BYTEMARK: A UNIFIED BYTE-LEVEL WATERMARK FOR DETECTION ACROSS LANGUAGE MODELS

Anonymous authors
Paper under double-blind review

ABSTRACT

As watermarking of LLM-generated text enters deployment, users increasingly need to check text whose source is unknown. Existing detectors read the watermark through a specific tokenizer or embedding pipeline, and even if all providers adopted the same watermarking method, differences in these representations would leave their watermarks incompatible, and the text alone does not reveal which representation produced it. We propose ByteMark, a *unified watermarking scheme* that defines its signal on the UTF-8 bytes of published text. Models with different tokenizers embed this signal under one public specification and key, and a single detector verifies the text from the text and key alone. When the tokenizer or embedding model of a baseline detector is replaced, its detection degrades sharply, whereas ByteMark, which uses neither, retains its detection rate; ByteMark also remains detectable when participating models continue or rewrite the text.

1 INTRODUCTION

Large language models (LLMs) generate text for question answering, writing assistance, and other instruction-driven tasks (Brown et al., 2020; Ouyang et al., 2022; Dathathri et al., 2024). When generated text is presented as human-written work, it can undermine authorship assessment and mislead readers about the origin of content (Kirchenbauer et al., 2023; Kuditipudi et al., 2024; Dathathri et al., 2024). LLMs can also hallucinate plausible but factually incorrect content, potentially misleading readers (Lin et al., 2022). These risks also extend to training-data curation: repeatedly training on generated text can lead to *model collapse*, reducing the diversity and performance of subsequent models (Shumailov et al., 2024; Kazdan et al., 2025). Identifying whether text was generated by an LLM can help assess claims of human authorship and track the composition of training data. Publication venues have already taken steps to identify LLM-generated text: ICML has used prompt injection to detect LLM use in peer reviews, and the NeurIPS 2026 Position Paper Track has screened submissions for AI-generated content (Agarwal et al., 2026; NeurIPS 2026 Position Paper Track Chairs, 2026).

Such identification nonetheless remains difficult. An instruction hidden in a paper, such as a prompt injection, can be found and removed by the reviewer (Liu et al., 2026). Zero-shot detectors such as DetectGPT rely on linguistic differences and can lose accuracy with a surrogate scoring model or under a different generator and decoding strategy (Mitchell et al., 2023; Dugan et al., 2024), and human readers do not reliably distinguish LLM-generated responses from human ones (Jones & Bergen, 2024).

A complementary approach is watermarking. By deliberately embedding a recognizable signal into generated text, watermarking gives verifiers evidence that can be tested, without relying solely on incidental differences between human and generated writing (Kirchenbauer et al., 2023; Kuditipudi et al., 2024). Regulation has begun to mandate such marking: the European Union AI Act, applicable from August 2026, requires that generated outputs be "marked in a machine-readable format and detectable as artificially generated" (European Parliament and Council of the European Union, 2024, Art. 50). We therefore expect providers of generative models to *participate* in some form of watermarking.

Existing watermarks embed this signal in tokens or in sentences. Token-level methods alter token selection through logit bias or keyed sampling, and their detectors aggregate keyed token statistics

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.

(Kirchenbauer et al., 2023; Kuditipudi et al., 2024; Dathathri et al., 2024). These detectors do not need the generator's weights and are often described as model-agnostic (Kirchenbauer et al., 2023; Chen et al., 2025; Wu et al., 2024b), yet their signal is defined on token IDs, which differ across tokenizers for the same text. Sentence-level methods map candidate sentences to a semantic embedding and resample until a candidate satisfies the watermark condition; the discarded candidates reduce computational efficiency and increase latency (Hou et al., 2024a;b; Dabiriaghdam & Wang, 2025; Li et al., 2026). Their detectors require the embedding pipeline used to define the watermark, and some also require stored artifacts derived from it, such as cluster centroids (Hou et al., 2024b; Dabiriaghdam & Wang, 2025). In both families, detection therefore depends on a representation that the text does not reveal and that, for closed-source models, may not be publicly available (Section 2).

This dependence does not match practical verification. An area chair checking a peer review, an instructor assessing student work, or a curator filtering training data typically receives the text without knowing which model, if any, produced it. Whoever runs the test, the reader or a service acting for the reader, holds the text and the watermark key but not the generating model or its components. We therefore distinguish two detection tasks:

Presumed-source verification.
: Verifying a watermark associated with a presumed source, given its tokenizer or embedding pipeline.

Unknown-source detection.
: Detecting a watermark from any participating model without knowing how the text was produced.

In unknown-source detection, a negative result under one assumed representation cannot establish that a passage is unwatermarked unless the verifier has searched among candidates known to include the generating representation.

These representations multiply as watermarking enters deployment. Token-level watermarking is already deployed in Gemini and Claude (Dathathri et al., 2024; Anthropic, 2026). *Even if all providers adopted the same watermarking method, the method alone would not make their watermarks compatible*, because each provider applies it through its own tokenizer or embedding pipeline. A single document may also combine several models: multi-agent systems assign the writing and revision of shared text to agents that need not run the same model (Wu et al., 2024a; Wang et al., 2025). Training corpora also collect text written over many years, during which tokenizers and embedding pipelines change.

We therefore seek a common watermark that participating providers embed under a shared key, with three properties.

- **Model independence.** Detection should depend only on the text, the watermark key, and a fixed public specification, so that one detection rule applies across participating models and gives the same result for a given text as models, tokenizers, and embedding pipelines change.
- **Continuity across models.** The same detector should remain effective when participating models continue or rewrite the text.
- **Low generation overhead.** Because deployed language models serve a large volume of requests, embedding should add limited computation and latency during generation.

To meet these requirements, we propose ByteMark, which defines its signal on the UTF-8 byte stream of the published text. During generation, each participating model biases its token selection with a shared byte-level scoring rule; at detection, a verifier applies the same rule to the text's bytes using the shared key and public specification. Because detection requires neither a tokenizer nor a semantic pipeline, one detector serves every participating model, and ByteMark functions as a *unified watermarking scheme* for heterogeneous generators.

Contributions.

1. We distinguish presumed-source verification from unknown-source detection and show empirically that replacing the tokenizer or embedding model removes most of the signal of existing token-level and sentence-level detectors, limiting their use for unknown-source detection.
2. We propose ByteMark, which defines the watermark on the UTF-8 byte stream so that heterogeneous generators share one signal definition and one detection rule. By construction, the

detection score is determined solely by the text, key, and public specification, without reference to the generating tokenizer. Under an ideal pseudorandom function, the detection statistic has an exact finite-sample null distribution, which yields thresholds at any target false-positive rate without calibration data.

3. We demonstrate model independence and continuity across models by applying a single ByteMark detector to outputs of heterogeneous generators and to text continued or rewritten by other participating models, without selecting a representation at model boundaries. When a participating model rewrites the text and re-embeds the watermark, the rewritten output remains detectable. We also quantify generation overhead through the additional candidate-token work and latency of sentence-level methods alongside ByteMark's own byte-scoring overhead.

2 Related Work

We review token-level and sentence-level watermarking methods in terms of their generation procedures and detection requirements. A watermark detector can operate without querying the generating model yet still depend on its representation pipeline. We examine how embedding coordinates, implementation configurations, and token IDs enter the detection statistic and how changing the pipeline affects detection.

Sentence-level watermarking. Sentence-level watermarking methods typically rely on a semantic embedding pipeline to select candidate sentences and detect the embedded signal. Their detection statistics are defined in the geometry of that embedding space: hyperplane regions (Hou et al., 2024a), k-means or fuzzy clusters (Hou et al., 2024b; Diaa et al., 2026; Zhang et al., 2025a), similarity intervals between adjacent sentences (Dabiriaghdam & Wang, 2025), and projections onto keyed pivot vectors or directions (Huo et al., 2026a;b; Li et al., 2026). Two embedding models can capture similar relational structure without preserving these hyperplane regions, clusters, similarity intervals, or pivot projections. Our embedding-model replacement experiments show score degradation (Table 3), and the detectors are undefined for encoders with a different output dimension (Appendix D.3). When such a watermark is deployed, this dependence creates a long-term compatibility trade-off:

- **Retain the original pipeline.** A fixed embedding pipeline preserves compatibility with previously embedded watermarks but cannot benefit from newer embedding models with improved text representations (Wang et al., 2024; Lee et al., 2025; Zhang et al., 2025b). Retaining the original pipeline also requires maintaining an aging deployment environment, whose software dependencies may eventually become deprecated or unsupported.
- **Update the pipeline.** A new pipeline may fail to detect watermarks embedded with the original one (Section 4.1), so updating risks losing backward compatibility.

Sentence-level resampling and candidate selection can generate multiple continuations for each retained sentence, increasing the number of tokens generated per final output token (Hou et al., 2024a;b; Dabiriaghdam & Wang, 2025). This extra generation can increase latency and reduce final-output throughput, depending on how candidate generation is batched or parallelized. Such overhead matters at the scale of current use: agentic coding tasks consume on the order of a thousand times more tokens than code chat or reasoning tasks (Bai et al., 2026), and token consumption is a major computational and economic cost for deployed systems (Chen et al., 2026). Section 4.3 evaluates throughput and candidate-token work separately.

Token-level watermarking. Token-level watermarking methods embed the signal at each decoding step by modifying the token probabilities or the sampling rule. KGW and its descendants bias a keyed green region in the model vocabulary (Kirchenbauer et al., 2023; Zhao et al., 2024; Kirchenbauer et al., 2024). Distortion-free reweighting and deterministic sampling alter the sampling rule but retain randomness indexed by the token context or position (Kuditipudi et al., 2024; Wu et al., 2024b; Hu et al., 2024; Feng et al., 2025); SynthID adds a multi-layer tournament signal (Dathathri et al., 2024). Semantic-conditioned variants map an embedding model's output to token-level biases through learned or projected transforms, and therefore retain a vocabulary-specific mapping (Liu et al., 2024; He et al., 2024; Ren et al., 2024; Schäfer et al., 2026).

The same text can receive different token IDs and segmentations under different tokenizers (Table 1). For token-based watermarks, these differences change the inputs to the detection rule and can therefore change the detector's keyed assignments and score.

Table 1: Tokenization of three words under six fixed tokenizers. Each input has a leading space and is encoded without special tokens. Each cell shows token pieces (top) and IDs (bottom); centered dots separate tokens.

Word	Pythia	Qwen2.5	OpenLLaMA	GPT-Neo	OPT	TinyLlama
`model`	`model` 1566	`model` 1614	`model` 2042	`model` 2746	`model` 1421	`model` 1904
`watermark`	`waterm·ark` 37385·782	`watermark` 88006	`water·mark` 1536·3630	`water·mark` 1660·4102	`water·mark` 514·6920	`wat·erm·ark` 16699·837·935
`paraphrase`	`par·aphr·ase` 1061·24596·511	`paraph·rase` 62230·10632	`par·aphr·ase` 1302·29486·542	`paraph·rase` 38624·22789	`paraph·rase` 40127·34338	`par·ap·hr·ase` 610·481·1092·559

Section 4.1 measures this effect systematically (Table 2), and Appendix H discusses watermarking through formatting, synonym substitution, and prompting.

3 METHOD

Section 2 traced the difficulty to the carrier: embedding coordinates, token IDs, and their implementation configurations exist only inside components the verifier may not hold. We therefore look for a watermark carrier inside the text itself. We introduce that construction through a toy example, LetterMark, which uses the letters of English text. It serves only as an illustration, yet it already detects from the text alone; Section 3.2 applies the same pattern to a carrier that covers arbitrary text.

3.1 TOY EXAMPLE: LETTERMARK

LetterMark works as follows (Figure 1). At each generation step, a keyed hash of the last h letters of the completion selects a green list of letters. Since a token is generated as a whole but may contain several letters, each candidate token is scored letter by letter with a rolling window, and its score R is its fraction of green letters. The fraction γ of candidates with the highest R receives a logit increase of $\delta > 0$, and the model samples as usual with its own tokenizer.

The verifier holds only the published text and the key. It slides the same window over the letters, recomputes each green list, and counts green letters; an unusually high rate indicates the watermark.

Letters can serve as the carrier because they have two properties:

- **Publicly accessible.** The letters are part of the published text and can be recovered by any verifier without access to private artifacts.
- **Representation-independent.** The letter sequence is determined by the published text itself, rather than by a model-specific representation. The same text therefore provides the same carrier across models and over time.

3.2 EMBEDDING A WATERMARK THROUGH THE BYTE STREAM

LetterMark illustrates a publicly accessible, representation-independent carrier, but letters exist only in alphabetic scripts. General text needs a carrier with the same two properties that also covers numerals, punctuation, non-Latin scripts, and the other characters a text may contain. We propose ByteMark, which takes the UTF-8 byte stream of the text as its watermark carrier. UTF-8 maps Unicode code points to byte sequences that a verifier can recover from the published text without information from the generator.

For a secret key k, byte context c, and next byte b, define

$$u_b(c) = \frac{(\mathrm{int}_{64} \circ F_k)(c, b)}{2^{64}} \in [0, 1), \tag{1}$$

where int_{64} reads the first eight bytes of a PRF output as a big-endian unsigned integer, and F_k is a keyed pseudorandom function (PRF) under the secret key k: a deterministic function whose

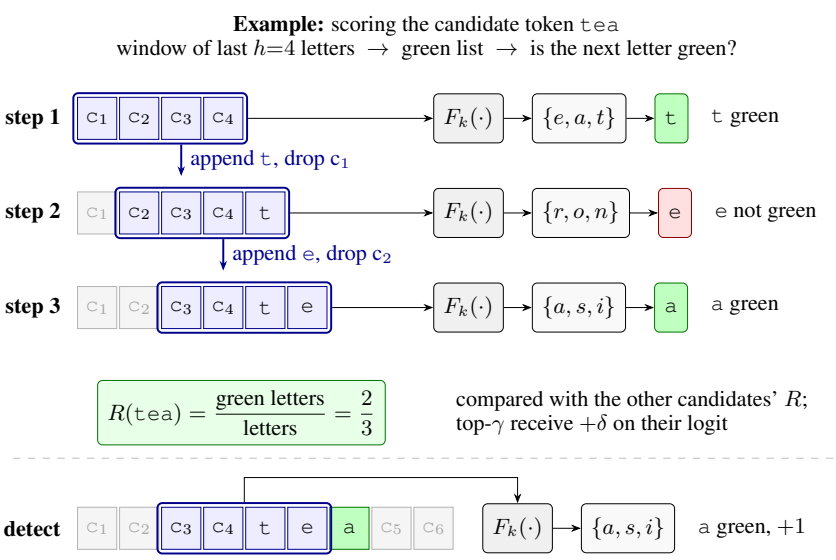


Figure 1: LetterMark scoring the candidate token `tea` after context letters $c_1c_2c_3c_4$ (top) and detecting the mark (bottom). The blue window holds the last $h = 4$ letters, and its keyed hash selects the green list for the next letter; after each check the letter is appended and the window slides by one. Two of the three letters are green, so $R(\texttt{tea}) = 2/3$. The verifier repeats the same scan on the published text without a tokenizer.

outputs are indistinguishable from uniform random values without the key, built from cryptographic primitives widely used in secure communications (Rescorla, 2018; Nystrom, 2005). The public specifications of these primitives support reproducible computation across platforms.

