# LATENT CHAINS FOR ROBUST AND TAMPER-EVIDENT WATERMARK OF DIFFUSION LLMS

**Anonymous authors**
Paper under double-blind review

## ABSTRACT

Diffusion large language models (dLLMs) are emerging as an alternative to autoregressive LLMs, with blockwise generation becoming a prominent design. LLM watermarking provides a promising approach for identifying dLLM-generated text. However, existing dLLM watermarks achieve detectability and robustness but provide limited tamper evidence, leaving them vulnerable to piggyback spoofing attacks that modify watermarked text while preserving its attribution to the targeted model. To provide tamper evidence, we introduce Latent Chain Watermark (LCW), which uses a fixed latent chain within each block and lets each finalized block select the latent chain for the next block. To overcome the tension between robustness and tamper evidence, one favours unaffected by modification, the other favours sensitive to modification. We adopt two detections: (1) robust watermark detection searches over all candidate changes to recover watermark evidence, (2) tamper detection measures how modifications disrupt block boundaries and latent chains as tamper evidence. We further develop efficient methods to accelerate detection. At a 5% false-positive rate, LCW achieves 95.1% watermark detection, 80.2% watermark survival under 10% token-level editing, and 82.2% tamper detection after three token edits. Additional experiments demonstrate that LCW's applicability beyond diffusion generation.

## 1 INTRODUCTION

Diffusion large language models (dLLMs) are emerging as a competitive alternative to autoregressive large language models (arLLMs), using parallel token generation and iterative refinement to improve generation efficiency (Nie et al., 2025; Song et al., 2025; Ye et al., 2025). Among current dLLM architectures, blockwise generation has emerged as a prominent design (Arriola et al., 2025; Wu et al., 2026b). It generates blocks sequentially while updating multiple token positions within each block in parallel, enabling variable-length generation and reuse of the key–value (KV) cache for previously finalized blocks (Wu et al., 2026a; Bie et al., 2025).

As these models become more capable and widely deployed, their generated text can be misused for purposes such as spreading misinformation (Pan et al., 2023) and facilitating academic misconduct (Kasneci et al., 2023), underscoring the urgent need for reliable methods to identify machine-generated text (European Parliament and Council of the European Union, 2024; Anthropic, 2026). LLM watermarking has emerged as a promising approach to this problem by embedding statistical evidence into generated text that can later reveal whether it was produced by a target model (Kirchenbauer et al., 2023; Zhao et al., 2023). One of the most widely used approaches is green-red list watermarking. At each token generation, the preceding tokens and a secret key are used to partition the vocabulary into green and red lists, while generation is biased toward green-list tokens. Detection reconstructs these lists and applies a statistical test to the resulting green token count (Kirchenbauer et al., 2023; Zhao et al., 2023; Zhou et al., 2024).

As highlighted by prior work (Pang et al., 2024; Zhou et al., 2024), a reliable watermark should satisfy three key properties: *detectability*, which distinguishes watermarked from non-watermarked text; *robustness*, which preserves detectability under text modifications; and *tamper evidence*, which reveals whether watermarked text has been altered. The latter two address two widely studied security threats. Removal attacks seek to eliminate the watermark signal and thereby evade detection (Krishna et al., 2023; Zhang et al., 2025). Piggyback spoofing attacks, in contrast, modify a wa-



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.

termarked text while preserving its watermark, allowing semantically altered or harmful content to remain attributed to the target model (Pang et al., 2024; Zhou et al., 2024). These requirements are inherently in tension: robustness favors unaffected to text modifications, whereas tamper evidence requires sensitivity to them.

This tension is particularly difficult to resolve for blockwise dLLMs. Existing dLLM watermark methods generally avoid making the green list at one position depend on provisional tokens, which supports reliable detection and robustness but limits sensitivity to post-generation modifications (Bagchi et al., 2025; Gloaguen et al., 2026; Wu et al., 2025; Raban et al., 2026; Zhao et al., 2023; Xie et al., 2025). In contrast, tamper-evident arLLM watermarks exploit sequential generation by deriving each green list from finalized preceding tokens. Modifying these tokens changes the green lists reconstructed at subsequent positions, leaving detectable evidence of tampering (Zhou et al., 2024; Li et al., 2026). This dependency does not directly extend to blockwise dLLMs, where multiple positions within a block are refined in parallel and remain provisional until finalization. Consequently, tamper-evident watermarking for blockwise dLLMs faces two key challenges: *(i) how to establish tamper evidence without relying on provisional tokens*, and *(ii) how to preserve robustness while still providing reliable evidence of tampering.*

We address these challenges with *Latent Chain Watermark* (LCW; Figure 1). For **the first challenge**, LCW avoids relying on provisional tokens to determine green lists. Within a block, a fixed latent chain determines the green list at each token position, allowing watermarking under parallel and non-sequential generation. Once the block is finalized, its tokens select the latent chain for the next block, so modifying a finalized block can change the latent chain for the next block and create tamper sensitive structure. For **the second challenge**, LCW resolves the tension between robustness and tamper evidence by treating modification-induced changes differently during detection. Editing can shift block boundaries, alter the latent chain, and change the within-block alignment between tokens and latent-chain positions. Robust watermark detection treats these changes as variables and searches over candidate boundaries, latent chains, and within-block alignments to recover the watermark evidence. Tamper detection instead measures how these changes affect the watermark score, turning disruptions of block and latent chain into tamper evidence. To make these searches practical, we further develop block-level beam search with within-block dynamic programming (DP) to accelerate these searches.

We evaluate LCW on DreamReasoner (Wu et al., 2026b) and LLaDA (Nie et al., 2025) in terms of watermark detectability, robustness under editing, and tamper evidence, and conduct extensive ablation studies on its key components. LCW maintains high watermark detection after token-level editing (80.2%) while effectively detecting modifications (82.2%). By organizing generated tokens into blocks, we further evaluate LCW under autoregressive generation and tree-based speculative decoding, demonstrating that LCW can generalize beyond dLLMs.

Our contributions are:

- We propose *Latent Chain Watermark* (LCW) for blockwise dLLMs. LCW uses fixed latent chains to avoid dependence on provisional tokens and introduces tamper evidence between blocks. We design robust watermark detection and tamper detection procedures to achieve robustness while retaining sensitivity to tampering, together with block-level beam search and within-block DP for efficient detection.
- We comprehensively evaluate LCW in terms of detectability, robustness, and tamper evidence under watermark-removal and piggyback-spoofing attacks. Our ablations further examine the effects of latent chain design and detection search.
- We show that the latent chain construction is not specific to diffusion generation by extending LCW to autoregressive generation and tree-based speculative decoding, and experimentally verify its effectiveness in both settings.

## 2 Background and Related Work

**Autoregressive and Diffusion Large Language Models.** Autoregressive large language models (arLLMs) generate a token sequence $X = (x_0, \ldots, x_{N_X-1})$ over a vocabulary $\mathcal{V}$ from left to right, sampling each token conditioned on its preceding prefix. Diffusion LLMs (dLLMs), in contrast, gen-

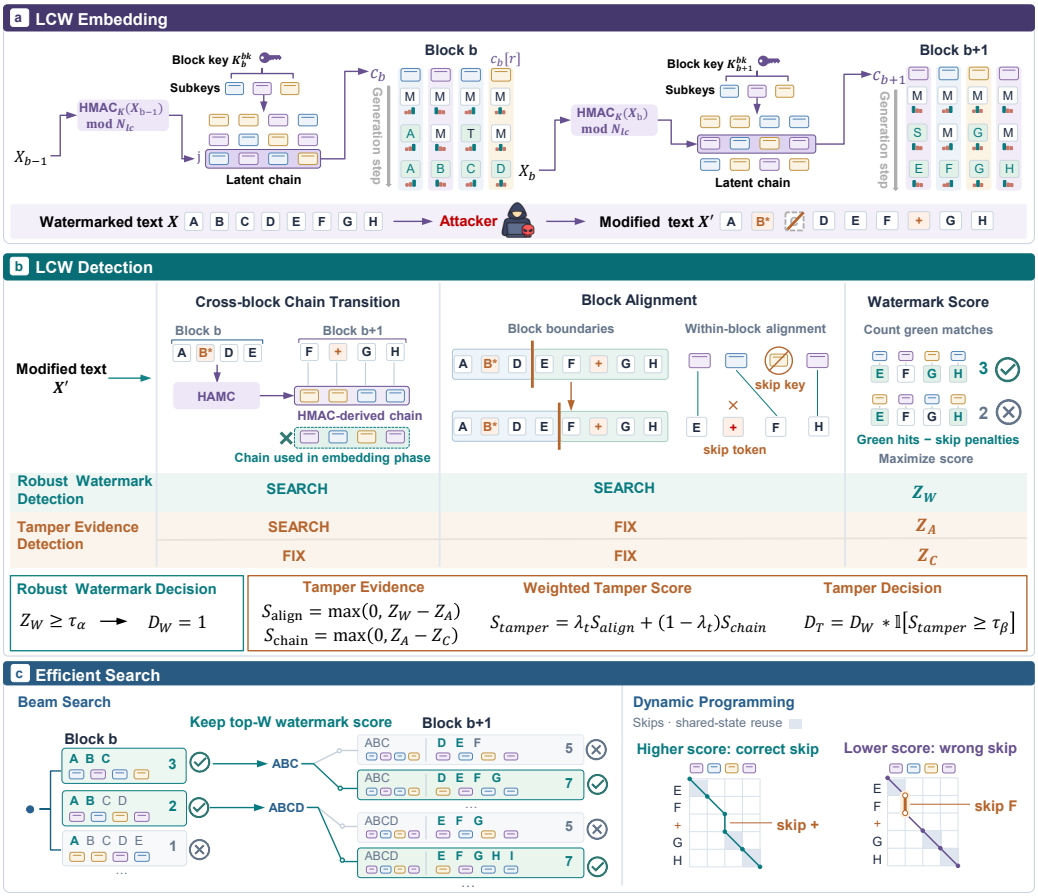



Figure 1: Overview of Latent Chain Watermark (LCW), which embeds watermark and tamper evidence through latent chains and uses robust watermark detection and tamper detection to achieve detectability, robustness, and tamper evidence efficiently.

