CROSSMARK: WEAVING CROSS-GRANULARITY WATERMARKS FOR ROBUST DETECTION OF LLM-GENERATED TEXT AND CODE

Anonymous authors
Paper under double-blind review

ABSTRACT

Watermarks for large language model (LLM) defined at a single granularity struggle to simultaneously achieve dense detection evidence, low generation distortion, robustness to local insertion, and resilience to rewriting. We introduce CrossMark, a cross-granularity watermarking framework that combines complementary evidence across token, local-unit, and global-unit levels. Its key idea is to let different granularities specialize in different attack regimes while reinforcing one another during detection. Dense token-level evidence supports reliable clean detection, local-unit keyed preferences expose insertion-induced inconsistencies, and rewrite-stable semantic representations preserve global evidence after substantial paraphrasing or code transformation. CrossMark further couples these signals rather than treating them independently: local consistency dynamically calibrates token statistics and identifies suspicious units for semantic verification, producing a unified joint detection score. Experiments on natural-language and code benchmarks show that CrossMark delivers the strongest or near-strongest clean detection, leads evaluated baselines under local-unit insertion and single-pass LLM rewriting, and preserves competitive text quality and code correctness. Ablations further reveal a clear division of robustness: local evidence is the primary contributor against insertion, whereas global semantic evidence provides the dominant signal under rewriting.

1 INTRODUCTION

Large language models (LLMs) now generate fluent natural language and functionally useful source code, increasing the need for reliable provenance and usage auditing. Generation-time watermarking addresses this need by embedding a secret statistical signal during decoding that can later be verified without retaining individual generation records (Kirchenbauer et al., 2023a; Kuditipudi et al., 2023; Zhao et al., 2023).

Most existing methods place the signal at one granularity. Token-level watermarks create dense evidence by biasing a keyed subset of the vocabulary (Kirchenbauer et al., 2023a). They detect intact outputs effectively, but stronger biases can distort generation quality, particularly for code, where small lexical changes may affect syntax or functionality (Ajith et al., 2024; Lee et al., 2024; Kim et al., 2025). They are also vulnerable when insertion dilutes the proportion of watermarked tokens or when LLM rewriting replaces much of the original surface form. Semantic watermarks reduce dependence on exact tokens by operating over sentence- or span-level representations (Hou et al., 2024; Liu et al., 2023). However, larger units provide fewer independent observations and remain vulnerable when an attack changes their boundaries or rewrites an entire span. No single watermark granularity therefore simultaneously provides dense evidence, low distortion, insertion robustness, and rewrite robustness.

Our central observation is that these properties are complementary across three operational granularities. Tokens provide abundant but surface-sensitive evidence. Local units, defined as sentences for natural language and statements or logical lines for code, expose insertion at the level where new content is introduced. Global units, defined as overlapping sentence spans or complete functions, contain enough semantic information to support rewrite-robust verification. These units are domain-specific rather than an identical linguistic hierarchy for text and code, but they serve comparable

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

roles in generation and detection. Their different observation counts and attack sensitivities make them natural complements. Fine-grained evidence supports statistical power, local evidence identifies suspicious insertions, and global evidence preserves provenance when lexical realization changes substantially.

Based on this observation, we propose CROSSMARK, a cross-granularity watermarking framework that embeds all three signals within one shared candidate pool. Each active path receives a lightweight token-level red–green bias. At a local-unit boundary, candidates are retained according to a secret-key-conditioned hash preference. At a global-unit boundary, a learned semantic encoder assigns candidates to categories, and candidates in the key-preferred category remain active. The global representation is trained with candidate–rewrite pairs as positives and alternative generation paths as negatives, encouraging the representation to distinguish watermarkable paths while preserving category assignments after meaning-preserving rewriting. Unlike independent post-hoc watermarks, the three signals accumulate along the same generation paths.

The granularities also interact during detection rather than acting as three independent detectors. Local hash evidence provides an independent score and weights the token evidence recovered from each unit. This weighting reduces the influence of inserted units whose hashes are inconsistent with the secret preference. It additionally filters suspicious local units before one branch of global semantic verification. The detector combines token, local, and global scores into one empirical joint statistic. This design uses local evidence to reduce insertion dilution and global evidence to recover signal after substantial rewriting. Figure 1 summarizes the joint injection and cross-level verification process.

We evaluate CROSSMARK on natural-language and code generation under clean, local-unit insertion, and single-pass LLM rewriting settings. The experiments cover multiple generation backbones and code languages, allowing us to examine whether higher-level evidence remains useful when the token watermark varies in strength across models. We measure detection, text quality, code correctness, efficiency, and the individual and combined contributions of the three granularities.

Our contributions are as follows:

- We formulate LLM watermarking across token, local-unit, and global-unit granularities, which provide complementary evidence under insertion and rewriting.
- We introduce a joint generation framework that combines a lightweight token bias, keyed local-unit preference, and rewrite-robust global semantic preference within one candidate pool.
- We develop cross-level detection in which local hash evidence weights token observations and filters one global semantic view before the three scores are combined.
- We evaluate the framework on natural-language and code benchmarks and show that the three granularities improve robustness without a systematic loss of generation quality.

2 RELATED WORK

Generation-time LLM watermarking. Most proactive watermarks for large language models embed a keyed statistical preference during decoding. The red–green list framework biases next-token selection, while subsequent work improves distribution preservation, distortion control, robustness, or multi-bit capacity (Kirchenbauer et al., 2023a; Hu et al., 2024; Kuditipudi et al., 2023; Yoo et al., 2023). These methods provide dense evidence because many token decisions contribute to detection, but their effectiveness and quality impact remain tied to the strength and stability of token-level preferences.

Semantic and edit-robust watermarking. Watermark robustness has been studied under paraphrasing, adversarial editing, and bounded edit-distance channels (Kirchenbauer et al., 2023b; Abdelnabi & Fritz, 2021; Qiang et al., 2023; Golowich & Moitra, 2024). Semantic methods reduce dependence on exact lexical choices by partitioning an embedding space or conditioning watermark decisions on semantic representations (Hou et al., 2024; Liu et al., 2023). Such methods improve resistance to lexical rewriting, but a detector operating at only one semantic scale has fewer observations and can remain sensitive to inserted units or disrupted unit boundaries.

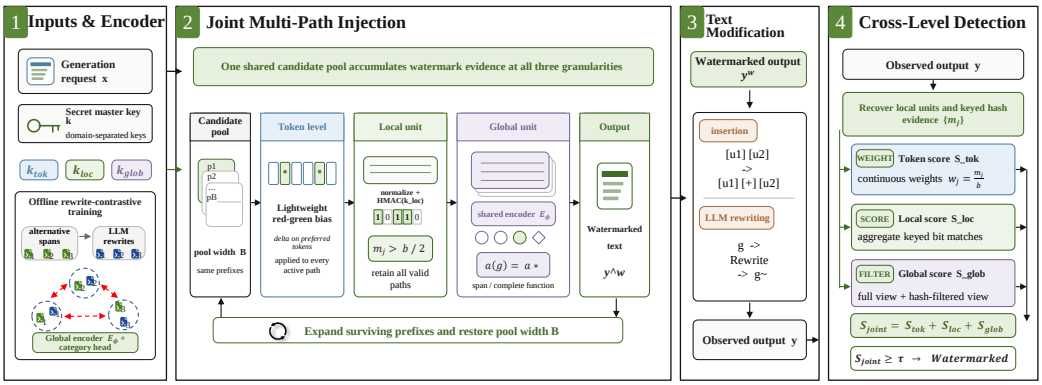


Figure 1: Overview of CROSSMARK. A secret master key is separated into token-, local-, and global-level keys. One shared candidate pool accumulates watermark evidence at all three granularities through a lightweight red–green token bias, keyed local-unit hash filtering, and a rewrite-robust global semantic preference. After local-unit insertion or LLM rewriting, recovered local evidence weights token-level detection, provides an independent local score, and filters one global semantic view before the three scores are combined.

Watermarking generated code. Code watermarking must preserve syntax and executable behavior despite the low entropy of many generation positions. Existing approaches restrict watermarking to high-entropy tokens, adapt thresholds to context, protect syntax explicitly, use grammar information, or apply post-generation transformations (Lee et al., 2024; Zhang et al., 2025; Kim et al., 2025; Guan et al., 2024; Wang & et al., 2024). Semantic-preserving transformations have also been used to improve persistence under code edits (Yang et al.). These systems are primarily designed around a single carrier granularity. In contrast, our framework jointly embeds token-, local-unit-, and global-unit-level evidence and introduces cross-level verification in which local hash evidence weights token statistics and filters the input to global semantic detection.

3 PROBLEM FORMULATION

Let x be a generation request, k a secret master key, and $y^w \sim \mathcal{G}_\theta^w(\cdot \mid x, k)$ a watermarked natural-language or code output. CrossMark derives separate token-, local-, and global-level keys from k, so evidence at one granularity does not reuse the key material of another.

We organize each output into three operational granularities. Tokens provide fine-grained observations. Local units are sentences in natural language and statements or logical lines in code. Global units are overlapping spans of consecutive sentences or complete functions. These units are domain-specific carriers rather than claims that text and code share an identical linguistic hierarchy. If $y = (u_1, \ldots, u_M)$ is the recovered local-unit sequence, a domain-specific extractor further constructs global units $\mathcal{G}(y) = \{g_1, \ldots, g_T\}$.

We study two modifications that target different carriers. A local-unit insertion attack adds sentences or behavior-preserving code statements. An LLM rewrite changes the lexical and structural realization of a span or function while preserving its intended meaning or functionality. The attacker does not know the secret key or repeatedly query the detector. A domain-appropriate utility verifier $\mathcal{V}(y, x)$ represents acceptable content preservation for natural language and syntactic and functional validity for code.

Given an observed output y, the detector computes a continuous joint score $\mathcal{S}_{\text{joint}}(y, k)$ and predicts a watermark when the score exceeds a threshold τ. The objective is to retain separation between watermarked and unwatermarked outputs after the considered modifications while preserving utility. The joint score is not assigned a standard-normal probability interpretation. Appendix A provides the complete notation, attack constraints, and score definitions.

4 METHODOLOGY

4.1 CROSS-GRANULARITY INJECTION

Figure 1 summarizes CrossMark. A single pool of B_{pool} candidate paths accumulates watermark evidence at all three granularities. The framework does not generate three independent watermarked outputs and combine them afterward.

At the token level, every active path receives the standard keyed red–green logit bias (Kirchenbauer et al., 2023a). A pseudorandom function partitions the vocabulary at each decoding step, and a strength parameter δ raises the logits of preferred tokens. Because local and global evidence supplement this dense token signal, CrossMark need not rely exclusively on a strong token bias.

At each local-unit boundary, the completed unit is conservatively normalized and hashed with HMAC-SHA-256 under the local key. CrossMark truncates the hash to b bits and counts matches with a fixed target string. All candidate paths whose match count exceeds the random-match mean $b/2$ remain active. The gate uses neither the previous unit nor its absolute position, so inserting a new unit adds hash evidence without changing the hashes recovered from original units. Retaining every valid candidate also preserves alternatives for later global selection instead of greedily choosing the highest local score.

The surviving prefixes are expanded until the pool returns to width B_{pool}. For natural language, a global check is applied whenever an overlapping span of r sentences becomes available. For code, local checks occur at statement boundaries and the global check is applied to the completed function. A secret global key identifies one preferred semantic category. Paths assigned to that category survive, after which generation continues from the same shared pool. Thus, token, local, and global preferences are embedded as nested constraints within one decoding process.

4.2 REWRITE-CONTRASTIVE GLOBAL REPRESENTATION

The global carrier must satisfy two competing requirements. Alternative candidate paths from the same context must remain distinguishable so that a key can prefer one path over another, while a selected candidate and its meaning-preserving rewrite should remain in the same category. We train separate natural-language and code encoders with the same rewrite-contrastive construction. For each context, the generator produces multiple candidates and an LLM independently rewrites each candidate. Candidate–rewrite pairs provide positive supervision, while different candidates from the same context serve as hard negatives.

The training objective combines representation and category consistency with path separation, category balance, and assignment confidence. The resulting encoder maps each global unit to one of K learned categories. These categories have no fixed semantic interpretation. During generation, the global key selects a preferred category, and the clusterer acts as a discrete gate over candidate paths. This directly connects representation learning to the alternatives available during injection and to the rewriting process used in the robustness evaluation.

4.3 CROSS-LEVEL DETECTION

Detection recovers the same three forms of evidence, but local evidence also controls how the other levels are evaluated. First, matching local hash bits are aggregated into a normalized local score S_{loc}. Second, each local match fraction weights the green-token observations inside that unit, producing a hash-weighted token score S_{tok}. Units with weak local evidence therefore contribute less to token detection, which limits dilution from inserted unwatermarked content.

Global verification uses two views. The full view encodes the observed output without removal. The filtered view first discards local units that fail the keyed majority gate and then reconstructs the global input. Natural-language sentence windows are rebuilt after filtering, allowing original neighboring content to become adjacent again when an inserted sentence is removed. For code, suspicious statements are removed while the function boundary is preserved. The two standardized global statistics are averaged into S_{glob}.

The detector combines the three correlated components as $\mathcal{S}_{\text{joint}} = S_{\text{tok}} + S_{\text{loc}} + S_{\text{glob}}$. This design gives local hashes three roles: they provide direct evidence, weight token statistics, and construct a cleaner input for one branch of semantic verification. Complete equations and the normalization procedure are reported in Appendix A.

