RANDOMNESS REUSE IN LLM WATERMARKS: SECURITY BOUNDS AND BLACK-BOX SPOOFING

Anonymous authors
Paper under double-blind review

ABSTRACT

Generative watermarking is increasingly used to identify LLM-generated content. While theoretical security notions and practical attacks have been studied, their relationship and whether attacks exploit general design properties or implementation details remain unclear. We study these questions through formal analysis and black-box spoofing attacks on SynthID-Text. We first connect two watermark security goals usually analyzed separately: undetectability and spoofing resistance, showing that a successful spoofing attack also yields a distinguisher between watermarked and clean LLM generation when one terminal verification query on a fresh candidate is allowed. We then derive a general security bound for context-dependent watermarking schemes in which randomness reuse explicitly degrades security. Our experiments test whether this reuse is exploitable in practice. On SynthID-Text, we developed black-box spoofing attacks that recover watermark bias from model outputs. An implementation-specific weakness in the public Hugging Face implementation further strengthens the attack, yielding joint detection-and-automated-quality rates of 82.6% and 88.7% on Gemma-2B and Gemma-9B prompts, respectively.

1 INTRODUCTION

Large language models (LLMs) are now widely used for content generation, increasing interest in methods for identifying the provenance of generated text. Text watermarking addresses this problem by embedding a detectable statistical signal during generation, and a growing literature has proposed a variety of watermarking schemes (Kirchenbauer et al., 2023; Zhao et al., 2024; Hu et al., 2024; Kuditipudi et al., 2024; Christ et al., 2024; Wu et al., 2024). Watermarking is also moving toward deployment: Google evaluated SynthID-Text on nearly 20 million Gemini responses before deploying it in Gemini (Dathathri et al., 2024), while Anthropic has announced plans to watermark future Claude-generated text (Anthropic, 2026). The EU AI Act further requires providers of generative AI systems to support machine-readable marking of AI-generated content (European Union, 2024). These developments make the security of practical watermarking systems increasingly important.

Prior spoofing attacks. Existing spoofing analyses of text watermarks are largely construction- and attack-specific. Several works have shown that watermark information can be learned without access to the secret key. Jovanović et al. (2024b) develop an automated approach that estimates hidden watermark behavior from black-box outputs and uses it to generate spoofed text, later applying the same idea to SynthID-Text (Jovanović et al., 2024a). DITTO (An et al., 2026) instead fine-tunes a student model on watermarked outputs, while Adaptive Stealing (Zhang et al., 2026) estimates how the next-token distribution changes across contexts. Related work also studies how common watermark design choices can create security trade-offs (Pang et al., 2024). These attacks establish that particular constructions can be spoofed, but provide limited guidance about which structural properties make a watermark vulnerable or whether the same attack principle should transfer to other constructions. This motivates a more general question: can watermark attacks be understood at the level of the underlying construction, rather than requiring a new attack analysis for each scheme?

Undetectability and spoofing resistance. Two security properties are particularly relevant for text watermarking. *Undetectability* asks whether an attacker without the secret key can distinguish watermarked generation from clean LLM generation, while *spoofing resistance* asks whether an

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.

> YouTube doesn't actually have a system where they "pay every YouTuber." The earning structure is based on how creators generate revenue from their content. **1. Ad Revenue:** Creators participating in the YouTube Partner Program (YPP) can monetize their videos with ads. When people watch ads on these videos, YouTube pays the creator a share of the ad revenue. [. . .]

Figure 1: Spoofed text accepted by the Gemma-9B detector; the genuine watermarked answer to the same prompt is rejected.

attacker can produce a fresh sequence that is accepted as watermarked by the detector (Gu et al., 2024; Jovanović et al., 2024b). Christ et al. (2024) formalize undetectability through cryptographic indistinguishability, while Gloaguen et al. (2025) show that practical black-box access can suffice to distinguish watermarked from unwatermarked models and recover information about the watermark. Some constructions aim to satisfy both properties (Lin et al., 2026), but the relationship between the two notions remains unclear. In particular, it is unclear what a successful spoofing attack implies about undetectability, or under what conditions information that reveals the presence of a watermark can also be reused to spoof it. This raises a second question: under what conditions can information learned from watermarked generation be turned into a successful spoofing attack?

Scope and approach. We study context-dependent LLM watermarking through formal security analysis and black-box spoofing attacks. Our formal contribution has two parts. First, we connect spoofing resistance and undetectability by showing that a successful spoofing attack also gives a way to distinguish watermarked from clean generation when one final verification query is allowed. Second, we derive a security bound that separates the effect of the sampling rule from reuse of hidden watermark randomness when contexts repeat. The bound makes context reuse an explicit source of security degradation and, more generally, captures the effect of hidden-randomness reuse. To our knowledge, this is the first work to formally connect spoofing resistance and undetectability in this way and to derive a cryptographic security bound that explicitly captures context reuse in context-dependent watermarking. This effect does not rely on a weakness in the pseudorandom function: even an ideal random function returns the same value on repeated inputs. Longer contexts reduce such repeats but make the watermark more brittle to token edits, since one edit affects more subsequent contexts.

We use SynthID-Text as a concrete case study of whether this randomness reuse is exploitable in practice. Our exact-context attack exploits the mechanism above: repeated watermark contexts receive the same hidden randomness, allowing the attacker to estimate changes in next-token probabilities. We then identify an additional, implementation-specific form of reuse in the public Hugging Face implementation (Ghaisas et al., 2024). Different token windows can receive the same watermark values, allowing observations to be pooled across contexts and greatly reducing the sparsity of the exact-context attack. Thus, both attacks exploit hidden randomness reuse, through repeated contexts in the first case and a public equivalence between different token windows in the second. Although prior work has already shown that SynthID-Text can be spoofed (Jovanović et al., 2024a; An et al., 2026; Zhang et al., 2026), to our knowledge it has not identified or exploited this public equivalence. Our experiments therefore separate the general context-induced reuse studied in our analysis from the additional reuse introduced by this implementation.

Key contributions. Our main contributions are:

- Section 3 formalizes the connection between spoofing resistance and undetectability and shows how a successful spoofing attack implies distinguishability when one final verification query is allowed.
- Section 4 derives a security bound for context-dependent watermarking in which randomness reuse appears explicitly as a source of security degradation, even when the hidden mapping is an ideal random function.
- Sections 5 and 6 develops an exact-context black-box spoofing attack on SynthID-Text. It reaches 94.5% detector acceptance on Qwen-0.5B and 82.4% on Gemma-2B.
- Section 6 identifies an additional implementation-specific weakness in the public Hugging Face Transformers implementation of SynthID-Text. Different token windows can receive

the same watermark values, allowing observations to be pooled across them. The resulting attack achieves 82.6% and 88.7% joint detection-and-quality rates on frozen held-out Gemma-2B and Gemma-9B prompts, respectively; on Gemma-9B, this exceeds the 81.8% rate of the deployment's own watermarked output. Figure 1 gives an example.

2 SAMPLING-BASED WATERMARKING SCHEME

Let $\mathcal{V}$ denote the token vocabulary, and let $\mathcal{Y} = \bigcup_{\ell \geq 0} \mathcal{V}^\ell$ be the set of finite token sequences. A prompt is an initial token sequence $X \in \mathcal{Y}$, and an output is denoted by $Y = (Y_1, \ldots, Y_\ell) \in \mathcal{Y}$. Here $Y_i \in \mathcal{V}$ is the i-th generated token, $Y_{<i}$ denotes the tokens generated before step i, and ℓ is the output length.

Clean generation. At generation step i, the language-model context is $X \| Y_{<i}$, where $\|$ denotes sequence concatenation. The language model defines the next-token distribution

$$p_i(\cdot) = p_{\mathsf{LM}}(\cdot \mid X \| Y_{<i}) \in \Delta(\mathcal{V}),$$

where $\Delta(\mathcal{V})$ is the set of probability distributions over $\mathcal{V}$. Clean generation samples $Y_i \sim p_i$. Repeating this procedure defines the randomized clean generator Clean, and we write $Y \leftarrow \mathsf{Clean}(X)$ for an output generated from prompt X.

Watermarked generation. A watermarked generator uses the same underlying language model but modifies the next-token sampling rule using a secret key. Throughout this work, we consider context-keyed sampling-based watermarks. Let $\mathcal{K}$ be the key space and $K \in \mathcal{K}$ the watermarking key. The watermarking rule depends on a short context window. Let

$$S_i = \mathsf{ctx}_n(Y_{<i}) \in \mathcal{V}^n$$

denote the previous n-token context used for watermarking. If fewer than n tokens have been generated, S_i is defined using a fixed padding or initialization rule. A keyed pseudorandom function $F_K : \mathcal{V}^n \to \mathcal{R}$ maps S_i to a watermark seed in $\mathcal{R}$. A watermark-specific sampling rule $\mathsf{Samp}_{\mathsf{wm}} : \Delta(\mathcal{V}) \times \mathcal{R} \to \Delta(\mathcal{V})$ then maps the language-model distribution p_i and seed $F_K(S_i)$ to the key-dependent next-token distribution

$$p^{\mathsf{wm}}_{i,K}(\cdot) = \mathsf{Samp}_{\mathsf{wm}}\big(p_i, F_K(S_i)\big).$$

Watermarked generation samples $Y_i \sim p^{\mathsf{wm}}_{i,K}$. Repeating this procedure defines the randomized keyed generator Wm_K, and we write $Y \leftarrow \mathsf{Wm}_K(X)$ for an output generated from prompt X using key K.