The context c contains at most the last h bytes of the generated completion. For a candidate token v with L_v bytes, we score each byte b_j using its rolling context $c_j = \mathrm{last}_h(c\|b_1\|\cdots\|b_{j-1})$, which includes earlier bytes of the candidate token. Here last_h retains at most the final h bytes, and $\|$ denotes concatenation. We then average the scores to bias the token logits:

$$R(v;c) = \frac{1}{L_v}\sum_{j=1}^{L_v} u_{b_j}(c_j), \qquad \ell'_v = \ell_v + \delta\,\mathbf{1}[v \in \mathrm{Top}_\gamma\, R]. \tag{2}$$

Here ℓ_v is the original logit, $\mathrm{Top}_\gamma\, R$ contains the highest-scoring fraction γ of candidates, and δ is the logit increase. This bias encourages tokens with larger R, favoring higher byte scores u across the watermarked text. The model samples from $\mathrm{softmax}(\ell')$.

Each generator applies the shared byte-level rule through its own token vocabulary, while detection depends only on the text's bytes, key, and public specification; Figure 2 contrasts these verification inputs with those of the evaluated baselines. The byte-level carrier also supports other injection rules: Appendix K outlines a possible extension to SynthID-style multilayer reweighting. We use the single-layer green-set logit bias for computational economy and implementation simplicity.

Because this specification is fixed and its computation deterministic, the detection procedure does not change when models or their tokenizers are updated.

Algorithms 1 and 2 in Appendix A give the generation and detection procedures, together with details on UTF-8, the PRF construction, and implementation rules.

3.3 Text-only detection

To establish a theoretical basis for detection, we first examine the distribution of byte scores under the null hypothesis: the text is unwatermarked and independent of the watermark key. In the ideal continuous-score model, distinct context–byte pairs receive independent $U[0,1)$ scores; for a text fixed independently of these scores, all probabilities and expectations below are over this random assignment. Detection tests whether the mean of the text's distinct byte scores exceeds its null expectation.

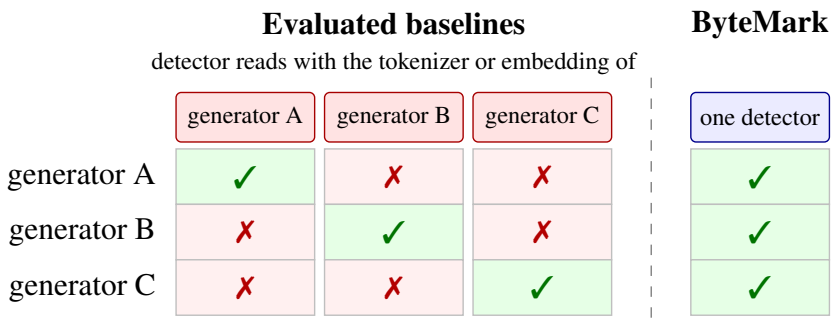


Figure 2: Each baseline detector recovers the watermark only with the representation of the generating source, whereas one ByteMark detector serves all three sources.

For a text x with byte stream $b_1, \ldots, b_N$, the verifier reconstructs $c_i = \text{last}_h(b_1 \| \cdots \| b_{i-1})$ and $u_i = u_{b_i}(c_i)$. We retain each distinct (c_i, b_i) pair once, following the treatment of repeated n-grams in prior work (Kirchenbauer et al., 2024; Fernandez et al., 2023). Let $m \geq 1$ be the number retained. Reindex their scores as $u_1, \ldots, u_m$ and let $\bar{u} = m^{-1} \sum_{i=1}^{m} u_i$. The index j above is local to a candidate token; before deduplication, i indexes the complete byte stream.

Null calibration.

Theorem 1 (Fixed-text null calibration)**.** *Under the null hypothesis, the m retained scores are independent $U[0,1)$ values, so $\mathbb{E}[\bar{u}] = 1/2$, $\text{Var}(\bar{u}) = 1/(12m)$, and $m\bar{u}$ has the m-term Irwin–Hall distribution with CDF*

$$F_m(t) = \frac{1}{m!} \sum_{j=0}^{\lfloor t \rfloor} (-1)^j \binom{m}{j} (t-j)^m, \qquad 0 \leq t \leq m. \tag{3}$$

Consequently, the standardized mean

$$z(x) = \sqrt{12m} \left(\bar{u} - \frac{1}{2} \right) \tag{4}$$

satisfies $\mathbb{E}[z] = 0$ and $\text{Var}(z) = 1$, and for every $m \geq 1$ and $0 < \alpha < 1$,

$$\Pr[z > \tau(m, \alpha)] = \alpha, \qquad \tau(m, \alpha) = \frac{F_m^{-1}(1-\alpha) - m/2}{\sqrt{m/12}}. \tag{5}$$

As $m \to \infty$,

$$z(x) \xrightarrow{d} \mathcal{N}(0,1), \qquad \tau(m, \alpha) \to \Phi^{-1}(1-\alpha), \tag{6}$$

for each fixed $0 < \alpha < 1$, where Φ is the standard normal CDF.

Proof. See Appendix B.3. □

The detector therefore uses $z(x)$ as its statistic and reports watermark evidence when $z(x) > \tau$ for a threshold τ; a larger z indicates a higher average byte score than the null mean $1/2$. In our experiments, we use empirically calibrated detection thresholds (Appendix C.2) so that reported operating points reflect the evaluated human pools. A deployment without calibration data can use the theoretical threshold; Appendix I.3 compares its nominal false-positive rates with the rates realized on human text.

We next show that the injection raises the expected token score: at each decoding step, the boost cannot lower it and raises it strictly whenever the boosted set has the higher conditional mean.

Proposition 1 (One-step score increase)**.** *Fix the decoding history and candidate scores $R(v; c)$. Let $p = \text{softmax}(\ell)$ and $q = \text{softmax}(\ell')$, where ℓ' applies a boost $\delta \geq 0$ to the highest-scoring candidate set $G = \text{Top}_\gamma R$. Write $p_G = \sum_{v \in G} p(v)$ and $q_G = \sum_{v \in G} q(v)$. For $0 < p_G < 1$, define the conditional means $\mu_G = \mathbb{E}_p[R \mid v \in G]$ and $\mu_{\bar{G}} = \mathbb{E}_p[R \mid v \notin G]$. Then*

$$\mathbb{E}_q[R] - \mathbb{E}_p[R] = (q_G - p_G)(\mu_G - \mu_{\bar{G}}) \geq 0. \tag{7}$$

The increase is strict if $\delta > 0$ and $\mu_G > \mu_{\bar{G}}$.

Proof. See Appendix B.2. □

A closed-form characterization of the document-level statistic under watermarking requires additional assumptions on the byte distribution and dependencies across decoding steps; we leave this analysis outside the scope of this work.

4 EXPERIMENTS

We empirically evaluate representation-independent detection, continuity across participating models, and generation efficiency.

- **Detection without tokenizer or embedding dependencies (Section 4.1).** Token-level and sentence-level detectors lose most of their signal when the detector's tokenizer or embedding model differs from the generator's, whereas ByteMark requires neither.
- **Detection continuity across participating models (Section 4.2).** We evaluate one unchanged detector on text continued by participating models or rewritten by them.
- **Generation efficiency (Section 4.3).** We measure output throughput and the number of candidate tokens generated per final output token.

Appendix C states the experimental protocol; the other appendices report additional experiments, analyses, and diagnostics.

4.1 DETECTION UNDER EMBEDDING MODEL AND TOKENIZER REPLACEMENT

We evaluate the representation dependence discussed in Section 2 by replacing the detector's representation while holding the generated text fixed. ByteMark uses neither component, so its rows simply report detection on the three datasets.

Table 2: TPR (%) of the token-level detectors at the same-pool empirical 1% threshold. Matched uses the generating tokenizer; each Replacement cell averages the five other tokenizers over the three generators of that dataset. ByteMark uses no tokenizer, so its cells are its detection rates.

Method	Replacement tokenizer			Matched
	C4	Wikipedia	LFQA	
ByteMark	**99.5**	**99.3**	**99.6**	99.5
KGW	1.5	1.3	1.7	99.6
SynthID	0.6	0.6	0.6	100.0
SIR	1.7	5.9	2.2	96.5
EXP	1.6	1.2	1.3	99.9
ITS	0.8	0.8	1.0	98.0

Table 3: TPR (%) of the sentence-level methods on C4 at the same-pool empirical 1% threshold. Matched uses the generating embedding model; Replacement averages the replacement models that can score the method (Appendix D.3).

Method	Replacement embedding	Matched
SimMark	9.2	91.8
SemStamp	3.8	99.1
k-SemStamp	3.7	96.6
AliMark	6.5	99.8

Results. Table 2 shows that every token-level baseline detects its watermark with its own tokenizer, but detection largely fails with replacement tokenizers, whereas ByteMark retains its detection rate in every domain. The complete generator–tokenizer matrices are in Tables 8–12 of Appendix D.

Table 3 reports the sentence-level methods under embedding-model replacement. With their own embedding model all four reach a TPR above 90%; with a replacement model the mean TPR falls to 4–9%. Appendix D.3 reports each replacement model separately.

For token-level detectors, searching over candidate tokenizers does not remove the dependence on the generating tokenizer. For Table 4, a search flags a human text whenever any of K candidate tokenizers exceeds its own same-pool threshold at level α. Its false-positive rate is several times the single-test rate and close to $1-(1-\alpha)^K$, the value for independent tests, so the candidates' false-positive sets overlap little at this threshold and the combined decision must be calibrated. Bonferroni correction achieves this without assuming independence by testing each candidate at level α/K (Appendix E). These stricter thresholds can reduce power relative to detection with the known tokenizer, and the search still requires the candidate set to contain the generating tokenizer.

Table 4: Same-pool empirical false-positive rate of an uncorrected search that flags a human text when any of $K = 6$ candidate tokenizers exceeds its own threshold at level $\alpha = 1\%$. Independent candidates would give $1-(1-\alpha)^K \approx 5.85\%$; identical rejection events would give α. Bonferroni correction tests each candidate at $\alpha/K \approx 0.167\%$.

Method	Observed	Independent
KGW	0.0580	0.0585
SynthID	0.0580	0.0585
SIR	0.0537	0.0585
EXP	0.0563	0.0585
ITS	0.0573	0.0585

Summary. Token-level detectors need the generating tokenizer and sentence-level detectors need the generating embedding model; with a replacement tokenizer or embedding model, most of the watermark signal is lost. ByteMark depends on neither, so tokenizer replacement leaves its detection rate in Table 2 unchanged. This is the model-independence property of Section 1, and it is what unknown-source detection requires: given the text and the key, one detector verifies text from any participating model without identifying its source. Searching over candidate tokenizers does not remove this requirement for the baselines: it must be calibrated for multiple testing and still needs the generating tokenizer among its candidates.

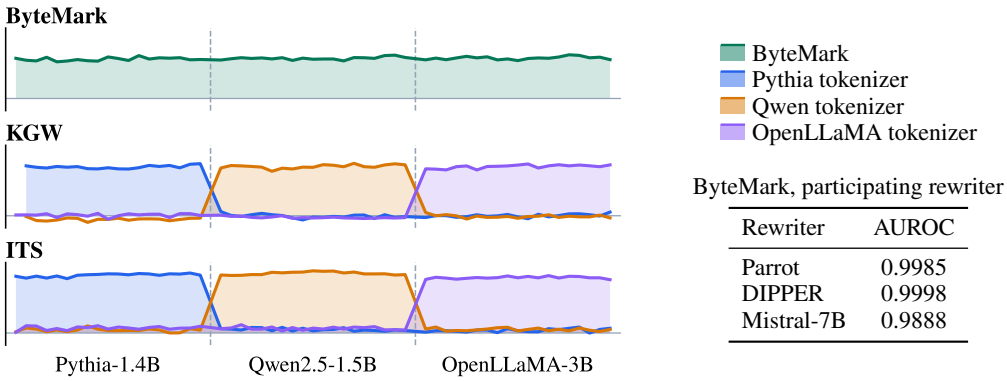
4.2 Detection continuity across model transitions

We test whether one unchanged detector remains effective when participating models continue a text or re-embed the same watermark during rewriting.

Relay generation. In this experiment, Pythia-1.4B, Qwen2.5-1.5B, and OpenLLaMA-3B write successive segments of a C4 completion in turn, each embedding the watermark under the same key (Appendix F). Figure 3 (left) plots the mean local signal in a sliding window of five tokens along the relay. A KGW or ITS detector reads the whole text with one fixed tokenizer, so it finds the signal only in the segment written with that tokenizer, and its curve stays near zero in the other two. The ByteMark curve holds its level across all three segments and shows no visible change at either switch: the byte context carries over each model boundary, and one detector reads the text as a single byte stream.

Rewriting with re-embedding. In this experiment, Parrot, DIPPER, and Mistral-7B-Instruct rewrite watermarked Pythia-1.4B completions, each embedding the same ByteMark watermark as it rewrites. The unchanged ByteMark detector separates the rewritten outputs from human text under all three rewriters (Figure 3, right). This result concerns the re-embedded signal. Whether the original signal survives rewriting by a non-participating model is a robustness question shared by all watermarking methods, and Appendix G.2 reports that setting.

Summary. A token-level detector reads only the part of a relayed document written with its own tokenizer, whereas one ByteMark detector follows the watermark across model switches and after rewriting by participating models. This is the continuity-across-models property of Section 1.



ByteMark, participating rewriter

Rewriter	AUROC
Parrot	0.9985
DIPPER	0.9998
Mistral-7B	0.9888

Figure 3: Detection across model transitions. Left: mean local z in a sliding window of five tokens along the C4 relay; dashed lines mark model switches. Right: ByteMark AUROC after rewriting by participating models of Pythia-1.4B completions.

4.3 Generation efficiency

Sentence-level watermarks generate several candidates, keep one, and discard the rest (Section 2). We measure this cost in two ways: the number of candidate tokens generated per final output token, which counts the computation spent on discarded text and depends on the sampling mechanism rather than the implementation; and final-output tokens per second, the speed a user observes.

Table 5 reports both on the same requests. The four sentence-level methods generate a large multiple of the tokens they keep and reach only a small fraction of the unwatermarked throughput. ByteMark provides the contrast: it samples each token once, and its added cost is scoring the vocabulary's bytes with a hardware-accelerated keyed function at each step. A sentence-level watermark therefore pays for its detection with generation efficiency, whereas ByteMark keeps its throughput close to that of unwatermarked generation.