5 EXPERIMENTS

5.1 EXPERIMENTAL SETUP

Datasets and models. We evaluate natural-language watermarking on C4 (Raffel et al., 2020) and code watermarking on HumanEval-Python (Chen et al., 2021), HumanEval-Java and HumanEval-C++ (Zheng et al., 2023), MBPP (Austin et al., 2021), and ClassEval (Du et al., 2023). Clean generations use Llama-3.1-8B-Instruct (Dubey et al., 2024) and Qwen2.5-7B-Instruct (Team, 2024). C4, HumanEval-Python, and MBPP form the main clean-generation comparison, while ClassEval supports code insertion and whole-class rewriting. Cross-language results are reported in Appendix C.

Attacks and baselines. We evaluate local-unit insertion and one-pass LLM rewriting. Natural-language insertion adds lines sampled from the same dataset, while code insertion adds behavior-preserving dead code. In both cases, the number of inserted lines is 50% of the original line count. We compare CROSSMARK with WLLM, SWEET, Unigram, ACW, SIR, and SemStamp using their public implementations. Methods that expose a watermark-strength parameter use 3.5, and other settings retain their defaults.

Configuration and metrics. CROSSMARK uses green-token proportion $\gamma = 0.5$, token strength $\delta = 3.5$, local hash length $b = 3$, $K = 2$ semantic categories, and an active pool of $B_{\text{pool}} = 8$ paths. We report AUROC and TPR at $\text{FPR} \leq 5\%$. Natural-language quality is measured with BERTScore (Zhang et al., 2020) and perplexity, while code correctness is measured with Pass@1. Complete dataset assignments, attack models, clusterer training, decoding settings, runtime environment, and prompts are provided in Appendix B and Appendix G.

5.2 CLEAN-GENERATION RESULTS

Table 1 compares detection performance and generation quality without attacks. C4 evaluates natural-language generation, while HumanEval-Python and MBPP evaluate code generation. The table reports both AUROC and $\text{TPR@FPR} \leq 5\%$ for every dataset. Natural-language quality is measured with BERTScore and PPL, while code correctness is measured with Pass@1.

Detection across generation models. Table 1 shows that CROSSMARK provides the strongest clean-output detection on C4 and HumanEval-Python under both backbones. Under Llama-3.1-8B-Instruct, the gains over the strongest baseline are moderate but consistent. AUROC increases from 0.9727 to 0.9796 on C4 and from 0.9343 to 0.9534 on HumanEval-Python. The corresponding TPR gains are 0.0139 and 0.0410. On MBPP, WLLM obtains a slightly higher AUROC, 0.9612 versus 0.9514, whereas CROSSMARK obtains the highest TPR, 0.8534 versus 0.8521. This difference illustrates why both metrics are needed. AUROC measures ranking over the full threshold range, while TPR reflects separation in the low-false-positive operating region.

The contrast is larger under Qwen2.5-7B-Instruct. Relative to the strongest baseline, CROSSMARK improves AUROC by 0.2099 on C4, 0.1712 on HumanEval-Python, and 0.1297 on MBPP. TPR improves by 0.3072, 0.1895, and 0.2690, respectively. These results should not be interpreted as complete backbone invariance. The common token strength $\delta = 3.5$ produces a much weaker token-level signal under Qwen, as reflected by the broadly lower baseline scores. The local and global components provide additional evidence, so weakening the token component does not reduce the joint detector to the same extent. The resulting advantage is therefore a reduced dependence on backbone-specific token-bias calibration.

Generation quality. The detection gains do not coincide with a systematic deterioration in output quality. On C4, CROSSMARK remains close to the unwatermarked reference under both backbones. Its BERTScore/PPL values are 0.890/5.96 under Llama and 0.939/5.12 under Qwen, compared with

Table 1: Clean-generation results on C4, HumanEval-Python, and MBPP. TPR denotes $\mathrm{TPR@FPR} \leq 5\%$. Higher values are better for AUROC, TPR, BERTScore, and Pass@1, while lower PPL is better. Baseline strength is $\delta = 3.5$ where applicable. Best results among watermarked methods within each backbone block are shown in bold. A dash denotes a metric that is not applicable to unwatermarked text.

Method	C4				HumanEval-Py			MBPP		
	AUROC	TPR	BERTScore	PPL	AUROC	TPR	Pass@1	AUROC	TPR	Pass@1
Llama-3.1-8B-Instruct										
No watermark	–	–	0.891	5.82	–	–	0.677	–	–	0.486
WLLM	0.9658	0.7264	0.861	7.94	0.9343	0.5765	0.530	**0.9612**	0.8521	0.366
SWEET	0.9182	0.5137	0.878	6.87	0.8720	0.4156	0.494	0.8990	0.6732	0.354
Unigram	0.9315	0.5849	0.852	8.31	0.8814	0.4738	0.470	0.8304	0.5447	0.300
ACW	0.9581	0.6628	**0.895**	6.52	0.8088	0.3241	0.518	0.8459	0.4035	0.416
SIR	0.9474	0.6152	0.869	7.15	0.8384	0.3687	0.476	0.8704	0.4397	0.296
SemStamp	0.9727	0.6981	0.887	6.08	0.8126	0.3529	0.442	0.8385	0.4123	0.338
CROSSMARK	**0.9796**	**0.7403**	0.890	**5.96**	**0.9534**	**0.6175**	**0.557**	0.9514	**0.8534**	**0.442**
Qwen2.5-7B-Instruct, baselines at $\delta = 3.5$										
No watermark	–	–	0.945	5.14	–	–	0.811	–	–	0.638
WLLM	0.6289	0.1706	0.917	7.01	0.6117	0.1507	0.780	0.6215	0.1804	0.623
SWEET	0.6154	0.1287	0.927	6.07	0.5746	0.1108	**0.811**	0.6123	0.1602	0.661
Unigram	0.6493	0.1568	0.911	7.34	0.5634	0.0906	**0.811**	0.6273	0.1907	0.658
ACW	0.6471	0.1595	0.937	5.76	0.5260	0.0609	0.793	0.5932	0.1206	**0.669**
SIR	0.6876	0.2553	0.921	6.31	0.6071	0.1405	0.799	0.6316	0.2008	0.615
SemStamp	0.7118	0.2732	0.932	5.37	0.5984	0.1326	0.745	0.6087	0.1715	0.552
CROSSMARK	**0.9217**	**0.5804**	**0.939**	**5.12**	**0.7829**	**0.3402**	0.805	**0.7613**	**0.4698**	0.650

0.891/5.82 and 0.945/5.14 without a watermark. Code utility varies across datasets, but CROSSMARK remains competitive with the watermarked baselines. We therefore interpret the quality results conservatively. They show that the added levels do not impose a uniform quality penalty, rather than that higher-level watermarking improves generation quality.

Independent threshold transfer. To verify that the clean-result gains do not depend on selecting an operating threshold from each target test ROC, we additionally calibrate thresholds on a separate calibration set and transfer them unchanged to the target datasets. The realized FPR remains close to the nominal 5% operating point across all six backbone–target combinations (4.8–5.5%), and the larger CROSSMARK–WLLM gap under Qwen persists under this deployment-style setting. Full transferred-threshold results with 95% confidence intervals are reported in Appendix D.1.

5.3 ROBUSTNESS UNDER INSERTION AND REWRITE

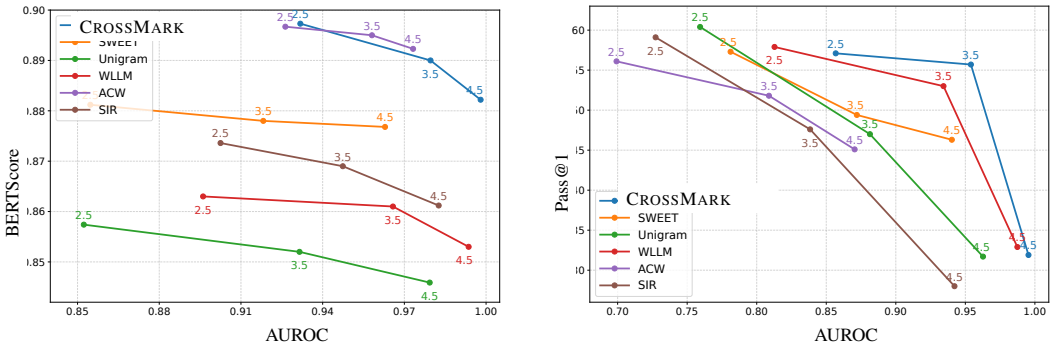
Table 2 places natural-language and code robustness results in adjacent column groups. The row blocks indicate the attack family. Under *Insertion*, C4 uses CP-50 mixing and ClassEval uses behavior-preserving statement insertion. Under *Rewrite*, C4 is rewritten by Llama-2-7B-Chat (Touvron et al., 2023) and ClassEval is rewritten at the whole-class level by CodeLlama-13B-Instruct (Roziere et al., 2023).

Insertion robustness. Under local-unit insertion, CROSSMARK achieves the highest AUROC and TPR in both domains. On C4, it improves over the strongest baseline from 0.884 to 0.913 AUROC and from 0.416 to 0.490 TPR. On ClassEval, the gains are larger, from 0.812 to 0.861 AUROC and from 0.312 to 0.402 TPR. Inserted sentences or statements add evidence that is approximately random with respect to the secret key. Token-only evidence is therefore diluted as the inserted content grows. The local hash component counters this effect in two ways. It directly contributes local evidence and identifies units that should receive less token weight or be removed before filtered global verification. The larger gains in TPR than in AUROC indicate that this interaction is particularly useful in the upper tail of the negative-score distribution.

Rewrite robustness. The global component becomes more important under LLM rewriting. SemStamp is the strongest baseline in both domains, which confirms the value of semantic evidence over token identity. CROSSMARK nevertheless raises C4 AUROC from 0.831 to 0.887 and ClassEval AUROC from 0.712 to 0.802. The ClassEval TPR gain is especially pronounced, increasing from 0.198

Table 2: Robustness under insertion and rewrite attacks. Natural-language and code results are reported side by side. BERT denotes BERTScore. Best results within each attack block are shown in bold.

Method	C4			ClassEval		
	AUROC	TPR@5% FPR	BERT	AUROC	TPR@5% FPR	Pass@1
Insertion						
WLLM	0.842	0.384	0.812	0.812	0.312	0.580
SWEET	0.803	0.247	0.828	0.745	0.201	0.577
Unigram	0.884	0.416	0.805	0.768	0.243	0.567
ACW	0.866	0.352	**0.841**	0.701	0.156	0.573
SIR	0.851	0.338	0.819	0.732	0.198	0.570
SemStamp	0.846	0.361	0.836	0.725	0.187	0.557
CROSSMARK	**0.913**	**0.490**	0.825	**0.861**	**0.402**	**0.597**
Rewrite						
WLLM	0.674	0.199	0.846	0.650	0.163	0.543
SWEET	0.684	0.145	0.849	0.635	0.121	0.543
Unigram	0.772	0.221	0.842	0.671	0.131	0.530
ACW	0.707	0.132	0.861	0.572	0.059	0.520
SIR	0.680	0.143	0.852	0.585	0.086	0.537
SemStamp	0.831	0.381	**0.872**	0.712	0.198	0.517
CROSSMARK	**0.887**	**0.410**	0.867	**0.802**	**0.318**	**0.577**



(a) Natural-language generation. **(b) Code generation.**

Figure 2: Quality–detectability trade-offs under method-native watermark-strength sweeps. Points closer to the upper-right corner provide stronger detection while preserving generation quality. SemStamp is omitted because it does not expose a directly comparable continuous strength parameter.

to 0.318. Whole-class code rewriting can replace identifiers, statements, and control flow at once, so little of the original token-level pattern may remain. The rewrite-contrastive global representation instead tests whether the rewritten function remains in the key-preferred generation-path category. This difference explains why the relative gain is larger for code rewriting than for natural-language rewriting.

The robustness gains are not obtained by accepting clearly lower-quality attacked outputs. CROSSMARK retains competitive BERTScore on C4 and achieves the highest ClassEval Pass@1 under both insertion and rewriting. These utility metrics do not establish that the attacks are equally strong for every method, but they show that the detection advantage is not accompanied by a lower rate of valid code outputs.

5.4 QUALITY–DETECTABILITY TRADE-OFF

We vary each method using its native watermark-strength parameter. For natural language, the comparison uses C4 generations from Llama-3.1-8B-Instruct and plots BERTScore against AUROC. For code, the comparison uses HumanEval-Python generations from Llama-3.1-8B-Instruct and plots Pass@1 against AUROC.

Table 3: Ablation of the token, local, and global watermark components. The *Insertion* and *Rewrite* blocks use the same attacks as Table 2. Best results within each condition block are shown in bold. Rows marked † are detection-only controls evaluated on the same Full outputs; rows marked ‡ are WLLM multi-path controls. Generation-quality entries are omitted for these control rows.