Example 1 (SynthID-Text by Dathathri et al. (2024))**.** SynthID-Text instantiates this framework using a sliding n-token context and Tournament sampling. Dathathri et al. use $n = 4$ in their experiments. Related context-dependent schemes can also be expressed in this form (Kirchenbauer et al., 2023; Wu et al., 2024). SynthID-Text additionally applies repeated-context masking. Let $A_i \in \{0, 1\}$ indicate whether watermarking is active at position i: $A_i = 0$ if the current context has already been used for watermarking within the retained context history, and $A_i = 1$ otherwise. SynthID-Text therefore samples $Y_i \sim p_i^{\mathsf{SynthID}}$, where

$$p_i^{\mathsf{SynthID}} = \begin{cases} \mathsf{Samp}_{\mathsf{wm}}\big(p_i, F_K(S_i)\big), & A_i = 1, \\ p_i, & A_i = 0\,. \end{cases}$$

Thus, when a context is repeated within the masking history, the corresponding token is sampled directly from the clean language-model distribution rather than watermarked. In its non-distortionary configuration, SynthID-Text uses two competitors per tournament match, for which Tournament sampling is single-token non-distortionary: averaging over the random seed yields the original next-token distribution (Dathathri et al., 2024).

Detection. The scheme uses a keyed detector $\mathsf{Detect}_K : \mathcal{Y} \to \{0, 1\}$, where $\mathsf{Detect}_K(Y) = 1$ means that Y is detected as watermarked under key K. We define the detector false-positive rate as

$$\Pr_{K \leftarrow \mathcal{K}}[\mathsf{Detect}_K(Y) = 1] \leq \alpha_{\mathsf{det}} \qquad \text{for every fixed admissible sequence } Y.$$

For SynthID-Text, detection compares a key-dependent score with a threshold; we represent this procedure by Detect_K.

3 SECURITY DEFINITIONS

Threat model. We consider two security notions for LLM watermarking. In both settings, the attacker does not know the watermark key or any internal watermark value, but has generation access to the unwatermarked model. We first give the intuition before formalizing the two notions.

Undetectability: The attacker interacts with a challenge oracle, such as a model provider, that answers generation queries using either the watermarked generator or the clean generator. Note that the attacker will also get additional clean LLM oracle access. Undetectability asks whether the attacker can determine which generator is being used without knowing the watermark key.

Spoofing resistance: The attacker has generation-query access to the deployed watermarked service and access to the clean LLM model. It observes the returned token sequences and tries to produce a fresh sequence that is accepted by the watermark detector, without knowing the watermark key. In the taxonomy of Jovanović et al. (2024b), this corresponds to the $D0$ setting.

Freshness. Both notions use the following freshness condition. Let $\Phi : \mathcal{Y} \times \mathcal{Y} \to \{0, 1\}$ be a public closeness predicate, where $\Phi(Y, Y') = 1$ means that Y is considered too close to an observed output Y'. We require $\Phi(Y, Y) = 1$, so exact replay is excluded. Following prior works by Fairoze et al. (2025); Lin et al. (2026), let $\mathcal{Q}$ denote the set of generation outputs observed by the attacker and define

$$\mathsf{Fresh}_{\mathcal{Q}}(Y) = 1 \iff \forall Y' \in \mathcal{Q} : \Phi(Y, Y') = 0.$$

Thus, replaying an observed output cannot be made fresh by changing its prompt. For example, $\Phi(Y, Y') = \mathbf{1}\{d_{\mathsf{edit}}(Y, Y') \le r\}$ excludes outputs within r token edits of an observed sequence.

Verification-augmented undetectability. Ordinary undetectability, as defined by Christ et al. (2024), gives a distinguisher access only to generation-oracle and clean LLM oracle outputs. To relate undetectability to spoofing, we strengthen this experiment with one terminal verification query and call the resulting notion *verification-augmented undetectability*. Let $b \in \{0, 1\}$ and sample $K \leftarrow \mathcal{K}$. The distinguisher receives adaptive access to

$$\mathcal{O}_b(X) = \begin{cases} \mathsf{Clean}(X), & b = 0, \\ \mathsf{Wm}_K(X), & b = 1, \end{cases} \qquad \mathcal{O}_{\mathsf{clean}}(X) = \mathsf{Clean}(X),$$

After completing its generation queries, the distinguisher may submit one candidate $(X^\star, Y^\star)$ for verification before making its final decision. Let $\mathcal{Q}$ contain all generation outputs observed during the interaction. If $\mathsf{Fresh}_{\mathcal{Q}}(Y^\star) = 1$, the challenger returns $\mathsf{Detect}_K(Y^\star)$; otherwise it returns 0. The distinguisher then guesses whether $\mathcal{O}_b$ uses the watermarked or clean generator. No further oracle queries are allowed after verification. Allowing this terminal query can only strengthen the distinguisher; ordinary undetectability is recovered by not using it.

Spoofing resistance. Sample $K \leftarrow \mathcal{K}$. A spoofing attacker $\mathcal{A}$ receives adaptive access to Wm_K and Clean. Let $\mathcal{Q}$ contain all generation outputs observed by $\mathcal{A}$. The attacker outputs one candidate $(X^\star, Y^\star)$ and succeeds exactly when

$$F_{\mathsf{spoof}} = \{\mathsf{Fresh}_{\mathcal{Q}}(Y^\star) = 1 \ \wedge \ \mathsf{Detect}_K(Y^\star) = 1\}.$$

The spoofing success probability of $\mathcal{A}$ is the probability that the event F_{spoof} happens.

From spoofing to distinguishability. The two security notions above are directly related. If an attacker can use generation queries to produce a fresh output that the watermark detector accepts, then that acceptance can be used to distinguish the watermarked generator from the clean one. Thus, a successful spoofing adversary can be converted into a distinguisher for verification-augmented undetectability.

To quantify this connection, we use the cryptographic power-bound framework of Beyne & Chen (2024), which views indistinguishability as a binary hypothesis-testing problem. Let α be the probability of incorrectly declaring the clean generator to be watermarked, while $1 - \beta$ be the probability of correctly identifying the watermarked generator. A verification-augmented undetectability power bound is a non-decreasing function $f_{\mathsf{und}}(\cdot)$ such that every allowed distinguisher satisfies $1 - \beta \le f_{\mathsf{und}}(\alpha)$. For the distinguisher induced by a spoofing attacker, its false-positive rate will

satisfy $\alpha \leq \alpha_{\mathsf{det}}$, where α_{det} is the detector's false-positive rate. The formal proof of Theorem 1 is given in Appendix B.

Theorem 1. *Let $\mathcal{A}$ be a spoofing adversary with detector false positive rate α_{det}. For every allowed verification-augmented undetectability distinguisher $\mathcal{D}$, with false positive and negative rates resp. α and β, satisfying $1 - \beta \leq f_{\mathsf{und}}(\alpha)$. Then the spoofing success probability of $\mathcal{A}$ satisfies*

$$\Pr[F_{\mathsf{spoof}}] \leq f_{\mathsf{und}}(\alpha_{\mathsf{det}}).$$

4 Security Analysis

We study how reuse of hidden watermark randomness affects security. In context-dependent watermarking, repeated contexts are one source of such reuse. Section 5 develops two attacks exploiting different forms of this reuse, while Section 6 evaluates them experimentally.

By Theorem 1, a bound on verification-augmented undetectability also bounds spoofing success. We now analyze the context-keyed sampling-based watermarking scheme defined in Section 2. To isolate the role of context reuse from additional cryptographic machinery, we first replace the pseudorandom function F_K with a uniformly random function $H : \mathcal{V}^n \to \mathcal{R}$. The same random function is used by watermarked generation and terminal verification. This preserves the two properties central to our analysis: repeated contexts reuse the same hidden seed, and generation and verification depend on the same hidden function. The extension to the full keyed construction, in which H is replaced by F_K, is given in Appendix C.

We use two power bounds. The function f_{perf} bounds the distinguishing power introduced by watermark sampling, relative to clean sampling, when every watermark-active position receives a fresh independent seed. The function f_{link} captures the additional distinguishing power available from terminal verification when generation and verification use the same hidden random function. Let $A_{j,t} \in \{0, 1\}$ indicate whether position t of response j is watermark-active, and let $S_{j,t}$ denote its n-token watermark context. We define the active context-reuse event

$$B_n = \{\exists (j, t) \neq (j', t') : S_{j,t} = S_{j',t'} \ \wedge \ A_{j,t} = A_{j',t'} = 1\}.$$

The security-relevant phenomenon is reuse of hidden watermark value, rather than context equality itself. In the ideal model, $H(S)$ is hidden, but equality of two observed active contexts guarantees reuse of the same value: $S_{j,t} = S_{j',t'}$ implies $H(S_{j,t}) = H(S_{j',t'})$. We therefore use context reuse to define B_n. The pooled attack in Section 5 exploits an implementation-specific analogue, where different token windows can be publicly identified as receiving the same watermark values. The theorem covers only context-induced reuse, while both attacks exploit the broader phenomenon of reused hidden watermark information. Recall that α is the probability of incorrectly declaring the clean generator to be watermarked, while $1 - \beta$ is the probability of correctly identifying the watermarked generator.

Theorem 2 (Context-keyed sampling-based watermark verification-augmented undetectability)**.** *Let $f_{\mathsf{perf}}, f_{\mathsf{link}} : [0, 1] \to [0, 1]$ be non-decreasing power bounds described above. If the probability that the event B_n happens in the watermarked model is at most ε, then any distinguisher with false-positive rate α and false-negative rate β satisfies*

$$1 - \beta \leq f_{\mathsf{link}}(\min\{1,\ f_{\mathsf{perf}}(\alpha) + \varepsilon\}).$$

The proof for the idealized construction and its extension to the full keyed construction are given in Appendix C. For samplers that preserve the next-token distribution under fresh randomness, such as inverse-transform and exponential-minimum sampling (Kuditipudi et al., 2024), f_{perf} can be the identity, giving $1 - \beta \leq f_{\mathsf{link}}(\min\{1, \alpha + \varepsilon\})$. If f_{link} is also the identity, terminal verification adds no distinguishing power and the bound reduces to that for ordinary undetectability. We leave a characterization of when f_{link} can be taken as the identity to future work. Thus, active context reuse contributes the additive ε term, separated from the generation-verification linkage captured by f_{link}. It is one form of hidden watermark value reuse; Section 5 studies it directly and identifies an additional implementation-specific form.