erate text through iterative denoising, allowing multiple positions to be updated in parallel. A prominent dLLM design is blockwise generation, adopted by models such as LLaDA (Nie et al., 2025) and DreamReasoner (Wu et al., 2026b). It partitions $X$ into consecutive blocks $X_0, \ldots, X_b, \ldots$, each containing $N_{\text{bw}}$ tokens upon completion. Blocks are generated sequentially, while token positions within each block are iteratively refined in parallel until finalization. This design supports variable-length generation and KV-cache reuse for flexible and efficient decoding.

**LLM Watermarking.** LLM watermarking embeds a statistically detectable signal by biasing token sampling. A representative approach is the green–red list watermark (Kirchenbauer et al., 2023; Zhou et al., 2024; Zhao et al., 2023). At position $t$, the secret key $K$ and a hash of the generated prefix determine a pseudorandom green list $\mathcal{G}_t \subset \mathcal{V}$, with $|\mathcal{G}_t| = \gamma|\mathcal{V}|, \gamma \in (0,1)$. To preserve generation quality, soft watermarking is typically used: a positive bias $\delta$ is added to the logits of green-list tokens, increasing their sampling probability. At each position $t$, the detector reconstructs the green list from $K$ and the preceding context in $X'$ and counts the tokens falling into their respective green lists, yielding $n_G$. It then computes $z = \frac{n_G - \gamma N_{X'}}{\sqrt{N_{X'}\gamma(1-\gamma)}}$. The $z$-score quantifies the *watermark evidence* in $X'$, and $D_W(X') = 1$ if it exceeds a threshold.

**Robustness and Tamper Evidence.** In practical use, a watermark should first provide reliable detectability, distinguishing watermarked text with $D_W(X) = 1$ from unwatermarked text with $D_W(X) = 0$. Post-generation editing introduces two additional security requirements: robustness and tamper evidence. *Robustness* requires an edited watermarked text $X'$ to remain detectable,

i.e., $D_W(X') = 1$, thereby resisting removal attacks that seek to erase the watermark and evade provenance tracing (Zhang et al., 2025). *Tamper evidence*, in contrast, requires modifications to be identified. For watermark-positive text, i.e., $D_W(X') = 1$, a decision $D_T(X')$ detects whether the text has been altered. This counters piggyback spoofing attacks, which change the meaning or inject harmful content while preserving watermark detection, causing manipulated text to remain attributed to the target model and potentially harm the model owner's reputation (Pang et al., 2024; Zhou et al., 2024). These objectives are inherently in tension: robust watermark detection should tolerate editing, whereas tamper detection must remain sensitive to the edits. Existing arLLM methods address this trade-off with complementary watermark signals, for example, Bileve (Zhou et al., 2024) and DualGuard (Li et al., 2026) use dual watermark streams for watermark and tamper detection. They both rely on a key autoregressive assumption: preceding token is already finalized when watermark signals for subsequent tokens are constructed.

**Watermarking Diffusion Language Models.** Existing dLLM watermarks primarily target detectability and robustness, using different mechanisms to embed watermark signals under parallel generation. GSJ averages over uncertain context (Gloaguen et al., 2026), DMark uses predicted and bidirectional context (Wu et al., 2025), and LR-DWM exploits available neighboring tokens (Raban et al., 2026). While such tolerance enables watermarking under parallel generation and improves robustness, it conflicts with the sensitivity to modification required for tamper detection. Order-agnostic approaches instead avoid dependence on generation order and are therefore naturally compatible with parallel generation: Unigram uses a fixed green list across positions (Zhao et al., 2023), while Combinatorial Watermarking assigns vocabulary subsets according to a cyclic pattern (Xie et al., 2025). However, because these assignments are independent of generated token values, edits do not systematically disrupt the watermark structure; moreover, under soft watermarking, a non-green token can arise during normal generation even without tampering (Zhou et al., 2024). Meanwhile, tamper-evident arLLM mechanisms rely on finalized preceding tokens and therefore do not directly apply within a diffusion block. Therefore, a reliable dLLM watermark must therefore support parallel generation while remaining both robust and sensitive to tampering.

# 3 Latent Chain Watermark

This section presents Latent Chain Watermark (LCW) for tamper-evident and robust watermark in blockwise dLLMs. We first introduce the threat model, then describe watermark embedding, robust watermark detection, tamper detection, and finally an efficient search to accelerate detection.

## 3.1 Threat Model

**Adversary goal:** We consider an adversary who modifies a watermarked text before detection, either to remove the watermark or to preserve watermark while evading tamper detection (Krishna et al., 2023; Zhang et al., 2025). **Adversary capabilities and knowledge:** Following prior work and current deployment settings (Anthropic, 2026; Zhang et al., 2025), we assume that the adversary knows the watermarking and detection algorithms but not the secret master key $K$. The detector is private and provides no query access or feedback (Anthropic, 2026); moreover, following prior settings (Zhou et al., 2024), the adversary cannot repeatedly query the target LLM to collect additional watermarked samples or train surrogate models. The adversary may modify the text through token- or character-level substitutions, deletions, and insertions, as well as sentence-level paraphrasing (Krishna et al., 2023; Zhang et al., 2025).

## 3.2 Watermark Embedding

Watermark embedding in blockwise dLLMs should avoid dependence on provisional within-block tokens while retaining tamper evidence that depends on finalized content. LCW achieves this with a fixed latent chain within each block and uses each finalized block to select the latent chain for the next (Figure 1 (a)). The full algorithm is given in Appendix A.1.

**Latent chain.** A latent chain determines the green list used at each within-block position. LCW first derives $N_{\mathrm{bk}}$ block keys $K_0^{\mathrm{bk}}, \ldots, K_{N_{\mathrm{bk}}-1}^{\mathrm{bk}}$ from the master secret $K$. Each block key is further used to derive $N_{\mathrm{sk}}$ subkeys, from which LCW constructs $N_{\mathrm{lc}}$ candidate latent chains. Each latent

chain is a length-$N_{\mathrm{bw}}$ ordered sequence formed by pseudo-randomly selecting subkeys, where the subkey at latent chain position $r$ determines the green list assigned to within-block position $r$. [1] Each block key therefore defines a distinct set of $N_{\mathrm{lc}}$ candidate latent chains. For block $X_b$, LCW assigns $K_b^{\mathrm{bk}} = K_{(b+\phi) \bmod N_{\mathrm{bk}}}^{\mathrm{bk}}$, where $\phi \in \{0, \ldots, N_{\mathrm{bk}} - 1\}$ is sampled uniformly once for each generated sequence $X$. One of the candidate latent chains associated with $K_b^{\mathrm{bk}}$ is then selected for $X_b$. The $j$-th candidate latent chain associated with $K_b^{\mathrm{bk}}$ and its green lists are defined as

$$c_b = \mathrm{Chain}(K_b^{\mathrm{bk}}, j) = \{c_b[r]\}_{r=0}^{N_{\mathrm{bw}}-1}, \qquad \mathcal{G}_{b,r} = \mathrm{GreenList}\left(K_b^{\mathrm{bk}}, c_b[r]\right), \tag{1}$$

where $j \in 0, \ldots, N_{\mathrm{lc}} - 1$, $c_b[r]$ denotes the subkey at latent-chain position $r$, and $\mathcal{G}_{b,r} \subset \mathcal{V}$ is the green list for the corresponding token position in $X_b$. Once $c_b$ is selected, every update at position $r$ uses the same $\mathcal{G}_{b,r}$, independent of the current token values and the within-block generation order. LCW then applies the soft green-list bias defined in Section 2.

**Chain transition.** Once block $X_b$ is finalized, its tokens determine the latent chain used by the next block:

$$c_{b+1} = \mathrm{Chain}\big(K_{b+1}^{\mathrm{bk}}, \mathrm{HMAC}_K(X_b) \bmod N_{\mathrm{lc}}\big)\,. \tag{2}$$

where $\mathrm{HMAC}_K(X_b) \bmod N_{\mathrm{lc}}$ selects one of the $N_{\mathrm{lc}}$ candidate latent chains associated with $K_{b+1}^{\mathrm{bk}}$ [2]. For the first block, LCW uses $c_0 = \mathrm{Chain}(K_0^{\mathrm{bk}}, 0)$, with $K_0^{\mathrm{bk}}$ determined by the sampled phase. After generation, if $X_b$ is modified to $X_b'$, chain transition may select a different latent chain for $X_{b+1}$. This difference makes the chain transition sensitive to post-generation modification.