Table 5: Generation efficiency with OpenLLaMA-3B on 50 C4 texts. Relative is the share of the unwatermarked rate (63.82 tok/s). *Different backend (vLLM). Appendix J.1 gives the measurement conditions.

Method	Final tok/s	Relative (%)	Generated / final
Unwatermarked	63.82	100.0	1.00
ByteMark	62.19	97.4	1.00
SemStamp	1.73	2.7	35.38
k-SemStamp	6.17	9.7	10.10
SimMark	0.49	0.8	119.36
AliMark*	18.22	28.6	100.68

5 Conclusion

The token-level and sentence-level watermark tests we evaluate verify a presumed source: their detectors need the source's tokenizer or embedding pipeline, which the text does not reveal, so a negative result under one assumed source says little about whether a passage is watermarked. ByteMark removes this dependence by defining the watermark on the UTF-8 bytes of the published text, so that detection needs only the text, the key, and the public specification. In our experiments, replacing the tokenizer or the embedding model leaves the token-level and sentence-level baselines with little of their signal, whereas ByteMark's detection rate is unchanged; one detector follows a text across a relay of heterogeneous generators and recovers the watermark that a participating rewriter re-embeds; and sampling each output token once keeps throughput close to unwatermarked generation, with an overhead that shrinks as models grow. A byte-level watermark thus offers **unified detection** of text from any participating model without identifying its source. Appendix L discusses the limitations.

Ethics Statement

This work uses public corpora and public models and involves no human subjects. A watermark detector supports claims about the origin of a text, and a false claim can harm the person it is attributed to. We therefore report detection at frozen thresholds calibrated on held-out human text and state the realized false-positive rates, and we recommend that a positive detection be treated as evidence rather than proof. The key must remain secret; its loss exposes the watermark of every participant, as discussed in Appendix L.

Reproducibility Statement

The core implementation of ByteMark is provided in the supplementary material; the complete code will be released as open source after cleanup.

AI Use Statement

Generative AI tools were used to summarize literature, analyze manuscript structure, translate draft material, edit the manuscript for readability and consistency, and assist with editing the experimental code. We take responsibility for the final content of this work, including text and artifacts produced with the aid of generative AI.

References

Alekh Agarwal, Miroslav Dudik, Sharon Li, Martin Jaggi, Nihar B. Shah, Katherine Gorman, and Gautam Kamath. On violations of LLM review policies. ICML Blog, March 2026. URL https://blog.icml.cc/2026/03/18/on-violations-of-llm-review-policies/. Published March 18, 2026; updated March 26, 2026. Accessed September 8, 2026.

Anthropic. How Claude's text watermark works. Anthropic, August 2026. URL https://www.anthropic.com/news/claude-text-watermark. Published August 14, 2026. Accessed September 25, 2026.

Longju Bai, Zhemin Huang, Xingyao Wang, Jiao Sun, Rada Mihalcea, Erik Brynjolfsson, Alex Pentland, and Jiaxin Pei. How do AI agents spend your money? analyzing and predicting token consumption in agentic coding tasks, 2026. URL https://arxiv.org/abs/2604.22750.

Stella Biderman, Hailey Schoelkopf, Quentin Gregory Anthony, Herbie Bradley, Kyle O'Brien, Eric Hallahan, Mohammad Aflah Khan, Shivanshu Purohit, Usvsn Sai Prashanth, Edward Raff, Aviya Skowron, Lintang Sutawika, and Oskar Van Der Wal. Pythia: A suite for analyzing large language models across training and scaling. In *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pp. 2397–2430. PMLR, 2023. URL https://proceedings.mlr.press/v202/biderman23a.html.

Sid Black, Leo Gao, Phil Wang, Connor Leahy, and Stella Biderman. GPT-Neo: Large scale autoregressive language modeling with Mesh-TensorFlow. Zenodo, 2021. URL https://doi.org/10.5281/zenodo.5297715.

Vladimir Blagojevic. LFQA dataset. Hugging Face dataset repository, n.d. URL https://huggingface.co/datasets/vblagoje/lfqa. Accessed 25 September 2026.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel Ziegler, Jeffrey Wu, Clemens Winter, Chris Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin (eds.), *Advances in Neural Information Processing Systems*, volume 33, pp. 1877–1901. Curran Associates, Inc.,

2020. URL https://proceedings.neurips.cc/paper_files/paper/2020/file/1457c0d6bfcb4967418bfb8ac142f64a-Paper.pdf.

Ruibo Chen, Yihan Wu, Junfeng Guo, and Heng Huang. Improved unbiased watermark for large language models, 2025. URL https://arxiv.org/abs/2502.11268.

Yuxi Chen, Junming Chen, Chenyu He, Yiwei Li, Yicheng Ji, Yifan Wu, Dingyu Yang, Lansong Diao, Lidan Shou, Hongliang Zhang, Huan Li, and Gang Chen. Token economics for LLM agents: A dual-view study from computing and economics, 2026. URL https://arxiv.org/abs/2605.09104.

Amirhossein Dabiriaghdam and Lele Wang. SimMark: A robust sentence-level similarity-based watermarking algorithm for large language models. In Christos Christodoulopoulos, Tanmoy Chakraborty, Carolyn Rose, and Violet Peng (eds.), *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 30785–30806, Suzhou, China, November 2025. Association for Computational Linguistics. ISBN 979-8-89176-332-6. doi: 10.18653/v1/2025.emnlp-main.1567. URL https://aclanthology.org/2025.emnlp-main.1567/.

Prithiviraj Damodaran. Parrot: Paraphrase generation for NLU. https://github.com/PrithivirajDamodaran/Parrot_Paraphraser, 2021. Version 1.0.

Sumanth Dathathri, Abigail See, Sumedh Ghaisas, Po-Sen Huang, Rob McAdam, Johannes Welbl, Vandana Bachani, Alex Kaskasoli, Robert Stanforth, Tatiana Matejovicova, Jamie Hayes, Nidhi Vyas, Majd Al Merey, Jonah Brown-Cohen, Rudy Bunel, Borja Balle, Taylan Cemgil, Zahra Ahmed, Kitty Stacpoole, Ilia Shumailov, Ciprian Baetu, Sven Gowal, Demis Hassabis, and Pushmeet Kohli. Scalable watermarking for identifying large language model outputs. *Nature*, 634(8035):818–823, Oct 2024. ISSN 1476-4687. doi: 10.1038/s41586-024-08025-4. URL https://doi.org/10.1038/s41586-024-08025-4.

Abdulrahman Diaa, Jonathan Petit, and Florian Kerschbaum. Semantic watermarking with order-robust detection over sub-sentence units, 2026. URL https://arxiv.org/abs/2608.27666.

Liam Dugan, Alyssa Hwang, Filip Trhlík, Andrew Zhu, Josh Magnus Ludan, Hainiu Xu, Daphne Ippolito, and Chris Callison-Burch. RAID: A shared benchmark for robust evaluation of machine-generated text detectors. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar (eds.), *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 12463–12492, Bangkok, Thailand, August 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.acl-long.674. URL https://aclanthology.org/2024.acl-long.674/.

European Parliament and Council of the European Union. Regulation (EU) 2024/1689 (Artificial Intelligence Act). Official Journal of the European Union, 2024. URL https://eur-lex.europa.eu/eli/reg/2024/1689/oj. Article 50(2). Accessed September 9, 2026.

Xiaoyan Feng, He Zhang, Yanjun Zhang, Leo Yu Zhang, and Shirui Pan. BiMark: Unbiased multilayer watermarking for large language models. In Aarti Singh, Maryam Fazel, Daniel Hsu, Simon Lacoste-Julien, Felix Berkenkamp, Tegan Maharaj, Kiri Wagstaff, and Jerry Zhu (eds.), *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 17049–17067. PMLR, 13–19 Jul 2025. URL https://proceedings.mlr.press/v267/feng25u.html.

Pierre Fernandez, Antoine Chaffin, Karim Tit, Vivien Chappelier, and Teddy Furon. Three bricks to consolidate watermarks for large language models. In *IEEE International Workshop on Information Forensics and Security (WIFS)*, 2023. URL https://arxiv.org/abs/2308.00113.

Xinyang Geng and Hao Liu. OpenLLaMA: An open reproduction of LLaMA. Model and code repository, 2023. URL https://github.com/openlm-research/open_llama.

Zhiwei He, Binglin Zhou, Hongkun Hao, Aiwei Liu, Xing Wang, Zhaopeng Tu, Zhuosheng Zhang, and Rui Wang. Can watermarks survive translation? on the cross-lingual consistency of text

watermark for large language models. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar (eds.), *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 4115–4129, Bangkok, Thailand, August 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.acl-long.226. URL https://aclanthology.org/2024.acl-long.226/.

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 Kevin Duh, Helena Gomez, and Steven Bethard (eds.), *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 2024a. Association for Computational Linguistics. doi: 10.18653/v1/2024.naacl-long.226. URL https://aclanthology.org/2024.naacl-long.226/.

Abe Hou, Jingyu Zhang, Yichen Wang, Daniel Khashabi, and Tianxing He. k-SemStamp: A clustering-based semantic watermark for detection of machine-generated text. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar (eds.), *Findings of the Association for Computational Linguistics: ACL 2024*, pp. 1706–1715, Bangkok, Thailand, August 2024b. Association for Computational Linguistics. doi: 10.18653/v1/2024.findings-acl.98. URL https://aclanthology.org/2024.findings-acl.98/.

Zhengmian Hu, Lichang Chen, Xidong Wu, Yihan Wu, Hongyang Zhang, and Heng Huang. Unbiased watermark for large language models. In B. Kim, Y. Yue, S. Chaudhuri, K. Fragkiadaki, M. Khan, and Y. Sun (eds.), *International Conference on Learning Representations*, volume 2024, pp. 45408–45436, 2024. URL https://proceedings.iclr.cc/paper_files/paper/2024/file/c5b00c5bdcc6fe35907dbcca03d27652-Paper-Conference.pdf.

Jiahao Huo, Shuliang Liu, Bin Wang, Junyan Zhang, Yibo Yan, Aiwei Liu, Xuming Hu, and Mingxun Zhou. PMark: Towards robust and distortion-free semantic-level watermarking with channel constraints, 2026a. URL https://arxiv.org/abs/2509.21057.

Jiahao Huo, Wenjie Qu, Yibo Yan, Kening Zheng, Jiaheng Zhang, Xuming Hu, Philip S. Yu, and Mingxun Zhou. SAMark: A self-anchored text watermarking with paragraph-level paraphrase robustness, 2026b. URL https://arxiv.org/abs/2605.25796.

Cameron R. Jones and Benjamin K. Bergen. People cannot distinguish GPT-4 from a human in a Turing test, 2024. URL https://arxiv.org/abs/2405.08007.

Joshua Kazdan, Rylan Schaeffer, Apratim Dey, Matthias Gerstgrasser, Rafael Rafailov, David L. Donoho, and Sanmi Koyejo. Collapse or thrive: Perils and promises of synthetic data in a self-generating world. In Aarti Singh, Maryam Fazel, Daniel Hsu, Simon Lacoste-Julien, Felix Berkenkamp, Tegan Maharaj, Kiri Wagstaff, and Jerry Zhu (eds.), *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 29469–29494. PMLR, 13–19 Jul 2025. URL https://proceedings.mlr.press/v267/kazdan25a.html.

John Kelsey, Shu-jen Chang, and Ray Perlner. SHA-3 derived functions: cSHAKE, KMAC, TupleHash, and ParallelHash. Technical Report NIST Special Publication 800-185, National Institute of Standards and Technology, 2016.

John Kirchenbauer, Jonas Geiping, Yuxin Wen, Jonathan Katz, Ian Miers, and Tom Goldstein. A watermark for large language models. In Andreas Krause, Emma Brunskill, Kyunghyun Cho, Barbara Engelhardt, Sivan Sabato, and Jonathan Scarlett (eds.), *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pp. 17061–17084. PMLR, 23–29 Jul 2023. URL https://proceedings.mlr.press/v202/kirchenbauer23a.html.

John Kirchenbauer, Jonas Geiping, Yuxin Wen, Manli Shu, Khalid Saifullah, Kezhi Kong, Kasun Fernando, Aniruddha Saha, Micah Goldblum, and Tom Goldstein. On the reliability of watermarks for large language models. In B. Kim, Y. Yue, S. Chaudhuri, K. Fragkiadaki, M. Khan, and

Y. Sun (eds.), *International Conference on Learning Representations*, volume 2024, pp. 49660–49704, 2024. URL https://proceedings.iclr.cc/paper_files/paper/2024/file/d78e9e4316e1714fbb0f20be66f8044c-Paper-Conference.pdf.

Kalpesh Krishna, Yixiao Song, Marzena Karpinska, John Wieting, and Mohit Iyyer. Paraphrasing evades detectors of AI-generated text, but retrieval is an effective defense. In *Advances in Neural Information Processing Systems*, volume 36, 2023. URL https://papers.nips.cc/paper_files/paper/2023/hash/575c450013d0e99e4b0ecf82bd1afaa4-Abstract-Conference.html.

Rohith Kuditipudi, John Thickstun, Tatsunori Hashimoto, and Percy Liang. Robust distortion-free watermarks for language models. *Transactions on Machine Learning Research*, 2024. URL https://arxiv.org/abs/2307.15593.

Taku Kudo and John Richardson. SentencePiece: A simple and language independent subword tokenizer and detokenizer for neural text processing. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pp. 66–71. Association for Computational Linguistics, 2018. doi: 10.18653/v1/D18-2012. URL https://aclanthology.org/D18-2012/.

Chankyu Lee, Rajarshi Roy, Mengyao Xu, Jonathan Raiman, Mohammad Shoeybi, Bryan Catanzaro, and Wei Ping. NV-Embed: Improved techniques for training LLMs as generalist embedding models. In Y. Yue, A. Garg, N. Peng, F. Sha, and R. Yu (eds.), *International Conference on Learning Representations*, volume 2025, pp. 79310–79333, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/file/c4bf73386022473a652a18941e9ea6f8-Paper-Conference.pdf.

Yuexin Li, Wenjie Qu, Linyu Wu, Yulin Chen, Yufei He, Tri Cao, Bryan Hooi, and Jiaheng Zhang. AliMark: Enhancing robustness of sentence-level watermarking against text paraphrasing, 2026. URL https://arxiv.org/abs/2605.29434.

Stephanie Lin, Jacob Hilton, and Owain Evans. TruthfulQA: Measuring how models mimic human falsehoods. In Smaranda Muresan, Preslav Nakov, and Aline Villavicencio (eds.), *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 3214–3252, Dublin, Ireland, May 2022. Association for Computational Linguistics. doi: 10.18653/v1/2022.acl-long.229. URL https://aclanthology.org/2022.acl-long.229/.