Context length, collisions, and robustness. For SynthID-Text, the watermark seed depends on a short sliding window of previous tokens. In our corpus measurements, 46.35%–70.40% of 160-token windows contain at least one repeated 4-token context, but repeated contexts account for only

1.06%–2.23% of positions. Recurrence falls as the context grows: at $n = 8$, only 4.20%–8.49% of windows contain a repeated context. These are raw recurrence measurements on natural text, not estimates of the collision probability ε in Theorem 2; that probability also depends on masking, initialization, and the adversary's queries. Full measurements are given in Appendix D.

Longer contexts reduce reuse, but make the watermark less robust to token edits. With an n-token sliding window, changing an interior token Y_t changes the subsequent contexts $S_{t+1}, \ldots, S_{t+n}$, so watermark evidence at up to n subsequent positions is affected. This is consistent with prior evaluations showing that SynthID-Text detection degrades under editing and paraphrasing (Dathathri et al., 2024; Han et al., 2025). The public implementation also notes that larger context length can improve detectability while making the watermark more brittle to modifications (Ghaisas et al., 2024). Figure 2 shows this trade-off: mean raw recurrence falls from about 100% at $n = 1$ to 6.06% at $n = 8$, while one edit affects between 1 and 8 subsequent contexts.

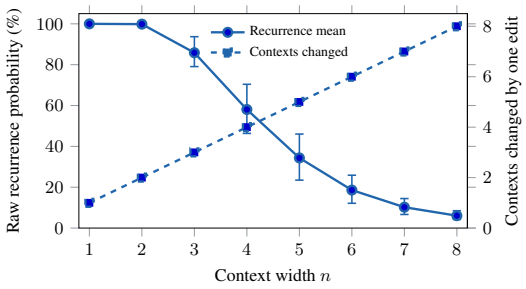

Figure 2: Effect of context width. The solid curve shows mean raw recurrence probability across the four corpus/tokenizer settings, with minimum–maximum ranges. The dashed curve shows the number of subsequent contexts changed by one interior token edit.

5 Spoofing Attack

We first estimate how the watermark changes token probabilities and then use these estimates in two attacks. The exact-context attack exploits repeated use of the same watermark context, while the pooled attack exploits an additional source of reuse in the public Hugging Face implementation.

Attack assumptions. As in Section 3, the attacker has query access to the deployed watermark model but does not know the watermark key. We assume the same key is reused across queries, as in prior watermark-stealing attacks (Jovanović et al., 2024b). The exact-context estimator also requires the deployed tokenizer. We evaluate two settings. In the *matched* setting, the attacker runs the deployed base model locally; in the *mismatched* setting, another model in the same family with compatible tokenization is used instead. The latter corresponds to the more restrictive $B0$ setting of Jovanović et al. (2024b). The pooled estimator makes one additional assumption: the attacker can compute the key-independent part of the mapping from token windows to watermark values. This holds for the open-source implementation we evaluate because its hashing procedure is public. This assumption is specific to the pooled attack; the exact-context attack does not require it.

Estimating the watermark bias. For a watermark context $S \in \mathcal{V}^n$ and candidate token $v \in \mathcal{V}$, let $\pi^{\mathsf{wm}}(v \mid S)$ and $\pi^{\mathsf{clean}}(v \mid S)$ denote the probability of generating v after observing watermark context S, under watermarked and clean generation, respectively. These distributions condition only on the watermark context S, rather than the full prompt and generation history. We estimate the watermark bias through

$$r(v \mid S) = \frac{\pi^{\mathsf{wm}}(v \mid S)}{\pi^{\mathsf{clean}}(v \mid S)}.$$

The numerator is estimated from harvested watermarked outputs and the denominator from the local reference model described above. We denote the resulting estimate by $\widehat{r}(v \mid S)$. We use only watermark-active positions; details are given in Appendix E.1. Repeated observations of the same context are useful because the watermark uses the same hidden randomness whenever that context is evaluated under the same key. The estimator can therefore combine these observations to learn which next tokens are favored by the watermark. This is the form of reuse captured by the analysis in Section 4.

Exact-context decoding. At step i, the decoder looks up candidate tokens for the current watermark context S_i. A candidate is discarded if its estimated probability in the watermarked data is below an *emission floor* τ, or if choosing it would produce a successor context that is not in the table. If

no candidate remains, the decoder falls back to the reference model. Let $p_i^{\mathsf{ref}}(v) = p_{\mathsf{ref}}(v \mid X \| Y_{<i})$ be the reference model's next-token probability; in the matched setting, $p_i^{\mathsf{ref}}(v) = p_i(v)$. When candidates are available, beam search scores them as

$$\mathrm{score}_i(v) = \lambda_r \log \widehat{r}(v \mid S_i) + \lambda_w \log \widehat{\pi}^{\mathsf{wm}}(v \mid S_i) + \lambda_m \log p_i^{\mathsf{ref}}(v),$$

where $\widehat{\pi}^{\mathsf{wm}}$ is the empirical next-token distribution estimated from the watermarked harvest. The three terms favour tokens with a strong estimated watermark bias, tokens well supported by the harvested data, and tokens likely under the reference model, respectively. Beam search also penalizes repeated contexts and immediate token repetition. Our default attack sets $\lambda_r = 1$ and $\lambda_w = \lambda_m = \tau = 0$, so the score depends only on the estimated ratio, while the table and repetition constraints remain. We vary λ_m and τ in Appendix F.4.

Bucket-pooled decoding. The exact-context attack can only use contexts for which it has learned a usable next-token estimate. Many contexts encountered during generation have no such entry in the table. The public Hugging Face implementation creates a second kind of repetition, shown beside the first in Figure 3. For each context S_i and candidate token v, the complete window $S_i \| v$ is mapped to one of 2^{16} public hash buckets. Different windows can map to the same bucket and, in the implementation we evaluate, then receive the same watermark values (Appendix E.2).

This lets the attacker share observations between different windows. Instead of learning a separate estimate for every exact context-token pair, we learn one estimate for each hash bucket. Thus, even if a particular context-token pair was never observed in the harvest, the attacker can still use observations from other windows that fall into the same bucket. In our experiments, the exact-context table provides usable candidates at only 11.4% of positions, whereas every candidate token can be assigned to a bucket. At each step, the attack slightly increases the probability of candidate tokens whose buckets are estimated to be favored by the watermark, while otherwise following the reference model. The exact hashing, estimation, and decoding rules are given in Appendix E.2.

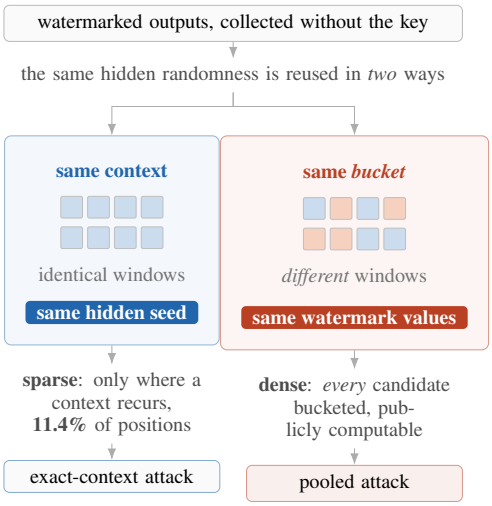

Figure 3: Both attacks exploit reused watermark randomness. Exact-context reuse occurs only when a context repeats, while the public implementation also lets different windows share a bucket and so the same watermark values without sharing a seed. The first source is sparse, the second covers every candidate, which is why pooling steers more gently.

Difference from prior ratio-based watermark stealing. Jovanović et al. (2024b) also compare watermarked and clean outputs and use their probability ratio to estimate which candidate tokens are favored by the watermark. For red-green watermarks (Kirchenbauer et al., 2023), this score estimates how likely a token is to be green. When full-context estimates are too sparse, their method also uses estimates from shorter contexts. Our exact-context attack learns which next tokens are favored for each observed context. The pooled attack can also learn across different contexts: when two context-token combinations fall into the same public hash bucket, the implementation gives them the same watermark values, so the attack can use their observations together. The authors later application (Jovanović et al., 2024a) of watermark stealing to SynthID-Text reports lower spoofing success, which they attribute in part to Tournament sampling, where tokens can receive different watermark biases rather than a common green-token boost.

6 Experimental Evaluation

Following the spoofing model in Section 3, we evaluate whether generation queries to the watermarked and clean generators expose enough watermark signal to produce detector-accepted text,

and how attack performance changes with query budget and model substitution. Experiments use the public Hugging Face implementation of SynthID-Text. The exact-context attack tests the context-induced reuse captured by Theorem 2, while the pooled attack tests a denser, implementation-specific form of the same underlying reuse of hidden watermark value.

Experimental setup. Our main deployments are `Qwen2.5-0.5B-Instruct` and `gemma-2-2b-it`; we additionally evaluate `gemma-2-9b-it`. Unless stated otherwise, we use the matched setting of Section 5. The watermark uses four previous tokens as context (`ngram_len`=5) and 30 tournament layers, following Dathathri et al. (2024). Harvested responses contain at most 256 new tokens, and attack outputs contain 40, 160 (default), and 450 tokens experiments. The 450 tokens experiments match the length used by Jovanović et al. (2024a).

Harvesting and evaluation use ELI5-derived prompts (Fan et al., 2019). During development of the Gemma-2B pooled attack, the watermark estimator is built from 591,542 harvested watermarked responses. Separately, we use 128 development prompts only to choose the attack settings, and report final results on a disjoint set of 512 evaluation prompts. For Qwen, the exact-context attack is also evaluated on a separate set of prompts. For each prompt, we compare the attack output with a genuine watermarked output and an unwatermarked output from the same model. The exact generation settings for these reference outputs differ between Qwen and Gemma and are given in Appendix E.1. The attack itself uses beam search in all cases. Detector training and implementation details are given in Appendices E.1 and E.4.

We use two detector thresholds. Some experiments use a posterior threshold of 0.9. For the main results, we use a stricter threshold chosen separately for each model from $3{,}000$ clean outputs, giving an estimated false-positive rate below 10^{-3}. This follows the low-FPR setting used in prior SynthID spoofing work (Jovanović et al., 2024a). For the pooled attack, we also report the joint rate, which counts outputs that pass both the detector and the automated quality check. Freshness is checked separately in Appendix E.4, and only in part. The joint rate is therefore not the formal event F_{spoof}, which requires freshness rather than quality. Further implementation and detector details are given in Appendices E.1 and E.4.