Configuration	C4			ClassEval		
	AUROC	TPR@5% FPR	BERT	AUROC	TPR@5% FPR	Pass@1
No attack						
Token only	0.884	0.602	0.870	0.856	0.506	0.605
Local only	0.812	0.499	0.860	0.789	0.421	0.598
Global only	0.836	0.533	0.858	0.812	0.449	0.600
Token + Local	0.932	0.671	0.878	0.901	0.563	0.610
Token + Global	0.922	0.657	0.875	0.892	0.552	0.608
Local + Global	0.874	0.588	0.865	0.847	0.495	0.602
Independent sum†	0.966	0.696	–	0.914	0.566	–
+ Token weighting†	0.971	0.720	–	0.924	0.604	–
+ Global filtering†	0.974	0.718	–	0.935	0.597	–
Full	**0.980**	**0.740**	**0.890**	**0.945**	**0.620**	**0.615**
Insertion						
Token only	0.707	0.270	0.798	0.681	0.226	0.578
WLLM 1-path‡	0.842	0.384	–	0.812	0.312	–
WLLM 8-path‡	0.861	0.417	–	0.808	0.339	–
Local only	0.686	0.248	0.802	0.663	0.208	0.572
Global only	0.645	0.204	0.793	0.626	0.173	0.568
Token + Local	0.872	0.446	0.815	0.825	0.367	0.588
Token + Global	0.769	0.336	0.810	0.735	0.279	0.585
Local + Global	0.748	0.314	0.806	0.717	0.261	0.575
Independent sum†	0.862	0.400	–	0.811	0.318	–
+ Token weighting†	0.876	0.448	–	0.822	0.378	–
+ Global filtering†	0.880	0.447	–	0.834	0.350	–
Full	**0.913**	**0.490**	**0.825**	**0.861**	**0.402**	**0.597**
Rewrite						
Token only	0.558	0.104	0.845	0.545	0.090	0.555
Local only	0.577	0.122	0.848	0.560	0.104	0.550
Global only	0.810	0.338	0.855	0.742	0.264	0.565
Token + Local	0.616	0.158	0.858	0.591	0.130	0.560
Token + Global	0.829	0.356	0.862	0.757	0.278	0.570
Local + Global	0.837	0.363	0.860	0.763	0.283	0.568
Independent sum†	0.846	0.386	–	0.784	0.280	–
+ Token weighting†	0.857	0.395	–	0.786	0.292	–
+ Global filtering†	0.891	0.407	–	0.796	0.323	–
Full	**0.887**	**0.410**	**0.867**	**0.802**	**0.318**	**0.577**

Analysis. Figure 2 shows that increasing the native watermark strength improves detection for every method but generally reduces output quality. The proposed framework shifts this frontier rather than eliminating the trade-off. On C4, CROSSMARK reaches the highest-AUROC region while retaining BERTScore above 0.88 across the sweep. At comparable quality, its points lie to the right of the token-oriented baselines, which indicates that local and global evidence reduce the amount of token bias needed to reach a target detection level.

The same pattern appears on HumanEval-Python, although functional correctness falls sharply for all strong token-bias settings. At the moderate $\delta = 3.5$ operating point reported in Table 1, CROSSMARK reaches 0.9534 AUROC with 0.557 Pass@1. WLLM reaches 0.9343 AUROC with 0.530 Pass@1. At the strongest setting, both methods approach saturated AUROC but lose a substantial fraction of correct programs. Higher-level evidence therefore moves the quality–detectability frontier outward, but an excessively strong token bias can still dominate the final utility cost.

5.5 MULTI-GRANULARITY ABLATION

We evaluate the token, local, and global components individually and in all pairwise combinations. Attack conditions are represented as row blocks, while C4 and ClassEval remain in adjacent column groups.

Granularity-specific roles. Table 3 separates the roles of the three components by removing each level from both generation and detection. Without an attack, token-only watermarking is the strongest single component because it provides the largest number of observations. Pairing token evidence with either higher level improves both AUROC and TPR, and the full configuration further raises AUROC from the strongest pair by 0.048 on C4 and 0.044 on ClassEval. Dense token evidence therefore remains the clean-output foundation, while the other levels add complementary signals.

Insertion changes the most useful pairing. Token + Local is the strongest two-level configuration, reaching 0.872 AUROC on C4 and 0.825 on ClassEval, compared with 0.707 and 0.681 for Token only. Local only is not a strong standalone detector. Its main contribution is the interaction with token evidence through hash-aware weighting and filtering. Global evidence provides a further gain in the full model, increasing AUROC to 0.913 and 0.861. This result supports the proposed cross-level mechanism rather than a simple claim that three independent scores are better than one.

Rewriting produces the opposite hierarchy. Global only reaches 0.810 AUROC on C4 and 0.742 on ClassEval, while Token only and Local only remain near 0.55. Every strong two-level configuration under rewriting contains the global component, and Local + Global is the best pair in both domains. The full model still improves over that pair by 0.050 AUROC on C4 and 0.039 on ClassEval, showing that residual token evidence remains useful even when it is no longer the dominant signal.

Across all six domain–condition comparisons, the full configuration improves AUROC over the strongest two-level generation alternative by 0.036–0.050 and improves TPR by 0.035–0.069. At the same time, quality does not decline systematically as components are added. We do not interpret the occasional quality increases as a causal benefit because each configuration is generated separately.

The † rows isolate the two cross-level operations in the detector on identical Full outputs. On clean outputs, both token weighting and global filtering improve over an independent sum of the three component scores, with the Full detector giving the highest AUROC and TPR overall. Under insertion, token weighting contributes the larger single-step TPR gain, while adding both operations gives the strongest result. Under rewriting, global filtering is the more useful interaction: on C4, adding it to the independent sum raises AUROC from 0.846 to 0.891, whereas token weighting alone reaches 0.857. The small non-monotonic differences between the single-interaction and Full rows under rewriting indicate that the contribution of each cross-level operation is attack dependent rather than uniformly additive.

Compute-matched multi-path control. The ‡ rows in Table 3 separate the benefit of multi-path search from the added local and global constraints under 50% insertion. Increasing WLLM from one path to eight raises C4 AUROC/TPR from 0.842/0.384 to 0.861/0.417, but does not account for the Full result. At nearly identical realized candidate-token cost (7.06$\times$ for WLLM 8-path versus 7.12$\times$ for Full CROSSMARK), Full reaches 0.913/0.490 on C4 and 0.861/0.402 on ClassEval. The additional robustness therefore cannot be attributed solely to spending the same multi-path search budget on a token-level watermark.

Additional analyses. Semantic-clusterer training dynamics and measured runtime results are reported in Appendix E and Appendix F, respectively.

6 CONCLUSION

We introduced CROSSMARK, a multi-granularity watermarking framework that combines token-level red–green evidence, keyed local-unit preferences, and a rewrite-contrastive global semantic signal. The three levels are injected through a shared multi-path generation process and interact again during detection, where local hash evidence weights token statistics and filters the input to one branch of global verification. Experiments on natural-language and code generation show that the resulting detector remains effective on clean outputs and provides complementary robustness to local-unit insertion and single-pass LLM rewriting. The ablations further identify distinct roles for the three levels. Token evidence supplies dense observations, local evidence is most useful against insertion, and global evidence provides the main signal after extensive rewriting.

Ethical Considerations

Watermarking can support content provenance, auditing, and responsible use of large language model outputs. However, detection errors may lead to incorrect attribution, so a CrossMark score should not be used as the sole basis for punitive or high-stakes decisions. The method also has dual-use implications: watermarking can enable legitimate tracing, while knowledge of the detection mechanism may help an adversary weaken or remove the signal. CrossMark detects a key-dependent generation pattern and does not establish the identity or intent of a human author.

AI Use Statement

Generative AI tools were used only for formatting checks and English grammar and language proofreading. All scientific content, methodology, experiments, analyses, and conclusions were produced and verified by the authors.

References

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

Anirudh Ajith, Sameer Singh, and Danish Pruthi. Downstream trade-offs of a family of text watermarks. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pp. 14039–14053, Miami, Florida, USA, November 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.findings-emnlp.821. URL https://aclanthology.org/2024.findings-emnlp.821/.

Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, Misha Denil, Nal Sepassi, Nora Kassner, Charles Peng, Itamar Pieters, et al. Program synthesis with large language models. *arXiv preprint arXiv:2108.07732*, 2021.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021.

Xueying Du, Mingwei Liu, Kaixin Wang, Hanlin Wang, Junwei Liu, Yixuan Chen, Jiayi Feng, Chaofeng Sha, Xin Peng, and Yiling Lou. ClassEval: A manually-crafted benchmark for evaluating LLMs on class-level code generation. *arXiv preprint arXiv:2308.01861*, 2023. doi: 10.48550/arXiv.2308.01861.

Abhimanyu Dubey et al. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024. doi: 10.48550/arXiv.2407.21783.

Aaron Gokaslan and Vanya Cohen. Openwebtext corpus. Open-source reproduction of the WebText corpus, 2019. URL https://skylion007.github.io/OpenWebTextCorpus/.

Noah Golowich and Ankur Moitra. Edit distance robust watermarks via indexing pseudo-random codes. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2024. URL https://proceedings.neurips.cc/paper_files/paper/2024/hash/24c53bfa5b53fc2cf05644f5a7a26bb0-Abstract-Conference.html. NeurIPS 2024, Proceedings page lists DOI 10.52202/079017-0652.

Batu Guan, Yao Wan, Zhangqian Bi, Zheng Wang, Hongyu Zhang, Pan Zhou, and Lichao Sun. Codeip: A grammar-guided multi-bit watermark for large language models of code. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pp. 9243–9258, 2024.

Ari Holtzman, Jan Buys, Li Du, Maxwell Forbes, and Yejin Choi. The curious case of neural text degeneration. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=rygGQyrFvH.

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 *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pp. 4067–4082, Mexico City, Mexico, June 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.naacl-long.226. URL https://aclanthology.org/2024.naacl-long.226/.

Zhengmian Hu, Lichang Chen, Xidong Wu, Yihan Wu, Hongyang Zhang, and Heng Huang. Unbiased watermark for large language models. In *International Conference on Learning Representations (ICLR)*, 2024. URL https://openreview.net/forum?id=uWVC5FVidc.

Hamel Husain, Ho-Hsiang Wu, Tiferet Gazit, Miltiadis Allamanis, and Marc Brockschmidt. CodeSearchNet challenge: Evaluating the state of semantic code search. *arXiv preprint arXiv:1909.09436*, 2019. doi: 10.48550/arXiv.1909.09436.

Jungin Kim, Shinwoo Park, and Yo-Sub Han. Marking code without breaking it: Code watermarking for detecting llm-generated code. *arXiv preprint arXiv:2502.18851*, 2025. Accepted to Findings of EACL 2026.

John Kirchenbauer, Jonas Geiping, Yuxin Wen, Jonathan Katz, Ian Miers, and Tom Goldstein. A watermark for large language models. *arXiv preprint arXiv:2301.10226*, 2023a.

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. *arXiv preprint arXiv:2306.04634*, 2023b.

Rohith Kuditipudi, John Thickstun, Tatsunori Hashimoto, and Percy Liang. Robust distortion-free watermarks for language models. *arXiv preprint arXiv:2307.15593*, 2023.

Taehyun Lee, Seokhee Hong, Jaewoo Ahn, Ilgee Hong, Hwaran Lee, Sangdoo Yun, Jamin Shin, and Gunhee Kim. Who wrote this code? watermarking for code generation. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 4890–4911. Association for Computational Linguistics, 2024. URL https://aclanthology.org/2024.acl-long.268.

Chaofan Li, Jianlyu Chen, Yingxia Shao, Defu Lian, and Zheng Liu. Towards a generalist code embedding model based on massive data synthesis. *arXiv preprint arXiv:2505.12697*, 2025. doi: 10.48550/arXiv.2505.12697.

Aiwei Liu, Leyi Pan, Xuming Hu, Shiao Meng, and Lijie Wen. A semantic invariant robust watermark for large language models. *arXiv preprint arXiv:2310.06356*, 2023.

Jianmo Ni, Gustavo Hernández Ábrego, Noah Constant, Ji Ma, Keith B. Hall, Daniel Cer, and Yinfei Yang. Sentence-T5: Scalable sentence encoders from pre-trained text-to-text models. In *Findings of the Association for Computational Linguistics: ACL 2022*, pp. 1864–1874, 2022. doi: 10.18653/v1/2022.findings-acl.146.

Jipeng Qiang, Shiyu Zhu, Yun Li, Yi Zhu, Yunhao Yuan, and Xindong Wu. Natural language watermarking via paraphraser-based lexical substitution. *Artificial Intelligence*, pp. 103859, 2023.

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.

Baptiste Roziere, Jonas Gehring, Fabian Gloeckle, Sten Sootla, Itai Gat, Xiaoqing Ellen Tan, Yossi Adi, Jingyu Liu, Romain Sauvestre, Tal Remez, Jeremy Rapin, Artyom Kozhevnikov, Ivan Evtimov, Joanna Bitton, Manish Bhatt, Cristian Canton Ferrer, Aaron Grattafiori, Wenhan Xiong, Alexandre Defossez, Jade Copet, Faisal Azhar, Hugo Touvron, Louis Martin, Nicolas Usunier, Thomas Scialom, and Gabriel Synnaeve. Code llama: Open foundation models for code. *arXiv preprint arXiv:2308.12950*, 2023. doi: 10.48550/arXiv.2308.12950.

Qwen Team. Qwen2.5: A party of foundation models, September 2024. URL https://qwenlm.github.io/blog/qwen2.5/.

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.

Yan Wang and et al. Efficient and universal watermarking for llm-generated code detection. *arXiv preprint arXiv:2402.07518*, 2024.

Borui Yang, Mingxuan Ma, Liyao Xiang, Nan Chen, Xin Zhang, Linghe Kong, and Xinghao Jiang. Codegenguard: A watermark for code generation models. In *The Fourteenth International Conference on Learning Representations*.

KiYoon Yoo, Wonhyuk Ahn, Jiho Jang, and Nojun Kwak. Robust natural language watermarking through invariant features. *arXiv preprint arXiv:2305.01904*, 2023.

Tianyi Zhang, Varsha Kishore, Felix Wu, Kilian Q. Weinberger, and Yoav Artzi. BERTScore: Evaluating text generation with BERT. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=SkeHuCVFDr.