Aiwei Liu, Leyi Pan, Xuming Hu, Shiao Meng, and Lijie Wen. A semantic invariant robust watermark for large language models. In B. Kim, Y. Yue, S. Chaudhuri, K. Fragkiadaki, M. Khan, and Y. Sun (eds.), *International Conference on Learning Representations*, volume 2024, pp. 6499–6519, 2024. URL https://proceedings.iclr.cc/paper_files/paper/2024/file/1a2131ebe25bd55e4fc734126ea583ed-Paper-Conference.pdf.

Yepeng Liu, Xuandong Zhao, Christopher Kruegel, Dawn Song, and Yuheng Bu. In-context watermarks for large language models, 2026. URL https://arxiv.org/abs/2505.16934.

Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. Pointer sentinel mixture models. arXiv preprint arXiv:1609.07843, 2016. URL https://arxiv.org/abs/1609.07843.

Eric Mitchell, Yoonho Lee, Alexander Khazatsky, Christopher D Manning, and Chelsea Finn. DetectGPT: Zero-shot machine-generated text detection using probability curvature. In Andreas Krause, Emma Brunskill, Kyunghyun Cho, Barbara Engelhardt, Sivan Sabato, and Jonathan Scarlett (eds.), *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pp. 24950–24962. PMLR, 23–29 Jul 2023. URL https://proceedings.mlr.press/v202/mitchell23a.html.

NeurIPS 2026 Position Paper Track Chairs. AI-generated papers in the NeurIPS 2026 position paper track. NeurIPS Blog, June 2026. URL https://blog.neurips.cc/2026/06/02/ai-generated-papers-in-the-neurips-2026-position-paper-track/. Published June 2, 2026. Accessed September 8, 2026.

Jianmo Ni, Chen Qu, Jing Lu, Zhuyun Dai, Gustavo Hernández Ábrego, Ji Ma, Vincent Y. Zhao, Yi Luan, Keith B. Hall, Ming-Wei Chang, and Yinfei Yang. Large dual encoders are generalizable retrievers. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pp. 9844–9855, 2022. doi: 10.18653/v1/2022.emnlp-main.669.

M. Nystrom. Identifiers and Test Vectors for HMAC-SHA-224, HMAC-SHA-256, HMAC-SHA-384, and HMAC-SHA-512. RFC 4231, December 2005. URL https://www.rfc-editor.org/info/rfc4231/.

Jack O'Connor, Jean-Philippe Aumasson, Samuel Neves, and Zooko Wilcox-O'Hearn. BLAKE3: One function, fast everywhere. https://github.com/BLAKE3-team/BLAKE3-specs/blob/master/blake3.pdf, 2020.

Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, John Schulman, Jacob Hilton, Fraser Kelton, Luke Miller, Maddie Simens, Amanda Askell, Peter Welinder, Paul F Christiano, Jan Leike, and Ryan Lowe. Training language models to follow instructions with human feedback. In S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh (eds.), *Advances in Neural Information Processing Systems*, volume 35, pp. 27730–27744. Curran Associates, Inc., 2022. doi: 10.52202/068431-2011. URL https://proceedings.neurips.cc/paper_files/paper/2022/file/b1efde53be364a73914f58805a001731-Paper-Conference.pdf.

Qwen, An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, Huan Lin, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jingren Zhou, Junyang Lin, Kai Dang, Keming Lu, Keqin Bao, Kexin Yang, Le Yu, Mei Li, Mingfeng Xue, Pei Zhang, Qin Zhu, Rui Men, Runji Lin, Tianhao Li, Tianyi Tang, Tingyu Xia, Xingzhang Ren, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yu Wan, Yuqiong Liu, Zeyu Cui, Zhenru Zhang, and Zihan Qiu. Qwen2.5 technical report. arXiv preprint arXiv:2412.15115, 2025. URL https://arxiv.org/abs/2412.15115.

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. URL https://jmlr.org/papers/v21/20-074.html.

Jie Ren, Han Xu, Yiding Liu, Yingqian Cui, Shuaiqiang Wang, Dawei Yin, and Jiliang Tang. A robust semantics-based watermark for large language model against paraphrasing. In Kevin Duh, Helena Gomez, and Steven Bethard (eds.), *Findings of the Association for Computational Linguistics: NAACL 2024*, pp. 613–625, Mexico City, Mexico, June 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.findings-naacl.40. URL https://aclanthology.org/2024.findings-naacl.40/.

E. Rescorla. The Transport Layer Security (TLS) Protocol Version 1.3. RFC 8446, August 2018. URL https://www.rfc-editor.org/info/rfc8446/.

Stefano Giovanni Rizzo, Flavio Bertini, and Danilo Montesi. Content-preserving text watermarking through Unicode homoglyph substitution. In *Proceedings of the 20th International Database Engineering & Applications Symposium (IDEAS)*, 2016. doi: 10.1145/2938503.2938510.

Ryoma Sato, Yuki Takezawa, Han Bao, Kenta Niwa, and Makoto Yamada. Embarrassingly simple text watermarks. arXiv preprint arXiv:2310.08920, 2023.

Jonas Schäfer, Cezary Pilaszewicz, and Gerhard Wunder. Robust text watermarking for large language models via dual semantic embeddings, 2026. URL https://arxiv.org/abs/2606.31602.

Ilia Shumailov, Zakhar Shumaylov, Yiren Zhao, Nicolas Papernot, Ross Anderson, and Yarin Gal. AI models collapse when trained on recursively generated data. *Nature*, 631(8022):755–759, Jul 2024. ISSN 1476-4687. doi: 10.1038/s41586-024-07566-y. URL https://doi.org/10.1038/s41586-024-07566-y.

J. H. Song, R. Poovendran, J. Lee, and T. Iwata. The AES-CMAC algorithm. RFC 4493, Internet Engineering Task Force, 2006. URL https://www.rfc-editor.org/rfc/rfc4493.

Hongjin Su, Weijia Shi, Jungo Kasai, Yizhong Wang, Yushi Hu, Mari Ostendorf, Wen-tau Yih, Noah A. Smith, Luke Zettlemoyer, and Tao Yu. One embedder, any task: Instruction-finetuned text embeddings. In *Findings of the Association for Computational Linguistics: ACL 2023*, pp. 1102–1121, 2023. doi: 10.18653/v1/2023.findings-acl.71.

The Unicode Consortium. The Unicode standard, version 16.0.0, 2024. URL https://www.unicode.org/versions/Unicode16.0.0/.

Umut Topkara, Mercan Topkara, and Mikhail J. Atallah. The hiding virtues of ambiguity: Quantifiably resilient watermarking of natural language text through synonym substitutions. In *Proceedings of the 8th Workshop on Multimedia and Security (MM&Sec)*, pp. 164–174, Geneva, Switzerland, 2006. doi: 10.1145/1161366.1161397.

W3Techs. Usage statistics of character encodings for websites. https://w3techs.com/technologies/overview/character_encoding, 2026. Survey dated 25 September 2026; accessed 25 September 2026.

Junlin Wang, Jue Wang, Ben Athiwaratkun, Ce Zhang, and James Zou. Mixture-of-agents enhances large language model capabilities. In *International Conference on Learning Representations (ICLR)*, 2025. arXiv:2406.04692.

Liang Wang, Nan Yang, Xiaolong Huang, Linjun Yang, Rangan Majumder, and Furu Wei. Improving text embeddings with large language models. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar (eds.), *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 11897–11916, Bangkok, Thailand, August 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.acl-long.642. URL https://aclanthology.org/2024.acl-long.642/.

Xiaopu Wang, Zelin He, Chengyuan Liu, and Runze Li. Beyond heuristic tuning: Power-calibrated LLM watermarking. In *Proceedings of the 43rd International Conference on Machine Learning (ICML)*, 2026. URL https://arxiv.org/abs/2607.05694.

Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Beibin Li, Erkang Zhu, Li Jiang, Xiaoyun Zhang, Shaokun Zhang, Jiale Liu, Ahmed Hassan Awadallah, Ryen W. White, Doug Burger, and Chi Wang. AutoGen: Enabling next-gen LLM applications via multi-agent conversation. In *Conference on Language Modeling (COLM)*, 2024a. arXiv:2308.08155.

Yihan Wu, Zhengmian Hu, Junfeng Guo, Hongyang Zhang, and Heng Huang. A resilient and accessible distribution-preserving watermark for large language models. In Ruslan Salakhutdinov, Zico Kolter, Katherine Heller, Adrian Weller, Nuria Oliver, Jonathan Scarlett, and Felix Berkenkamp (eds.), *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 53443–53470. PMLR, 21–27 Jul 2024b. URL https://proceedings.mlr.press/v235/wu24h.html.

François Yergeau. UTF-8, a transformation format of ISO 10646. RFC 3629, Internet Engineering Task Force, 2003. URL https://www.rfc-editor.org/rfc/rfc3629.

Junyan Zhang, Shuliang Liu, Aiwei Liu, Yubo Gao, Jungang Li, Xiaojie Gu, and Xuming Hu. CoheMark: A novel sentence-level watermark for enhanced text quality, 2025a. URL https://arxiv.org/abs/2504.17309.

Peiyuan Zhang, Guangtao Zeng, Tianduo Wang, and Wei Lu. TinyLlama: An open-source small language model. arXiv preprint arXiv:2401.02385, 2024. URL https://arxiv.org/abs/2401.02385.

Susan Zhang, Stephen Roller, Naman Goyal, Mikel Artetxe, Moya Chen, Shuohui Chen, Christopher Dewan, Mona Diab, Xian Li, Xi Victoria Lin, Todor Mihaylov, Myle Ott, Sam Shleifer, Kurt Shuster, Daniel Simig, Punit Singh Koura, Anjali Sridhar, Tianlu Wang, and Luke Zettlemoyer. OPT: Open pre-trained transformer language models. arXiv preprint arXiv:2205.01068, 2022. URL https://arxiv.org/abs/2205.01068.

Yanzhao Zhang, Mingxin Li, Dingkun Long, Xin Zhang, Huan Lin, Baosong Yang, Pengjun Xie, An Yang, Dayiheng Liu, Junyang Lin, Fei Huang, and Jingren Zhou. Qwen3 embedding: Advancing text embedding and reranking through foundation models, 2025b. URL https://arxiv.org/abs/2506.05176.

Xuandong Zhao, Prabhanjan Ananth, Lei Li, and Yu-Xiang Wang. Provable robust watermarking for AI-generated text. In B. Kim, Y. Yue, S. Chaudhuri, K. Fragkiadaki, M. Khan, and Y. Sun (eds.), *International Conference on Learning Representations*, volume 2024, pp. 43738–43772, 2024. URL https://proceedings.iclr.cc/paper_files/paper/2024/file/beae9ed5316bcc48e616754c06c11875-Paper-Conference.pdf.

SUPPLEMENTARY MATERIAL

A SPECIFICATION OF BYTE-LEVEL GENERATION AND DETECTION

The construction connects two interfaces: a generator chooses tokens, while a verifier reads text. Both evaluate the same keyed function on the same sequence of UTF-8 context–byte pairs. Token boundaries are needed to apply the generation bias, but disappear from the detection procedure.

A.1 UTF-8 REPRESENTATION

UTF-8 provides a standardized byte representation for Unicode text: it encodes each Unicode scalar value in one to four bytes (The Unicode Consortium, 2024; Yergeau, 2003). It is also prevalent on the web: in W3Techs' 25 September 2026 survey, UTF-8 accounts for 99.1% of websites with an identified character encoding (W3Techs, 2026). ByteMark uses the published text's UTF-8 byte stream as a common carrier across generators, so a verifier can apply the same detection rule without knowing the generator's token boundaries.

In our experiments, every text is produced directly as UTF-8 bytes through the tokenizer's byte table, so no transcoding is involved. For text that a deployment receives in another encoding, we recommend transcoding it to UTF-8 before detection. Text in another Unicode encoding, such as UTF-16 or UTF-32, can be converted without additional Unicode normalization. Text in a legacy character encoding, such as ISO-8859-1, Windows-1252, or Shift JIS, can be decoded to Unicode through the published mapping of its declared encoding and then re-encoded as UTF-8.

A.2 TOKENIZER COVERAGE OF THE BYTE INTERFACE

This subsection checks whether diverse tokenizer vocabularies can supply the exact output bytes needed to score candidate tokens during generation. For each token ID v, the byte table must define $\beta(v)$, its contribution to the UTF-8 output (Section A.5). We surveyed 35 public causal-language-model tokenizers with pairwise distinct byte tables, including byte-level BPE and SentencePiece-style vocabularies (Table 6). Every ID had a defined entry: ordinary tokens contribute a nonempty byte sequence, while designated zero-width special tokens contribute no output bytes.

UTF-8 byte-interface compatibility. A token ID v is compatible if the byte table assigns it a determinate output-byte sequence $\beta(v)$ under the tokenizer's decoding rules, without replacing unresolved byte fragments with a replacement character. Let $\mathcal{V}$ include all vocabulary IDs, including added and special tokens. We measure compatibility as

$$\text{Compatibility}(\mathcal{V}) = \frac{|\{v \in \mathcal{V} : \beta(v) \text{ is defined}\}|}{|\mathcal{V}|}.$$

For example, OPT-1.3B's end-of-sequence token </s> maps to the empty byte sequence and is therefore compatible. Further mapping details are given in the reference implementation.

Table 6: Token-to-byte mapping coverage for the 35 surveyed causal-language-model vocabularies. Every vocabulary ID has a defined byte contribution (100% UTF-8 byte-interface compatibility).

Tokenizer	IDs	Tokenizer	IDs
xglm-564M	256,008	deepseek-llm-7b-base	100,015
bloom-560m	250,680	Jamba-v0.1	65,536
MiniMax-M2	200,054	falcon-7b	65,024
gpt-oss-120b	200,019	gpt-j-6b	50,400
Ling-1T	157,153	phi-2	50,295
Seed-OSS-36B-Base	155,121	OLMoE-1B-7B-0924	50,280
Qwen3-Coder-480B-A35B	151,669	gpt-neox-20b	50,277
Qwen2.5-72B	151,665	opt-1.3b	50,265
Qwen2-0.5B	151,646	gpt-neo-1.3B	50,257
GLM-4.5	151,365	starcoder2-15b	49,152
Falcon3-1B-Base	131,072	granite-3.1-2b-base	49,152
DeepSeek-V3	128,815	SmolLM2-1.7B	49,152
Nemotron-Super-49B	128,256	Mistral-Large-Instruct-2411	32,768
EuroLLM-1.7B	128,000	Phi-3-mini-4k-instruct	32,011
EXAONE-3.5-2.4B-Instruct	102,400	Llama-2-7b	32,000
ERNIE-4.5-300B-A47B-Base-PT	101,304	Mixtral-8x22B	32,000
stablelm-2-1_6b	100,289	open_llama_3b_v2	32,000
OLMo-2-32B	100,278		

A defined contribution may be a byte fragment or an empty special-token entry; individual tokens need not decode as complete UTF-8 characters. This is a mapping check, not a generation benchmark.