Pooled attack. Table 1 summarizes the main pooled-attack results. At 160 tokens, the attack reaches detector acceptance rates of 84.6% on Gemma-2B and 89.6% on Gemma-9B. Most accepted outputs also pass the automated quality check, giving joint rates of 82.6% and 88.7%, respectively. The judged quality of the attack outputs remains close to that of genuine watermarked and unwatermarked outputs. On Gemma-9B, the attack is accepted more often than the model's own watermarked output, 89.6% versus 81.8%, while also achieving a higher joint rate, 88.7% versus 81.8%. These results substantially improve on the exact-context attack, which can reach high detector acceptance but produces much poorer text (Appendix F.3). One likely reason is sparsity: the exact-context table provides usable candidates at only 11.4% of positions, whereas the pooled attack can score every candidate through its bucket. This lets the attack make smaller adjustments at more positions while still following the reference model closely. Whether a much larger exact-context harvest can close this gap remains untested. Example outputs are shown in Appendix A, and results for Qwen-0.5B are reported in Appendix E.2. These experiments concern the public Hugging Face implementation evaluated in this work.

Detector acceptance increases sharply with output length. At 450 tokens, the pooled attack is accepted on 99.6% of Gemma-2B outputs and on all 512 Gemma-9B outputs. The corresponding joint rates are 98.8% and 95.9%, close to the genuine watermarked outputs at 98.4% and 96.1%, respectively. On Gemma-2B, the 0.4-point difference in joint rate is not statistically significant ($p = 0.77$). At 40 tokens, the picture is very different: even genuine watermarked outputs are accepted only 13.7% of the time on Gemma-2B and 1.6% on Gemma-9B. The low attack success at this length therefore largely reflects the difficulty of detecting the watermark in short text, rather than only a limitation of the attack. Further length results are reported in Appendix F.3.

All main results use the Bayesian detector. Appendix F.2 repeats the 160-token evaluation with the simpler mean-g detector and again finds high attack acceptance.

Query cost analysis. Table 2 shows that collecting more watermarked outputs generally helps, but with diminishing returns. Qwen-0.5B already reaches 94.5% acceptance with 30k responses and

Table 1: Bucket-pooled attack on 512 held-out prompts at each deployment's calibrated low-FPR threshold. Joint requires both detector acceptance and the automated quality cutoff. Gemma-2B settings were fixed before evaluation; Gemma-9B settings were selected on development prompts. Comparisons should be made within each deployment.

	Gemma-2B			Gemma-9B		
Decoder	Accepted	Judge	Joint	Accepted	Judge	Joint
40-*token continuations*						
Bucket-pooled bonus	3.3%	7.76	2.5%	3.7%	8.00	2.7%
Genuine keyed output	13.7%	7.53	10.5%	1.6%	8.05	1.6%
Unwatermarked output	0/512	7.76	0/512	0/512	8.19	0/512
160-*token continuations*						
Bucket-pooled bonus	84.6%	8.21	82.6%	89.6%	8.52	88.7%
Genuine keyed output	98.4%	8.03	93.4%	81.8%	8.74	81.8%
Unwatermarked output	0/512	8.31	0/512	0/512	8.76	0/512
450-*token continuations*						
Bucket-pooled bonus	99.6%	8.56	98.8%	100.0%	8.37	95.9%
Genuine keyed output	99.6%	8.47	98.4%	96.1%	9.12	96.1%
Unwatermarked output	0/512	8.72	0/512	0/512	9.15	0/512

Table 2: Exact-context detector acceptance versus collected query budget on 512 development prompts; each response contains at most 256 new tokens. Qwen uses posterior > 0.9, while the Gemma rows use their calibrated low-FPR thresholds. *Full* comprises 896,564 responses for Qwen and 591,542 for Gemma-2B. † marks an alternative table construction; threshold and construction details are in Appendix E.3.

Deployment	30k	90k	Full	Keyed	Clean
Qwen-0.5B	94.5%	97.5%	99.4%	100%	1/512
Gemma-2B	82.4%	86.9%†	92.4%	99.2%	0/512
Gemma-9B	24.0%	49.8%	—	85.7%	1/512

increases only to 99.4% with the full harvest; Gemma-2B follows a similar pattern. The effect is much larger for Gemma-9B, where acceptance increases from 24.0% at 30k responses to 49.8% at 90k. Thus, additional data matters most when the smaller harvest provides a weaker estimate, while gains become modest once the attack is already highly successful. Each query returns at most 256 new tokens. The pooled attack reuses the same harvest and requires no additional queries; further budget details are given in Appendices E.2 and E.3.

Sensitivity and scope. Appendix E.5 reports our model-mismatch experiments, where the attacker uses a different within-family reference model. We also test the same learned attack on Dolly creative-writing prompts instead of the ELI5-derived prompts used for harvesting. The attack remains effective in these settings, but we do not test unrelated model families or broader domain shifts.

7 Conclusion

We connect spoofing resistance with undetectability and show how reuse of hidden watermark randomness can weaken context-dependent LLM watermarks. The exact-context attack exploits the context-induced reuse captured by our analysis, while the pooled attack shows how an implementation-specific equivalence can create a much denser source of the same reusable watermark value. On the public SynthID-Text implementation, we demonstrate black-box spoofing attacks, with the pooled attack achieving high joint detection-and-automated-quality rates. How attack performance scales with larger harvests remains future work.

AI USE STATEMENT

In this work, we used generative AI tools to help develop and refine hypotheses and the theoretical framework; design and review experiments; formulate mathematical claims and assist with proofs; implement research code; and interpret experimental results. We also used these tools to draft and edit parts of the paper, search and summarize related work, and prepare figures and tables.

We did not use generative AI to generate synthetic datasets, translate text, clean or reformat datasets, or perform qualitative or thematic data analysis. All AI-assisted work was reviewed by the authors. Mathematical arguments, experimental results, code, and cited prior work were checked before inclusion. We take responsibility for the final content of the paper, including material produced with the aid of generative AI.

ETHICS STATEMENT

This work studies spoofing attacks against a public implementation of SynthID-Text. Publishing such attacks may make watermark spoofing easier, so we report the implementation-specific weakness together with a mitigation that removes the public hash-bucket equivalence. Our results concern the public Hugging Face implementation and do not establish the same weakness in Gemini's production system.

Our experiments use public datasets and publicly available models. We did not obtain secret watermark keys, private model weights, or private user data.

REPRODUCIBILITY STATEMENT

Section 6 describes the experimental setup, detector thresholds, evaluation sets, and query budgets. Appendix E gives the watermark configuration, estimator, hash-bucket construction, mitigation, and additional experimental checks, while Appendix F describes the quality evaluation and attack ablations. Appendices B and C give the complete proofs and assumptions for the formal results. Anonymous source code and evaluation scripts are included in the supplementary material.

REFERENCES

Hyeseon An, Shinwoo Park, Suyeon Woo, and Yo-Sub Han. DITTO: A spoofing attack framework on watermarked LLMs via knowledge distillation. In *Proceedings of the 19th Conference of the European Chapter of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 4922–4936, Rabat, Morocco, March 2026. Association for Computational Linguistics. doi: 10.18653/v1/2026.eacl-long.229.

Anthropic. How claude's text watermark works. `https://www.anthropic.com/news/claude-text-watermark`, August 2026. Accessed 2026-09-19.

Tim Beyne and Yu Long Chen. Information-theoretic security with asymmetries. In Leonid Reyzin and Douglas Stebila (eds.), *Advances in Cryptology – CRYPTO 2024*, volume 14923 of *Lecture Notes in Computer Science*, pp. 463–494. Springer, 2024. doi: 10.1007/978-3-031-68385-5_15. URL `https://eprint.iacr.org/2024/658`.

Miranda Christ, Sam Gunn, and Or Zamir. Undetectable watermarks for language models. In Shipra Agrawal and Aaron Roth (eds.), *Proceedings of Thirty Seventh Conference on Learning Theory*, volume 247 of *Proceedings of Machine Learning Research*, pp. 1125–1139. PMLR, 2024. URL `https://proceedings.mlr.press/v247/christ24a.html`.

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, 2024. doi: 10.1038/s41586-024-08025-4.

European Union. Regulation (eu) 2024/1689 laying down harmonised rules on artificial intelligence (artificial intelligence act). Official Journal of the European Union, 2024. Article 50.

Jaiden Fairoze, Sanjam Garg, Somesh Jha, Saeed Mahloujifar, Mohammad Mahmoody, and Mingyuan Wang. Publicly-detectable watermarking for language models. *IACR Communications in Cryptology*, 1(4), January 2025. doi: 10.62056/ahmpdkp10. URL https://cic.iacr.org/p/1/4/31.

Angela Fan, Yacine Jernite, Ethan Perez, David Grangier, Jason Weston, and Michael Auli. ELI5: Long form question answering. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pp. 3558–3567. Association for Computational Linguistics, 2019. doi: 10.18653/v1/P19-1346. URL https://aclanthology.org/P19-1346/.

Sumedh Ghaisas, Sumanth Dathathri, Ryan Mullins, Joao Gante, Marc Sun, and Raushan Turganbay. Introducing SynthID text. Hugging Face Blog, October 2024. URL https://huggingface.co/blog/synthid-text.

Thibaud Gloaguen, Nikola Jovanović, Robin Staab, and Martin Vechev. Black-box detection of language model watermarks. In *International Conference on Learning Representations*, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/hash/c0f7ee1901fef1da4dae2b88dfd43195-Abstract-Conference.html.

Chenchen Gu, Xiang Li, Percy Liang, and Tatsunori Hashimoto. On the learnability of watermarks for language models. In *International Conference on Learning Representations*, 2024.

Xia Han, Qi Li, Jianbing Ni, and Mohammad Zulkernine. Robustness assessment and enhancement of text watermarking for google's SynthID. *arXiv preprint arXiv:2508.20228*, 2025.