### 3.3 Robust Watermark Detection

Post-generation modifications can disrupt the watermark structure in two ways. First, modifications can change the block alignment, including block boundaries and within-block alignments, i.e., the correspondence between tokens within a block and positions in its latent chain. As a result, tokens in $X'$ may be assigned to different blocks or different latent-chain positions from those used during generation. Second, modifying the content of a block can change the chain transition and therefore cause the detector to reconstruct a different latent chain for the following block.

For watermark detection, our objective is to determine whether watermark evidence remains in $X'$, rather than whether the edited text preserves the embedding-time structure. To improve robustness, LCW therefore treats these disrupted structures as unknown and searches over all candidate block boundaries, latent chains, and within-block alignments to recover the strongest surviving watermark evidence, illustrated in Figure 1 (b). The detailed procedure is provided in Appendix A.3. For block boundaries, we consider candidate block lengths in $[N_{\mathrm{bw}} - \eta,\ N_{\mathrm{bw}} + \eta]$, where $\eta$ controls the maximum deviation from the original block width. For each candidate block, the detector evaluates the candidate latent chains associated with its block key and selects the one that best matches the surviving watermark evidence.

Since modifications can cause unmatched positions in within-block alignments, i.e., an insertion can introduce an additional token, while a deletion can leave a subkey in the latent-chain without a corresponding token. For within-block alignments, the detector therefore allows skip tokens and skip keys. Allowing unmatched elements, however, could artificially increase the watermark score by excluding non-green tokens. LCW therefore penalizes every unmatched position when computing the watermark score. LCW defines the penalized watermark score as follows

$$Z_{\mathrm{LCW}}(p) = \frac{h(p) - \lambda_p \sum_b e_b(p) - \gamma T(p)}{\sqrt{T(p)\gamma(1-\gamma)}}, \tag{3}$$

where $p$ specifies a candidate configuration including block boundaries, latent chains, and within-block alignments, $h(p)$ is the number of green tokens under $p$, $T(p) > 0$ is the number of input tokens covered by $p$, $e_b(p)$ is the number of unmatched positions in block $b$, and $\lambda_p \geq 0$ is the penalty applied to each unmatched position. The terms $h(p) - \gamma T(p)$ and $\sqrt{T(p)\gamma(1-\gamma)}$ follow the conventional z-score (Kirchenbauer et al., 2023; Zhao et al., 2023), as mentioned in Section 2.

[1] We call the chain *latent* because it is an internal watermark structure that is not explicitly encoded in the generated text.

[2] HMAC is a keyed hash-based message authentication code (Krawczyk et al., 1997).

Watermark detection returns the largest score found by the search:

$$Z_W(X') = \max_p Z_{\mathrm{LCW}}(p), \qquad D_W(X') = \mathbb{1}(Z_W(X') \geq \tau_\alpha). \tag{4}$$

where the maximization is over the candidate configurations considered by the detector, $Z_W(X')$ is the resulting watermark score, and $\tau_\alpha$ is calibrated on non-watermarked text at target FPR $\alpha$.

### 3.4 Tamper Evidence Detection

Tamper evidence detection determines whether a watermark-positive sequence has been modified. As discussed in Section 3.3, modifications can disrupt the block alignments and chain transitions. Thus, tamper detection measures these effects on the watermark score as evidence of modification, illustrated in Figure 1 (b). The detailed procedure is provided in Appendix A.4. LCW therefore measures two sources of structural inconsistency. First, it measures the watermark score loss caused by enforcing the block alignments including block boundary and within-block alignments. Second, it measures the watermark score loss caused by enforcing the chain transition.

**Block alignment.** To measure tamper evidence caused by changes in block boundaries and within-block alignment, LCW removes the corresponding search flexibility used in robust watermark detection. Specifically, the detector partitions $X'$ into consecutive blocks of width $N_{\mathrm{bw}}$, without allowing the boundary deviation $\eta$, and directly matches each within-block token to latent chain position. For each block, the detector still searches over its candidate latent chains and selects the one that yields the strongest watermark score, while the chain transition between consecutive blocks is not enforced. The alignment-based tamper evidence is then measured by the score loss relative to $Z_W(X')$:

$$S_{\mathrm{align}}(X') = \max(0,\, Z_W(X') - Z_A(X')). \tag{5}$$

where $Z_W(X')$ is the robust watermark score (see equation 4) and $Z_A(X')$ is the score obtained after fixing the block boundaries and within-block alignment. For an unmodified sequence, the fixed block boundaries and direct within-block alignment should remain consistent with those used during embedding, so enforcing them causes little score loss. Insertions or deletions can shift subsequent token positions, causing smaller $Z_A(X')$ and larger $S_{\mathrm{align}}(X')$.

**Chain transition.** To further measure tamper evidence caused by changes in the chain transition, LCW further removes the latent chain search flexibility retained in $Z_A(X')$. Specifically, we keep the same fixed block boundaries and within-block alignment used in $Z_A(X')$, but no longer allow each block to select the best candidate latent chain. Instead, for each block, the detector uses the latent chain determined by the preceding block through the chain transition in equation 2, the resulting watermark score is $Z_C(X')$. We then measure chain-based tamper evidence by the score loss:

$$S_{\mathrm{chain}}(X') = \max(0, Z_A(X') - Z_C(X')). \tag{6}$$

If a finalized block has been modified, the reconstructed transition may select a different latent chain for the following block, reducing $Z_C(X')$ and increasing $S_{\mathrm{chain}}(X')$.

**Tamper score and decision.** $S_{\mathrm{align}}$ and $S_{\mathrm{chain}}$ capture complementary effects of modifications. $S_{\mathrm{align}}$ reflects changes in token positions, while $S_{\mathrm{chain}}$ reflects inconsistencies in adjacent blocks. Since an edit may affect either or both, LCW combines them into a tamper score and defines the final decision as

$$S_{\mathrm{tamper}}(X') = \lambda_t S_{\mathrm{align}}(X') + (1-\lambda_t) S_{\mathrm{chain}}(X'), \qquad D_T(X') = D_W(X') * \mathbb{1}(S_{\mathrm{tamper}}(X') \geq \tau_\beta). \tag{7}$$

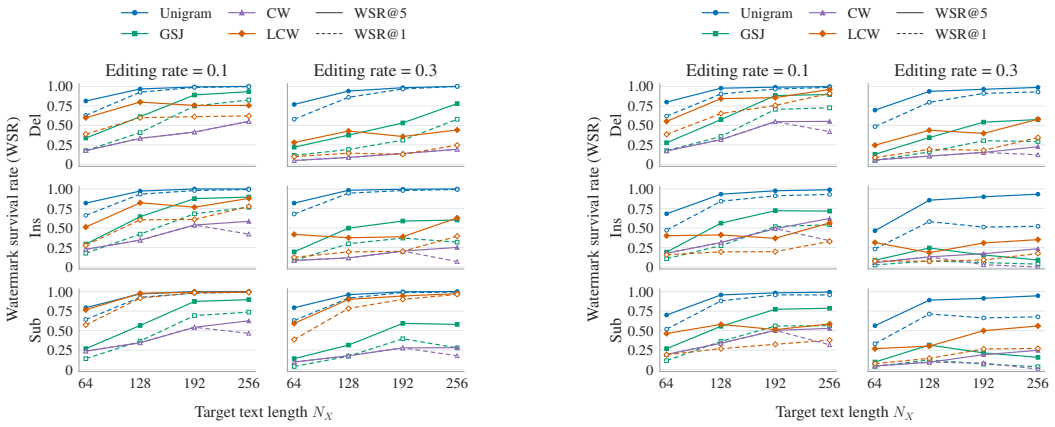
where $\lambda_t \in [0,1]$ controls the relative contribution of the two tamper scores, and $\tau_\beta$ is the threshold for tamper detection. $\tau_\beta$ is calibrated on watermark-positive unmodified texts at tamper FPR $\beta$.

### 3.5 Efficient Search

The detection procedures above require joint search over block boundaries, latent chains, and within-block alignments. Exhaustive enumeration is prohibitively expensive because the number of candidate configurations grows rapidly across both blocks and alignments. LCW therefore decomposes the search into two levels, as illustrated in Figure 1(c): block-level beam search efficiently explores boundaries and latent chains, while within-block dynamic programming (DP) finds the best alignment for each candidate block and chain.

Table 1: Watermark detectability on DreamReasoner across target lengths.

| Method | $N_X = 64$ | | | $N_X = 128$ | | | $N_X = 192$ | | | $N_X = 256$ | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | TPR@5 | TPR@1 | AUC | TPR@5 | TPR@1 | AUC | TPR@5 | TPR@1 | AUC | TPR@5 | TPR@1 | AUC |
| Unigram | 0.809 | **0.662** | 0.951 | 0.967 | 0.926 | **0.997** | **1.000** | 0.983 | 0.999 | **1.000** | 0.993 | **1.000** |
| GSJ | 0.344 | 0.214 | 0.796 | 0.732 | 0.535 | 0.958 | 0.946 | 0.866 | 0.987 | 0.960 | 0.890 | 0.995 |
| CW | 0.301 | 0.301 | 0.649 | 0.462 | 0.462 | 0.729 | 0.692 | 0.692 | 0.845 | 0.736 | 0.612 | 0.862 |
| **LCW (ours)** | **0.819** | 0.649 | **0.958** | **0.983** | **0.950** | 0.994 | **1.000** | **1.000** | **1.000** | **1.000** | **1.000** | **1.000** |




(a) Token-level edits

(b) Character-level edits

Figure 2: DreamReasoner robustness under token-and character-level edits at two editing rates.