A.3 Pseudorandom function

ByteMark requires a keyed pseudorandom function (PRF) that maps each context–byte input to a pseudorandom score; the function is not used to encrypt the text. The main instantiation is AES-CMAC under a 128-bit key (Song et al., 2006): AES is a standardized block cipher, and CMAC turns it into a message authentication code whose output serves as the PRF value. The same key and context–byte input always reproduce the same score. Other standardized keyed PRFs can take its place without changing the construction; the reference implementation also supports keyed BLAKE3 (O'Connor et al., 2020) and KMAC256 (Kelsey et al., 2016).

A.4 Detector key format

A ByteMark key is a single string containing the construction, text encoding, PRF, byte-context width, key material, and checksum, in that order. For the main configuration, it has the form `bytemark.utf8.aes-cmac.h8.SECRET.CHECK`. Under the fixed ByteMark specification, this string determines the watermark's keyed byte-level signal. A verifier given the published text and this string can compute the continuous detection statistic without the generating model or its tokenizer.

A.5 Byte table and candidate scores

To score a candidate token, the generator needs its exact output-byte sequence $\beta(v) = (b_1, \ldots, b_{L_v})$. The byte table recovers this sequence from the tokenizer's representation, including byte-level BPE and SentencePiece byte-fallback pieces. Decoding an isolated token can replace an incomplete UTF-8 fragment, so the table is not built from per-token `decode()` calls. Special tokens omitted from the output map to the empty sequence; output-head rows without token strings are masked.

Using the main-text notation, $u_b(c)$ is the keyed score of byte b in context c, with the fixed key k implicit. For a candidate's jth byte, let $c_j = \mathrm{last}_h(c \| b_1 \| \cdots \| b_{j-1})$. Here last_h retains at most the final h bytes. The candidate score, including the empty-sequence case, is

$$R(v; c) = \begin{cases} \dfrac{1}{L_v} \displaystyle\sum_{j=1}^{L_v} u_{b_j}(c_j), & L_v > 0, \\ 0, & L_v = 0. \end{cases} \tag{8}$$

Algorithm 1 ByteMark generation through a token-based model

Require: model M, byte table β, prompt p, key k, context width h, boosted fraction γ, logit increment δ

Set y and c to the empty sequence
while the chosen stopping rule permits a step **do**
  $\ell \leftarrow M(p\|y)$
  for all active vocabulary entries v **do**
    Compute $R(v; c)$ using Equation 8
  end for
  $G \leftarrow \mathrm{Top}_\gamma R$, the $\lfloor \gamma|\mathcal{V}| \rfloor$ highest-scoring entries
  $\ell'_v \leftarrow \ell_v + \delta\, \mathbf{1}[v \in G]$ for each v
  Sample v from $\mathrm{softmax}(\ell')$; append v to y
  $c \leftarrow \mathrm{last}_h(c\|\beta(v))$
end while
Return the generated token sequence y

A.6 Detection from the completion bytes

Detection scans the completion's UTF-8 encoding and retains one observation per distinct context–byte pair. The rolling context advances at every byte, including repeated pairs. For bytes $b_1, \ldots, b_N$, use the main-text definitions $c_i = \mathrm{last}_h(b_1\| \cdots \|b_{i-1})$ and $u_i = u_{b_i}(c_i)$. After retaining each distinct pair once, reindex the retained scores as $u_1, \ldots, u_m$ and write their mean as $\bar{u}$. The continuous statistic is

$$z(x) = \sqrt{12m}\left(\bar{u} - \frac{1}{2}\right). \tag{9}$$

Algorithm 2 ByteMark detection without a tokenizer or embedding model

Require: nonempty text x, key k, context width h, threshold τ

Set c to the empty byte sequence; $D \leftarrow \varnothing$; $S \leftarrow 0$; $m \leftarrow 0$
for all bytes b in $\mathrm{UTF8}(x)$, in order **do**
  if $(c, b) \notin D$ **then**
    Add (c, b) to D; $S \leftarrow S + u_b(c)$; $m \leftarrow m + 1$
  end if
  $c \leftarrow \mathrm{last}_h(c\|b)$
end for
$\bar{u} \leftarrow S/m$; $z(x) \leftarrow \sqrt{12m}(\bar{u} - 1/2)$
return $z(x)$ and the decision $z(x) > \tau$

B Reconstruction and null calibration of the detection statistic

B.1 Invariance of the byte scores to token boundaries

For an emitted token v with $\beta(v) = (b_1, \ldots, b_{L_v})$, generation scores its jth byte as $u_{b_j}(c_j)$, where $c_j = \mathrm{last}_h(c\|b_1\| \cdots \|b_{j-1})$. Let i be this byte's position in the completed text x. Detection encounters the same preceding bytes and therefore reconstructs $c_i = c_j$ and $u_i = u_{b_j}(c_j)$. This equality holds for every emitted byte, regardless of the token boundaries. Thus, the text, key, and public specification suffice to reconstruct the byte scores.

B.2 Effect of the logit bias

The watermark signal is a shift in the keyed byte scores of the emitted text. At a fixed decoding step, the logit bias shifts probability toward the highest-scoring candidates. We show below that this cannot decrease the expected candidate score.

The size of the shift at one step follows from the change in sampling probabilities. Condition on the step's history and logits. As in the main text, let $p = \mathrm{softmax}(\ell)$, $q = \mathrm{softmax}(\ell')$, and $G = \mathrm{Top}_\gamma R$, and write $p_G = \sum_{v \in G} p(v)$ and $q_G = \sum_{v \in G} q(v)$. At temperature one, adding δ to the logits in G multiplies their unnormalized softmax weights by e^δ, so the normalizing total becomes $(1 - p_G) + e^\delta p_G = 1 + (e^\delta - 1)p_G$. Hence

$$q(v) = \frac{p(v)e^{\delta \mathbf{1}[v \in G]}}{1 + (e^\delta - 1)p_G}, \qquad q_G - p_G = \frac{(e^\delta - 1)p_G(1 - p_G)}{1 + (e^\delta - 1)p_G}. \tag{10}$$

This is the conditional probability transformation analyzed for logit-based watermarking by Wang et al. (2026), applied here to $G = \mathrm{Top}_\gamma R$.

Proof of Proposition 1. Fix the history, key, and logits at a generation step, so that c, all scores $R(v; c)$, and G are fixed before sampling. Write $R(v) = R(v; c)$ and $\bar{G} = \mathcal{V} \setminus G$, and assume $\delta \geq 0$. Expectations below are over the sampled token only: $\mathbb{E}_s[R] = \sum_{v \in \mathcal{V}} s(v)R(v)$ for $s \in \{p, q\}$. First suppose $0 < p_G < 1$, and define

$$\mu_G = \frac{\sum_{v \in G} p(v)R(v)}{p_G}, \qquad \mu_{\bar{G}} = \frac{\sum_{v \in \bar{G}} p(v)R(v)}{1 - p_G}.$$

Let $Z = 1 + (e^\delta - 1)p_G$. The preceding transformation gives $q_G = e^\delta p_G / Z$ and $1 - q_G = (1 - p_G)/Z$. Hence

$$\frac{q(v)}{q_G} = \frac{p(v)}{p_G} \quad (v \in G), \qquad \frac{q(v)}{1 - q_G} = \frac{p(v)}{1 - p_G} \quad (v \in \bar{G}).$$

Thus both conditional means are unchanged by the boost. Splitting each expectation over the two sets yields

$$\begin{aligned} \mathbb{E}_q[R] - \mathbb{E}_p[R] &= \big[q_G \mu_G + (1 - q_G)\mu_{\bar{G}}\big] - \big[p_G \mu_G + (1 - p_G)\mu_{\bar{G}}\big] \\ &= (q_G - p_G)(\mu_G - \mu_{\bar{G}}) \\ &= \frac{(e^\delta - 1)p_G(1 - p_G)}{Z}(\mu_G - \mu_{\bar{G}}). \end{aligned} \tag{11}$$

By the definition of $G = \mathrm{Top}_\gamma R$, every $v \in G$ and $w \in \bar{G}$ satisfy $R(v) \geq R(w)$, including ties at the cutoff. Therefore

$$\mu_G - \mu_{\bar{G}} = \frac{\sum_{v \in G} \sum_{w \in \bar{G}} p(v)p(w)\big(R(v) - R(w)\big)}{p_G(1 - p_G)} \geq 0.$$

Since $Z > 0$ and $e^\delta - 1 \geq 0$, it follows that $\mathbb{E}_q[R] \geq \mathbb{E}_p[R]$. For $0 < p_G < 1$, the inequality is strict exactly when $\delta > 0$ and $\mu_G > \mu_{\bar{G}}$. If $p_G = 0$ or $p_G = 1$, the boost leaves $q = p$, so the expected score is unchanged without requiring either undefined conditional mean.

This conditional, one-step result does not determine the document-level signal: candidate lengths, shared prefixes, repeated context–byte inputs, and the contexts inherited by later steps all affect it, so we measure that signal empirically.

Without the bias, every fixed nonempty candidate has expected score $1/2$ under an ideal PRF, and a candidate with L_v distinct inputs has variance $1/(12L_v)$. Averaging in Equation 8, rather than summing, keeps the expected score from growing linearly with candidate length. The coverage diagnostics in Section I examine which byte classes contribute this evidence.

B.3 Proof of Theorem 1

Proof. Moments. Fix a text independently of the ideal PRF, with $m > 0$ distinct context–byte inputs. Thus the retained $u_1, \ldots, u_m$ are independent uniform values with expectation $1/2$ and variance $1/12$. Consequently,

$$\mathbb{E}[\bar{u}] = \frac{1}{2}, \qquad \mathrm{Var}(\bar{u}) = \frac{1}{12m}, \qquad \mathbb{E}[z] = 0, \qquad \mathrm{Var}(z) = 1.$$

Exact distribution. For a real argument $t \in [0, m]$, the CDF $F_m(t) = \Pr[m\bar{u} \leq t]$ equals the volume of $\{(u_1, \ldots, u_m) \in [0, 1]^m : \sum_{i=1}^m u_i \leq t\}$. Without the upper bounds $u_i \leq 1$, this simplex has

volume $t^m/m!$. For a subset $J \subseteq \{1, \ldots, m\}$ of size j, requiring $u_i > 1$ for every $i \in J$ and subtracting one from these coordinates gives volume $\max(t-j, 0)^m/m!$. Inclusion–exclusion over the $\binom{m}{j}$ subsets of each size therefore yields

$$F_m(t) = \frac{1}{m!} \sum_{j=0}^{\lfloor t \rfloor} (-1)^j \binom{m}{j} (t-j)^m, \qquad 0 \le t \le m. \tag{12}$$

Exact threshold. Since $z(x) = (m\bar{u} - m/2)/\sqrt{m/12}$, for a threshold τ,

$$\Pr[z > \tau] = 1 - F_m\Big(m/2 + \tau\sqrt{m/12}\Big).$$

The CDF F_m is continuous and strictly increasing on $[0, m]$. Hence, for every $0 < \alpha < 1$, the ideal upper-tail level-α threshold is

$$\tau(m, \alpha) = \frac{F_m^{-1}(1-\alpha) - m/2}{\sqrt{m/12}}. \tag{13}$$

Substitution gives $\Pr[z > \tau(m, \alpha)] = \alpha$ exactly.

Asymptotic calibration. The central limit theorem gives $z(x) \xrightarrow{d} \mathcal{N}(0, 1)$ as $m \to \infty$. Because the standard normal CDF Φ is continuous and strictly increasing, the corresponding quantiles converge: $\tau(m, \alpha) \to \Phi^{-1}(1-\alpha)$ for each fixed $0 < \alpha < 1$. □

B.4 Deduplication of repeated inputs

A repeated context–byte pair reproduces exactly the same PRF score, rather than an independent observation. Fix a text x containing N context–byte occurrences and m distinct pairs, where $m \le N$. Let $u_1, \ldots, u_m$ be the distinct-pair scores, and let $A_i \ge 1$ denote the multiplicity of pair i, so that $\sum_{i=1}^m A_i = N$. Deduplication retains m terms; counting every occurrence retains N terms, which can be grouped into m terms weighted by A_i. Grouping repeated terms does not remove their multiplicities.

Counting all occurrences with an independence assumption. Under the fixed-text ideal null of Theorem 1, normalizing the sum over all N occurrences as if they were independent gives

$$z_{\text{naive}}(x) = \frac{\sum_{i=1}^m A_i(u_i - 1/2)}{\sqrt{N/12}}, \qquad \operatorname{Var}(z_{\text{naive}}) = \frac{\sum_{i=1}^m A_i^2}{N}.$$

This variance exceeds one whenever any input repeats. Standard-normal calibration is therefore unjustified and can understate the false-positive rate. Kirchenbauer et al. (2024, Appendix A.3) report this discrepancy for repeated token n-grams and show that excluding repeats improves calibration. Fernandez et al. (2023, Sections III.B–III.C) address both finite-sample tail approximation and repeated-input dependence, combining non-asymptotic tests with exclusion of repeated context–token tuples.

Counting all occurrences with the correct null variance. Retaining the multiplicities and accounting for their covariance gives

$$z_{\text{rep}}(x) = \frac{\sum_{i=1}^m A_i(u_i - 1/2)}{\sqrt{\frac{1}{12}\sum_{i=1}^m A_i^2}}.$$

For fixed multiplicities under the ideal null, its denominator is the standard deviation of the numerator: the A_i copies of u_i are identical, so their variance contribution is $A_i^2/12$, not $A_i/12$. Unit variance alone does not make this weighted sum standard normal, so its tail probabilities still require a suitable null distribution.

Counting each distinct input once. ByteMark removes the multiplicity weights and uses

$$z(x) = \frac{\sum_{i=1}^m (u_i - 1/2)}{\sqrt{m/12}}.$$

The denominator uses the number m of distinct inputs because these m scores are independent under the ideal null. Theorem 1 gives the resulting exact null distribution; standard-normal calibration is its large-m approximation.

Standardized signal under a common shift. We compare $z(x)$ and $z_{\mathrm{rep}}(x)$ with the same m distinct inputs and fixed multiplicities $A_1, \ldots, A_m$, using their respective null standard deviations. Assume a common-shift alternative $\mathbb{E}[u_i] = 1/2 + \varepsilon$ for every distinct pair, with $\varepsilon > 0$ and the multiplicities held fixed. The expected scores, measured in their respective null standard deviations, are

$$\mathbb{E}[z(x)] = \sqrt{12m}\,\varepsilon, \qquad \mathbb{E}[z_{\mathrm{rep}}(x)] = \sqrt{12}\,\varepsilon\,\frac{\sum_{i=1}^{m} A_i}{\sqrt{\sum_{i=1}^{m} A_i^2}}.$$

Their ratio is therefore

$$\frac{\mathbb{E}[z_{\mathrm{rep}}(x)]}{\mathbb{E}[z(x)]} = \frac{\sum_{i=1}^{m} A_i}{\sqrt{m}\sqrt{\sum_{i=1}^{m} A_i^2}} \le 1,$$

where Cauchy–Schwarz gives $(\sum_i A_i)^2 \le m \sum_i A_i^2$, with equality exactly when all multiplicities are equal. Thus, under the common-shift assumption, counting repeated inputs provides no benefit in standardized mean signal.