Zhengmian Hu, Lichang Chen, Xidong Wu, Yihan Wu, Hongyang Zhang, and Heng Huang. Unbiased watermark for large language models. In *International Conference on Learning Representations*, 2024.

Nikola Jovanović, Thibaud Gloaguen, and Martin Vechev. Probing google deepmind's SynthID-Text watermark. SRI Lab, ETH Zurich, December 2024a. URL https://www.sri.inf.ethz.ch/blog/probingsynthid.

Nikola Jovanović, Robin Staab, and Martin Vechev. Watermark stealing in large language models. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 22570–22593. PMLR, 2024b. URL https://proceedings.mlr.press/v235/jovanovic24a.html.

John Kirchenbauer, Jonas Geiping, Yuxin Wen, Jonathan Katz, Ian Miers, and Tom Goldstein. A watermark for large language models. In *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pp. 17061–17084. PMLR, 2023.

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

Huijia Lin, Kameron Shahabi, and Min Jae Song. Unforgeable watermarks for language models via robust signatures. In Nadia Heninger and Mike Rosulek (eds.), *Advances in Cryptology – CRYPTO 2026*, volume 16806 of *Lecture Notes in Computer Science*, pp. 99–131. Springer, 2026. doi: 10.1007/978-3-032-35415-0_4. URL https://eprint.iacr.org/2026/282. Full version: Cryptology ePrint Archive, Report 2026/282.

Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. Pointer sentinel mixture models. In *International Conference on Learning Representations*, 2017. URL https://openreview.net/forum?id=Byj72udxe.

Qi Pang, Shengyuan Hu, Wenting Zheng, and Virginia Smith. No free lunch in llm watermarking: Trade-offs in watermarking design choices. In *Advances in Neural Information Processing Systems*, volume 37, pp. 138756–138788, 2024. doi: 10.52202/079017-4402.

Yihan Wu, Zhengmian Hu, Junfeng Guo, Hongyang Zhang, and Heng Huang. A resilient and accessible distribution-preserving watermark for large language models. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 53443–53470. PMLR, 2024.

Shuhao Zhang, Yuli Chen, Jiale Han, Bo Cheng, and Jiabao Ma. Beyond a fixed seal: Adaptive stealing watermark in large language models. In *Findings of the Association for Computational Linguistics: ACL 2026*, pp. 20678–20695, San Diego, California, United States, July 2026. Association for Computational Linguistics. doi: 10.18653/v1/2026.findings-acl.1036.

Xuandong Zhao, Prabhanjan Ananth, Lei Li, and Yu-Xiang Wang. Provable robust watermarking for ai-generated text. In *International Conference on Learning Representations*, 2024.

A Example Outputs

This appendix shows what the pooled attack actually writes, so that the quality claims in Table 1 can be checked rather than taken on trust. Each example gives the attack's detector posterior and judge score beside the deployment's own watermarked answer to the same prompt: the question is not whether the text reads well, but whether the two are distinguishable.

Content note. The outputs are unedited model text. The three prompts shown are benign, but the evaluation set is drawn from a public question-answering corpus that contains questions on sensitive topics, and neither the prompts nor the outputs were filtered.

The first two examples are *not* a random sample. They are two of the 454 Gemma-9B and 506 Gemma-2B outputs that were both accepted by the detector and above the automated quality cutoff, chosen because their topics are neutral; other qualifying outputs answer questions about drug use, indigenous disadvantage and political persecution, which would distract from the point being made here. The third is a failure: an output the detector accepted and the judge rejected.

Gemma-9B, 160 **tokens.** *How does YouTube make enough money to pay every YouTuber?*

Pooled attack (posterior 0.9475, accepted; judge 9.3):

> YouTube doesn't actually have a system where they "pay every YouTuber." The earning structure is based on how creators generate revenue from their content. It's a partnership that works like this: **1. Ad Revenue:** Creators participating in the YouTube Partner Program (YPP) can monetize their videos with ads. When people watch ads on these videos, YouTube pays the creator a share of the ad revenue. The amount a creator earns depends on factors like: number of views; ad engagement (clicks, watch time); viewer demographics and location; type of ads shown (some ads pay more than others). **2. Other Revenue Sources: Channel Memberships:** Viewers can [. . .]

Deployment's own watermarked output (posterior 0.8601, *not* accepted at this deployment's threshold):

> YouTube doesn't directly pay every YouTuber. Here's a breakdown of how YouTube makes money and how YouTubers earn: **How YouTube Makes Money: Advertising:** This is YouTube's primary revenue source. Ads are displayed before, during, or alongside videos. Advertisers pay YouTube based on views or clicks. **YouTube Premium:** Subscribers pay a monthly fee to watch YouTube without ads, access exclusive content, and download videos for offline viewing. **Channel memberships:** Some creators offer exclusive perks, badges, and content to paying members. **Super Chat & Super Stickers:** During live streams, viewers can pay to have their messages highlighted. **Merchandise Shelf:** Creators can sell merchandise directly through their videos. [. . .]

On this prompt the ordering is reversed: the spoofed text is accepted as watermarked and judged 9.3, while the deployment's own watermarked answer falls below its detector's threshold.

Gemma-2B, 450 **tokens.** *When light is reflected, does it "stop" at the point of reflection?*

Pooled attack (posterior 1.0000, accepted; judge 8.7):

> Not exactly! Here's a breakdown of light reflection: **Reflection is a change in direction:** Light doesn't "stop" at the point of reflection, but rather changes its path. **Bounce back:** The light bounces back from the surface towards the original direction it came from. **Incident and reflected rays:** There are two rays in this scenario: the *incident ray*, the incoming ray from the light source, and the *reflected ray*, the ray that bounces back after hitting a surface. **Think of it like this:** Imagine you throw a ball at a wall. The ball doesn't stop after hitting; instead, it bounces back. The ball's energy and direction change, but it doesn't simply disappear. Likewise, light carries energy and changes direction when it hits a surface. **Key Points:** Surface properties affect reflection: the angle of incidence and the properties of the surface (like roughness) determine how light reflects. [. . .]

Deployment's own watermarked output (posterior 1.0000):

> Here's a breakdown of light reflection and whether it stops: **Reflection IS NOT stopping:** Light beams don't "stop" at the moment they hit a surface. However, it is possible to perceive that it appears to stop when viewing a reflection. **How it works:** When light hits a surface, some of it bounces back at a different angle. This process is called reflection. **The role of the surface:** The type of surface and its material properties greatly influence how light bounces. A smooth surface like a mirror reflects light more directly, resulting in a sharp image. A rough surface scatters light in multiple directions, providing diffusive effects. **Our perception:** Our eyes interpret reflected light as coming from a specific direction. [. . .]

Both texts here are accepted. The comparison a reader can make directly is whether the two columns are distinguishable by quality; the judge scores in Table 1 assert that they are not, and these examples let that assertion be checked rather than taken on trust.

An accepted output that fails the quality bar. Ten of the 433 accepted Gemma-2B outputs fall below the automated quality cutoff; they are the difference between the 84.6% acceptance rate and the 82.6% joint rate. One of the ten is shown here.

Why doesn't our moon have an actual name?

Pooled attack (posterior 1.0000, judge 5.0):

> Our moon doesn't have an official name in the sense of a given title like "King James VI" or "Queen Elizabeth" because: **1. Ancient Greek Roots:** The moon has been named "Luna" in Latin for centuries, based on the ancient Roman goddess of the moon. It's been attributed to the moon since at least 1200 BC, long before the name of our country was established. It originates from the Roman goddess Circe, who held power over the moon in Greek mythology. The Greek word "Selene" also means moon. **2. Descriptive Language:** The moon's description and evolution in different cultures through time would suggest other names are less concrete, but more descriptive [. . .]

Deployment's own watermarked output (posterior 1.0000):

> This is a misconception! The Moon actually does have a proper name, and it has been used for centuries. **The Moon's official name is simply "The Moon."** However, some ancient cultures and groups did refer to it using various names, and those which are more well-known include: **Luna:** This is the Latin word meaning "moon" and is still used in modern-day scientific and astronomical contexts. **Callisto:** In Greek mythology, Callisto represents a chaste priestess who was turned into a bear for her punishment. **Theia:** The name refers to an ancient Greek theory of the Moon's formation. While all these names are equally fitting, the sad reality is that throughout history local interpretations [. . .]

The failure is factual drift rather than incoherence: the text is fluent and on topic, but it attributes lunar mythology to Circe and dates Latin usage against "the name of our country". The deployment's own answer to this prompt is also inaccurate, offering Callisto and Theia as names for the Moon, so this failure mode is not specific to the attack. The automated judge scores accuracy as one of its three dimensions, which is why both the attack's successes and this failure turn on factual content rather than on fluency.

B Proof of Theorem 1

For any attacker $\mathcal{A}$ attacking the spoofing resistance of the watermarking scheme, we construct a verification-augmented undetectability distinguisher $\mathcal{D}$ that simulates $\mathcal{A}$'s generation oracles. More precisely, $\mathcal{D}$ receives access to the generation oracle

$$\mathcal{O}_b(X) = \begin{cases} \mathsf{Clean}(X), & b = 0, \\ \mathsf{Wm}_K(X), & b = 1, \end{cases}$$

and the clean oracle

$$\mathcal{O}_{\mathsf{clean}}(X) = \mathsf{Clean}(X).$$

When $\mathcal{A}$ makes a watermarked-generation query X, $\mathcal{D}$ forwards X to $\mathcal{O}_b$ and returns the response. When $\mathcal{A}$ makes a clean-generation query, $\mathcal{D}$ forwards X to $\mathcal{O}_{\mathsf{clean}}$. After completing all queries, $\mathcal{A}$ outputs a candidate $(X^\star, Y^\star)$. The distinguisher submits this candidate for its single terminal verification and outputs the returned bit.

In doing so, $\mathcal{D}$ implicitly bets on being in the watermarked world when the candidate is accepted, and on being in the clean world when the candidate is rejected. The distinguisher makes exactly the same number of generation queries as $\mathcal{A}$.