Yu Zhang, Shuliang Liu, Xu Yang, and Xuming Hu. Catmark: A context-aware thresholding framework for robust cross-task watermarking in large language models. *arXiv preprint arXiv:2510.02342*, 2025. doi: 10.48550/arXiv.2510.02342.

Xuandong Zhao, Prabhanjan Ananth, Lei Li, and Yu-Xiang Wang. Provable robust watermarking for ai-generated text. *arXiv preprint arXiv:2306.17439*, 2023.

Qinkai Zheng, Xiao Xia, Xu Sui, Haohua Han, Jiangbin Yang, Kehan Guo, Zhanhui Liu, Zhaoxi Zuo, , Fan Wu, et al. Codegeex: A pre-trained model for code generation with 13 billion parameters. In *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pp. 5620–5631, 2023.

A Detailed Formulation and Cross-Granularity Scoring

This section provides the formal definitions and scoring details omitted from the main text.

A.1 Formal Problem Setup

Let x denote a generation request and k a secret master key. A watermark-enabled language model produces an output

$$y^w \sim \mathcal{G}_\theta^w(\cdot \mid x, k),$$

where y^w may be natural-language text or source code. We derive domain-separated keys k_{tok}, k_{loc}, and k_{glob} from k for token-, local-unit-, and global-unit-level watermarking, respectively.

We organize an output at three operational granularities. Tokens form the fine-grained generation sequence. A local unit is a sentence for natural language and a statement or logical code line for source code. A global unit contains a larger semantic context. We use consecutive sentence spans for natural language and complete functions for source code. These definitions are domain-specific and do not assume that natural-language and code structures form an identical linguistic hierarchy. Instead, they specify the units over which watermark evidence is embedded and recovered.

Let

$$y = (u_1, u_2, \ldots, u_M)$$

denote the local-unit sequence extracted from an output. Each local unit u_j consists of one or more tokens. The global-unit extractor constructs

$$\mathcal{G}(y) = \{g_1, g_2, \ldots, g_T\}.$$

For natural language, each g_t is a sliding window of consecutive local units, so neighboring global units may overlap. Overlapping spans reduce dependence on a single segmentation boundary and

allow the detector to reconstruct new candidate spans after suspicious inserted units are removed. For code, global units are complete functions and do not overlap.

We consider two modifications that target different granularities. A local-unit insertion attack inserts new sentences or code statements into a watermarked output. Since the attacker does not know the secret key, the inserted units are not expected to follow the local hash preference. A global rewriting attack uses an LLM to rewrite a text span or function while preserving its intended meaning or functionality. The attacker is assumed not to know the secret key and cannot repeatedly query the detector.

Let $\mathcal{V}(y, x) \in \{0, 1\}$ be a domain-appropriate utility verifier. For natural language, $\mathcal{V}$ represents preservation of the requested content and acceptable generation quality. For code, it represents syntactic validity and functional correctness when executable tests are available. An attack $\mathcal{A}$ produces

$$y^a \sim \mathcal{A}(\cdot \mid y^w, x)$$

subject to

$$\mathcal{V}(y^w, x) = \mathcal{V}(y^a, x) = 1.$$

Given an observed output y and the secret key k, the detector computes a joint score $\mathcal{S}_{\text{joint}}(y, k)$ and predicts that the output is watermarked when

$$\mathcal{S}_{\text{joint}}(y, k) \geq \tau.$$

The joint score is used as a continuous detection score and is not assumed to follow a standard normal distribution. Our objective is to preserve watermark separability after the considered modifications while maintaining output utility:

$$\max \ \Pr[\mathcal{S}_{\text{joint}}(y^a, k) \geq \tau \mid \mathcal{V}(y^w, x) = \mathcal{V}(y^a, x) = 1]\,. \tag{1}$$

A.2 Overview

CrossMark jointly embeds watermark evidence at token, local-unit, and global-unit levels. The three levels are integrated into one multi-path generation process.

At every token step, each active generation path receives a standard red–green list logit bias. When a local unit is completed, its normalized representation is hashed with the local key. Paths whose hash evidence exceeds the random-match mean remain in the candidate pool. Once a global unit becomes available, a rewrite-robust semantic clusterer determines whether the corresponding path belongs to the key-preferred category. Paths satisfying both the local and global preferences remain active and are expanded during the next generation step.

The framework maintains a single candidate-pool width B_{pool} across local-unit boundaries. It does not independently generate a token-watermarked output, a local-watermarked output, and a global-watermarked output. Instead, the same candidate paths accumulate watermark evidence at all three granularities. Candidate quality is controlled by top-p decoding from the base model distribution (Holtzman et al., 2020). We therefore do not introduce an additional weighted objective that combines model likelihood with the local hash score.

During detection, the local hash evidence serves three roles. It provides an independent local-unit score, continuously weights the token-level evidence, and filters suspicious local units before one branch of global semantic verification. The final detector combines the token-, local-, and global-level scores.

A.3 Token-Level Watermarking

We use the standard red–green list watermark (Kirchenbauer et al., 2023a). At token position i, a pseudorandom function parameterized by k_{tok} partitions the vocabulary $\mathcal{W}$ into a green list $\mathcal{W}_i^G$ and its complement:

$$\mathcal{W}_i^G = \text{PRF}\left(k_{\text{tok}}, y_{<i}\right), \qquad |\mathcal{W}_i^G| = \gamma|\mathcal{W}|,$$

where γ is the green-list proportion. Given the original model logit $\ell_i(v)$ for token v, the watermarked logit is

$$\widetilde{\ell}_i(v) = \ell_i(v) + \delta\, \mathbb{I}\big[v \in \mathcal{W}_i^G\big]\,,$$

where δ controls the token-level watermark strength. Top-p filtering and sampling are then applied to the adjusted distribution.

The same token-level procedure is applied independently to every active path in the multi-path candidate pool. We do not skip low-entropy positions. Since higher-level evidence supplements token detection, δ can be set below the strength required by a token-only detector, reducing the extent to which the watermark perturbs the original token distribution.

A.4 KEYED LOCAL-UNIT PREFERENCE

Let $\phi_{\mathrm{loc}}(\cdot)$ be a deterministic local-unit normalization operator. For natural language, it performs only conservative normalization, such as Unicode and whitespace normalization. For code, it additionally preserves language-specific logical-unit boundaries. The operator is not designed to provide invariance to unrestricted stylistic or structural rewriting.

For a completed local unit u_j, we compute

$$\mathbf{h}_j = \mathrm{Trunc}_b\left(\mathrm{HMAC}_{\mathrm{SHA256}}\left(k_{\mathrm{loc}}, \phi_{\mathrm{loc}}(u_j)\right)\right), \qquad \mathbf{h}_j \in \{0,1\}^b.$$

The hash input does not contain the previous local unit or its absolute position. An insertion therefore adds new hash evidence without changing the hash values recovered from the original units.

Let $\mathbf{r} \in \{0,1\}^b$ be a fixed target direction. Any fixed target vector can be used. Our experiments use the all-zero vector as a simple instantiation. The local match count is

$$m_j = \sum_{q=1}^{b} \mathbb{I}[h_{j,q} = r_q], \qquad 0 \le m_j \le b.$$

For an unwatermarked local unit, each keyed bit matches its corresponding target bit with probability approximately $1/2$, giving an expected match count of $b/2$. A local candidate is therefore consistent with the preferred direction when

$$m_j > \frac{b}{2}.$$

Unlike a single-candidate selection rule, this condition defines a retention gate. All paths satisfying Equation A.4 remain available for later global-unit selection. The method does not rank retained paths according to their exact match counts.

A.5 REWRITE-CONTRASTIVE GLOBAL REPRESENTATION

The global watermark requires a representation that separates alternative generation paths while remaining stable when one selected path is rewritten. We train separate natural-language and code encoders because their global-unit structures and representation requirements differ. Both encoders use the same rewrite-contrastive training principle.

For a shared generation context at global position t, the language model produces N_{cand} alternative candidates

$$\mathcal{C}_t = \{g_{t,1}, g_{t,2}, \ldots, g_{t,N_{\mathrm{cand}}}\}.$$

Each candidate is independently rewritten by an LLM, producing

$$\mathcal{B}_t = \{(g_{t,n}, \widetilde{g}_{t,n})\}_{n=1}^{N_{\mathrm{cand}}}.$$

A candidate and its rewrite form a positive pair. Different candidates generated from the same context form hard negatives. The construction therefore treats rewrite equivalence and generation-path distinction as separate supervision signals.

For domain $d \in \{\mathrm{NL}, \mathrm{Code}\}$, an encoder E_ϕ^d, a pooling layer, and a projection network map a normalized global unit g to an embedding $\mathbf{z}(g)$. A shared category head within each domain produces

$$\mathbf{q}(g) = \mathrm{softmax}\left(W\mathbf{z}(g) + \mathbf{b}\right), \qquad a(g) = \arg\max_c q_c(g),$$

where $a(g) \in \{1, \ldots, K\}$. The categories are learned and are not assigned fixed semantic meanings.

We train the clusterer using six complementary objectives:

$$\begin{aligned}\mathcal{L}_{\text{cluster}} = \lambda_{\text{sup}}\mathcal{L}_{\text{sup}} + \lambda_{\text{emb}}\mathcal{L}_{\text{emb}} + \lambda_{\text{cat}}\mathcal{L}_{\text{cat}} \\ + \lambda_{\text{sep}}\mathcal{L}_{\text{sep}} + \lambda_{\text{bal}}\mathcal{L}_{\text{bal}} + \lambda_{\text{conf}}\mathcal{L}_{\text{conf}}.\end{aligned} \tag{2}$$

Here, $\mathcal{L}_{\text{sup}}$ is a supervised contrastive objective that attracts each candidate to its rewrite while separating other same-context paths. The embedding-consistency loss $\mathcal{L}_{\text{emb}}$ directly increases candidate–rewrite similarity. The category-consistency loss $\mathcal{L}_{\text{cat}}$ aligns their predicted category distributions. The path-separation loss $\mathcal{L}_{\text{sep}}$ discourages different same-context candidates from collapsing into the same category. The balance loss $\mathcal{L}_{\text{bal}}$ prevents global category collapse, while the confidence loss $\mathcal{L}_{\text{conf}}$ discourages ambiguous assignments.

A.6 Keyed Global-Category Preference

The global key determines one preferred semantic category for an output. Let d_x be an optional stable task or document identifier. It must remain unchanged under local insertion and global rewriting. The preferred category is

$$a^* = \text{Int}\left(\text{HMAC}_{\text{SHA256}}(k_{\text{glob}}, d_x)\right) \bmod K.$$

When no external identifier is available, a fixed domain string can be used in place of d_x. The same a^* is used for all global units in the output. The preference therefore does not depend on the observed number or position of sentences, statements, spans, or functions.

For a candidate global unit g, the discrete retention condition is

$$a(g) = a^*.$$

During generation, this condition is used as a gate rather than as a continuous likelihood adjustment. All candidates assigned to the preferred category remain available for subsequent generation.

A.7 Cross-Granularity Multi-Path Generation

Let

$$\mathcal{P}_j = \{p_{j,1}, \ldots, p_{j,B_{\text{pool}}}\}$$

denote the active path pool while generating local unit u_j. Every active path is expanded under the token-watermarked distribution in Equation A.3. When the local unit is completed, the generator computes its keyed match count and retains

$$\mathcal{P}_j^{\text{loc}} = \left\{ p \in \mathcal{P}_j : m_j(p) > \frac{b}{2} \right\}.$$

All paths in $\mathcal{P}_j^{\text{loc}}$ survive. The method does not select only the candidate with the largest number of target bits.

For natural language, let a sliding window of r consecutive local units define the most recent global span $g_j(p)$ on path p. Once at least r local units are available, the active pool is further filtered by

$$\mathcal{P}_j^{\text{joint}} = \left\{ p \in \mathcal{P}_j^{\text{loc}} : a(g_j(p)) = a^* \right\}.$$

Because the sentence spans overlap, a new global preference check is performed whenever another local unit is completed. A sentence can therefore contribute to several consecutive global candidates.

The surviving prefixes are expanded again during the next token-generation step until the active pool returns to width B_{pool}. The candidate width is therefore maintained across local-unit boundaries rather than restarted independently for each span.

For code, local-unit filtering is applied whenever a statement or logical line is completed. The surviving paths continue to represent alternative versions of the same partially generated function. The global category gate is applied only when the function is complete:

$$\mathcal{P}_f^{\text{joint}} = \left\{ p \in \mathcal{P}_f^{\text{loc}} : a(g_f(p)) = a^* \right\},$$

where $g_f(p)$ is the completed function on path p. Thus, the same candidate pool accumulates token-level evidence throughout decoding, local hash evidence at statement boundaries, and global semantic evidence at the function boundary.

When several paths remain after the final applicable gate, the base-model likelihood can be used only as a tie-breaking criterion among already valid candidates. It does not trade off watermark preference against generation quality during the local or global filtering stages.

A.8 CROSS-LEVEL WATERMARK DETECTION

Given an observed output, the detector extracts its local units, applies the same conservative normalization, and reconstructs the keyed hash bits. The match count m_j provides both local watermark evidence and a measure of how consistent local unit u_j is with the expected preference.

Local-unit score. Let M be the number of recovered local units and $N = bM$ the total number of recovered bits. Let $n_{\text{loc}} = \sum_{j=1}^{M} m_j$ be the total number of target-bit matches. We compute

$$S_{\text{loc}} = \frac{n_{\text{loc}} - N/2}{\sqrt{N/4}}.$$

We use this quantity as a normalized local evidence score. In the joint detector, it is not interpreted as an independent standard-normal test statistic.