C Evaluation protocol

Generators, tokenizers, and corpora. The main experiments use Pythia-1.4B (Biderman et al., 2023), Qwen2.5-1.5B (Qwen et al., 2025), and OpenLLaMA-3B (Geng & Liu, 2023), with their released tokenizers spanning byte-level BPE and SentencePiece (Kudo & Richardson, 2018). The tokenizers of OPT-1.3B (Zhang et al., 2022), GPT-Neo-1.3B (Black et al., 2021), and TinyLlama-1.1B (Zhang et al., 2024) are used only for detection. Prompts and unwatermarked human texts come from C4 (Raffel et al., 2020), Wikipedia articles in WikiText-103 (Merity et al., 2016), and LFQA (Blagojevic, n.d.).

Baselines. Among token-level methods, we compare with the token-identity methods KGW (Kirchenbauer et al., 2023), EXP and ITS (Kuditipudi et al., 2024), and SynthID-Text (Dathathri et al., 2024), as well as SIR (Liu et al., 2024), whose semantic features condition a tokenizer-specific vocabulary mapping. The sentence-level baselines are SemStamp (Hou et al., 2024a), k-SemStamp (Hou et al., 2024b), SimMark (Dabiriaghdam & Wang, 2025), and AliMark (Li et al., 2026). We evaluate them under representation mismatch by replacing the detector's native embedding model while holding the generated text fixed. Each baseline is run through its authors' released generation and detection implementation. EXP and ITS follow the authors' conditional randomization procedure (Kuditipudi et al., 2024).

C.1 Cohorts and evaluation lengths

The token-level comparisons span C4, Wikipedia, and LFQA with Pythia-1.4B, Qwen2.5-1.5B, and OpenLLaMA-3B. The semantic comparisons use C4 with the generation models of the original implementations and with Pythia-1.4B; some additional tests use Wikipedia with Pythia-1.4B. In both families, each source condition uses 500 generation requests and each domain supplies 1,000 shared human texts. Token-level generation uses a budget of 200 native tokens. The matched-detection and tokenizer-replacement tables score prefixes of at most 200 reference tokens; shorter texts retain their original length. Tokenizer replacement scores each generated text with its native tokenizer and with five alternatives, without additional generation.

Detection metrics are computed over valid, scoreable texts. Paired analyses use the common valid subset. Invalid entries, whether rejected generations or unscoreable texts, are excluded and not replaced.

Sentence-level protocols retain their specified segmentation, candidate generation, stopping rules, and timeout settings. Embedding-model replacement retains the generated text and the method-specific watermark parameters; incompatible configurations are reported as unscoreable rather than assigned zero detection rates.

C.2 Thresholds, true positive rates, and reference distributions

For the displayed empirical comparisons, a detector's threshold is selected from the shared human scores at a target 1% exceedance rate. TPR is then measured on scoreable watermarked positives. AUROC compares positive and human scores.

C.3 Method configurations

ByteMark uses AES-CMAC-128 with $h = 8$ and the same $(\gamma, \delta) = (.25, 2)$ as KGW (Kirchenbauer et al., 2024); generation uses temperature one, disabled top-k, and top-$p = 1$. For both ByteMark and KGW, the logit increment δ sets the watermark strength, and adjusting δ trades detection rate against generation quality (Kirchenbauer et al., 2023).

Generation length. ByteMark, KGW, and SynthID set `min_new_tokens` to the budget, which suppresses EOS before 200 tokens. If the byte stream is incomplete at the budget, generation may continue for at most three additional tokens to complete a UTF-8 character; a completion that remains invalid UTF-8 is rejected without replacement. EXP and ITS run the authors' fixed number of sampling steps, where EOS neither stops nor is suppressed.

Semantic methods retain their author-specified preprocessing, generation, and detection procedures (Dabiriaghdam & Wang, 2025; Hou et al., 2024a;b; Li et al., 2026); only the component named in each replacement experiment is changed. Structural incompatibility, such as a replacement dimension inconsistent with stored centroids, is recorded as undefined.

C.4 Overview of the experiments

Experiment	Varied component	Held fixed
Matched detection	None (reference condition)	Generator, text, and detector
Tokenizer replacement	Detector tokenizer or vocabulary mapping	Generated texts
Candidate search	Detector over a finite candidate set	Human texts
Embedding replacement	Detection embedding model	Generation and stored geometric artifacts
Generator replacement	Generating model	Author watermark parameters
Model relay	Generator at each relay stage	Watermark key and byte context
Editing and rewriting	Text after generation	Clean-text detection threshold
Statistical diagnostics	Length, statistic, character class, human corpus	Key and detector

D Detection under representation replacement

We first evaluate matched detection, then replace the detector's representation while keeping the generated text fixed. For token-level methods the verifier replaces the tokenizer; for semantic methods it replaces the detection encoder. A separate experiment changes the generation model instead, which separates dependence on the generator from dependence on the detection representation. Search over candidate tokenizers is treated separately in Section E.

D.1 Matched detection

Table 7 compares matched detection across source models and domains. All methods exhibit strong detection performance under matched conditions across the evaluated source models and domains.

Table 7: Matched detection. Entries are AUROC / TPR.

Domain	Generator	ByteMark	KGW	SynthID	SIR	EXP	ITS
C4	Pythia	0.9997 / 0.996	0.9998 / 0.996	0.9995 / 0.998	0.9993 / 0.984	0.9982 / 0.996	0.9911 / 0.980
	Qwen	1.0000 / 0.998	1.0000 / 1.000	1.0000 / 1.000	0.9900 / 0.944	1.0000 / 1.000	0.9994 / 0.992
	OpenLLaMA	0.9989 / 0.992	0.9998 / 0.994	1.0000 / 1.000	0.9991 / 0.986	0.9990 / 0.996	0.9944 / 0.980
WIKI	Pythia	1.0000 / 1.000	1.0000 / 1.000	1.0000 / 1.000	0.9991 / 0.994	1.0000 / 1.000	0.9994 / 0.994
	Qwen	0.9976 / 0.978	0.9994 / 0.984	1.0000 / 1.000	0.9887 / 0.914	1.0000 / 1.000	0.9877 / 0.928
	OpenLLaMA	1.0000 / 1.000	0.9999 / 0.996	1.0000 / 1.000	0.9991 / 0.982	1.0000 / 1.000	0.9953 / 0.982
LFQA	Pythia	1.0000 / 1.000	1.0000 / 1.000	1.0000 / 1.000	0.9987 / 0.986	0.9990 / 1.000	0.9987 / 0.994
	Qwen	0.9996 / 0.990	0.9992 / 0.998	1.0000 / 1.000	0.9863 / 0.906	0.9998 / 0.998	0.9949 / 0.980
	OpenLLaMA	0.9999 / 0.998	1.0000 / 1.000	1.0000 / 1.000	0.9995 / 0.988	0.9995 / 1.000	0.9982 / 0.990

D.2 TOKENIZER REPLACEMENT

We hold the generated texts fixed and replace the detector's tokenizer or vocabulary mapping. Under tokenizer mismatch, the other methods largely lose their detection capability, whereas ByteMark's detection scores remain unchanged. Tables 8–12 report the replacement results, with cells with TPR below 0.2 shaded pale yellow.

SIR is included here because its detector uses a vocabulary mapping as well as semantic conditioning. The model-matched comparison aligns each generator with its native tokenizer before testing replacements.

Table 8: KGW: matched versus replacement tokenizers. Each cell shows AUROC (top) and TPR (bottom). Cells with TPR below 0.2 are shaded pale yellow.

		Matched	Replacement detector tokenizer					
Domain	Generator	Native	Pythia	Qwen	OpenLLaMA	OPT	GPT-Neo	TinyLLaMA
C4	Pythia	0.9998 0.996	/	0.4560 0.016	0.5053 0.010	0.4954 0.020	0.4181 0.006	0.4368 0.012
	Qwen	1.0000 1.000	0.4591 0.006	/	0.4385 0.010	0.5373 0.032	0.4868 0.012	0.4901 0.014
	OpenLLaMA	0.9998 0.994	0.5164 0.008	0.5127 0.016	/	0.5766 0.038	0.4702 0.018	0.4466 0.012
Wiki	Pythia	1.0000 1.000	/	0.4048 0.006	0.3132 0.006	0.4135 0.014	0.3410 0.006	0.4224 0.002
	Qwen	0.9994 0.984	0.4955 0.010	/	0.2649 0.002	0.5724 0.030	0.3625 0.008	0.4321 0.004
	OpenLLaMA	0.9999 0.996	0.4134 0.008	0.3885 0.004	/	0.5755 0.054	0.5505 0.042	0.4150 0.004
LFQA	Pythia	1.0000 1.000	/	0.4772 0.016	0.5465 0.024	0.5427 0.034	0.4128 0.002	0.3994 0.004
	Qwen	0.9992 0.998	0.5296 0.025	/	0.4546 0.004	0.5696 0.025	0.4360 0.008	0.4419 0.006
	OpenLLaMA	1.0000 1.000	0.5844 0.030	0.5139 0.012	/	0.6097 0.048	0.4590 0.012	0.4025 0.008

The matched reference is shown in the Native column; a diagonal stroke marks that same tokenizer, not a missing result.

Table 9: SynthID: matched versus replacement tokenizers. Each cell shows AUROC (top) and TPR (bottom). Cells with TPR below 0.2 are shaded pale yellow.

		Matched	Replacement detector tokenizer					
Domain	Generator	Native	Pythia	Qwen	OpenLLaMA	OPT	GPT-Neo	TinyLLaMA
C4	Pythia	0.9995 0.998	/	0.5134 0.002	0.5063 0.002	0.4874 0.000	0.5054 0.006	0.4857 0.008
	Qwen	1.0000 1.000	0.5045 0.008	/	0.5237 0.012	0.4874 0.000	0.4845 0.010	0.4814 0.004
	OpenLLaMA	1.0000 1.000	0.4914 0.010	0.4866 0.002	/	0.5015 0.000	0.4813 0.006	0.5121 0.020
Wiki	Pythia	1.0000 1.000	/	0.5100 0.004	0.4919 0.008	0.5013 0.010	0.5200 0.004	0.5113 0.004
	Qwen	1.0000 1.000	0.5103 0.006	/	0.5016 0.008	0.4846 0.006	0.5174 0.008	0.5201 0.006
	OpenLLaMA	1.0000 1.000	0.5147 0.004	0.5035 0.000	/	0.4971 0.006	0.4960 0.000	0.5079 0.010
LFQA	Pythia	1.0000 1.000	/	0.4869 0.008	0.5215 0.004	0.4985 0.006	0.4529 0.000	0.4994 0.000
	Qwen	1.0000 1.000	0.4706 0.004	/	0.4820 0.006	0.4960 0.010	0.4618 0.006	0.5208 0.002
	OpenLLaMA	1.0000 1.000	0.5077 0.018	0.4875 0.002	/	0.4925 0.020	0.4679 0.006	0.4948 0.000

The matched reference is shown in the Native column; a diagonal stroke marks that same tokenizer, not a missing result.

Table 10: SIR: matched versus replacement tokenizers. Each cell shows AUROC (top) and TPR (bottom). Cells with TPR below 0.2 are shaded pale yellow.

		Matched	Replacement detector tokenizer					
Domain	Generator	Native	Pythia	Qwen	OpenLLaMA	OPT	GPT-Neo	TinyLLaMA
C4	Pythia	0.9993 0.984	/	0.4772 0.014	0.4222 0.004	0.5239 0.018	0.5181 0.024	0.6779 0.050
	Qwen	0.9900 0.944	0.4178 0.004	/	0.4782 0.004	0.3649 0.002	0.5204 0.014	0.4417 0.000
	OpenLLaMA	0.9991 0.986	0.3171 0.006	0.4777 0.042	/	0.5013 0.008	0.5469 0.042	0.4942 0.018
Wiki	Pythia	0.9991 0.994	/	0.5871 0.058	0.4288 0.006	0.4766 0.004	0.7159 0.104	0.8081 0.086
	Qwen	0.9887 0.914	0.4894 0.018	/	0.6036 0.012	0.3878 0.006	0.7836 0.144	0.6972 0.028
	OpenLLaMA	0.9991 0.982	0.4727 0.056	0.6503 0.120	/	0.5981 0.040	0.7228 0.106	0.7492 0.090
LFQA	Pythia	0.9987 0.986	/	0.4372 0.014	0.4291 0.006	0.5706 0.024	0.5803 0.030	0.6609 0.038
	Qwen	0.9863 0.906	0.4568 0.006	/	0.4320 0.004	0.3886 0.004	0.5924 0.036	0.5196 0.006
	OpenLLaMA	0.9995 0.988	0.3126 0.006	0.4773 0.032	/	0.5401 0.016	0.6114 0.074	0.5138 0.028

The matched reference is shown in the Native column; a diagonal stroke marks that same tokenizer, not a missing result.

Table 11: EXP: matched versus replacement tokenizers. Each cell shows AUROC (top) and TPR (bottom). Cells with TPR below 0.2 are shaded pale yellow.

		Matched	Replacement detector tokenizer					
Domain	Generator	Native	Pythia	Qwen	OpenLLaMA	OPT	GPT-Neo	TinyLLaMA
C4	Pythia	0.9982 0.996	/	0.5282 0.006	0.5313 0.026	0.5131 0.008	0.5280 0.006	0.5180 0.010
	Qwen	1.0000 1.000	0.4874 0.004	/	0.5620 0.054	0.4876 0.014	0.4926 0.010	0.5202 0.020
	OpenLLaMA	0.9990 0.996	0.4713 0.014	0.4928 0.014	/	0.4923 0.014	0.5125 0.016	0.5157 0.016
Wiki	Pythia	1.0000 1.000	/	0.5452 0.006	0.5075 0.018	0.4644 0.004	0.4763 0.010	0.4979 0.016
	Qwen	1.0000 1.000	0.5067 0.020	/	0.4888 0.018	0.4471 0.004	0.5104 0.018	0.4714 0.008
	OpenLLaMA	1.0000 1.000	0.5532 0.012	0.5589 0.010	/	0.4982 0.012	0.4800 0.008	0.4611 0.016
LFQA	Pythia	0.9990 1.000	/	0.5139 0.018	0.5202 0.016	0.5184 0.004	0.4909 0.010	0.5187 0.020
	Qwen	0.9998 0.998	0.5159 0.010	/	0.4870 0.012	0.4885 0.004	0.4711 0.006	0.4896 0.016
	OpenLLaMA	0.9995 1.000	0.5016 0.018	0.4888 0.026	/	0.5129 0.010	0.4666 0.004	0.4935 0.022

The matched reference is shown in the Native column; a diagonal stroke marks that same tokenizer, not a missing result.