**Block-level beam search.** Searching over block boundaries and latent chains causes the number of candidate configurations to grow exponentially with the number of blocks. However, many partial configurations already yield weak watermark evidence and are unlikely to become competitive after further expansion. LCW therefore uses beam search to prune these configurations early. At each step, LCW retains only the top $W$ configurations, where $W$ is the beam width.

**Within-block DP.** For a fixed token span and latent chain, LCW uses DP to find the best within-block alignment. Naive enumeration repeatedly solves the same alignment problems over token and latent-chain pairs. Instead, the DP operates over pairs of token and latent chains, storing the best partial score for each pair. Each prefix pair is therefore solved once and reused, avoiding combinatorial enumeration. The detailed procedure is provided in Appendix A.2.

## 4 Experiments

We evaluate LCW along the three objectives motivating its design: watermark detectability, robustness to editing, and tamper evidence for watermark-positive edited text. Our experiments compare LCW with representative watermark baselines under raw generation, token-level and character-level edits, and tampering. Ablation studies (Appendix C.1), text quality evaluation (Appendix C.2), semantic piggyback spoofing(Appendix C.3), paraphrase attack (Appendix C.4) and additional results are reported in Appendix C.

**Experimental Setup.** We conduct our experiments on two publicly available blockwise dLLMs, DreamReasoner-8B (Wu et al., 2026b) and LLaDA-8B-Instruct (Nie et al., 2025). For evaluation, we randomly sample texts from the realnewslike subset of C4 (Raffel et al., 2020) and use the first 50 tokens of each text as the generation prompt. We compare LCW with Unigram (Zhao et al., 2023), GSJ (Gloaguen et al., 2026), and Combinatorial Watermarking (CW) (Xie et al., 2025). Additional details on the models, watermark settings, and detection settings are provided in Appendix B. We report the main results on DreamReasoner, providing LLaDA results in Appendix C.5.

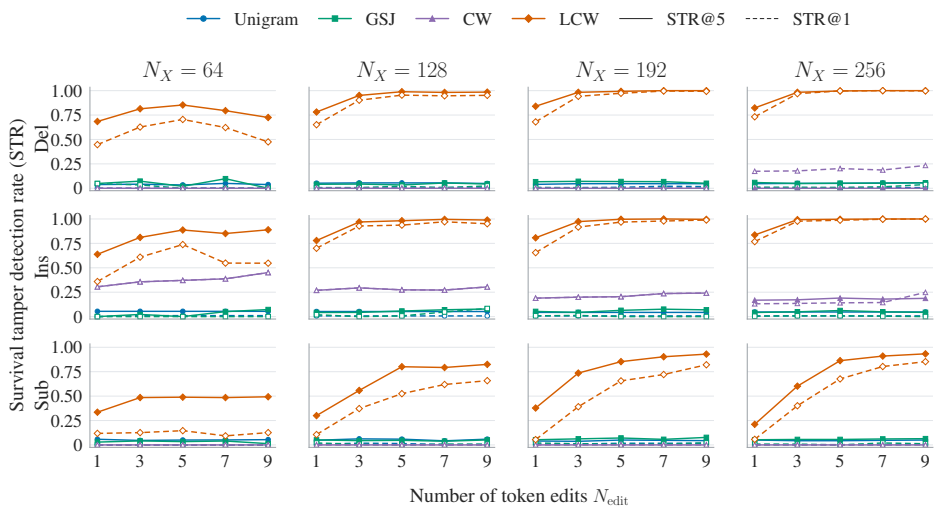



Figure 3: DreamReasoner tamper evidence under token-level edits. Rows show deletion, insertion, and substitution, and columns show target text lengths.

**Metrics.** We evaluate *detectability* using TPR and AUC, *robustness* using watermark survival rate (WSR), and *tamper evidence* using survival tamper detection rate (STR). WSR is the fraction of edited watermarked texts that remain watermark positive, while STR is the fraction identified as tampered among those that remain watermark positive. For TPR, WSR, and STR, @5 and @1 denote operating points with 5% and 1% FPR, respectively; for STR, the same target FPR is used for both watermark and tamper detection.

### 4.1 Detectability

We first evaluate watermark detectability on unedited text. As shown in Table 1, LCW achieves strong detection across target lengths. At $N_X = 64$, LCW obtains a TPR@5 of 0.819 and an AUC of 0.958, outperforming Unigram (0.809/0.951), GSJ (0.344/0.796), and CW (0.301/0.649). Although its TPR@1 is slightly lower than Unigram at this shortest length (0.649 vs. 0.662), it remains substantially higher than GSJ (0.214) and CW (0.301). At $N_X = 256$, LCW reaches 1.000 for both TPR@5 and TPR@1, compared with 1.000/0.993 for Unigram, 0.960/0.890 for GSJ, and 0.736/0.612 for CW. Overall, LCW remains competitive with Unigram while substantially outperforming GSJ and CW across text lengths.

### 4.2 Tamper Evidence

We treat any post-generation modification as tampering, and evaluate tamper evidence under token-level deletion, insertion, and substitution with $N_{\text{edit}} \in \{1, 3, 5, 7, 9\}$. The tampering with low $N_{\text{edit}}$ is particularly challenging because only a small portion of the watermark structure is disrupted. Figure 3 reports DreamReasoner results. LCW directly uses its tamper detector, CW uses its native combinatorial edit detector, while Unigram and GSJ use the decrease in their watermark scores after editing as tamper evidence. LCW consistently provides substantially stronger tamper evidence than the baselines. LCW also detects single-token modifications, with mean STR@5 of 0.782, 0.765, and 0.306 for deletion, insertion, and substitution, respectively. Tamper detection generally becomes stronger as more edits are introduced, for example, substitution STR@5 at $N_X = 128$ increases from 0.3 for $N_{\text{edit}} = 1$ to 0.825 for $N_{\text{edit}} = 9$. Deletion and insertion are easier to detect because they shift token positions and directly affect within-block alignment, whereas substitution preserves token positions and relies more heavily on inconsistencies in chain transition. We further evaluate LCW under semantic piggyback spoofing attacks (Pang et al., 2024; Zhou et al., 2024) in Appendix C.3. We also discuss a potential limitation of tamper detection in Appendix C.6.

### 4.3 ROBUSTNESS

We evaluate robustness under token-level and character-level deletion, insertion, and substitution. Token-level attacks directly modify generated tokens, while character-level attacks modify characters within selected tokens. Figure 2 reports DreamReasoner results at editing rates of 10% and 30%. Averaged over the four $N_X$, LCW achieves WSR@5 of 0.727, 0.745, and 0.932 under 10% token-level deletion, insertion, and substitution, respectively. For substitution, LCW remains close to the strong Unigram baseline (0.941) and exceeds GSJ (0.651) and CW (0.437). At a 30% editing rate, substitution WSR@5 remains 0.852. Results on paraphrasing are reported in Appendix C.4.

## 5 DISCUSSION

**Detection Efficiency.** Detection time increases with text length but remains modest. At $N_X = 256$, robust watermark detection takes about 2.1 seconds per text, while tamper detection, which requires additional searches, takes about 3.1 seconds. This demonstrates that block-level beam search with within-block DP effectively controls the search cost. Detailed runtime results are provided in Appendix C.7.

**Applicability Beyond Diffusion.** Although LCW is designed for blockwise dLLMs, it does not depend on diffusion generation itself. By dividing the generation process into consecutive blocks, LCW can be applied to more generation paradigms. LCW can further be applied to generation settings where standard arLLM watermarks are difficult to use, such as tree-based speculative decoding (Hu et al., 2026; Ringel & Romano, 2026), where different branches have different contexts but may share the same candidate token, causing conflicting watermark assignments for arLLM watermarks. LCW instead uses the same green list at the same position.

Table 2: LCW beyond diffusion at $N_X = 128$. Robustness is evaluated under 10% token-level substitution, and tamper evidence under three token-level substitutions.

| Backend | Method | TPR@5 | Sub WSR@5 | Sub STR@5 |
|---|---|---|---|---|
| Qwen | Bileve | 0.739 | 0.595 | 0.234 |
| | LCW | 0.990 | 0.960 | 0.828 |
| JetSpec | LCW | 0.960 | 0.931 | 0.418 |
| DDTree | LCW | 0.970 | 0.960 | 0.510 |

We evaluate LCW on autoregressive generation with Qwen3.5-9B (Qwen Team, 2026) and two tree-based speculative decoding backends, JetSpec (Hu et al., 2026) and DDTree (Ringel & Romano, 2026). On Qwen3.5-9B, LCW outperforms Bileve (Zhou et al., 2024), which is designed for robust and tamper-evident arLLM watermarking, across detectability, robustness, and tamper detection. LCW also remains effective on JetSpec and DDTree, demonstrating that it can generalize beyond dLLMs. Full implementation details and results are reported in Appendix C.8.