When $\mathcal{D}$ interacts with the watermarked world, the simulated experiment is identical to the spoofing experiment. Therefore, its power is exactly

$$1 - \beta = \Pr[F_{\mathsf{spoof}}].$$

When $\mathcal{D}$ interacts with the clean world, both generation oracles are independent of the secret key K. Hence the candidate $(X^\star, Y^\star)$ produced by $\mathcal{A}$ is independent of K. Consequently, the probability that the terminal verification accepts is bounded by the detector's false positive rate α_{det}. Applying the assumed verification-augmented power bound gives

$$\Pr[F_{\mathsf{spoof}}] = 1 - \beta \le f_{\mathsf{und}}(\alpha) \le f_{\mathsf{und}}(\alpha_{\mathsf{det}})\,.$$

This concludes the proof.

C Proof of Theorem 2

We retain the following tools for the subsequent security analysis.

Lemma 3 (Power-bound composition (Beyne & Chen, 2024)). *Let P, Q and X be distributions on a common domain. If every distinguisher from P to X with false positive and negative rates α_1 and β_1 satisfies $1 - \beta_1 \le f(\alpha_1)$, and if every distinguisher from X to Q with false positive and negative rates α_2 and β_2 satisfies $1 - \beta_2 \le g(\alpha_2)$ with f and g non-decreasing, then for all distinguishers from P to Q,*

$$1 - \beta \le g(f(\alpha))\,,$$

with α the false positive rate and β the false negative rate.

Lemma 4 (Power-bound H-coefficient (Beyne & Chen, 2024)). *Let P and Q be distributions on $\mathcal{T} = \mathcal{T}_{\mathsf{g}} \sqcup \mathcal{T}_{\mathsf{b}}$. If $Q(\mathcal{T}_{\mathsf{b}}) \le \varepsilon_{\mathsf{b}}$ and $P(E) \ge (1-\varepsilon_{\mathsf{g}})Q(E)$ for all $E \subseteq \mathcal{T}_{\mathsf{g}}$, then for every distinguisher from P to Q,*

$$1 - \beta \le \varepsilon_{\mathsf{b}} + \frac{\alpha}{1 - \varepsilon_{\mathsf{g}}}\,,$$

with α the false positive rate and β the false negative rate.

We prove here the full keyed version of Theorem 2.

Theorem 5 (Full keyed context-reuse bound). *Let $f_{\mathsf{perf}}, f_{\mathsf{prf}}, f_{\mathsf{link}} : [0,1] \to [0,1]$ be non-decreasing power bounds described above. If the probability that the event B_n happens in the watermarked model is at most ε, then any distinguisher with false-positive rate α and false-negative rate β satisfies*

$$1 - \beta \le f_{\mathsf{link}}(f_{\mathsf{prf}}(\min\{1,\, f_{\mathsf{perf}}(\alpha) + \varepsilon\}))\,.$$

The concrete form of f_{prf} depends on the PRF security bound. For an ideal random function, f_{prf} is the identity, recovering Theorem 2.

Proof of Theorem 5. Every transcript contains the result of at most q_{ch} generation queries and at most q_{clean} clean-reference queries. We also include the detector for one terminal verification response; this can only improve the distinguisher. In terms of the security game, the one terminal verification response is given after the distinguisher finishes its interaction with the oracles but before it outputs its decision. In the ideal world, the key is a uniform random dummy. Let P and Q denote the transcript distributions in the clean and watermarked worlds of the verification-augmented undetectability game, respectively. Thus, if E denotes the event that the distinguisher outputs "watermarked",

$$P(E) = \alpha, \qquad Q(E) = 1 - \beta.$$

For response j at position t, write $p_{j,t}(v) = p_{\mathsf{LM}}(v \mid X_j \| Y_{j,<t})$ for the multi-response counterpart of $p_i(v)$ in Section 2. We introduce three intermediate distributions. All experiments use the same clean-reference oracle $\mathcal{O}_{\mathsf{clean}}$, masking rule, and terminal detector Detect_K. The hybrids differ in the challenge generator as follows. In the table, $R_{j,t}$ is independently uniform at every active position, $H : \mathcal{V}^n \to \mathcal{R}$ is a uniform random function independent of K, and $K' \leftarrow \mathcal{K}$ is independent of K.

World	Seed at an active position	Next-token distribution
P	–	$p_{j,t}$
P_1	$R_{j,t} \leftarrow \mathcal{R}$, fresh	$\mathsf{Samp}_{\mathsf{wm}}(p_{j,t}, R_{j,t})$
P_2	$H(S_{j,t})$	$\mathsf{Samp}_{\mathsf{wm}}(p_{j,t}, H(S_{j,t}))$
P_3	$F_{K'}(S_{j,t})$	$\mathsf{Samp}_{\mathsf{wm}}(p_{j,t}, F_{K'}(S_{j,t}))$
Q	$F_K(S_{j,t})$	$\mathsf{Samp}_{\mathsf{wm}}(p_{j,t}, F_K(S_{j,t}))$

The resulting sequence of experiments is

$$P \xrightarrow{f_{\mathsf{perf}}} P_1 \longrightarrow P_2 \xrightarrow{f_{\mathsf{prf}}} P_3 \xrightarrow{f_{\mathsf{link}}} Q.$$

The main-body theorem considers the ideal random-function construction. The full keyed theorem adds the $P_2 \to P_3$ transition that replaces the random function by the PRF.

Fresh-seed sampling: $P \to P_1$. It measures the difference caused by the watermark sampling rule in comparison with clean LLM sampling, when every active position receives fresh independent randomness. Hence

$$P_1(E) \le f_{\mathsf{perf}}(P(E)) = f_{\mathsf{perf}}(\alpha).$$

If, for every $p \in \Delta(\mathcal{V})$ and $v \in \mathcal{V}$,

$$\mathbb{E}_R[\mathsf{Samp}_{\mathsf{wm}}(p, R)(v)] = p(v),$$

then fresh-seed sampling has the clean conditional next-token law at each step. Its generation randomness is independent of the detector key, just as in P. The same terminal verification therefore preserves equality of the complete transcript laws, giving $P = P_1$.

Context reuse: $P_1 \to P_2$. The distributions P_1 and P_2 are identical until an active watermark context repeats. Indeed, when a context S appears for the first time, $H(S)$ is uniformly random, just like the fresh seed used in P_1. Therefore, for a context that has not appeared before, the two experiments are identically distributed. The same holds for every new context. The distributions can differ only when a context appears again: P_1 draws a new independent seed, whereas P_2 must reuse the previous value $H(S)$. This motivates the active context-collision bad event

$$B_n = \{\exists (j,t) \neq (j',t') : S_{j,t} = S_{j',t'} \ \wedge \ A_{j,t} = A_{j',t'} = 1\}.$$

On $\overline{B_n}$, every active context is queried at most once, so the complete generation transcripts of P_1 and P_2 are identical. Both experiments also use the same key K for terminal verification. Therefore, writing $\mathcal{T}_{\mathsf{bad}} = B_n$ and $\mathcal{T}_{\mathsf{good}} = \mathcal{T} \setminus B_n$, we have $P_1(E) = P_2(E)$ for every $E \subseteq \mathcal{T}_{\mathsf{good}}$. If $P_2(B_n) \le \varepsilon$, then Lemma 4, with $\varepsilon_{\mathsf{good}} = 0$ and $\varepsilon_{\mathsf{bad}} = \varepsilon$, gives

$$P_2(E) \le \min\{1, P_1(E) + \varepsilon\} \le \min\{1,\ f_{\mathsf{perf}}(\alpha) + \varepsilon\}.$$

Random function to PRF: $P_2 \to P_3$. The only change in this transition is the seed generator used by the watermarked challenge oracle: P_2 uses the random function H, whereas P_3 uses $F_{K'}$ for an independent key K'. The verification key K is unchanged and independent of both H and K'.

Because the generator's key dependence is entirely through $F_{K'}$, a distinguisher between P_2 and P_3 gives a distinguisher between a random function and the keyed PRF. The reduction samples K itself for terminal verification and uses its PRF challenge only to answer seed queries made during watermarked generation. Therefore, we have

$$P_3(E) \le f_{\mathsf{prf}}(P_2(E))\,,$$

where f_{prf} depends on the used PRF, and the exact value can be found in cryptographic literature.

Key linkage: $P_3 \to Q$. Finally, P_3 and Q use the same watermarking algorithm and sampling rule. The difference is only the relation between the generation and verification keys. In P_3, generation uses an independent key K', while verification uses K; in Q, both use K. By the definition of f_{link},

$$Q(E) \le f_{\mathsf{link}}(P_3(E))\,.$$

Combining the four transitions and using monotonicity of the power bounds gives

$$\begin{aligned} 1-\beta = Q(E) &\le f_{\mathsf{link}}(f_{\mathsf{prf}}(P_2(E))) \\ &\le f_{\mathsf{link}}(f_{\mathsf{prf}}(\min\{1,\, f_{\mathsf{perf}}(\alpha)+\varepsilon\}))\,, \end{aligned}$$

which proves the theorem. □

D Raw Context-Recurrence Measurements

The analysis in Section 4 motivates measuring how often the ordered seed inputs used by a watermarking scheme recur in natural text. Here, we will consider SynthID-Text. These measurements characterize raw recurrence only and are not estimates of ε in Theorem 2, which additionally requires both occurrences to be watermark-active in the complete security-game transcript.

For a tokenized sequence $Y^{(j)}$, let $I_n(Y^{(j)}) = 1$ if at least one n-token context occurs more than once and 0 otherwise. For M sampled sequences, we estimate

$$\widehat{P}(B_n^{\mathsf{raw}}) = \frac{1}{M}\sum_{j=1}^{M} I_n(Y^{(j)}).$$

We sample $10{,}000$ 160-token windows from WikiText-103 (Merity et al., 2017) and $10{,}000$ from the Reddit-derived ELI5 corpus (Fan et al., 2019), and tokenize each corpus with the Qwen2.5-0.5B-Instruct and Gemma-2-2B-IT tokenizers. We evaluate context widths $n = 1, \ldots, 8$; the SynthID configuration used in our main experiments has $n = 4$.