Hash-weighted token score. For local unit u_j, we define the continuous local confidence weight

$$w_j = \frac{m_j}{b}.$$

Let $I_i = 1$ when observed token y_i belongs to its reconstructed green list and $I_i = 0$ otherwise. Let $\mathcal{I}_j$ denote the token positions contained in u_j. The hash-weighted token score is

$$S_{\text{tok}} = \frac{\sum_{j=1}^{M} w_j \sum_{i \in \mathcal{I}_j} (I_i - \gamma)}{\sqrt{\gamma(1-\gamma) \sum_{j=1}^{M} w_j^2 |\mathcal{I}_j|}}.$$

Local units with stronger keyed hash evidence therefore contribute more to token-level detection. Units with near-random local evidence are down-weighted rather than necessarily removed.

Full-view global score. For an observed global unit g_t, let $\boldsymbol{\ell}_t$ be its clusterer logits. We measure its alignment with the preferred category using the logit margin

$$R(g_t, a^*) = \ell_{t,a^*} - \frac{1}{K-1} \sum_{c \neq a^*} \ell_{t,c}.$$

Since category logits may have different null distributions, the detector estimates a mean μ_a and standard deviation σ_a for each preferred category from the corresponding unwatermarked model generations. It computes

$$z_t = \frac{R(g_t, a^*) - \mu_{a^*}}{\sigma_{a^*}}.$$

If the full observed output yields T_{full} global units, its full-view global score is

$$S_{\text{glob}}^{\text{full}} = \frac{1}{\sqrt{T_{\text{full}}}} \sum_{t=1}^{T_{\text{full}}} z_t^{\text{full}}.$$

Hash-filtered global score. For global semantic verification, local hash evidence is used as a binary filter rather than a continuous weight. We retain

$$u_j \quad \text{iff} \quad m_j > \frac{b}{2}.$$

This produces a filtered local-unit sequence y^{filter}.

For natural language, the detector removes local units that fail Equation A.8 and then reconstructs overlapping sentence spans from the remaining sequence. It does not preserve the span boundaries observed before filtering. Reconstructing the sliding windows after removal allows originally separated local units to become adjacent again when an inserted sentence is filtered out.

For code, the detector preserves the function boundary and removes suspicious statements within the function before semantic encoding. The resulting text remains a discrete filtered view. Local units are either retained or removed. They are not assigned continuous weights before being passed to the semantic clusterer.

If the filtered output yields T_{filter} global units, we compute

$$S_{\text{glob}}^{\text{filter}} = \frac{1}{\sqrt{T_{\text{filter}}}} \sum_{t=1}^{T_{\text{filter}}} z_t^{\text{filter}}.$$

The final global score averages the complete and filtered views:

$$S_{\text{glob}} = \frac{S_{\text{glob}}^{\text{full}} + S_{\text{glob}}^{\text{filter}}}{2}.$$

Joint detection score. The final multi-granularity score is

$$\mathcal{S}_{\text{joint}}(y, k) = S_{\text{tok}} + S_{\text{loc}} + S_{\text{glob}}.$$

The three components are correlated because local hash evidence participates in token weighting and global filtering. We therefore treat $\mathcal{S}_{\text{joint}}$ only as a continuous detection score. We do not assign it a standard-normal probability interpretation.

B Detailed Experimental Configuration

B.1 Datasets, Models, Baselines, and Metrics

We evaluate natural-language watermarking on C4 and code watermarking on HumanEval-Python, HumanEval-Java, HumanEval-C++, MBPP, and ClassEval. C4 is used for the main natural-language clean-generation and robustness experiments. HumanEval-Python is used for the main code-generation comparison and the quality–detectability analysis. MBPP provides an additional cross-dataset evaluation, while HumanEval-Java and HumanEval-C++ evaluate cross-language transfer. ClassEval is used for behavior-preserving statement insertion and whole-class rewriting.

Clean-generation experiments use Llama-3.1-8B-Instruct (Dubey et al., 2024) and Qwen2.5-7B-Instruct (Team, 2024). Natural-language rewriting uses Llama-2-7B-Chat (Touvron et al., 2023), and whole-class code rewriting uses CodeLlama-13B-Instruct (Roziere et al., 2023). Each output is rewritten once. Natural-language continuations and single-function code generations are capped at 400 new tokens. All generation and rewriting experiments use random seed 42, while other decoding parameters retain the defaults of the corresponding model or public implementation.

Evaluation sample construction. For C4, we sample 1,000 dataset prompts. For HumanEval-Python, HumanEval-Java, HumanEval-C++, MBPP, and ClassEval, we use every instance in the corresponding benchmark. Each instance produces one watermarked positive sample and one unwatermarked model generation as the negative sample under the same backbone, dataset prompt, and decoding configuration. Where an attack is evaluated, the corresponding watermarked output is transformed once to produce one attacked sample. The point estimates in the main tables use the original experimental runs. Appendix D.6 additionally reports uncertainty over three independently sampled keys and two decoding seeds per key. C4 uses 1,000 paired positive and negative samples, while the other datasets use the complete benchmark size. Attacked sample counts equal the number of evaluated watermarked positives.

We compare CrossMark with WLLM, SWEET, Unigram, ACW, SIR, and SemStamp. Their public implementations are used without a fixed repository commit. Methods that expose a watermark-strength parameter use 3.5, while all other parameters retain their default values. A blank table entry indicates that the corresponding result is unavailable or was not evaluated.

Detection is measured with AUROC and TPR at an empirical $\text{FPR} \leq 5\%$ operating point. Natural-language quality is measured with BERTScore (Zhang et al., 2020) against the reference continuation and perplexity, where higher BERTScore and lower perplexity are preferred. Code correctness is measured with Pass@1. Efficiency is measured as per-sample generation and detection wall-clock time, together with peak memory, under the same hardware and inference configuration for the Llama-3.1-8B-Instruct + C4 setting.

Table 4: Domain-specific instantiations of the multi-granularity watermarking framework.

Setting	Natural language	Code
Local unit	One complete sentence	One executable statement or logical line
Global unit	Three-sentence span with overlapping windows	One complete function
Green-token proportion	$\gamma = 0.5$	$\gamma = 0.5$
Token strength	$\delta_{\mathrm{NL}} = 3.5$	$\delta_{\mathrm{Code}} = 3.5$
Hash function	HMAC-SHA-256	HMAC-SHA-256
Hash target	All-zero b-bit string	All-zero b-bit string
Hash length	$b = 3$	$b = 3$
Candidate acceptance	$m_j > b/2$	$m_j > b/2$
Semantic clusters	$K = 2$	$K = 2$
Active candidate pool	$B_{\mathrm{pool}} = 8$	$B_{\mathrm{pool}} = 8$
Semantic encoder	`sentence-transformers/sentence-t5-xl` (Ni et al., 2022)	`BAAI/bge-code-v1` (Li et al., 2025)
Clusterer training corpus	100,000 OpenWebText contexts (Gokaslan & Cohen, 2019)	100,000 Python functions or methods from CodeSearchNet (Husain et al., 2019)
Clusterer candidates per context	$N_{\mathrm{cand}} = 16$	$N_{\mathrm{cand}} = 16$
Rewrites per candidate	1	1
Candidate and rewrite model	Llama-3.1-8B-Instruct	Llama-3.1-8B-Instruct
Optimization schedule	One epoch, 100,000 optimization steps, seed 42	One epoch, 100,000 optimization steps, seed 42
Maximum new tokens	400	400
Other decoding parameters	Model defaults	Model defaults

B.2 Domain-Specific Watermark Settings

B.3 Insertion, Rewriting, and Runtime Settings

CP-50 is defined by line count. For a source containing L lines, the attack inserts $\lfloor 0.5L \rfloor$ additional lines at randomly sampled line positions. Natural-language insertion draws the added lines from original texts in the same dataset. Code insertion adds behavior-preserving dead-code statements. Rewriting is applied once to each attacked output. The generation and rewriting models use a maximum of 400 new tokens and random seed 42, while all remaining decoding parameters use their default values.

All experiments are conducted on three NVIDIA RTX 5090 GPUs with an x86_64 CPU and 251.55 GB RAM. The environment uses Linux 7.0.0-28-generic with glibc 2.39, Python 3.11.15, CUDA 13.0, PyTorch 2.9.0+cu130, and Transformers 4.44.2.

C Additional Code-Generation Results

Table 5 reports clean-generation results on HumanEval-Java and HumanEval-C++. These datasets are moved to the appendix because the main comparison focuses on one natural-language dataset and two representative code datasets.

Cross-language results. Table 5 shows that a clusterer trained only on Python functions transfers beyond its training language, but the transfer is not uniform. Under Llama, CrossMark obtains the best HumanEval-C++ detection, reaching 0.9059 AUROC and 0.6133 TPR. On HumanEval-Java, however, ACW is stronger, with 0.8668 AUROC and 0.4523 TPR compared with 0.8002 and 0.3371 for CrossMark. The Pass@1 of CrossMark is higher than ACW in this setting, 0.378 versus 0.354, so the detection gap cannot be explained only by lower functional correctness. This result bounds the cross-language claim. The Python-trained semantic categories do not transfer equally well to every model–language combination.

Table 5: Clean-generation results on HumanEval-Java and HumanEval-C++. TPR denotes $\mathrm{TPR@FPR} \leq 5\%$. Higher values are better for all metrics. Code baselines use $\delta = 3.5$. Best results among watermarked methods within each backbone block are shown in bold.

Method	HumanEval-Java			HumanEval-C++		
	AUROC	TPR	Pass@1	AUROC	TPR	Pass@1
Llama-3.1-8B-Instruct						
No watermark	–	–	0.561	–	–	0.470
WLLM	0.7961	0.3253	0.335	0.9013	0.5918	0.348
SWEET	0.7679	0.3017	**0.415**	0.8580	0.4482	0.354
Unigram	0.8140	0.3864	0.244	0.8223	0.3721	0.171
ACW	**0.8668**	**0.4523**	0.354	0.8047	0.2876	0.293
SIR	0.7406	0.2571	0.360	0.8068	0.3142	0.274
SemStamp	0.6934	0.1783	0.294	0.7851	0.3244	0.306
CROSSMARK	0.8002	0.3371	0.378	**0.9059**	**0.6133**	**0.393**
Qwen2.5-7B-Instruct, baselines at $\delta = 3.5$						
No watermark	–	–	0.689	–	–	0.555
WLLM	0.7020	0.2556	0.640	0.6280	0.1704	0.543
SWEET	0.6711	0.2200	0.665	0.6173	0.1581	0.543
Unigram	0.5813	0.1167	0.659	0.6828	0.2335	0.573
ACW	0.5734	0.1076	0.683	0.5986	0.1366	0.476
SIR	0.6273	0.1696	**0.707**	0.7859	0.3522	**0.579**
SemStamp	0.6872	0.2340	0.589	0.6147	0.1560	0.500
CROSSMARK	**0.8793**	**0.6213**	0.664	**0.7866**	**0.4142**	0.564

Under Qwen, the pattern again reflects the weakness of a fixed token bias across backbones. On HumanEval-Java, CROSSMARK reaches 0.8793 AUROC and 0.6213 TPR, compared with 0.7020 and 0.2556 for the strongest baseline values. On HumanEval-C++, its AUROC is nearly tied with SIR, 0.7866 versus 0.7859, but its TPR is higher, 0.4142 versus 0.3522. These gains indicate that local and global evidence can preserve detectability when the token-level signal induced by $\delta = 3.5$ is weak. Overall, CROSSMARK leads detection in three of the four backbone–language settings, while the Llama–Java exception shows that cross-language semantic transfer remains a limitation rather than a solved problem.

D CALIBRATION, DETECTOR CONTROLS, AND STATISTICAL UNCERTAINTY

D.1 INDEPENDENT THRESHOLD CALIBRATION AND TRANSFER

The clean-result tables in the main text report TPR at an operating point selected from each target test ROC. We additionally evaluate a deployment-style setting in which the detector threshold is fixed on a separate calibration set and then transferred to the target dataset without readjustment. Table 6 reports the realized FPR and TPR at these transferred thresholds for CROSSMARK and WLLM.

Across the six backbone–target combinations, the realized FPR remains close to the nominal 5% operating point (4.8–5.5%). The transferred-threshold TPR also closely tracks the corresponding clean-result operating-point TPR. For example, CROSSMARK reaches 73.4%, 60.9%, and 84.3% TPR under Llama and 57.8%, 33.8%, and 46.8% under Qwen on C4, HumanEval, and MBPP, respectively. The larger CrossMark–WLLM gap under Qwen therefore persists when the threshold is fixed before target evaluation rather than selected from the target ROC.

D.2 FALSE POSITIVES ON HUMAN-WRITTEN AND OUT-OF-DISTRIBUTION DATA

Table 7 applies the fixed calibrated thresholds to human-written negatives and to shifted domains or programming languages under Llama-3.1-8B-Instruct. The closer-domain C4 and Python sets remain near the nominal 5% operating point, whereas arXiv, Java, and C++ generally show higher false-positive rates across methods.

Table 6: Detection with independently calibrated thresholds transferred to the target datasets. FPR and TPR are measured at the fixed transferred threshold; each cell reports the point estimate with a 95% confidence interval.