## 6 CONCLUSION

We study watermarking for blockwise dLLMs, where parallel generation makes it challenging to achieve robustness while retaining tamper evidence. We introduce Latent Chain Watermark (LCW), which uses a fixed latent chain within each block and lets each finalized block determine the latent chain for the following block. Robust watermark detection searches over changes in block boundaries, latent chains, and within-block alignments to recover watermark evidence, while tamper detection measures how modifications affect block alignment and chain transitions, using the resulting watermark score losses as tamper evidence. Experiments on DreamReasoner and LLaDA show that LCW achieves strong detectability and robustness while providing tamper evidence across post-generation modifications ranging from single-token to multi-token edits. Additional results on autoregressive and tree-based speculative decoding further demonstrate that LCW can extend beyond diffusion generation. Together, these results show that LCW provides a practical way to combine robustness with tamper evidence across different generation paradigms.

### AI USE STATEMENT

Generative AI tools were used to aid and polish the writing of this paper, including improving clarity and grammar. All AI-assisted content was reviewed and revised by the authors. The authors take full responsibility for the final content of the paper.

### ETHICS STATEMENT

Our work aims to improve the traceability of dLLM-generated text by developing watermarking methods that support both robust detection and tamper detection. Such techniques may help identify model-generated content and detect post-generation modifications. At the same time, publicly describing watermarking mechanisms may provide adversaries with additional information for designing targeted removal or spoofing attacks. We believe that studying these vulnerabilities and developing stronger watermarking methods provides substantial benefits for improving the reliability and security of watermark mechanisms, and that these benefits outweigh the potential risks of disclosure.

### REPRODUCIBILITY STATEMENT

To support reproducibility, we provide the implementation of LCW and the code used for our experiments at `https://anonymous.4open.science/r/LatentChainWatermark-7714`. Detailed experimental settings, watermark configurations, detection parameters, and additional results are provided in the Appendix C. The algorithms for watermark embedding, robust watermark detection, and tamper detection are also described in Appendix A.

## REFERENCES

Anthropic. How claude marks AI-generated content, 2026. URL `https://support.claude.com/en/articles/16266773-how-claude-marks-ai-generated-content`. Accessed: September 4, 2026.

Marianne Arriola, Aaron Gokaslan, Justin Chiu, Zhihan Yang, Zhixuan Qi, Jiaqi Han, Subham Sahoo, and Volodymyr Kuleshov. Block diffusion: Interpolating between autoregressive and diffusion language models. In *International Conference on Learning Representations*, volume 2025, pp. 50726–50753, 2025.

Avi Bagchi, Akhil Bhimaraju, Moulik Choraria, Daniel Alabi, and Lav R Varshney. Watermarking discrete diffusion language models. *arXiv preprint arXiv:2511.02083*, 2025.

Tiwei Bie, Maosong Cao, Kun Chen, Lun Du, Mingliang Gong, Zhuochen Gong, Yanmei Gu, Jiaqi Hu, Zenan Huang, Zhenzhong Lan, et al. Llada2. 0: Scaling up diffusion language models to 100b. *arXiv preprint arXiv:2512.15745*, 2025.

European Parliament and Council of the European Union. Regulation (eu) 2024/1689 of the european parliament and of the council of 13 june 2024 laying down harmonised rules on artificial intelligence and amending regulations (ec) no 300/2008, (eu) no 167/2013, (eu) no 168/2013, (eu) 2018/858, (eu) 2018/1139 and (eu) 2019/2144 and directives 2014/90/eu, (eu) 2016/797 and (eu) 2020/1828 (artificial intelligence act), 2024. URL `https://eur-lex.europa.eu/eli/reg/2024/1689/oj`.

Gemma Team. Gemma 4 technical report, 2026. URL `https://arxiv.org/abs/2607.02770`.

Thibaud Gloaguen, Robin Staab, Nikola Jovanović, and Martin Vechev. Watermarking diffusion language models. In *International Conference on Learning Representations*, volume 2026, pp. 100104–100149, 2026.

Lanxiang Hu, Zhaoxiang Feng, Yulun Wu, Haoran Yuan, Yujie Zhao, Yu-Yang Qian, Bojun Wang, Peng Zhao, Daxin Jiang, Yibo Zhu, et al. Jetspec: Breaking the scaling ceiling of speculative decoding with parallel tree drafting. *arXiv preprint arXiv:2606.18394*, 2026.

Enkelejda Kasneci, Kathrin Sessler, Stefan Küchemann, Maria Bannert, Daryna Dementieva, Frank Fischer, Urs Gasser, Georg Groh, Stephan Günnemann, Eyke Hüllermeier, Stephan Krusche, Gitta Kutyniok, Tilman Michaeli, Claudia Nerdel, Jürgen Pfeffer, Oleksandra Poquet, Michael Sailer, Albrecht Schmidt, Tina Seidel, Matthias Stadler, Jochen Weller, Jochen Kuhn, and Gjergji Kasneci. Chatgpt for good? on opportunities and challenges of large language models for education. *Learning and Individual Differences*, 103:102274, 2023. ISSN 1041-6080. doi: https://doi.org/10.1016/j.lindif.2023.102274. URL https://www.sciencedirect.com/science/article/pii/S1041608023000195.

John Kirchenbauer, Jonas Geiping, Yuxin Wen, Jonathan Katz, Ian Miers, and Tom Goldstein. A watermark for large language models. In *International conference on machine learning*, pp. 17061–17084. PMLR, 2023.

Hugo Krawczyk, Mihir Bellare, and Ran Canetti. Rfc 2104: Hmac: Keyed-hashing for message authentication. Technical Report 2104, RFC Editor, 1997. URL https://www.rfc-editor.org/info/rfc2104/.

Kalpesh Krishna, Yixiao Song, Marzena Karpinska, John Wieting, and Mohit Iyyer. Paraphrasing evades detectors of ai-generated text, but retrieval is an effective defense. *Advances in neural information processing systems*, 36:27469–27500, 2023.

Hao Li, Yubing Ren, Yanan Cao, Yingjie Li, Fang Fang, Shi Wang, and Li Guo. Dualguard: Dual-stream large language model watermarking defense against paraphrase and spoofing attack. In *Findings of the Association for Computational Linguistics: ACL 2026*, pp. 23338–23361, 2026.

Shen Nie, Fengqi Zhu, Zebin You, Xiaolu Zhang, Jingyang Ou, Jun Hu, Jun Zhou, Yankai Lin, Ji-Rong Wen, and Chongxuan Li. Large language diffusion models. *Advances in Neural Information Processing Systems*, 38:50608–50646, 2025.

OpenAI. Gpt-6 astra. Software repository and documentation, 2026. URL https://developers.openai.com/api/docs/models/gpt-6-astra. Accessed 2026-09-24.

Yikang Pan, Liangming Pan, Wenhu Chen, Preslav Nakov, Min-Yen Kan, and William Yang Wang. On the risk of misinformation pollution with large language models. In *Findings of the Association for Computational Linguistics: EMNLP*, pp. 1389–1403, 2023.

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

Qwen Team. Qwen3.5: Towards native multimodal agents, February 2026. URL https://qwen.ai/blog?id=qwen3.5.

Ofek Raban, Gal Chechik, and Ethan Fetaya. Lr-dwm: Efficient watermarking for diffusion language models. In *Findings of the Association for Computational Linguistics: ACL 2026*, pp. 43510–43517, 2026.

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.

Liran Ringel and Yaniv Romano. Accelerating speculative decoding with block diffusion draft trees. *arXiv preprint arXiv:2604.12989*, 2026.

Yuxuan Song, Zheng Zhang, Cheng Luo, Pengyang Gao, Fan Xia, Hao Luo, Zheng Li, Yuehang Yang, Hongli Yu, Xingwei Qu, et al. Seed diffusion: A large-scale diffusion language model with high-speed inference. *arXiv preprint arXiv:2508.02193*, 2025.

Chengyue Wu, Hao Zhang, Shuchen Xue, Shizhe Diao, Yonggan Fu, Zhijian Liu, Pavlo Molchanov, Ping Luo, Song Han, and Enze Xie. Fast-dllm v2: Efficient block-diffusion llm. In *International Conference on Learning Representations*, volume 2026, pp. 128353–128370, 2026a.

Linyu Wu, Linhao Zhong, Wenjie Qu, Yuexin Li, Yue Liu, Shengfang Zhai, Chunhua Shen, and Jiaheng Zhang. Dmark: Order-agnostic watermarking for diffusion large language models. *arXiv preprint arXiv:2510.02902*, 2025.

Zirui Wu, Lin Zheng, Jiacheng Ye, Shansan Gong, Xueliang Zhao, Yansong Feng, Wei Bi, and Lingpeng Kong. Dreamreasoner-8b: Block-size curriculum learning for diffusion reasoning models. *arXiv preprint arXiv:2606.19257*, 2026b.

Liyan Xie, Muhammad Siddeek, Mohamed Seif, Andrea J Goldsmith, and Mengdi Wang. Detecting post-generation edits to watermarked llm outputs via combinatorial watermarking. *arXiv preprint arXiv:2510.01637*, 2025.