At $n = 4$, 70.40% and 67.75% of WikiText windows contain a repeated context under the Qwen and Gemma tokenizers, respectively; the corresponding values on ELI5 are 47.73% and 46.35%. At $n = 8$, these values fall to 8.49%, 6.80%, 4.74%, and 4.20%.

As a control, we randomly permute the tokens within each window while preserving its token multiset. At $n = 4$, the resulting recurrence probability is only 0.14%–0.20%, indicating that most repetitions arise from sequential structure rather than unigram frequencies. Moreover, although 46.35%–70.40% of windows contain at least one repeated context at $n = 4$, only 1.06%–2.23% of positions have repeated contexts on average. These corpus measurements describe natural text and are not worst-case bounds for model-generated or adversarial sequences.

E Experimental Details and Security Checks

E.1 Configuration, detector, and estimator

Context construction and masking. We use context width $n = 4$. Contexts shorter than n are left-padded with zeros, so generation begins from $(0,0,0,0)$, $(0,0,0,Y_{j,1})$, $(0,0,Y_{j,1},Y_{j,2})$,

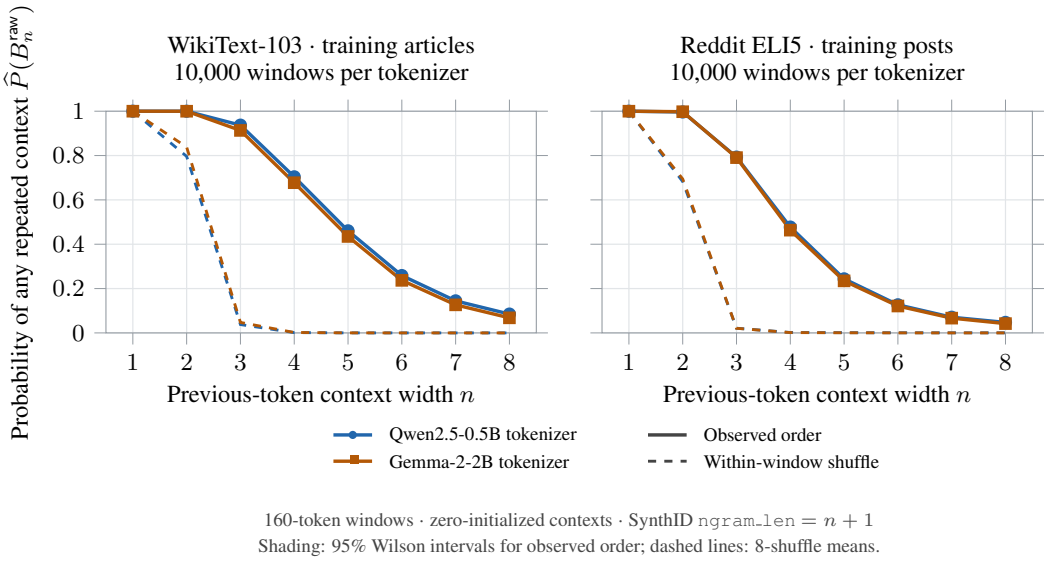

Figure 4: Probability that a 160-token window contains at least one repeated n-token context. Solid curves show the observed token order; dashed curves show the mean over eight within-window permutations. Shading denotes 95% Wilson intervals for the observed-order estimates.

Table 3: Experimental configuration of the public reference implementation; **not a known production configuration**.

Deployed models	`Qwen2.5-0.5B-Instruct`, `gemma-2-2b-it`, `gemma-2-9b-it`
Watermark	`ngram_len` $= 5$ ($n = 4$), 30 keys, table 2^{16}, seed 0, history 1024
Harvest decoding	256 new tokens, prompt capped at 64, $T = 1.0$, top-k 0, top-p 1.0
Harvest corpus	ELI5-derived prompts (Fan et al., 2019)
Harvest size	Qwen 896,564; Gemma-2B 591,542; Gemma-9B up to 90,002 responses
Estimator	$\eta = 0.01$, intersection support, 32 tokens/context
Exact-context decoding	beam width 32, 12 candidates/context; 160 tokens by default
Objective	$\widehat{r}$ only (ratio weight 1.0; $\widehat{\pi}^{\mathsf{wm}}$ weight 0, gate 0)
Penalties	repeated context -5.0, repeated token -2.0; successor context required
Evaluation	Bayesian detector; thresholds calibrated as in Section 6
Software	Python 3.13.1, PyTorch 2.9.1+cu128, Transformers 5.1.0

and $(0, Y_{j,1}, Y_{j,2}, Y_{j,3})$. The opening n positions are excluded from the harvested estimator. SynthID-Text also disables watermarking when a context repeats within the retained masking history (Dathathri et al., 2024); only watermark-active positions are used as estimator observations.

Ratio estimator. For $d \in \{\mathsf{wm}, \mathsf{ref}\}$, let $N_d(S, v)$ be the number of active occurrences of context S followed by token v, and let $N_d(S) = \sum_{u \in \mathcal{V}} N_d(S, u)$. On the shared observed support

$$U_S = \{v \in \mathcal{V} : N_{\mathsf{wm}}(S, v) > 0 \wedge N_{\mathsf{ref}}(S, v) > 0\},$$

we use additive smoothing with parameter $\eta > 0$:

$$\widehat{\pi}^d(v \mid S) = \frac{N_d(S, v) + \eta}{N_d(S) + \eta |U_S|}, \qquad \widehat{r}(v \mid S) = \frac{\widehat{\pi}^{\mathsf{wm}}(v \mid S)}{\widehat{\pi}^{\mathsf{ref}}(v \mid S)}.$$

Contexts with fewer than m observations are discarded.

E.2 Implementation-specific hash-bucket pooling

The pooled attack uses the same harvested outputs as the exact-context attack, but groups observations differently. In the public implementation we evaluate, different context–token windows can

map to the same public hash bucket and therefore receive the same watermark values. The attacker can compute these buckets from the public code without knowing the watermark key. This behavior is specific to the implementation and is separate from the repeated-context effect studied in our theorem.

Public hash buckets. The implementation computes a hash over the token IDs in each window using `accumulate_hash` and reduces the result modulo $M = 2^{16}$. For a complete window $w = (w_1, \ldots, w_{n+1})$, this can be written as

$$z(w) = C + \sum_{j=1}^{n+1} a^{\,n+2-j} w_j \pmod{M},$$

where C is independent of the window. For layer key k_ℓ, the table index is

$$[a(z(w) + k_\ell) + c_0] \bmod M.$$

Therefore, if two windows have the same value of $z(w)$ modulo M, they receive the same table index at every layer and hence the same watermark values. The public implementation thus places all windows into at most 2^{16} publicly identifiable buckets. This does not reveal the secret key, but it does reveal which different windows share the same watermark values.

Pooled estimator and decoder. Instead of estimating a separate watermark preference for every exact context–token pair, we combine observations that fall into the same public bucket. Let $\widehat{r}^{\mathsf{pool}}(b)$ be the estimated watermarked-to-reference ratio for bucket b. For context S_i and candidate token v, let $B_i(v) = z(S_i \| v)$. We use

$$b_i(v) = \min\left\{1, \frac{\max\{0, \log \widehat{r}^{\mathsf{pool}}(B_i(v))\}}{\log c}\right\}, \qquad q_i(v) = \frac{p_i^{\mathsf{ref}}(v) e^{\delta b_i(v)}}{\sum_{u \in \mathcal{V}} p_i^{\mathsf{ref}}(u) e^{\delta b_i(u)}}.$$

The reference model therefore determines the basic next-token distribution, while the bucket estimate gives a limited bonus to tokens that appear favored by the watermark. For Gemma-2B, we use $c = 20$, $\delta = 6$, and require at least 20 observations for a bucket. Settings for the other models are chosen before their final evaluation.

Mitigation. Simply changing where the key is inserted in the existing hash computation does not remove the problem: all 300 constructed collision pairs still collide. In contrast, hashing the complete window with a keyed function removes all 300 collisions.

We also test a practical patched implementation. It removes all public-bucket collisions among 5,933 tested pairs while preserving most of the watermark signal on Gemma-2B and nearly all of it on Qwen-0.5B. Using a new 30k-response harvest, the pooled attack is accepted on $107/128$ outputs with the original hashing and $0/128$ with the patch at posterior > 0.9. This experiment tests whether removing the public bucket equivalence blocks the attack; it is not a security proof for the patched construction.

Does pooling work on unseen windows? The full Gemma-2B harvest contains 3,474,000 distinct context–token pairs and covers 99.9% of the 65,536 buckets. Each occupied bucket contains 53.1 pairs on average. The learned bucket score correlates with the actual watermark preference measured using the secret key ($r = 0.78$).

To check that this is not only memorization of the windows used to build the table, we divide the observed windows into two disjoint halves. We estimate the bucket scores from one half and evaluate them on previously unseen windows from the other half. Across 24,601 such buckets, the correlation remains $r = 0.73$. Thus the bucket estimate transfers to new windows drawn from the same harvested distribution.

Qwen-0.5B. The same pooling effect also appears on Qwen-0.5B. Its harvest covers 95.7% of buckets, with 37.5 context–token pairs per occupied bucket. Using the Gemma-2B attack settings without modification gives $510/512$ detector acceptance, but poor text quality. Choosing a smaller bonus, $\delta = 4$, improves quality and gives 95.3% detector acceptance with a 27.9% joint rate on the Qwen development prompts. We therefore report the Qwen quality results separately from Gemma.

Table 4: Effect of harvest size on the Gemma-2B pooled attack. The fixed setting requires 20 observations per bucket. The adapted setting uses thresholds of 5 and 8 observations for the 30k and 90k harvests, respectively.

Harvest	Bucket occupancy	Pairs/bucket	Joint, fixed	Joint, adapted
30k	91.0%	6.6	66.4%	82.8%
90k	97.9%	14.2	80.5%	87.5%
Full	99.9%	53.1	83.6%	—

Table 5: Exact-context attack under changes to the local model and prompt type. Results within each block should be compared with each other rather than across blocks.