Backbone	Method	Target	FPR (%) [95% CI] (1 decimal place)	TPR (%) [95% CI] (1 decimal place)
Llama-3.1-8B	CROSSMARK	C4	4.9 [3.3, 6.7]	73.4 [68.1, 78.2]
	WLLM	C4	5.1 [3.6, 6.9]	70.7 [65.1, 76.4]
	CROSSMARK	HumanEval	4.9 [2.0, 9.0]	60.9 [53.4, 68.2]
	WLLM	HumanEval	5.5 [2.4, 9.5]	56.8 [49.2, 64.1]
	CROSSMARK	MBPP	5.2 [3.3, 7.5]	84.3 [80.1, 88.2]
	WLLM	MBPP	4.8 [3.1, 7.1]	83.6 [78.9, 88.1]
Qwen2.5-7B	CROSSMARK	C4	5.1 [3.5, 6.9]	57.8 [52.2, 63.1]
	WLLM	C4	4.9 [3.4, 6.6]	17.2 [12.1, 22.4]
	CROSSMARK	HumanEval	5.5 [2.3, 9.4]	33.8 [26.2, 41.3]
	WLLM	HumanEval	4.9 [2.1, 8.9]	15.2 [9.1, 22.1]
	CROSSMARK	MBPP	4.8 [3.0, 7.2]	46.8 [42.1, 52.2]
	WLLM	MBPP	5.2 [3.4, 7.5]	18.3 [13.9, 23.2]

Table 7: False-positive rates (%) on human-written and out-of-distribution negatives under Llama-3.1-8B-Instruct. Each cell reports the point estimate with a 95% confidence interval under the fixed calibrated threshold.

Method	C4 human FPR [95% CI]	arXiv OOD FPR [95% CI]	Python human FPR [95% CI]	Java OOD FPR [95% CI]	C++ OOD FPR [95% CI]
WLLM	4.9 [3.3, 6.7]	6.8 [4.8, 9.3]	5.5 [3.1, 8.4]	7.1 [5.0, 9.6]	6.9 [4.9, 9.3]
SWEET	5.1 [3.5, 6.9]	7.2 [5.1, 9.7]	4.9 [2.7, 7.7]	6.8 [4.8, 9.2]	7.3 [5.2, 9.8]
Unigram	4.8 [3.2, 6.6]	6.5 [4.5, 8.9]	4.9 [2.7, 7.7]	7.4 [5.3, 9.9]	6.6 [4.6, 9.0]
ACW	5.3 [3.6, 7.2]	7.6 [5.5, 10.0]	5.5 [3.1, 8.4]	7.0 [4.9, 9.4]	7.5 [5.4, 9.9]
SIR	4.7 [3.1, 6.5]	6.9 [4.9, 9.4]	4.9 [2.7, 7.7]	6.4 [4.5, 8.8]	7.1 [5.0, 9.6]
SemStamp	5.2 [3.5, 7.1]	6.6 [4.6, 9.0]	5.5 [3.1, 8.4]	7.3 [5.2, 9.8]	6.8 [4.8, 9.2]
CROSSMARK	5.0 [3.4, 6.8]	6.7 [4.7, 9.1]	4.9 [2.7, 7.7]	6.9 [4.9, 9.3]	6.5 [4.5, 8.9]

For CROSSMARK, FPR is 5.0% on C4 human text and 4.9% on Python human code, but rises to 6.7% on arXiv and 6.9%/6.5% on Java/C++. The same shift is visible across the baselines, suggesting that fixed thresholds are sensitive to domain and language changes in the negative distribution. We therefore do not interpret the nominal 5% calibration target as a distribution-free false- positive guarantee.

D.3 FUSION CONTROLS

Table 8 compares the equal-sum detector used in the paper with two learned linear fusion rules on clean outputs. The alternatives occasionally improve one metric, but no learned rule dominates across backbones and targets; the simple equal-sum fusion remains competitive without requiring a learned combiner.

The differences among fusion rules are small relative to the much larger cross-backbone differences in Table 1. On Llama, equal summation has the highest HumanEval AUROC and essentially matches the alternatives on C4 TPR, while regularized logistic fusion gives a modest MBPP AUROC/TPR increase. On Qwen, logistic fusion improves C4 and HumanEval point estimates, but does not improve MBPP AUROC and only changes MBPP TPR by 0.002. Shrinkage LDA likewise helps selected cells without producing a consistent cross-target advantage. These mixed gains suggest that the detector does not depend on a narrowly tuned learned fusion rule: a fixed equal-weight sum captures most of the available benefit while avoiding an additional fitted combiner.

Table 8: Fusion controls on clean outputs.

Backbone	Target	Fusion rule	AUROC (3 d.p.)	TPR (3 d.p.)
Llama	C4	Equal sum (paper)	0.980	0.740
		Regularized logistic	0.978	0.731
		Shrinkage LDA	0.984	0.739
	HumanEval	Equal sum (paper)	0.953	0.618
		Regularized logistic	0.939	0.624
		Shrinkage LDA	0.936	0.609
	MBPP	Equal sum (paper)	0.951	0.853
		Regularized logistic	0.961	0.856
		Shrinkage LDA	0.940	0.849
Qwen	C4	Equal sum (paper)	0.922	0.580
		Regularized logistic	0.935	0.602
		Shrinkage LDA	0.908	0.585
	HumanEval	Equal sum (paper)	0.783	0.340
		Regularized logistic	0.798	0.348
		Shrinkage LDA	0.789	0.346
	MBPP	Equal sum (paper)	0.761	0.470
		Regularized logistic	0.760	0.472
		Shrinkage LDA	0.746	0.477

Table 9: Local-hash selectivity under 50% insertion. R_{orig} and R_{ins} are the retained fractions of original and inserted local units.

Configuration	$R_{\text{orig}} \uparrow$ (3 d.p.)	$R_{\text{ins}} \downarrow$ (3 d.p.)	AUROC↑ (3 d.p.)	TPR↑ (3 d.p.)
C4: sentence insertion				
CROSSMARK, 3-bit ($m \geq 2$)	0.968	0.518	0.913	0.490
Shuffled local matches	0.819	0.821	0.876	0.424
CROSSMARK, 5-bit ($m \geq 4$)	0.909	0.203	0.937	0.544
WLLM	–	–	0.842	0.384
SemStamp	–	–	0.846	0.361
ClassEval: statement insertion				
CROSSMARK, 3-bit ($m \geq 2$)	0.926	0.480	0.861	0.402
Shuffled local matches	0.766	0.800	0.815	0.342
CROSSMARK, 5-bit ($m \geq 4$)	0.896	0.172	0.885	0.457
WLLM	–	–	0.812	0.312
SemStamp	–	–	0.725	0.187

D.4 LOCAL-HASH SELECTIVITY UNDER INSERTION

Table 9 directly measures whether the local hash retains original local units while rejecting inserted ones. The default 3-bit gate retains 96.8% of original C4 sentences and 92.6% of original ClassEval statements, while retaining only 51.8% and 48.0% of inserted units. Shuffling the locations of local matches removes this separation and also reduces end-to-end detection, indicating that the location information supplied by the keyed local hash is useful rather than merely its aggregate match count.

A stricter 5-bit gate further lowers inserted-unit retention and increases AUROC/TPR in both domains, while also lowering original-unit retention. We report this as a sensitivity result: stronger local selectivity improves the measured insertion detector but makes the gate more selective toward original units as well.

Table 10: Global semantic detector views on clean and 50%-insertion outputs.

	Clean		Insertion	
Global view	AUROC↑ (3 d.p.)	TPR↑ (3 d.p.)	AUROC↑ (3 d.p.)	TPR↑ (3 d.p.)
C4: sentence insertion				
Unfiltered global view	0.836	0.533	0.645	0.204
Hash-filtered global view	0.815	0.502	0.742	0.322
Dual-view global score (paper)	0.848	0.520	0.733	0.274
ClassEval: statement insertion				
Unfiltered global view	0.812	0.449	0.626	0.173
Hash-filtered global view	0.810	0.416	0.713	0.252
Dual-view global score (paper)	0.816	0.455	0.704	0.259

Table 11: 95% confidence intervals for the clean detection metrics in Table 1.

	C4		HumanEval-Python		MBPP	
Method	AUROC (3 d.p.)	TPR (3 d.p.)	AUROC (3 d.p.)	TPR (3 d.p.)	AUROC (3 d.p.)	TPR (3 d.p.)
Llama-3.1-8B-Instruct						
No watermark	–	–	–	–	–	–
WLLM	0.966 [0.937, 0.975]	0.726 [0.667, 0.766]	0.934 [0.920, 0.963]	0.577 [0.507, 0.632]	0.961 [0.949, 0.988]	0.852 [0.790, 0.901]
SWEET	0.918 [0.906, 0.943]	0.514 [0.458, 0.555]	0.872 [0.842, 0.887]	0.416 [0.360, 0.485]	0.899 [0.887, 0.926]	0.673 [0.611, 0.723]
Unigram	0.931 [0.906, 0.943]	0.585 [0.544, 0.643]	0.881 [0.865, 0.912]	0.474 [0.409, 0.528]	0.830 [0.801, 0.842]	0.545 [0.495, 0.607]
ACW	0.958 [0.929, 0.964]	0.663 [0.607, 0.706]	0.809 [0.793, 0.841]	0.324 [0.273, 0.392]	0.846 [0.834, 0.875]	0.404 [0.336, 0.447]
SIR	0.947 [0.940, 0.973]	0.615 [0.558, 0.659]	0.838 [0.821, 0.870]	0.369 [0.320, 0.439]	0.870 [0.859, 0.901]	0.440 [0.393, 0.502]
SemStamp	0.973 [0.964, 0.999]	0.698 [0.639, 0.742]	0.813 [0.800, 0.843]	0.353 [0.302, 0.423]	0.839 [0.827, 0.869]	0.412 [0.366, 0.475]
CROSSMARK	0.980 [0.950, 0.985]	0.740 [0.697, 0.801]	0.953 [0.923, 0.966]	0.618 [0.567, 0.687]	0.951 [0.937, 0.978]	0.853 [0.788, 0.895]
Qwen2.5-7B-Instruct						
No watermark	–	–	–	–	–	–
WLLM	0.629 [0.605, 0.642]	0.171 [0.112, 0.217]	0.612 [0.598, 0.641]	0.151 [0.081, 0.203]	0.622 [0.609, 0.648]	0.180 [0.119, 0.228]
SWEET	0.615 [0.604, 0.643]	0.129 [0.086, 0.185]	0.575 [0.562, 0.607]	0.111 [0.038, 0.160]	0.612 [0.600, 0.639]	0.160 [0.099, 0.208]
Unigram	0.649 [0.621, 0.661]	0.157 [0.099, 0.198]	0.563 [0.531, 0.576]	0.091 [0.024, 0.145]	0.627 [0.601, 0.638]	0.191 [0.126, 0.237]
ACW	0.647 [0.620, 0.661]	0.160 [0.101, 0.200]	0.526 [0.515, 0.561]	0.061 [0.011, 0.130]	0.593 [0.568, 0.607]	0.121 [0.055, 0.168]
SIR	0.688 [0.681, 0.716]	0.255 [0.217, 0.318]	0.607 [0.576, 0.622]	0.141 [0.072, 0.191]	0.632 [0.619, 0.658]	0.201 [0.154, 0.268]
SemStamp	0.712 [0.705, 0.740]	0.273 [0.231, 0.332]	0.598 [0.565, 0.610]	0.133 [0.068, 0.185]	0.609 [0.596, 0.637]	0.172 [0.103, 0.218]
CROSSMARK	0.922 [0.917, 0.952]	0.580 [0.523, 0.625]	0.783 [0.752, 0.795]	0.340 [0.288, 0.407]	0.761 [0.730, 0.773]	0.470 [0.424, 0.533]

D.5 HASH FILTERING AND GLOBAL SEMANTIC VERIFICATION

Table 10 isolates the global semantic score and compares the unfiltered, hash-filtered, and dual-view variants. Hash filtering has little effect on clean AUROC but substantially improves insertion robustness. The dual view trades some of the insertion-only gain for stronger clean performance, providing a compromise across the two conditions.

On C4 insertion, filtering raises the global-only AUROC/TPR from 0.645/0.204 to 0.742/0.322; on ClassEval it raises them from 0.626/0.173 to 0.713/0.252. The dual view is not uniformly the strongest insertion-only variant, but it preserves or improves clean AUROC while retaining much of the filtering gain.

D.6 CONFIDENCE INTERVALS ACROSS KEYS AND SEEDS

For the following tables, each reported result aggregates three independently sampled keys and two decoding seeds per key. The 95% confidence intervals are computed with sample-level bootstrap resampling. These intervals quantify the uncertainty of the reported aggregate metrics and are not used as a separate paired significance test.

Across the repeated keys and seeds, the clean-detection results preserve the backbone-dependent pattern reported in the main text. On Llama-3.1-8B-Instruct, CROSSMARK has the highest AUROC point estimate on C4 and HumanEval-Python and is essentially tied with WLLM on MBPP TPR; several of the corresponding marginal intervals overlap, so we treat these comparisons as differences

Table 12: 95% confidence intervals for the clean quality metrics in Table 1.