Table 12: ITS: matched versus replacement tokenizers. Each cell shows AUROC (top) and TPR (bottom). Cells with TPR below 0.2 are shaded pale yellow.

		Matched	Replacement detector tokenizer					
Domain	Generator	Native	Pythia	Qwen	OpenLLaMA	OPT	GPT-Neo	TinyLLaMA
C4	Pythia	0.9911 0.980	/	0.4878 0.006	0.4877 0.006	0.4936 0.002	0.4394 0.010	0.5230 0.004
	Qwen	0.9994 0.992	0.4831 0.010	/	0.4997 0.016	0.4874 0.004	0.4852 0.004	0.4812 0.004
	OpenLLaMA	0.9944 0.980	0.4988 0.016	0.5271 0.008	/	0.4894 0.008	0.4400 0.008	0.5652 0.016
Wiki	Pythia	0.9994 0.994	/	0.4388 0.008	0.3487 0.002	0.4777 0.004	0.4854 0.012	0.2893 0.000
	Qwen	0.9877 0.928	0.5881 0.012	/	0.3527 0.002	0.4898 0.006	0.3925 0.004	0.3406 0.000
	OpenLLaMA	0.9953 0.982	0.5820 0.018	0.4687 0.012	/	0.5016 0.016	0.4581 0.016	0.3501 0.006
LFQA	Pythia	0.9987 0.994	/	0.5069 0.006	0.5674 0.014	0.5384 0.012	0.4027 0.000	0.5226 0.010
	Qwen	0.9949 0.980	0.5132 0.014	/	0.5784 0.010	0.5671 0.010	0.4335 0.004	0.5153 0.010
	OpenLLaMA	0.9982 0.990	0.5319 0.010	0.5322 0.012	/	0.5503 0.016	0.3711 0.006	0.5538 0.016

The matched reference is shown in the Native column; a diagonal stroke marks that same tokenizer, not a missing result.

Tables 11 and 12 give the same matrices for EXP and ITS. Matched detection reaches AUROC of at least 0.98 in every group, whereas the replacement cells average AUROC 0.502 for EXP and 0.481 for ITS, with TPR of 0.014 and 0.009.

D.3 Encoder and generator replacement for semantic methods

Semantic watermarks score text in the embedding space of a specified encoder, and their detectors store artifacts derived from that space, such as cluster centers or a calibrated similarity interval. This

experiment holds the generated text fixed and replaces only the encoder used at detection, leaving every stored artifact unchanged. Each method is evaluated with the five encoders common to all methods; the matched embedding model is the one used at generation. A replacement whose output dimension differs from that of a stored artifact defines no detector and is reported as undefined rather than as a score of zero.

Table 13 reports the study on C4 with each method's author generation model. The matched embedding model detects the watermark reliably. After replacement, TPR falls in every scoreable configuration. Ranking performance varies across replacements: most AUROC values are near or below chance, while SimMark with instructor-base in place of instructor-large retains partial discrimination. Both encoders belong to the instructor model family (Su et al., 2023; Ni et al., 2022), suggesting that the replacement preserves some of the embedding-space structure used by the detector.

Table 13: Detection-encoder replacement on C4 with each method's author generation model. The matched embedding model is the one used at generation. TPR cells below 0.2 are shaded pale yellow.

Method	Detection encoder	Relation	AUROC	TPR
SimMark	instructor-large	matched	0.9889	0.9178
	semstamp-c4-sbert	replacement	0.3448	0.0080
	instructor-base	replacement	0.9051	0.3006
	mpnet-base-v2	replacement	0.2982	0.0240
	MiniLM-L6-v2	replacement	0.3780	0.0341
SemStamp	semstamp-c4-sbert	matched	0.9996	0.9914
	instructor-large	replacement	0.4718	0.0172
	instructor-base	replacement	0.4517	0.0237
	mpnet-base-v2	replacement	0.6314	0.1013
	MiniLM-L6-v2	replacement	0.4741	0.0108
k-SemStamp	semstamp-c4-sbert	matched	0.9938	0.9660
	instructor-large	replacement	0.4851	0.0140
	instructor-base	replacement	0.5131	0.0180
	mpnet-base-v2	replacement	0.7316	0.0780
	MiniLM-L6-v2	replacement	–	–
AliMark	mpnet-base-v2	matched	0.9999	0.9980
	semstamp-c4-sbert	replacement	0.8452	0.1880
	instructor-large	replacement	0.4281	0.0040
	instructor-base	replacement	0.6267	0.0020
	MiniLM-L6-v2	replacement	–	–

The k-SemStamp/MiniLM-L6-v2 and AliMark/MiniLM-L6-v2 detectors are structurally undefined (768-dimensional frozen centers or secret vectors versus a 384-dimensional replacement), not zero. Operating points use the same-pool empirical 1% threshold.

Table 14 evaluates the semantic methods with Pythia-1.4B as a general-purpose generator, keeping the authors' watermark parameters unchanged. For SimMark, SemStamp, and k-SemStamp, this replaces the fine-tuned OPT-1.3B released with SemStamp (Hou et al., 2024a). Matched detection performance declines for some methods; SimMark's TPR on C4 decreases from 0.92 to 0.62. The Wiki rows extend the matched-detection evaluation to a second dataset with the same generator and watermark parameters.

On the Pythia-generated C4 texts, encoder replacement also reduces TPR in every scoreable configuration.

Table 14: Semantic methods with Pythia-1.4B as the generation model and author watermark parameters. On C4 the detection encoder is replaced; on Wiki only the matched embedding model is evaluated. The matched embedding model is the one used at generation. TPR cells below 0.2 are shaded pale yellow.

Method	Data	Detection encoder	Relation	AUROC	TPR
SimMark	C4	instructor-large	matched	0.9733	0.6213
	Wiki	instructor-large	matched	0.9968	0.9299
	C4	semstamp-c4-sbert	replacement	0.4701	0.0021
	C4	instructor-base	replacement	0.8592	0.1444
	C4	mpnet-base-v2	replacement	0.2650	0.0084
	C4	MiniLM-L6-v2	replacement	0.2492	0.0167
SemStamp	C4	semstamp-c4-sbert	matched	0.9946	0.9505
	Wiki	semstamp-c4-sbert	matched	0.9964	0.9281
	C4	instructor-large	replacement	0.5116	0.0113
	C4	instructor-base	replacement	0.5540	0.0135
	C4	mpnet-base-v2	replacement	0.6104	0.0383
	C4	MiniLM-L6-v2	replacement	0.5396	0.0045
k-SemStamp	C4	semstamp-c4-sbert	matched	0.9872	0.9130
	Wiki	semstamp-c4-sbert	matched	0.9901	0.8621
	C4	instructor-large	replacement	0.5706	0.0081
	C4	instructor-base	replacement	0.6033	0.0101
	C4	mpnet-base-v2	replacement	0.6740	0.0425
	C4	MiniLM-L6-v2	replacement	–	–
AliMark	C4	mpnet-base-v2	matched	0.9969	0.9940
	Wiki	mpnet-base-v2	matched	1.0000	1.0000
	C4	semstamp-c4-sbert	replacement	0.8273	0.1860
	C4	instructor-large	replacement	0.4053	0.0000
	C4	instructor-base	replacement	0.6514	0.0000
	C4	MiniLM-L6-v2	replacement	–	–

A dash denotes a structurally undefined detector: for k-SemStamp and AliMark, MiniLM-L6-v2 produces 384-dimensional embeddings incompatible with the stored 768-dimensional centers or secret vectors.

E STATISTICAL CORRECTION OVER CANDIDATE TOKENIZERS

A token-level detector needs the tokenizer that produced the text, and Section D shows that with a wrong tokenizer, detection falls to near chance level. In practice the text often arrives without that information: the model behind an online service is undisclosed, or the text has been collected without source metadata. A verifier may still know the watermarking method and have access to a finite set of plausible tokenizers, such as those of the public models in use. This section analyzes that setting: the method is known, the generating tokenizer is unknown, and the verifier scores the text under each of K candidate tokenizers, obtaining $s_j(x)$ for candidate j, and must decide from these K scores together.

E.1 WHY SEARCH REQUIRES CORRECTION

A search declares a positive result when any candidate test rejects. Its rejection event is the union of the candidate rejection events, so applying a single-test threshold to every candidate can raise the overall false-positive rate above the nominal level. Calibration must therefore account for the combined decision.

E.2 Calibration of the combined decision

If each candidate supplies a valid null p-value p_j, so that $\Pr_0(p_j \le t) \le t$ for $0 \le t \le 1$, Bonferroni correction rejects when $\min_j p_j \le \alpha/K$. The union bound gives

$$\Pr_0\left(\min_j p_j \le \alpha/K\right) \le \sum_{j=1}^{K} \Pr_0(p_j \le \alpha/K) \le \alpha.$$

This guarantee requires neither independence nor identical score distributions. When valid candidate p-values are unavailable, one can instead use $M(x) = \max_j s_j(x)$ and calibrate its threshold from the joint null score distribution. A threshold τ satisfying $\Pr_0(M > \tau) \le \alpha$ controls the combined false-positive rate. Empirically, the verifier computes all candidate scores on each calibration text, takes their maximum, and uses the empirical quantile of these maxima as the threshold.

E.3 Detection cost of candidate search

If the generating tokenizer is included, Bonferroni tests it at level α/K rather than α, and maximum-score calibration sets a threshold on M that cannot be lower than the single-test threshold, since adding candidates cannot decrease the maximum on any text. As K increases, the per-candidate level becomes smaller, which can reduce detection power relative to direct detection with the known tokenizer.

If the generating tokenizer is absent, the correction still controls the false-positive rate, but no candidate reads the text in the representation that carries the watermark, so the search has almost no detection power.

E.4 Empirical false-positive inflation

The size of the cost depends on how much null variation the candidate scores share. If the candidates' rejection events at their calibrated levels were identical, the search would cost nothing; if they were independent, the search would pay the full multiplicity price, with family-wise rate $1-(1-\alpha/K)^K$ for candidate-wise level α/K, which differs from α only at second order in α.

Table 4 in Section 4.1 reports the false-positive rate of an uncorrected search over the $K = 6$ candidate tokenizers of Section D; it is close to the value for independent candidates.

ByteMark does not require this search, because its detector operates directly on the text's bytes and has no tokenizer to identify.

F Watermark signal continuity and preservation

This section gives the setup of the relay experiment whose results appear in Section 4.2. A common byte interface allows several generators to continue one watermark signal; a token-level detector, tied to one tokenizer, follows the signal only through the segments written with that tokenizer.

F.1 Motivation

The relay experiment tests whether one fixed detector can recover watermark evidence throughout a document generated by several models under the same key.

For a token-level watermark this is not ensured by the shared key alone. The generation bias and the detection statistic are both defined on the token sequence of a particular tokenizer, so the signal written by one model is only visible to a detector that retokenizes the text the same way. When the models in a relay use different tokenizers, a detector fixed to one of them reads the other segments as unmarked text, and the signal it can recover is limited to the share of the document written with its own tokenizer. Searching over candidate tokenizers does not remove this requirement. The verifier must have access to a candidate set known to include the tokenizers of all contributing models and must correct for multiple testing (Appendix E); even with complete coverage, each fixed-tokenizer detector primarily recovers signal from the segments generated with its own tokenizer. Without assured coverage, a negative search result cannot rule out a watermark tied to an omitted tokenizer.

ByteMark defines both the bias and the statistic on the UTF-8 bytes of the text. Every model in a relay scores its candidate tokens through the bytes they contribute, and the byte context that the PRF conditions on is carried across the model boundary. The detector therefore reads the relayed document as one byte stream and does not need to know that a switch occurred.

F.2 Experimental setup

Relay generation. Three models generate one C4 completion in turn: Pythia-1.4B writes the first segment from the original prompt, Qwen2.5-1.5B continues the accumulated text, and OpenLLaMA-3B continues it again. Each stage receives the full text produced so far as its prompt and generates up to 100 native tokens under its own tokenizer. The cohort is 500 original requests per method, drawn from the same frozen C4 prompts as the single-model experiments, with rejected or incomplete stages recorded and excluded rather than replaced. The methods compared are ByteMark, KGW, and ITS, each with the configuration of Section C, and each model in the chain embeds the same key.

Detection. ByteMark is detected once, with the tokenizer-free detector of Section A, over the whole completion. KGW and ITS are detected with one fixed tokenizer over the whole completion, once for each of the three tokenizers in the chain, so that every detector reads all three segments in the representation of one participant. Prompts are excluded from detection in every case.

Local signal. To show where in the document each detector finds the watermark, we compute a local statistic over windows of five consecutive tokens and plot its mean across texts against the normalized position in the document. Each of the three segments is mapped to unit length on the horizontal axis, so model switches fall at positions one and two. KGW and ITS count tokens of their detector tokenizer and use their respective standardized statistics on the tokens in the window, with the full-text context; for ITS the per-token key distances are standardized under the key alignment chosen by the document-level detector. ByteMark counts the generating model's tokens and applies the byte statistic of Equation 9 to the bytes they cover, counting each distinct context–byte pair once. Windows that would extend past the document are omitted, and a mean is shown only where at least half of the texts contribute a window. The dotted references in the figure mark the ideal null value of each statistic and are not decision thresholds; the curves describe the local signal and are not detection rates.

G Editing and rewriting attacks

We evaluate watermark detection after lightweight edits and paraphrasing, and separately examine re-embedding during rewriting.

G.1 Lightweight editing attacks

We evaluate two lightweight edits on all accepted C4 completions from the three source models. *Word deletion* splits the completion at whitespace and deletes each word independently with probability 0.15. *Token insertion* encodes the completion with the shared reference tokenizer (Pythia-1.4B), inserts $\lfloor 0.15\, n \rfloor$ tokens drawn uniformly from the vocabulary, excluding special tokens, at uniformly chosen positions among the n tokens, and decodes the result. Each completion uses its own fixed random seed, so every method's text is edited by the same operations. The edited text is then detected by each method's own detector. Table 15 reports the results. For ByteMark, KGW, SynthID, and EXP the AUC remains above 0.95 after both edits, so watermarked and human text remain well separated; ITS is affected more. A high AUROC after editing does not imply a high TPR at the threshold calibrated on clean text. A deployment that expects edited text can instead detect with a windowed maximum statistic such as WinMax (Kirchenbauer et al., 2024), with a threshold recalibrated for that statistic.

Table 15: C4 word-deletion and token-insertion results. Each cell is the AUC after the transformation.

