftarrow \dfrac{|s|_G - \gamma \sum_{i=h}^{|\boldsymbol{x}|} W_i}{\sqrt{\gamma(1-\gamma)\sum_{i=h}^{|\boldsymbol{x}|} W_i^2}}$
if $z > \tau$ **then**
  return $\boldsymbol{x}$ is watermarked
else
  return $\boldsymbol{x}$ is not watermarked
end if

A Watermark Detection

Algorithm 2 explains how SSG works with another detection algorithm like EWD.

B Curve of Watermark Strength

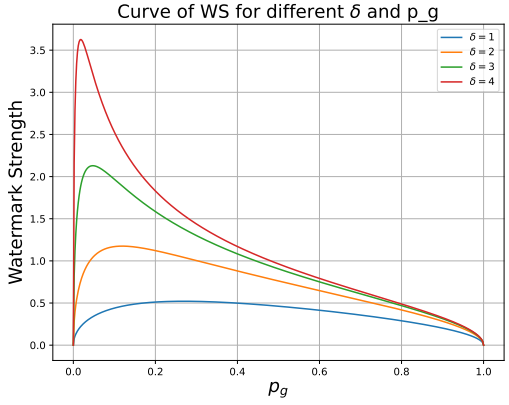

Figure 3: Curve of Watermark Strength over p_g and δ.

C Text Quality of SSG

We show that SSG and KGW are equivalent in the expected logit shift across the parameter space $\gamma \in (0, 1)$. Let $\mathbb{E}[\tilde{L}_i(\gamma)]$ be the expected logit of token v_i for a given γ.

In the KGW framework, the expectation is trivially:

$$\mathbb{E}_{KGW}[\tilde{L}_i(\gamma)] = L_i + \gamma\delta \tag{11}$$

In SSG, the probability of green tokens $P(v_i \in \mathcal{V}^g; \gamma)$ is piecewise-defined by the token's rank Z_i. The total bias injected into the vocabulary $\mathcal{V}$ is:

$$SSG(\gamma) = \sum_{v_j \in \mathcal{V}} \left(\mathbb{E}_{SSG}[\tilde{L}_j(\gamma)] - L_j \right) \tag{12}$$

$$= \sum_{v_j \in \mathcal{V}_{top}} 0.5\delta + \sum_{v_j \in \mathcal{V}_{rem}} \frac{\gamma|\mathcal{V}| - k/2}{|\mathcal{V}| - k}\delta \tag{13}$$

$$= \frac{k}{2}\delta + (\gamma|\mathcal{V}| - k/2)\delta = \gamma|\mathcal{V}|\delta \tag{14}$$

Dividing by the vocabulary size $|\mathcal{V}|$, the mean expected logit for SSG is:

$$\frac{1}{|\mathcal{V}|} \sum_{v_j \in \mathcal{V}} \mathbb{E}_{SSG}[\tilde{L}_j(\gamma)] = \bar{L} + \gamma\delta \tag{15}$$

Thus, the global expectation of the logit shift in SSG is identical to KGW:

$$\mathbb{E}_{SSG}[\tilde{L}(\gamma)] = \mathbb{E}_{KGW}[\tilde{L}(\gamma)] = L + \gamma\delta \tag{16}$$

This confirms that SSG preserves the quality quality of KGW.

D General Prompts

For Humaneval and MBPP, we use "Please write a Python function that solves a given task.\n Output only valid Python code without explanations. \n\n def solve_problem(input_data): \n" as our General Prompt for experiments without original prompts. While on GSM8k, the General Prompt is "Solve the following math word problem step by step, and in the last line output the final numeric answer in the format: \n #### <answer>.\n".

E Experiments

Model	Method	C4					CNN/DM				
		PPL	T@1	F1@1	T@5	F1@5	PPL	T@1	F1@1	T@5	F1@5
LLaMA-3-8B	UW	2.65	3.0	5.8	4.0	7.3	2.60	0.0	0.0	2.0	3.7
	KGW	3.01	81.0	89.0	90.0	92.3	2.86	72.0	83.2	91.0	92.9
	+SSG	2.80	93.0	95.9	95.0	95.0	2.84	98.0	98.5	99.0	97.1
	SWEET	3.01	91.0	94.8	93.0	93.9	2.89	88.0	93.1	99.0	97.1
	+SSG	2.80	94.0	96.8	95.2	95.8	2.84	95.0	95.0	99.0	97.1
	EWD	3.01	95.0	96.9	97.0	96.0	2.86	92.0	95.3	95.0	95.0
	+SSG	2.80	96.0	97.5	98.0	96.6	2.84	98.0	98.5	99.0	97.1

Table 5: Evaluation of watermarking methods on **C4** and **CNN/DailyMail** . All numbers are percentages.

Model	Method	Humaneval				MBPP			
		T@1	F1@1	T@5	F1@5	T@1	F1@1	T@5	F1@5
Qwen2.5-coder-7B	UW	0.6	1.2	3.7	6.7	1.9	3.6	5.8	10.5
	KGW	3.0	5.9	9.1	16.1	3.2	6.1	14.3	24.0
	SWEET	1.2	2.4	8.5	15.1	2.6	5.1	9.3	16.2
	EWD	5.5	10.3	11.0	18.9	4.8	9.0	8.7	15.4
	+SSG	7.3	13.6	20.1	32.2	2.4	4.6	6.1	11.0
LLaMA-3-8B	UW	6.1	11.4	11.6	19.9	1.6	3.1	4.5	8.2
	KGW	0.6	1.2	1.2	2.3	1.1	2.1	5.0	9.2
	SWEET	0.6	1.2	1.2	2.3	1.1	2.1	5.0	9.2
	EWD	7.3	13.6	23.8	37.0	4.0	7.6	12.7	21.6
	+SSG	28.0	43.6	53.0	67.2	5.6	10.4	19.6	31.5

Table 6: Evaluation of watermarking methods on **Humaneval** and **MBPP** when watermarked sentence has been paraphrased by GPT-4o-mini. All numbers are percentages.

Model	Method	T@1	F1@1	T@5	F1@5
DSMath-7B	UW	4.5	8.6	6.8	12.2
	KGW	1.5	3.0	6.1	11.0
	SWEET	25.0	39.8	44.7	59.9
	EWD	22.7	36.8	28.8	43.2
	+SSG	0.0	0.0	9.8	17.2
LLaMA-3-8B	UW	2.3	4.4	3.8	7.0
	KGW	0.0	0.0	13.6	23.1
	SWEET	0.0	0.0	13.6	23.1
	EWD	1.5	3.0	12.1	20.8
	+SSG	12.1	21.5	25.0	38.6

Table 7: Evaluation of watermarking methods on **GSM8K** when watermarked sentence has been paraphrased by GPT-4o-mini. All numbers are percentages.