Method	C4 BERTScore (3 d.p.)	C4 PPL (2 d.p.)	HumanEval Pass@1 (3 d.p.)	MBPP Pass@1 (3 d.p.)
Llama-3.1-8B-Instruct				
No watermark	0.891 [0.875, 0.892]	5.82 [5.59, 6.05]	0.677 [0.643, 0.732]	0.486 [0.447, 0.536]
WLLM	0.861 [0.852, 0.870]	7.94 [7.73, 8.16]	0.530 [0.478, 0.569]	0.366 [0.314, 0.405]
SWEET	0.878 [0.869, 0.887]	6.87 [6.65, 7.10]	0.494 [0.459, 0.554]	0.354 [0.302, 0.391]
Unigram	0.852 [0.835, 0.853]	8.31 [8.09, 8.52]	0.470 [0.415, 0.504]	0.300 [0.266, 0.359]
ACW	0.895 [0.886, 0.904]	6.52 [6.30, 6.75]	0.518 [0.462, 0.554]	0.416 [0.364, 0.453]
SIR	0.869 [0.868, 0.883]	7.15 [6.91, 7.39]	0.476 [0.439, 0.528]	0.296 [0.258, 0.349]
SemStamp	0.887 [0.870, 0.887]	6.08 [5.88, 6.29]	0.442 [0.407, 0.498]	0.338 [0.303, 0.398]
CROSSMARK	0.890 [0.881, 0.899]	5.96 [5.73, 6.19]	0.557 [0.506, 0.595]	0.442 [0.393, 0.482]
Qwen2.5-7B-Instruct				
No watermark	0.945 [0.936, 0.954]	5.14 [4.93, 5.36]	0.811 [0.775, 0.866]	0.638 [0.602, 0.694]
WLLM	0.917 [0.908, 0.926]	7.01 [6.79, 7.22]	0.780 [0.723, 0.817]	0.623 [0.572, 0.661]
SWEET	0.927 [0.925, 0.942]	6.07 [5.84, 6.29]	0.811 [0.756, 0.843]	0.661 [0.608, 0.699]
Unigram	0.911 [0.909, 0.930]	7.34 [7.13, 7.56]	0.811 [0.755, 0.847]	0.658 [0.622, 0.711]
ACW	0.937 [0.928, 0.946]	5.76 [5.53, 6.00]	0.793 [0.761, 0.850]	0.669 [0.617, 0.708]
SIR	0.921 [0.921, 0.939]	6.31 [6.10, 6.53]	0.799 [0.761, 0.850]	0.615 [0.580, 0.674]
SemStamp	0.932 [0.918, 0.933]	5.37 [5.15, 5.60]	0.745 [0.690, 0.781]	0.552 [0.516, 0.603]
CROSSMARK	0.939 [0.937, 0.954]	5.12 [4.88, 5.37]	0.805 [0.773, 0.862]	0.650 [0.616, 0.708]

Table 13: 95% confidence intervals for the robustness results in Table 2.

Attack	Method	C4			ClassEval		
		AUROC (3 d.p.)	TPR (3 d.p.)	BERTScore (3 d.p.)	AUROC (3 d.p.)	TPR (3 d.p.)	Pass@1 (3 d.p.)
Insertion	WLLM	0.842 [0.814, 0.871]	0.384 [0.339, 0.448]	0.812 [0.806, 0.833]	0.812 [0.787, 0.849]	0.312 [0.243, 0.364]	0.580 [0.541, 0.643]
	SWEET	0.803 [0.764, 0.819]	0.247 [0.198, 0.307]	0.828 [0.807, 0.835]	0.745 [0.723, 0.786]	0.201 [0.147, 0.268]	0.577 [0.517, 0.618]
	Unigram	0.884 [0.866, 0.919]	0.416 [0.370, 0.481]	0.805 [0.800, 0.829]	0.768 [0.747, 0.810]	0.243 [0.190, 0.312]	0.567 [0.511, 0.610]
	ACW	0.866 [0.832, 0.887]	0.352 [0.286, 0.399]	0.841 [0.832, 0.850]	0.701 [0.663, 0.728]	0.156 [0.107, 0.229]	0.573 [0.533, 0.632]
	SIR	0.851 [0.815, 0.870]	0.338 [0.269, 0.383]	0.819 [0.800, 0.823]	0.732 [0.694, 0.759]	0.198 [0.131, 0.252]	0.570 [0.512, 0.609]
	SemStamp	0.846 [0.818, 0.875]	0.361 [0.310, 0.424]	0.836 [0.816, 0.839]	0.725 [0.694, 0.757]	0.187 [0.134, 0.253]	0.557 [0.516, 0.615]
	CROSSMARK	0.913 [0.894, 0.953]	0.490 [0.430, 0.538]	0.825 [0.804, 0.829]	0.861 [0.838, 0.904]	0.402 [0.334, 0.453]	0.597 [0.535, 0.634]
Rewrite	WLLM	0.674 [0.637, 0.697]	0.199 [0.138, 0.247]	0.846 [0.840, 0.865]	0.650 [0.609, 0.674]	0.163 [0.097, 0.214]	0.543 [0.483, 0.584]
	SWEET	0.684 [0.648, 0.709]	0.145 [0.085, 0.194]	0.849 [0.848, 0.874]	0.635 [0.594, 0.657]	0.121 [0.072, 0.191]	0.543 [0.484, 0.587]
	Unigram	0.772 [0.736, 0.791]	0.221 [0.172, 0.283]	0.842 [0.836, 0.862]	0.671 [0.631, 0.694]	0.131 [0.077, 0.196]	0.530 [0.469, 0.573]
	ACW	0.707 [0.690, 0.745]	0.132 [0.087, 0.198]	0.861 [0.836, 0.861]	0.572 [0.549, 0.610]	0.059 [0.001, 0.121]	0.520 [0.480, 0.585]
	SIR	0.680 [0.644, 0.699]	0.143 [0.078, 0.190]	0.852 [0.830, 0.853]	0.585 [0.565, 0.628]	0.086 [0.017, 0.140]	0.537 [0.496, 0.595]
	SemStamp	0.831 [0.811, 0.868]	0.381 [0.331, 0.443]	0.872 [0.867, 0.890]	0.712 [0.687, 0.750]	0.198 [0.143, 0.267]	0.517 [0.477, 0.576]
	CROSSMARK	0.887 [0.871, 0.928]	0.410 [0.363, 0.474]	0.867 [0.864, 0.885]	0.802 [0.778, 0.843]	0.318 [0.247, 0.372]	0.577 [0.522, 0.621]

in point estimates rather than as paired significance claims. The separation is much larger on Qwen2.5-7B-Instruct: CROSSMARK reaches 0.922/0.580 on C4, 0.783/0.340 on HumanEval-Python, and 0.761/0.470 on MBPP, whereas the strongest baseline point estimates remain substantially lower on most detection metrics. The repeated-run intervals therefore support the main observation that the advantage of multi-granularity detection is especially pronounced when the backbone is more challenging for the baseline watermarkers.

The quality intervals show a different pattern from detection: CROSSMARK does not obtain its detection gains by consistently pushing generation quality in a single direction. On Llama, its C4 BERTScore (0.890) is close to the unwatermarked value (0.891), while its perplexity (5.96) is much closer to the unwatermarked baseline (5.82) than several token-level baselines with larger perplexity increases. Its HumanEval and MBPP Pass@1 values remain below the unwatermarked model, but are among the stronger watermarked results. On Qwen, CROSSMARK similarly keeps C4 perplexity essentially at the unwatermarked level (5.12 versus 5.14) and preserves code accuracy within the range occupied by other watermarking methods. These intervals reinforce the intended interpretation of Table 1: detection strength and output quality should be considered jointly, rather than treating small point-estimate differences in either quality metric as a separate statistical claim.

The robustness intervals preserve the attack-dependent ordering seen in the main table. Under insertion, CROSSMARK attains the highest detection point estimates on both C4 (0.913 AUROC, 0.490 TPR) and ClassEval (0.861, 0.402), while output-quality intervals remain comparable to the other attacked watermarked outputs. Rewriting is more destructive for all methods, but the relative benefit of CROSSMARK persists: its AUROC/TPR is 0.887/0.410 on C4 and 0.802/0.318

Table 14: 95% confidence intervals for the ablation results in Table 3.

Setting	Variant	C4			ClassEval		
		AUROC (3 d.p.)	TPR (3 d.p.)	BERTScore (3 d.p.)	AUROC (3 d.p.)	TPR (3 d.p.)	Pass@1 (3 d.p.)
No attack	Token only	0.884 [0.847, 0.905]	0.602 [0.537, 0.646]	0.870 [0.867, 0.890]	0.856 [0.829, 0.896]	0.506 [0.439, 0.558]	0.605 [0.546, 0.647]
	Local only	0.812 [0.792, 0.850]	0.499 [0.453, 0.562]	0.860 [0.838, 0.861]	0.789 [0.746, 0.814]	0.421 [0.354, 0.473]	0.598 [0.557, 0.658]
	Global only	0.836 [0.814, 0.871]	0.533 [0.490, 0.597]	0.858 [0.854, 0.879]	0.812 [0.785, 0.854]	0.449 [0.380, 0.499]	0.600 [0.542, 0.644]
	Token+Local	0.932 [0.892, 0.947]	0.671 [0.611, 0.720]	0.878 [0.856, 0.883]	0.901 [0.880, 0.943]	0.563 [0.497, 0.618]	0.610 [0.565, 0.667]
	Token+Global	0.922 [0.886, 0.941]	0.657 [0.593, 0.702]	0.875 [0.870, 0.898]	0.892 [0.867, 0.930]	0.552 [0.499, 0.620]	0.608 [0.547, 0.648]
	Local+Global	0.874 [0.836, 0.889]	0.588 [0.545, 0.656]	0.865 [0.857, 0.886]	0.847 [0.809, 0.872]	0.495 [0.427, 0.549]	0.602 [0.562, 0.661]
	Independent sum†	0.966 [0.939, 0.994]	0.696 [0.641, 0.754]	–	0.914 [0.883, 0.946]	0.566 [0.506, 0.627]	–
	+ Token weighting†	0.971 [0.941, 0.999]	0.720 [0.667, 0.776]	–	0.924 [0.892, 0.955]	0.604 [0.541, 0.665]	–
	+ Global filtering†	0.974 [0.946, 0.999]	0.718 [0.663, 0.773]	–	0.935 [0.901, 0.966]	0.597 [0.539, 0.656]	–
	Full	0.980 [0.951, 0.999]	0.740 [0.690, 0.805]	0.890 [0.871, 0.894]	0.945 [0.921, 0.987]	0.620 [0.567, 0.686]	0.615 [0.573, 0.672]
Insertion	Token only	0.707 [0.690, 0.749]	0.270 [0.206, 0.315]	0.798 [0.779, 0.804]	0.681 [0.655, 0.721]	0.226 [0.159, 0.278]	0.578 [0.522, 0.621]
	Local only	0.686 [0.668, 0.728]	0.248 [0.188, 0.297]	0.802 [0.797, 0.822]	0.663 [0.621, 0.686]	0.208 [0.156, 0.273]	0.572 [0.528, 0.629]
	Global only	0.645 [0.625, 0.686]	0.204 [0.156, 0.265]	0.793 [0.789, 0.815]	0.626 [0.604, 0.667]	0.173 [0.127, 0.246]	0.568 [0.524, 0.627]
	Token+Local	0.872 [0.855, 0.910]	0.446 [0.400, 0.511]	0.815 [0.794, 0.820]	0.825 [0.802, 0.865]	0.367 [0.315, 0.434]	0.588 [0.527, 0.631]
	Token+Global	0.769 [0.752, 0.807]	0.336 [0.272, 0.383]	0.810 [0.788, 0.813]	0.735 [0.693, 0.754]	0.279 [0.210, 0.331]	0.585 [0.526, 0.631]
	Local+Global	0.748 [0.727, 0.782]	0.314 [0.266, 0.378]	0.806 [0.788, 0.811]	0.717 [0.680, 0.743]	0.261 [0.207, 0.330]	0.575 [0.514, 0.613]
	Independent sum†	0.862 [0.834, 0.891]	0.400 [0.343, 0.455]	–	0.811 [0.781, 0.844]	0.318 [0.258, 0.377]	–
	+ Token weighting†	0.876 [0.845, 0.905]	0.448 [0.394, 0.503]	–	0.822 [0.791, 0.856]	0.378 [0.316, 0.437]	–
	+ Global filtering†	0.880 [0.854, 0.907]	0.447 [0.393, 0.504]	–	0.834 [0.800, 0.869]	0.350 [0.290, 0.409]	–
	Full	0.913 [0.894, 0.953]	0.490 [0.430, 0.538]	0.825 [0.804, 0.829]	0.861 [0.838, 0.904]	0.402 [0.334, 0.453]	0.597 [0.535, 0.634]
Rewrite	Token only	0.558 [0.522, 0.580]	0.104 [0.060, 0.169]	0.845 [0.833, 0.857]	0.545 [0.504, 0.572]	0.090 [0.041, 0.160]	0.555 [0.493, 0.594]
	Local only	0.577 [0.557, 0.614]	0.122 [0.061, 0.168]	0.848 [0.845, 0.870]	0.560 [0.532, 0.601]	0.104 [0.051, 0.170]	0.550 [0.491, 0.593]
	Global only	0.810 [0.790, 0.845]	0.338 [0.297, 0.406]	0.855 [0.854, 0.881]	0.742 [0.723, 0.786]	0.264 [0.214, 0.335]	0.565 [0.505, 0.607]
	Token+Local	0.616 [0.598, 0.653]	0.158 [0.096, 0.205]	0.858 [0.837, 0.865]	0.591 [0.566, 0.629]	0.130 [0.062, 0.183]	0.560 [0.503, 0.604]
	Token+Global	0.829 [0.797, 0.850]	0.356 [0.311, 0.422]	0.862 [0.856, 0.885]	0.757 [0.718, 0.781]	0.278 [0.205, 0.327]	0.570 [0.529, 0.628]
	Local+Global	0.837 [0.799, 0.854]	0.363 [0.297, 0.410]	0.860 [0.859, 0.882]	0.763 [0.737, 0.802]	0.283 [0.229, 0.351]	0.568 [0.511, 0.610]
	Independent sum†	0.846 [0.818, 0.873]	0.386 [0.329, 0.443]	–	0.784 [0.753, 0.816]	0.280 [0.221, 0.342]	–
	+ Token weighting†	0.857 [0.827, 0.884]	0.395 [0.342, 0.449]	–	0.786 [0.755, 0.820]	0.292 [0.230, 0.355]	–
	+ Global filtering†	0.891 [0.863, 0.918]	0.407 [0.352, 0.464]	–	0.796 [0.764, 0.827]	0.323 [0.264, 0.383]	–
	Full	0.887 [0.868, 0.927]	0.410 [0.362, 0.471]	0.867 [0.863, 0.888]	0.802 [0.760, 0.826]	0.318 [0.268, 0.387]	0.577 [0.539, 0.638]