Jiacheng Ye, Zhihui Xie, Lin Zheng, Jiahui Gao, Zirui Wu, Xin Jiang, Zhenguo Li, and Lingpeng Kong. Dream 7b: Diffusion large language models. *arXiv preprint arXiv:2508.15487*, 2025.

Zhaoxi Zhang, Xiaomei Zhang, Yanjun Zhang, He Zhang, Shirui Pan, Bo Liu, Asif Qumer Gill, and Leo Yu Zhang. Character-level perturbations disrupt llm watermarks. *arXiv preprint arXiv:2509.09112*, 2025.

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

Tong Zhou, Xuandong Zhao, Xiaolin Xu, and Shaolei Ren. Bileve: Securing text provenance in large language models against spoofing with bi-level signature. *Advances in Neural Information Processing Systems*, 37:56054–56075, 2024.

# A Algorithms

This section provides pseudocode for LCW embedding and detection using the notation defined in Section 3. In the detection algorithms, a path represents a candidate configuration and stores the current block index, the next input-token position, block-key phase, selected latent chains, within-block alignments, and accumulated green-token and unmatched-element counts. The beam $\mathcal{B}$ stores retained paths, and $\mathcal{N}$ collects their candidate extensions before beam pruning.

## A.1 Watermark Embedding

Algorithm 1 applies the green list determined by the current latent chain whenever a token position is updated. The latent chain remains fixed until the block is finalized, after which the finalized block selects the latent chain for the following block.

## A.2 Within-block Dynamic Programming

For a fixed token span and candidate latent chain, LCW searches for the alignment with the largest penalized green-token count. Let $D[i, r]$ store the best partial score after processing the first $i$ input tokens and the first $r$ latent-chain indices. Each state can be reached by matching a token, skipping a token, or skipping a subkey in latent chain. Because each prefix pair is solved once and reused by longer alignments, the dynamic program avoids enumerating all sequences of match and skip operations, as shown in Algorithm 2.

## A.3 Robust Watermark Detection

Algorithm 3 performs block-level beam search over candidate block boundaries and latent chains. For each candidate boundary, Algorithm 2 returns the best within-block alignment for every allowed latent chain. The resulting counts are accumulated along the path and used to compute the penalized watermark score in equation 3.

**Algorithm 1** Watermark Embedding

**Require:** Prompt; decoder; master secret $K$; $N_{\mathrm{bw}}, N_{\mathrm{bk}}, N_{\mathrm{lc}}, N_{\mathrm{sk}}, \gamma, \delta$
**Ensure:** Watermarked token sequence $X$

Derive $N_{\mathrm{bk}}$ block keys and $N_{\mathrm{sk}}$ subkeys from $K$
Construct $N_{\mathrm{lc}}$ candidate latent chains for each block key
Initialize $X \leftarrow ()$ and $b \leftarrow 0$
Sample $\phi$ uniformly from $\{0, \ldots, N_{\mathrm{bk}} - 1\}$
**while** generation continues **do**
    Assign $K_b^{\mathrm{bk}}$ according to equation 1
    **if** $b = 0$ **then**
        $c_b \leftarrow \mathrm{Chain}(K_b^{\mathrm{bk}}, 0)$
    **end if**
    Initialize the token positions of the current block
    **while** the current block is being generated or refined **do**
        Obtain the logits and positions scheduled for the next update
        **for** each scheduled within-block position $r$ **do**
            $t \leftarrow bN_{\mathrm{bw}} + r$
            $\mathcal{G}_{b,r} \leftarrow \mathrm{GreenList}(K_b^{\mathrm{bk}}, c_b[r])$
            $\tilde{\ell}_{t,v} \leftarrow \ell_{t,v} + \delta\, \mathbf{1}[v \in \mathcal{G}_{b,r}]$ for each $v \in \mathcal{V}$
        **end for**
        Apply the decoder update using the biased logits
    **end while**
    Append the finalized block to $X$ in position order
    **if** the block is complete and generation continues **then**
        $X_b \leftarrow (x_{bN_{\mathrm{bw}}}, \ldots, x_{bN_{\mathrm{bw}}+N_{\mathrm{bw}}-1})$
        Select $c_{b+1}$ according to equation 2
        $b \leftarrow b + 1$
    **else**
        **break**
    **end if**
**end while**
**return** $X$

### A.4 Tamper Detection

Algorithm 4 performs tamper detection, which reuses the robust watermark score $Z_W(X')$ and computes two additional scores using the same block-level search framework. For $Z_A(X')$, block boundaries are fixed at multiples of $N_{\mathrm{bw}}$, token position $r$ is matched directly to latent-chain position $r$, and each block may select any candidate latent chain. For $Z_C(X')$, the same boundaries and alignment are used, but each complete block additionally determines the latent chain of the following block according to equation 2.

## B Experimental Setup

**Datasets and Models.** We conduct our main experiments on two publicly available blockwise dLLMs, DreamReasoner-8B (Wu et al., 2026b) and LLaDA-8B-Instruct (Nie et al., 2025). Both models use a block width of $N_{\mathrm{bw}} = 32$ and 32 denoising steps per block. DreamReasoner uses temperature 0.8 and top-$p = 0.95$, while LLaDA uses temperature 0 with a linear scheduler. For evaluation, we randomly sample texts from the realnewslike subset of C4 (Raffel et al., 2020) and use the first 50 tokens of each text as the prompt. The models generate continuations with target lengths $N_X \in \{64, 128, 192, 256\}$.

**Watermark Settings.** We compare LCW with Unigram (Zhao et al., 2023), GSJ (Gloaguen et al., 2026), and Combinatorial Watermarking (CW) (Xie et al., 2025). GSJ represents watermarking designed specifically for uncertain token context in dLLMs, whereas Unigram and CW determine their watermark assignments without relying on generated token values. For CW, we partition the vocabulary into four equally sized subsets and use a length-8 combinatorial pattern that is repeated

**Algorithm 2** Within-block Dynamic Programming

**Require:** Token span $Y = (y_1, \ldots, y_m)$; candidate latent chain $c$; block key $K_b^{\mathrm{bk}}$; unmatched-element penalty $\lambda_p$
**Ensure:** Best alignment, green-token count $h$, and unmatched-element count $e$

Initialize $D[0,0] \leftarrow 0$
**for** $i = 1, \ldots, m$ **do**
    $D[i,0] \leftarrow D[i-1,0] - \lambda_p$
**end for**
**for** $r = 1, \ldots, N_{\mathrm{bw}}$ **do**
    $D[0,r] \leftarrow D[0,r-1] - \lambda_p$
**end for**
**for** $i = 1, \ldots, m$ **do**
    **for** $r = 1, \ldots, N_{\mathrm{bw}}$ **do**
        $\mathcal{G}_r \leftarrow \mathrm{GreenList}(K_b^{\mathrm{bk}}, c[r-1])$
        $s_{\mathrm{match}} \leftarrow D[i-1,r-1] + \mathbf{1}[y_i \in \mathcal{G}_r]$
        $s_{\mathrm{token}} \leftarrow D[i-1,r] - \lambda_p$
        $s_{\mathrm{key}} \leftarrow D[i,r-1] - \lambda_p$
        $D[i,r] \leftarrow \max\{s_{\mathrm{match}}, s_{\mathrm{token}}, s_{\mathrm{key}}\}$
    **end for**
**end for**
Trace back from $D[m, N_{\mathrm{bw}}]$ to recover the best alignment
Count matched green tokens as $h$ and skip token/skip key as $e$
**return** alignment, $h$, $e$

**Algorithm 3** Robust Watermark Detection

**Require:** Tokens $X'$; secret $K$; search settings; calibrated threshold $\tau_\alpha$
**Ensure:** Recovered configuration $p$; watermark score $Z_W(X')$; decision $D_W(X')$

$\mathcal{B} \leftarrow$ one initial path for each block-key phase $\phi$
**while** $\mathcal{B}$ contains an unfinished path **do**
    $\mathcal{N} \leftarrow$ completed paths already in $\mathcal{B}$
    **for** each unfinished path $p \in \mathcal{B}$ **do**
        **for** each candidate end boundary **do**
            Determine the block key from the phase and block index
            **for** each allowed candidate latent chain **do**
                Evaluate the candidate using Algorithm 2
            **end for**
            Extend $p$ using the candidate chain and alignment with the largest penalized score
            Add the extended path to $\mathcal{N}$
        **end for**
    **end for**
    Retain the top $W$ paths in $\mathcal{N}$ as $\mathcal{B}$
**end while**
$p \leftarrow$ the complete retained configuration with the largest $Z_{\mathrm{LCW}}(p)$
$Z_W(X') \leftarrow Z_{\mathrm{LCW}}(p)$
$D_W(X') \leftarrow \mathbf{1}[Z_W(X') \geq \tau_\alpha]$
**return** $p, Z_W(X'), D_W(X')$

periodically. We use CW's native local edit detector. For all watermarking methods, we set the green-list fraction to $\gamma = 0.25$ and the logit bias to $\delta = 2$. LCW uses $N_{\mathrm{sk}} = 6$ subkeys, $N_{\mathrm{lc}} = 4$ latent chains per block key, and $N_{\mathrm{bk}} = 2$ block keys. Watermark detection uses beam width $W = 8$, boundary deviation $\eta = 8$ and unmatched penalty $\lambda_p = 1.3$. Tamper detection uses $\lambda_t = 0.6$.