Experiment	Condition	Acceptance
Reference model	Matched Gemma-2B	99.4%
	Gemma-9B denominator	98.8%
Attack generator	Matched Gemma-9B	24.0%
	Gemma-2B generator	27.9%
Prompt domain	ELI5-derived	99.2%
	Dolly creative-writing instructions	91.4%

E.3 Query budget and harvest cost

Each query returns at most 256 new tokens. The largest collected harvests contain 896,564 responses for Qwen-0.5B, 591,542 for Gemma-2B, and 90,002 for Gemma-9B. Changing the local reference model does not require any additional queries to the watermarked service.

For Gemma-2B, the 90k entry in Table 2 uses a slightly different candidate filter. It reaches 86.9% acceptance, compared with 75.6% using the default construction. The same filter does not improve the other Gemma experiments, so this should be viewed as a special case rather than a general improvement.

For the pooled attack, more collected responses increase both the number of covered buckets and the number of observations available for each bucket. Table 4 shows the Gemma-2B results.

A larger harvest does not always improve the pooled attack. On Qwen-0.5B, the 30k and full harvests use the same best bonus weight, $\delta = 4$, but their joint rates are 25.0% and 19.5%, respectively. The difference is not significant ($p = 0.35$). We therefore do not claim that pooled-attack success always increases with harvest size.

E.4 Freshness

Our spoofing definition requires the generated text to be fresh rather than a replay of a harvested response. In the Gemma-2B and Qwen exact-context experiments, the longest consecutive sequence made only from transitions seen once in the harvest is eight tokens out of 160, with a median of four. This argues against simple replay. However, we did not retain every complete harvested response, so we cannot formally verify freshness against the full harvest.

E.5 Model and domain sensitivity

Table 5 tests whether the exact-context attack still works when we change the local model or the type of prompt. The attack remains effective for the model substitutions and prompt change that we tested. We do not test unrelated model families.

E.6 Repeated-context masking

SynthID-Text stops applying the watermark when the same context appears again within its recent history. We use the stored Gemma-2B outputs to estimate how often this would happen. We replay

Table 6: Fraction of positions marked by repeated-context masking in the stored Gemma-2B outputs.

	Emission floor τ						
	0	0.05	0.10	0.14	0.18	0.20	0.30
Attack, $J = 1$	0.02%	0.08%	0.45%	6.97%	7.14%	7.93%	5.60%
Attack, $J = 128$	17.2%	31.0%	31.7%	25.0%	14.7%	15.1%	12.6%
Keyed, $J = 1$				0.81%			
Keyed, $J = 128$				6.12%			

the contexts while keeping the most recent 1,024 occurrences in memory. Table 6 reports the fraction of positions that would be masked. Here, $J = 1$ means that the history is reset for every response, while $J = 128$ keeps the history across all 128 responses in the experiment.

These numbers only show how often masking would be triggered on the stored outputs. They do not tell us directly how much detector acceptance would change, because masking can also change which tokens are generated.

F Text-Quality Evaluation and Additional Attack Results

F.1 Evaluation protocol

We evaluate text quality using an automated judge. Following the criteria of Jovanović et al. (2024b), the judge scores each output for accuracy, consistency, and style. We use `Qwen2.5-7B-Instruct` as the main judge and treat scores of at least 6.5 as passing the quality check. If the judge does not return a valid score, we count that output as failing rather than dropping it. We also repeat the Gemma-2B evaluation with `gemma-2-9b-it` as a second judge. The joint rate used throughout the paper is the fraction of outputs that pass both the watermark detector and this quality check.

Previous spoofing attacks use different models, output lengths, quality tests, and detector thresholds (Jovanović et al., 2024b;a; An et al., 2026; Zhang et al., 2026). Their reported percentages are therefore not directly comparable with ours. We instead compare each attack with genuine watermarked and unwatermarked outputs generated under the same experimental setup.

F.2 The simpler mean-g detector

The main results use the Bayesian detector. We also test a much simpler detector that averages the g-values and checks whether this average exceeds a threshold. This lets us test whether the attack depends on the particular Bayesian detector used in the main experiments. We choose the mean-g threshold using clean outputs from the development prompts, targeting false-positive rates of 1% and 0.1%. We then keep the threshold fixed and evaluate it on the separate held-out set.

Table 7: Detector acceptance on the same 160-token held-out outputs using either the Bayesian detector from the main experiments or a simpler mean-g detector. Mean-g thresholds are chosen on development prompts at the stated false-positive rates.

	Bayesian	Mean g	
Decoder	Accepted	at 1% FPR	at 0.1% FPR
Gemma-2B			
Bucket-pooled bonus	84.6%	99.8%	98.8%
Genuine keyed output	98.4%	99.8%	99.8%
Unwatermarked output	$0/512$	$0/512$	$0/512$
Gemma-9B			
Bucket-pooled bonus	89.6%	93.2%	89.3%
Genuine keyed output	81.8%	87.9%	80.5%
Unwatermarked output	$0/512$	$1/512$	$0/512$

The attack remains effective with this simpler detector. On Gemma-2B, at a 0.1% false-positive rate, 98.8% of attack outputs are accepted, compared with 84.6% under the Bayesian detector. Genuine watermarked output reaches 99.8%, only one percentage point higher than the attack. On Gemma-9B, the attack is accepted on 89.3% of outputs, compared with 80.5% for genuine watermarked output. The attack therefore does not depend on the Bayesian detector used in the main experiments.

F.3 POOLED-ATTACK QUALITY AND CONTROLS

Brackets give two-sided 95% Wilson confidence intervals for success rates, in percentage units, not ranges of detector posterior probabilities. Table 1 gives the main pooled-attack results. On Gemma-2B, the attack has a joint rate of 82.6% $[79.1, 85.7]$. On Gemma-9B, the joint rate is 88.7% $[85.6, 91.1]$, compared with 81.8% $[78.3, 84.9]$ for genuine watermarked output on the same prompts. The difference on Gemma-9B is significant under a paired McNemar test ($p = 0.0006$). This is the only deployment in our experiments where the pooled attack has a higher joint rate than the genuine watermarked output.

For Gemma-2B, the joint rate on the 128 prompts used while developing the attack is 83.6%, close to the 82.6% measured on the separate 512-prompt evaluation set. Repeating the quality evaluation with the second automated judge gives the same joint rate. This reduces dependence on one particular judge.

Table 8 shows why we moved from the exact-context attack to bucket pooling. The exact-context attack can achieve high detector acceptance, but its text quality is poor. If we restrict the attack enough to preserve quality, detector acceptance disappears. The pooled attack is designed to avoid this trade-off. As a control, randomly permuting the learned bucket assignments also removes detector acceptance.

Table 8: Development controls showing the limitation of the exact-context attack and the importance of the learned bucket assignments.

Decoder	Accepted	Judge mean	Joint
Exact context, restricted search	92.4%	1.54	0/512
Exact context, bounded bonus	0/128	8.31	0/128
permuted buckets (control)	0/128	—	—

The paired comparisons at different output lengths tell the same story as Table 1. At 160 tokens, genuine Gemma-2B watermarked output has the higher joint rate ($p < 10^{-4}$). At 450 tokens, the two joint rates are very close, 98.8% for the attack and 98.4% for the genuine watermarked output, and the difference is not significant ($p = 0.77$). On Gemma-9B, the pooled attack has the higher joint rate ($p = 0.0006$).

Spoofing on existing text by paraphrasing. The attacker seeks to imprint a watermark on existing unwatermarked text by rewriting it without changing its content or knowing the key. Whereas Jovanović et al. (2024b) use DIPPER, we prompt Gemma-2B to rewrite Dolly passages with the exact-context attack, not the pooled decoder. Joint success requires both detection and a lexical paraphrase check based on content-word recall and limits on verbatim copying; this differs from the judge-based quality test above and does not guarantee semantic fidelity.

At a 160-token limit, joint rates are 19.5% for the attack and 18.8% for genuine watermarked rewrites of the same passages. However, their word counts exceed the source by 44% and 2%, respectively, potentially confounding detection with length. At a 110-token limit, $105/128$ attack outputs pass the paraphrase check, but only $14/128$ also pass detection: 10.9% $[6.6\%, 17.5\%]$, versus $23/128 = 18.0\%$ $[12.3\%, 25.5\%]$ for the genuine watermarked reference. Overlapping intervals alone do not establish whether the methods differ; that requires a paired comparison. These results concern only the exact-context attack; pooled-attack paraphrasing remains untested.

F.4 CHOOSING THE ATTACK STRENGTH

The parameter δ controls how strongly the pooled attack favors tokens that appear to carry the watermark. Increasing δ generally makes the output easier for the detector to recognize as watermarked, but pushing it too far also makes the text less natural. Table 9 shows this trade-off for Gemma-2B.

Table 9: Effect of the bonus weight δ on the Gemma-2B pooled attack, measured on 128 development prompts at 160 tokens. Perplexity is measured under the clean model; without steering ($\delta = 0$), it is 2.42.

δ	1	2	4	5	6	8
Mean g	0.5050	0.5111	0.5251	0.5358	0.5474	0.5878
Accepted (> 0.9)	0/128	2/128	47/128	98/128	121/128	128/128
Perplexity	2.43	2.53	2.78	3.27	3.94	9.38

For Gemma-9B, the attack is built from $90{,}002$ harvested responses, and we keep buckets with at least eight observations. We test $\delta \in \{6, 8, 10, 12, 16\}$ on 128 prompts. Detector acceptance increases from $10/128$ at $\delta = 6$ to $110/128$ at $\delta = 16$, while $127/128$ outputs still pass the quality check at $\delta = 16$. We then use $\delta = 16$ for the separate 512-prompt evaluation reported in Table 1. Thus, the Gemma-9B result tests one setting chosen from this earlier sweep. The Qwen bonus weight reported in Appendix E.2 is chosen in the same way from separate development experiments.

F.5 LIMITATIONS

Our quality results rely on automated judges rather than human evaluation. The pooled attack is specific to the public Hugging Face implementation we evaluate; we do not know whether Gemini's production system uses the same hash-bucket construction. Detector acceptance also does not by itself establish the formal freshness requirement; see Appendix E.4.