Method	Source model	Word deletion	Token insertion
ByteMark	Pythia	0.9994	0.9976
ByteMark	Qwen2.5	0.9995	0.9982
ByteMark	OpenLLaMA	0.9978	0.9944
SynthID	Pythia	0.9998	0.9987
SynthID	Qwen2.5	1.0000	0.9999
SynthID	OpenLLaMA	0.9992	0.9999
KGW	Pythia	0.9985	0.9992
KGW	Qwen2.5	0.9999	0.9999
KGW	OpenLLaMA	0.9997	0.9996
EXP	Pythia	0.9957	0.9685
EXP	Qwen2.5	0.9981	0.9573
EXP	OpenLLaMA	0.9972	0.9577
ITS	Pythia	0.6871	0.6092
ITS	Qwen2.5	0.7790	0.6091
ITS	OpenLLaMA	0.7280	0.5931

G.2 Re-embedding and survival under rewriting

We evaluate rewriting under two conditions. Without re-embedding, a non-participating rewriter adds no watermark, and detection measures survival of the original watermark. With ByteMark re-embedding, a participating rewriter embeds the same keyed signal while it rewrites and carries its byte context across window boundaries, and detection evaluates the newly embedded signal. Both conditions use the same source completions; the watermarked-rewriter results are discussed in Section 4.2, and the unwatermarked-rewriter results below. We use three rewriters on the 200-reference-token C4/Pythia-1.4B cohort; Parrot (Damodaran, 2021) rewrites the completions sentence by sentence. DIPPER-XXL (Krishna et al., 2023) rewrites the Pythia-1.4B completions in successive sentence windows with lexical diversity 60 and order diversity 20. It uses nucleus sampling ($p = 0.75$, temperature 1, no top-k filtering) in BF16. Each request has a single budget of 300 new decoder tokens, and text reaching this limit is retained as is. Each document and window uses an independently seeded random stream. Rewritten outputs are detected in full with each method's detector unchanged, and we report AUROC, which needs no threshold, on the valid outputs of each condition; generation masks and truncation indicators are retained with the data.

Instruction-model rewriting. The third rewriter is Mistral-7B-Instruct-v0.3 on the same source cohort. The prompt requests a rewrite preserving facts, meaning, and approximate length, without a preface. Generation uses BF16, nucleus sampling with $p = 0.75$, temperature 1, no top-k filtering, an independently seeded stream per document, and a cap of 300 new tokens. The non-participating rewriter adds no watermark; the ByteMark participating rewriter uses the same key and $h = 8$, $\gamma = 0.25$, $\delta = 2$ configuration as the source watermark. Detection reads the complete rewritten output. Table 16 reports AUROC under all three rewriters; under Mistral, ByteMark attains 0.989 with the participating rewriter compared with 0.835 with the non-participating rewriter.

Table 16: AUROC after rewriting C4/Pythia-1.4B completions with three rewriters. The rewriter adds no watermark except in the ByteMark (participating) row, where it re-embeds the ByteMark watermark.

Method	Parrot	DIPPER	Mistral-7B
ByteMark (participating)	**0.9985**	**0.9998**	**0.9888**
ByteMark (non-participating)	0.9690	0.7499	0.8348
KGW	0.9791	0.8635	0.8990
SynthID	0.9797	0.7036	0.7920
EXP	0.9643	0.7327	0.8237
ITS	0.5281	0.5459	0.6273

H Other related work

Other approaches embed watermarks through formatting, lexical substitution, or prompting. Formatting and steganographic carriers hide a payload in whitespace or glyph choice (Rizzo et al., 2016; Sato et al., 2023), lexical carriers substitute synonyms under a keyed rule (Topkara et al., 2006), and in-context watermarking asks the model to follow lexical, acrostic, or Unicode rules stated in the prompt (Liu et al., 2026). Whitespace and glyph carriers can be removed by ordinary copying and Unicode normalization; synonym substitution needs its dictionary at detection and alters wording; prompted rules depend on the model's instruction following, and a rule visible in the output can be inferred and suppressed by a further instruction.

I Statistical diagnostics

This section compares continuous and binary detection statistics, examines scores across Unicode character classes, and evaluates false-positive behavior on human text under fixed keys.

I.1 Continuous and binary statistics

A natural alternative to the continuous statistic counts bytes whose scores exceed $1 - \gamma$. This standardized green-byte count follows the form of the green-count statistic used in KGW (Kirchenbauer et al., 2023) and is computed from the same distinct byte scores:

$$z_{\text{bin}} = \frac{\sum_{i=1}^{m} \mathbf{1}[u_i > 1 - \gamma] - m\gamma}{\sqrt{m\gamma(1-\gamma)}}.$$

The continuous statistic retains score magnitudes discarded by the binary count. On the same generated texts, it achieves slightly higher mean AUROC and TPR across the nine evaluated conditions.

Table 17: Continuous and binary statistics on the same byte scores, using at most 200 reference tokens. Entries are AUROC / TPR, each at its own statistic's threshold.

Domain	Generator	Continuous	Binary	Δ (Cont. − Bin.)
C4	Pythia	0.9997 / 0.996	0.9997 / 0.994	0.0000 / +0.002
	Qwen	1.0000 / 0.998	0.9999 / 0.998	+0.0001 / 0.000
	OpenLLaMA	0.9989 / 0.992	0.9987 / 0.984	+0.0002 / +0.008
WIKI	Pythia	1.0000 / 1.000	0.9998 / 0.994	+0.0002 / +0.006
	Qwen	0.9976 / 0.978	0.9963 / 0.961	+0.0013 / +0.017
	OpenLLaMA	1.0000 / 1.000	0.9999 / 0.996	+0.0001 / +0.004
LFQA	Pythia	1.0000 / 1.000	0.9998 / 0.994	+0.0002 / +0.006
	Qwen	0.9996 / 0.990	0.9974 / 0.951	+0.0022 / +0.039
	OpenLLaMA	0.9999 / 0.998	0.9993 / 0.986	+0.0006 / +0.012

Δ reports continuous minus binary AUROC / TPR, computed from the displayed values; positive values favor continuous detection.

This ablation changes only the detection statistic, keeping the ByteMark-generated texts fixed.

I.2 Signal coverage across character classes

For each Unicode character in the accepted completions, we average the scores of its constituent UTF-8 bytes. We then group characters by class and compute the character count and mean character score for each group. Table 18 reports these means separately for C4, Wiki, and LFQA, together with the overall means weighted by character counts across corpora. The null reference mean is 0.5.

Table 18: Mean character scores by Unicode class across nine generation conditions. The null reference mean is 0.5.

Class	Characters	Mean u	C4	Wiki	LFQA
ASCII letter	1,456,711	0.5764	0.5756	0.5768	0.5767
Whitespace	324,076	0.5124	0.5114	0.5106	0.5151
ASCII punctuation	65,740	0.6258	0.6258	0.6270	0.6245
Digit	21,553	0.6216	0.6202	0.6201	0.6257
Other	3,239	0.5925	0.5881	0.6015	0.5939
Other letter	1,065	0.5841	0.5819	0.5938	0.5483
CJK	868	0.5922	0.5921	0.6019	0.5769
Emoji / symbol	174	0.5710	0.5969	0.5582	0.5423

Each character contributes the mean of its byte scores. Overall means are weighted by character counts across corpora. Data are ByteMark-watermarked completions generated by Pythia-1.4B, Qwen2.5-1.5B, and OpenLLaMA-3B on C4, Wiki, and LFQA.

I.3 Null moments and realized false-positive rates on human text

Theorem 1 gives the null distribution of z for an ideal PRF and a fixed text. This subsection compares the observed moments and false-positive rates on human text with this ideal-null reference: whether the detection scores have the predicted mean and variance, and how often the normal-approximation thresholds $z > 2$, $z > 3$, and $z > 4$ reject without any calibration data. We score five human corpora (C4, Wikipedia, LFQA, and the Chinese and German portions of mC4) under five keys and report, for each corpus, the mean and standard deviation of z and the realized false-positive rate pooled over the keys (Table 19). Two statistics are compared: the deduplicated statistic of Section A, which counts each distinct context–byte input once, and an all-occurrences statistic that counts every occurrence and normalizes as if the occurrences were independent.

On most evaluated corpora for both statistics the mean of z is close to zero, the standard deviation close to one, and the realized rates at $z > 2$ and $z > 3$ close to the nominal 2.28% and 0.135%. The exception is the all-occurrences statistic on the Chinese and, to a lesser degree, German text, where repeated context–byte pairs are frequent and the standard deviation rises to 1.33 and 1.14, with the rate at $z > 2$ reaching 5.1% and 4.0%. The deduplicated statistic keeps the standard deviation within 0.03 of one and the realized rates at 1.9–2.4% on every corpus, including the two where the all-occurrences statistic departs from unit variance.

Table 19: Detection-score moments on human text and the realized false-positive rate at the normal-approximation thresholds $z > 2$, $z > 3$, and $z > 4$, whose nominal rates are 2.28%, 0.135%, and 0.003%.

				Realized FPR		
Corpus	Statistic	Mean $\bar{z}$	SD s_z	$z > 2$	$z > 3$	$z > 4$
C4	Deduplicated	+0.0230	0.9863	0.0202	0.0012	0.0000
	All occurrences	+0.0190	1.0332	0.0272	0.0020	0.0000
Wiki	Deduplicated	+0.0250	0.9735	0.0216	0.0008	0.0000
	All occurrences	+0.0234	1.0005	0.0244	0.0004	0.0000
LFQA	Deduplicated	-0.0015	1.0117	0.0244	0.0020	0.0000
	All occurrences	-0.0012	1.0535	0.0298	0.0024	0.0006
mC4 zh	Deduplicated	-0.0161	0.9819	0.0189	0.0008	0.0000
	All occurrences	-0.0007	1.3303	0.0511	0.0131	0.0048
mC4 de	Deduplicated	+0.0176	0.9834	0.0211	0.0008	0.0000
	All occurrences	+0.0200	1.1382	0.0395	0.0063	0.0000

1,000 documents per corpus for C4, Wiki, and LFQA and 1,500 for mC4 Chinese and German, truncated to 400 characters and scored under five keys; realized rates pool the keys. Deduplicated counts each distinct context–byte input once; All occurrences counts every occurrence and normalizes as if the occurrences were independent.

J COMPUTATIONAL COST AND REPRODUCIBILITY

We report generation cost and, in a separate experiment, the agreement between independent Python, JavaScript, and C++ detector implementations.

J.1 GENERATION COST

Measurement environment. All generation-efficiency measurements run on the same machine: one laptop with an NVIDIA GeForce RTX 5090 Laptop GPU (24 GB), an AMD Ryzen AI 9 HX 370 CPU (24 threads), and 32 GB of memory, under Ubuntu 26.04 LTS with Python 3.11, PyTorch 2.10.0 (CUDA 12.8), and Transformers 5.3.0. The machine runs no other workload during the measurements; each method runs in its own process, one at a time, with the model loaded once and all requests issued sequentially at batch size one. The machine keeps fixed thermal limits throughout (CPU frequency capped near 3.0 GHz, GPU core clock at most 2,000 MHz), identical for every method.

Throughput is accepted final-output tokens divided by generation-call time. Timing includes final decode and CUDA synchronization but excludes model loading, warm-up, method setup, and scheduling. The OpenLLaMA-3B measurements use 50 frozen C4 requests, 200 native tokens, FP16, batch size one, and identical model revisions and prompt-token IDs.

J.2 CANDIDATE WORK IN SENTENCE-LEVEL SEMANTIC METHODS

Candidate-token work counts all sampled continuation tokens, including discarded sentence candidates, per final-output token. It is a work measure rather than a latency multiplier. Table 5 compares all six generation paths on the same 50 C4 texts with OpenLLaMA-3B on the same GPU at batch size one. The sentence-level methods achieve lower throughput than unwatermarked generation and sample many discarded candidate tokens for each retained token, whereas ByteMark samples no extra tokens and incurs the cost of scoring the vocabulary. Parallelizing candidates may alter latency while leaving discarded-token work a relevant measure of inference demand.

J.3 DETECTOR AGREEMENT ACROSS IMPLEMENTATIONS

We evaluate detector reproducibility on 100 texts from the main C4/OpenLLaMA experiment: the first 50 accepted watermarked completions and the first 50 human texts in original sample order, selected without reference to their detection scores. Independent JavaScript and C++ implementations are compared with the Python detector on the same frozen texts and AES-CMAC key, with context width $h = 8$. Table 20 summarizes the replayed texts and lists each compared quantity.

Table 20: Detector agreement on 100 texts from the C4/OpenLLaMA experiment (AES-CMAC, $h = 8$). Top: the replayed texts. Bottom: for each compared quantity, the percentage of texts or byte positions at which the JavaScript or C++ output is identical to that of the Python detector. All comparisons are exact, without numerical tolerance.

Texts	Count	Bytes	Distinct inputs	Median $z(x)$ [range]
Watermarked	50	21,326	20,866	4.81 [2.73, 7.13]
Human	50	22,206	21,359	0.12 [-2.16, 2.38]

		Match with Python	
Compared quantity	Comparisons	JavaScript	C++
Statistic ID and key; byte and character counts	100 texts	100%	100%
UTF-8 byte string	100 texts	100%	100%
Distinct PRF inputs and largest multiplicity	100 texts	100%	100%
Centered sum $T(x)$ (binary64)	100 texts	100%	100%
Null variance $V(x)$ (binary64)	100 texts	100%	100%
Score $z(x)$ (binary64)	100 texts	100%	100%
Context window and PRF message	43,532 bytes	100%	100%
AES-CMAC digest (first 64 bits)	43,532 bytes	100%	100%
Byte score u (binary64)	43,532 bytes	100%	100%
Deduplication group	43,532 bytes	100%	100%

All compared quantities match exactly across Python, JavaScript, and C++ on the 100 evaluated texts, including the binary64 representations of $z(x)$. This agreement is consistent with the standardized AES-CMAC primitive (Song et al., 2006) and the fixed byte-encoding and numerical-conversion rules of ByteMark.

User-side detection. The JavaScript detector can be integrated into a browser extension, an editor, or a reviewing tool. A verifier holding the key can run detection locally from the published text without a language model, a tokenizer, or a call to the provider.

K A BYTE-LEVEL EXTENSION TO MULTILAYER REWEIGHTING

We outline how byte-level scores could be used in the multilayer reweighting of SynthID-Text (Dathathri et al., 2024). For each layer d, we replace its token-based g-value with $R_d(v; c)$, the candidate's average byte score computed from $u_b^{(d)}(c)$. The layer index is included in the PRF input. The remaining reweighting follows the original construction, while detection reconstructs the layer scores from the published bytes and counts each distinct context–byte pair once per layer; a separately specified continuous-score statistic, such as z of Equation 4 computed over the distinct inputs of all layers, then aggregates the evidence. The calibration of the original SynthID detector, which assumes binary g-values, is not inherited. The evaluation of this extension is left for future work.

L LIMITATIONS AND OPEN QUESTIONS

Semantic robustness. As with token-level watermarks, rewriting by a non-participating model can weaken the ByteMark signal by changing the byte stream (Appendix G). An open question is whether a fixed, public semantic representation could improve robustness to rewriting while preserving detection from the text, key, and public specification alone.

Key management. Like other keyed watermarks, ByteMark depends on the secrecy of its key. Because one key serves every participating model, a wider deployment places the key with more parties and raises the risk of its loss, and a lost key exposes the watermark of every participant at once.