**Algorithm 4** Tamper Detection

**Require:** Tokens $X'$; master secret $K$; search settings; tamper weight $\lambda_t$; calibrated thresholds $\tau_\alpha, \tau_\beta$

**Ensure:** Watermark decision $D_W(X')$; tamper score $S_{\text{tamper}}(X')$; tamper decision $D_T(X')$

1: Run Algorithm 3 to obtain $Z_W(X')$ and $D_W(X')$
2: Fix block boundaries at $N_{\text{bw}}, 2N_{\text{bw}}, \ldots$
3: Match each token at within-block position $r$ directly to latent-chain position $r$
4: Compute $Z_A(X')$ while allowing each block to select any candidate latent chain
5: Compute $Z_C(X')$ using the same boundaries and alignment, while additionally enforcing the chain transition
6: $S_{\text{align}}(X') \leftarrow \max\{0, Z_W(X') - Z_A(X')\}$
7: $S_{\text{chain}}(X') \leftarrow \max\{0, Z_A(X') - Z_C(X')\}$
8: $S_{\text{tamper}}(X') \leftarrow \lambda_t S_{\text{align}}(X') + (1-\lambda_t) S_{\text{chain}}(X')$
9: $D_T(X') \leftarrow D_W(X')\,\mathbf{1}[S_{\text{tamper}}(X') \geq \tau_\beta]$
10: **return** $D_W(X'), S_{\text{tamper}}(X'), D_T(X')$

# C Additional Experiments

## C.1 Ablation Study

We ablate the main components of LCW to examine where its tamper evidence comes from and how its structural parameters affect detection. All experiments use DreamReasoner at $N_X = 128$.

**Watermark configuration.** Table 3 examines how LCW's watermark structure and key parameters affect detectability, robustness, and tamper evidence. Removing the chain transition leaves watermark detectability and robustness largely intact, but substantially weakens tamper detection: substitution STR@5 drops from 0.558 to 0.070, while deletion and insertion STR@5 decrease from 0.951/0.969 to 0.388/0.333. Similarly, setting $N_{\text{sk}} = N_{\text{lc}} = 1$ retains strong TPR@5 and WSR@5 but almost eliminates tamper evidence, including an STR@5 of 0 for substitution. These results show that strong watermark detection alone is insufficient for tamper evidence, and that both the chain transition and latent chain diversity are important for detecting modifications. Varying $N_{\text{sk}}$, $N_{\text{lc}}$, and $N_{\text{bk}}$ does not uniformly improve performance. For example, increasing $N_{\text{lc}}$ from 4 to 6 strengthens tamper detection, particularly for substitution, but does not consistently improve robustness. Block width has a clear effect on overall performance, and $N_{\text{bw}} = 32$ performs best among the evaluated configurations. Results at 1% show the same main structural trends and are reported in Table 4.

**Tamper detection weight.** Table 5 examines the complementary roles of $S_{\text{align}}$ and $S_{\text{chain}}$ evidence by varying $\lambda_t$. Increasing $\lambda_t$ consistently improves detection of deletion and insertion, with $S_{\text{align}}$-only scoring reaching STR@5 of 0.968 and 0.972, respectively. In contrast, substitution relies primarily on $S_{\text{chain}}$: $S_{\text{chain}}$-only scoring achieves STR@5 of 0.578, whereas $S_{\text{align}}$-only scoring drops to 0.041. These results confirm that the two evidence sources capture different effects of editing. The default $\lambda_t = 0.6$ provides a balanced combination, retaining strong deletion and insertion detection while preserving substantial sensitivity to substitution.

## C.2 Text quality

Following prior work (Gloaguen et al., 2026), we evaluate text quality using log perplexity (log(PPL)) computed with Gemma-4-12B (Gemma Team, 2026) as the reference model, where lower values indicate better quality. Table 6 compares LCW with unwatermarked generation and the watermarking baselines. LCW remains in a similar range to the other watermarking methods across target lengths, although it shows a moderate increase in log(PPL) relative to unwatermarked generation.

Table 3: Watermark configuration ablations on DreamReasoner at $N_X = 128$. We report TPR@5 for detectability, WSR@5 under 10% token-level edits for robustness, and STR@5 under $N_{\text{edit}} = 3$ token-level edits for tamper evidence. The default LCW uses $N_{\text{sk}} = 6$, $N_{\text{lc}} = 4$, $N_{\text{bk}} = 2$, $N_{\text{bw}} = 32$, and $\lambda_t = 0.6$.

| Variant | Detectability | Robustness | | | Tamper detection | | |
|---|---|---|---|---|---|---|---|
| | TPR@5 | Del | Ins | Sub | Del | Ins | Sub |
| No chain transition | 0.990 | 0.861 | 0.891 | 0.990 | 0.388 | 0.333 | 0.070 |
| $N_{\text{sk}} = N_{\text{lc}} = 1$ | 1.000 | 0.970 | 0.931 | 0.990 | 0.200 | 0.192 | 0.000 |
| $N_{\text{sk}} = 4$ | 0.980 | 0.861 | 0.832 | 0.960 | 0.969 | 0.979 | 0.684 |
| $N_{\text{sk}} = 8$ | 1.000 | 0.812 | 0.861 | 0.970 | 0.959 | 0.909 | 0.673 |
| $N_{\text{lc}} = 6$ | 0.990 | 0.792 | 0.782 | 0.980 | 0.990 | 0.979 | 0.760 |
| $N_{\text{bk}} = 4$ | 0.970 | 0.733 | 0.713 | 0.970 | 0.915 | 0.924 | 0.500 |
| $N_{\text{bw}} = 8$ | 0.842 | 0.426 | 0.713 | 0.743 | 0.771 | 0.733 | 0.337 |
| $N_{\text{bw}} = 16$ | 0.980 | 0.683 | 0.782 | 0.941 | 0.876 | 0.924 | 0.265 |
| LCW (default) | 0.983 | 0.799 | 0.823 | 0.977 | 0.951 | 0.969 | 0.558 |

Table 4: Watermark configuration ablations at the stricter 1% operating point. We report TPR@1, WSR@1, and STR@1 using the same configurations and editing settings as Table 3.

| Variant | Detectability | Robustness | | | Tamper detection | | |
|---|---|---|---|---|---|---|---|
| | TPR@1 | Del | Ins | Sub | Del | Ins | Sub |
| No chain transition | 0.990 | 0.802 | 0.851 | 0.980 | 0.258 | 0.253 | 0.010 |
| $N_{\text{sk}} = N_{\text{lc}} = 1$ | 0.990 | 0.941 | 0.871 | 0.980 | 0.202 | 0.194 | 0.000 |
| $N_{\text{sk}} = 4$ | 0.950 | 0.663 | 0.624 | 0.911 | 0.957 | 0.977 | 0.568 |
| $N_{\text{sk}} = 8$ | 0.980 | 0.634 | 0.545 | 0.941 | 0.936 | 0.856 | 0.378 |
| $N_{\text{lc}} = 6$ | 0.970 | 0.554 | 0.465 | 0.931 | 0.892 | 0.771 | 0.354 |
| $N_{\text{bk}} = 4$ | 0.931 | 0.653 | 0.545 | 0.921 | 0.912 | 0.930 | 0.379 |
| $N_{\text{bw}} = 8$ | 0.733 | 0.168 | 0.525 | 0.624 | 0.689 | 0.662 | 0.181 |
| $N_{\text{bw}} = 16$ | 0.980 | 0.475 | 0.653 | 0.921 | 0.782 | 0.856 | 0.082 |
| LCW (default) | 0.950 | 0.599 | 0.605 | 0.916 | 0.902 | 0.928 | 0.372 |

### C.3 Semantic Piggyback Spoofing Attack

Following prior work (Pang et al., 2024; Zhou et al., 2024), we further evaluate LCW against semantic piggyback spoofing attacks. The attacker aims to change the factual content or stance of a watermarked text while modifying as few tokens as possible, using token insertion, deletion, and substitution. We use GPT-6-Astra (OpenAI, 2026) to construct the attacks. For each attack, multiple edited candidates are generated, from which we select a candidate that satisfies the semantic attack objective while minimizing the number of modified tokens. We evaluate watermarked texts generated by DreamReasoner at $N_X = 128$ and $N_X = 256$, with three spoofing attacks applied to each text.