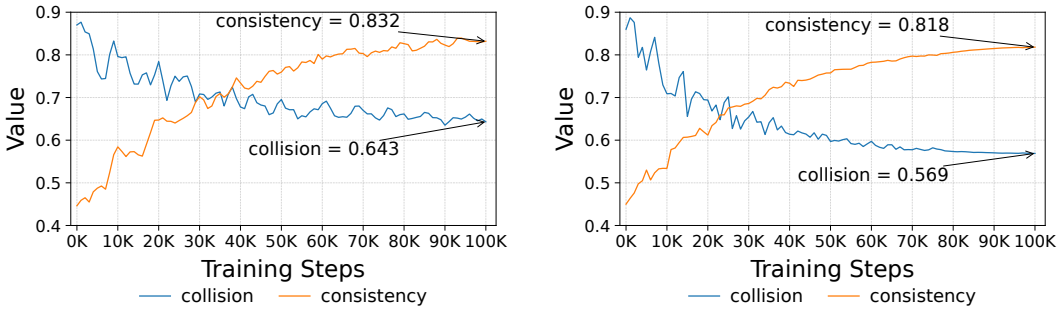
on ClassEval. SemStamp is the closest baseline under rewriting on C4, whereas the gap is larger on ClassEval. The wider TPR intervals under attack also indicate greater sample-level variability than in clean detection, which is expected when the attack changes how much watermark evidence survives in each individual output. We therefore rely on the consistent cross-dataset trend rather than on marginal-interval non-overlap to characterize robustness.

The ablation intervals clarify that the three evidence levels contribute in an attack-dependent manner. Without attack, combining views consistently raises detection above the corresponding single-view variants, and the full model has the strongest point estimates on both datasets. Under insertion, Token+Local is markedly stronger than Token+Global or Local+Global, consistent with the local view being particularly useful for identifying inserted units; the detection-side weighting/filtering steps then provide additional gains on top of the independent sum. Rewriting reverses part of this pattern: the Global-only detector is much stronger than Token-only or Local-only, and adding global filtering produces most of the gain among the detection-side operations. The full model remains competitive across both attack types, but not every incremental operation monotonically improves every metric—for example, under rewriting the global-filtered variant has a slightly higher C4 AUROC than Full, while Full has a slightly higher TPR. This is why we interpret the ablation as evidence of complementary, condition-dependent signals rather than claiming that each component yields an independent additive improvement.

The additional-language intervals further show that performance depends on both backbone and programming language. For Llama, CROSSMARK is strongest on C++ by point estimate (0.906 AUROC and 0.613 TPR), but Java is more mixed: ACW has the highest Java AUROC/TPR point estimates, while several intervals overlap substantially. On Qwen, CROSSMARK shows a large Java detection advantage, reaching 0.879 AUROC and 0.621 TPR; on C++, SIR and CROSSMARK have similar AUROC point estimates, while CROSSMARK has the higher TPR point estimate. Pass@1 does not track detection monotonically across methods or languages, which again separates watermark detectability from code-generation quality. Taken together, these results support cross-language applicability while also showing that the magnitude of the detection advantage is not uniform across all backbone–language combinations.

Table 15: 95% confidence intervals for the additional HumanEval-Java and HumanEval-C++ results.

	HumanEval-Java			HumanEval-C++		
Method	AUROC (3 d.p.)	TPR (3 d.p.)	Pass@1 (3 d.p.)	AUROC (3 d.p.)	TPR (3 d.p.)	Pass@1 (3 d.p.)
Llama-3.1-8B-Instruct						
No watermark	–	–	0.561 [0.510, 0.624]	–	–	0.470 [0.424, 0.533]
WLLM	0.796 [0.752, 0.823]	0.325 [0.248, 0.389]	0.335 [0.294, 0.401]	0.901 [0.874, 0.945]	0.592 [0.528, 0.673]	0.348 [0.304, 0.413]
SWEET	0.768 [0.723, 0.795]	0.302 [0.225, 0.364]	0.415 [0.369, 0.478]	0.858 [0.831, 0.904]	0.448 [0.386, 0.525]	0.354 [0.305, 0.416]
Unigram	0.814 [0.772, 0.844]	0.386 [0.323, 0.462]	0.244 [0.180, 0.289]	0.822 [0.791, 0.865]	0.372 [0.291, 0.430]	0.171 [0.123, 0.234]
ACW	0.867 [0.822, 0.893]	0.452 [0.390, 0.527]	0.354 [0.305, 0.416]	0.805 [0.762, 0.837]	0.288 [0.211, 0.350]	0.293 [0.243, 0.355]
SIR	0.741 [0.716, 0.785]	0.257 [0.196, 0.335]	0.360 [0.295, 0.408]	0.807 [0.767, 0.836]	0.314 [0.252, 0.393]	0.274 [0.211, 0.323]
SemStamp	0.693 [0.671, 0.740]	0.178 [0.100, 0.239]	0.294 [0.245, 0.359]	0.785 [0.742, 0.811]	0.324 [0.263, 0.404]	0.306 [0.256, 0.367]
CROSSMARK	0.800 [0.775, 0.842]	0.337 [0.255, 0.396]	0.378 [0.331, 0.446]	0.906 [0.860, 0.929]	0.613 [0.550, 0.692]	0.393 [0.332, 0.441]
Qwen2.5-7B-Instruct						
No watermark	–	–	0.689 [0.624, 0.733]	–	–	0.555 [0.489, 0.602]
WLLM	0.702 [0.678, 0.747]	0.256 [0.192, 0.333]	0.640 [0.574, 0.686]	0.628 [0.600, 0.669]	0.170 [0.110, 0.249]	0.543 [0.474, 0.588]
SWEET	0.671 [0.631, 0.700]	0.220 [0.162, 0.303]	0.665 [0.602, 0.713]	0.617 [0.596, 0.663]	0.158 [0.081, 0.222]	0.543 [0.479, 0.594]
Unigram	0.581 [0.557, 0.626]	0.117 [0.058, 0.200]	0.659 [0.614, 0.723]	0.683 [0.640, 0.709]	0.234 [0.171, 0.313]	0.573 [0.526, 0.635]
ACW	0.573 [0.547, 0.618]	0.108 [0.025, 0.167]	0.683 [0.621, 0.730]	0.599 [0.572, 0.641]	0.137 [0.073, 0.217]	0.476 [0.410, 0.519]
SIR	0.627 [0.584, 0.655]	0.170 [0.089, 0.230]	0.707 [0.662, 0.769]	0.786 [0.741, 0.812]	0.352 [0.273, 0.418]	0.579 [0.519, 0.628]
SemStamp	0.687 [0.661, 0.733]	0.234 [0.173, 0.312]	0.589 [0.543, 0.652]	0.615 [0.571, 0.644]	0.156 [0.099, 0.238]	0.500 [0.437, 0.548]
CROSSMARK	0.879 [0.849, 0.921]	0.621 [0.564, 0.703]	0.664 [0.603, 0.712]	0.787 [0.762, 0.836]	0.414 [0.353, 0.492]	0.564 [0.515, 0.626]



(a) Natural-language training dynamics. **(b) Code training dynamics.**

Figure 3: Training dynamics of the natural-language and code semantic clusterers. Each panel reports the path-collision rate and rewrite consistency throughout training. Lower collision and higher rewrite consistency indicate better separation between alternative generation paths and greater stability under semantics-preserving rewriting.

E SEMANTIC CLUSTERER ANALYSIS

We examine whether the semantic clusterers simultaneously reduce cross-cluster collisions and preserve cluster assignments after meaning-preserving rewrites.

Training dynamics. Path collision is the fraction of distinct candidate pairs from the same context that receive the same category. Rewrite consistency is the fraction of candidate–rewrite pairs that retain the same category. With $N_{\text{cand}} = 16$ candidates and $K = 2$ balanced categories, collision cannot reach zero. The minimum is $2\binom{8}{2}/\binom{16}{2} = 0.467$, obtained when the candidates divide evenly between the two categories. Collision values should therefore be interpreted relative to this feasible lower bound.

Figure 3 shows two complementary phases. Early training primarily separates alternative generation paths. Natural-language collision falls from approximately 0.87 to 0.70, while code collision falls from approximately 0.86 to below 0.65. Later training produces a steadier rise in rewrite consistency without reversing the earlier separation. At the final checkpoint, natural language reaches 0.643 collision and 0.832 consistency, while code reaches 0.569 collision and 0.818 consistency. The code collision rate is particularly close to the 0.467 lower bound. These trajectories show that rewrite stability is not obtained by collapsing all candidates into one category. The clusterers instead

Table 16: Online computational efficiency for the Llama-3.1-8B-Instruct + C4 main-experiment setting. Generation and detection times are measured per sample.

Method	Generation (s/sample) ↓	Detection (s/sample) ↓	Peak memory (GB) ↓
WLLM	19.94	13.33	18.50
Unigram	16.18	4.89	16.47
SWEET	14.75	11.17	17.31
ACW	25.47	7.78	16.69
SIR	13.44	10.10	17.25
SemStamp	17.52	15.21	21.78
CROSSMARK	18.10	6.72	20.64

```
Continue the following text, ensuring it flows naturally:
{dataset_sample}
```

Listing 1: Natural-language direct generation prompt.

preserve candidate–rewrite assignments while maintaining distinguishable alternatives from the same generation context.

F EFFICIENCY RESULTS

Table 16 reports measured online generation and detection cost for the Llama-3.1-8B-Instruct + C4 main-experiment setting under the same hardware and inference configuration.

Analysis. In this setting, CROSSMARK requires 18.10 s/sample for generation and 6.72 s/sample for detection, with 20.64 GB peak memory. Its generation time is lower than WLLM and ACW but higher than the other measured baselines; its detection time is lower than WLLM, SWEET, SIR, and SemStamp but higher than Unigram. The measurement therefore shows a moderate implementation-level overhead rather than uniformly faster execution. The compute-matched rows in Table 3 separately test whether the detection gain can be explained by additional candidate search.

G GENERATION AND REWRITING PROMPTS

The same direct-generation templates are used for evaluation and for constructing the corresponding clusterer candidates. The code-rewriting experiment uses the final prompt below.

```
# Python Solution Task
# Description: {docstring_content}

# IMPORTANT REQUIREMENTS:
# 1. Write complete, working Python code, enclosed in triple backticks.
# 2. Implement ALL logic within the function(s) below.
# 3. Include ALL necessary import statements at the beginning.
# 4. Do NOT change the function name or signature as specified in the
    problem.
# 5. You may write helper functions, but they must be included within
    the same code block.
# 6. Do NOT include any test code, example calls, or main block.

# Complete the following implementation:
{skeleton}
```

Listing 2: Code generation prompt for Python on HumanEval and MBPP.

```
// C++ Solution Task
// Language: C++17
// Write a correct and efficient C++ function to solve the following
    problem.
// Requirements:
// 1. Provide ONLY C++ code, no explanations.
// 2. Include all necessary #include directives.
// 3. Do NOT write any main() function or test code.
// 4. Implement the required function exactly as specified.

// Problem description and function signature:
{original}
```

Listing 3: Code generation prompt for C++ on HumanEval-X.

```
// Java Solution Task
// Language: Java (JDK 8)
// Write a correct and efficient Java method/class to solve the
    following problem.
// Requirements:
// 1. Provide ONLY Java code, no explanations.
// 2. Include necessary import statements if needed.
// 3. Do NOT write any main method or test code.
// 4. Implement the required method/class exactly as specified.

// Problem description and method/class signature:
{original}
```

Listing 4: Code generation prompt for Java on HumanEval-X.

````
You will be given a Python code snippet containing a class. Your task is
     to perform an extreme rewrite to maximize the lexical and
    structural difference from the original code, while strictly
    preserving its exact input/output behavior and semantics.

You MUST keep the class name and ALL method names completely unchanged.
    The number of methods must remain exactly the same. For absolutely
    EVERYTHING ELSE inside each method body, you must aggressively
    modify it:

1. Rename Identifiers: Rename all local variables, arguments (except `
    self`), and any module-level helper functions/variables to
    completely new, distinct names.
2. Alter Control Flow: Radically change the logic structure within each
    method. Convert `for` loops to `while` loops (or vice versa), use
    list comprehensions, invert `if/else` branch conditions, or replace
    simple iterations with recursion where safe.
3. Modify Operations: Use alternative but mathematically/logically
    equivalent Python constructs and operators (e.g., replace standard
    arithmetic with bitwise equivalents, or use different built-in
    functions that achieve the same result).
4. Strip Metadata: Remove all original comments, docstrings, and type
    hints.

Constraints:
- The class name, every method name, and the total number of methods
    must remain identical to the original.
- The resulting code must be valid Python 3 and pass the exact same
    functional tests as the original.
- Do not add any new external dependencies; use only the Python standard
     library.

Output format:
- Return ONLY the full rewritten Python code wrapped in a single ```
    python code block.
- Do not include any explanation, introductory text, or markdown outside
     the code block.
```python
{code}
```
````

Listing 5: Aggressive Python rewriting prompt used in the experiments.