Across all attacks, an average of 3.21 tokens are modified per text, with the number of modified tokens ranging from 1 to 14. At $N_X = 128$, LCW achieves STR@5 of 0.732 and STR@1 of 0.564; at $N_X = 256$, the corresponding results are 0.749 and 0.597. These results are close to those under $N_{\text{edit}} = 3$ token-level edits in Section 4.2, despite the edits here being explicitly selected to alter

Table 5: Tamper-score weight ablation on DreamReasoner at $N_X = 128$. $\lambda_t = 0$ uses only chain evidence, while $\lambda_t = 1$ uses only alignment evidence; LCW uses $\lambda_t = 0.6$ by default.

| Weight | Del | | Ins | | Sub | |
|---|---|---|---|---|---|---|
| | STR@5 | STR@1 | STR@5 | STR@1 | STR@5 | STR@1 |
| $\lambda_t = 0.0$ ($S_{\text{chain}}$ only) | 0.361 | 0.075 | 0.332 | 0.061 | 0.578 | 0.344 |
| $\lambda_t = 0.2$ | 0.684 | 0.242 | 0.661 | 0.242 | 0.578 | 0.344 |
| $\lambda_t = 0.4$ | 0.898 | 0.725 | 0.920 | 0.663 | 0.568 | 0.355 |
| $\lambda_t = 0.6$ (default) | 0.951 | 0.902 | 0.969 | 0.928 | 0.558 | 0.372 |
| $\lambda_t = 0.8$ | 0.968 | 0.917 | 0.976 | 0.936 | 0.466 | 0.064 |
| $\lambda_t = 1.0$ ($S_{\text{align}}$ only) | 0.968 | 0.917 | 0.972 | 0.932 | 0.041 | 0.011 |

Table 6: Text quality measured by log perplexity (log(PPL)); lower values indicate better text quality.

| Method | $N_X = 64$ | $N_X = 128$ | $N_X = 192$ | $N_X = 256$ |
|---|---|---|---|---|
| No-WM | 1.456 | 1.530 | 1.494 | 1.393 |
| Unigram | 1.750 | 1.740 | 1.620 | 1.448 |
| GSJ | 1.554 | 1.573 | 1.505 | 1.411 |
| LCW | 1.682 | 1.681 | 1.543 | 1.422 |

Table 7: Paraphrase robustness on DreamReasoner under DIPPER, reported as WSR@5.

| Method | $N_X = 128$ WSR@5 | $N_X = 256$ WSR@5 |
|---|---|---|
| Unigram | 0.881 | 0.990 |
| GSJ | 0.485 | 0.842 |
| CW | 0.228 | 0.495 |
| LCW | 0.436 | 0.663 |

the semantics of the text. This shows that LCW can effectively detect semantic piggyback spoofing attacks.

### C.4 PARAPHRASING

We evaluate paraphrase robustness on DreamReasoner using DIPPER with lexical diversity 20 and order diversity 0 (Krishna et al., 2023). As shown in Table 7, LCW retains watermark evidence after paraphrasing, achieving WSR@5 of 0.436 and 0.663 at target lengths 128 and 256, respectively. LCW consistently outperforms CW and achieves WSR close to GSJ at length 128. These results show that LCW can preserve detectable watermark evidence even under substantial paraphrasing.

### C.5 LLADA RESULTS

**Detectability.** Table 8 compares LLaDA detection across text lengths using the metrics in Section 4. Averaged equally over the four target lengths, LCW achieves a TPR@5 of 0.977 and a TPR@1 of 0.951, compared with 0.961 and 0.890 for Unigram. Its mean AUC is 0.995, and its TPR@5 increases from 0.936 at length 64 to 0.997 at length 256.

**Robustness.** Figure 4 reports LLaDA results for token-level and character-level edits at rates 0.1 and 0.3, using the protocol in Section 4.3. Averaged equally over the four target lengths, LCW's WSR@5 under 10% token-level edits is 0.715 for deletion, 0.717 for insertion, and 0.963 for substitution. At a 30% editing rate, substitution WSR@5 remains 0.880, whereas deletion and insertion WSR@5 fall to 0.320 and 0.452. Under 10% character-level edits, the corresponding means are 0.747, 0.304, and 0.428 for deletion, insertion, and substitution, indicating greater sensitivity to character-level insertion and substitution than to their token-level counterparts. Mean WSR@1 under 10% token-level substitution is 0.909.

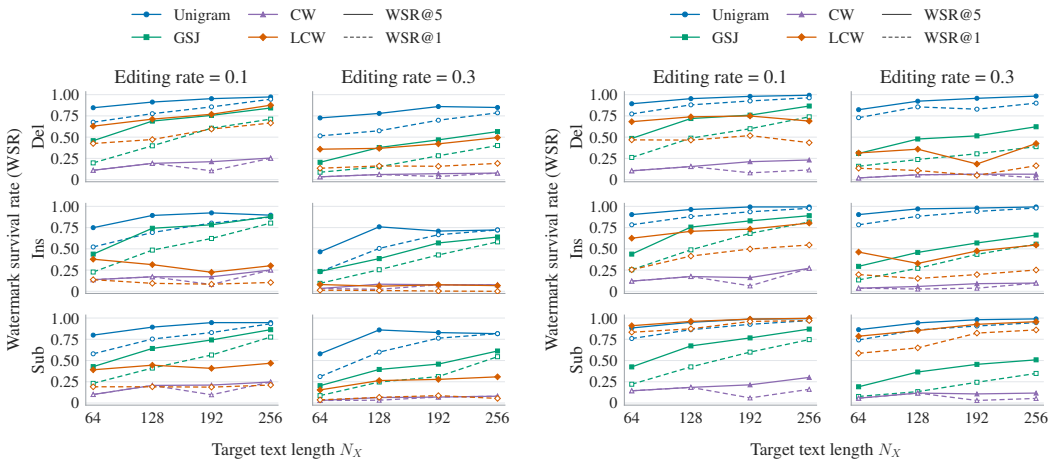
**Tamper evidence.** Figure 5 reports LLaDA STR@5 and STR@1 under token-level deletion, insertion, and substitution using the protocol in Section 4.2. With three edits, LCW's STR@5 averaged equally over the four target lengths is 0.984 for deletion, 0.986 for insertion, and 0.802 for substitution; the corresponding mean STR@1 values are 0.941, 0.939, and 0.605. Single-token edits also leave detectable evidence: mean STR@5 is 0.877 for deletion, 0.867 for insertion, and 0.477 for substitution. For substitution, this mean increases to 0.904 with nine edits, showing that additional edits make tampering easier to detect among texts that retain detectable watermarks.

### C.6 LIMITATIONS

LCW's chain-transition evidence requires a following block. When a finalized block $X_b$ is modified, the detector may reconstruct a different latent chain for $X_{b+1}$, allowing the resulting inconsistency to provide tamper evidence. For the final block, however, no subsequent block exists, so modifications

Table 8: Watermark detectability on LLaDA across target lengths.

| Method | $N_X = 64$ | | | $N_X = 128$ | | | $N_X = 192$ | | | $N_X = 256$ | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | TPR@5 | TPR@1 | AUC | TPR@5 | TPR@1 | AUC | TPR@5 | TPR@1 | AUC | TPR@5 | TPR@1 | AUC |
| Unigram | 0.903 | 0.783 | 0.980 | 0.953 | 0.870 | 0.991 | 0.993 | 0.933 | 0.998 | 0.993 | 0.973 | 0.999 |
| GSJ | 0.555 | 0.331 | 0.915 | 0.849 | 0.629 | 0.974 | 0.886 | 0.786 | 0.983 | 0.943 | 0.886 | 0.990 |
| CW | 0.184 | 0.184 | 0.589 | 0.288 | 0.288 | 0.642 | 0.304 | 0.094 | 0.639 | 0.365 | 0.247 | 0.675 |
| LCW | 0.936 | 0.880 | 0.985 | 0.980 | 0.943 | 0.996 | 0.997 | 0.987 | 1.000 | 0.997 | 0.993 | 1.000 |




(a) Character-level edits

(b) Token-level edits

Figure 4: LLaDA robustness under token-level and character-level edits at editing rates 0.1 and 0.3. Rows show deletion, insertion, and substitution; solid and dashed lines report WSR@5 and WSR@1, respectively.

confined to this block cannot produce chain-transition evidence. In this case, LCW can only on block-alignment evidence. This limitation is particularly relevant to substitutions in the final block, which may leave both block boundaries and within-block alignment unchanged and are therefore more difficult to detect. We further evaluate this setting on texts with $N_X = 256$ by applying a single token edit ($N_{\text{edit}} = 1$) at a randomly selected position within the final block. LCW achieves STR@5 of 0.421 for insertion, 0.368 for deletion, and 0.158 for substitution.

It's worth noting that this limitation is not specific to LCW. Tamper-evident watermarking methods for arLLMs, such as Bileve (Zhou et al., 2024) and DualGuard (Li et al., 2026), can face a similar challenge near the end of a sequence, where there are insufficient subsequent tokens to expose tamper evidence caused by modification. Extending tamper evidence to the final block without requiring additional visible metadata is a potential direction for future work.

Table 9: Detection runtime on DreamReasoner across target lengths. Runtime is reported in seconds per text using a single CPU.

| $N_X$ | Watermark Detection | Tamper Detection |
|---|---|---|
| 64 | 0.214 | 0.311 |
| 128 | 0.868 | 1.199 |
| 192 | 1.520 | 2.106 |
| 256 | 2.097 | 3.057 |

### C.7 Additional Results of Detection Efficiency

LCW detection remains efficient despite searching over block boundaries, latent chains, and within-block alignments. As shown in Table 9, for texts of lengths 64, 128, 192, and 256 tokens, watermark detection takes 0.214, 0.868, 1.520, and 2.097 seconds per text on average, respectively. The runtime increases with text length, but remains around two seconds even at $N_X = 256$. Tamper

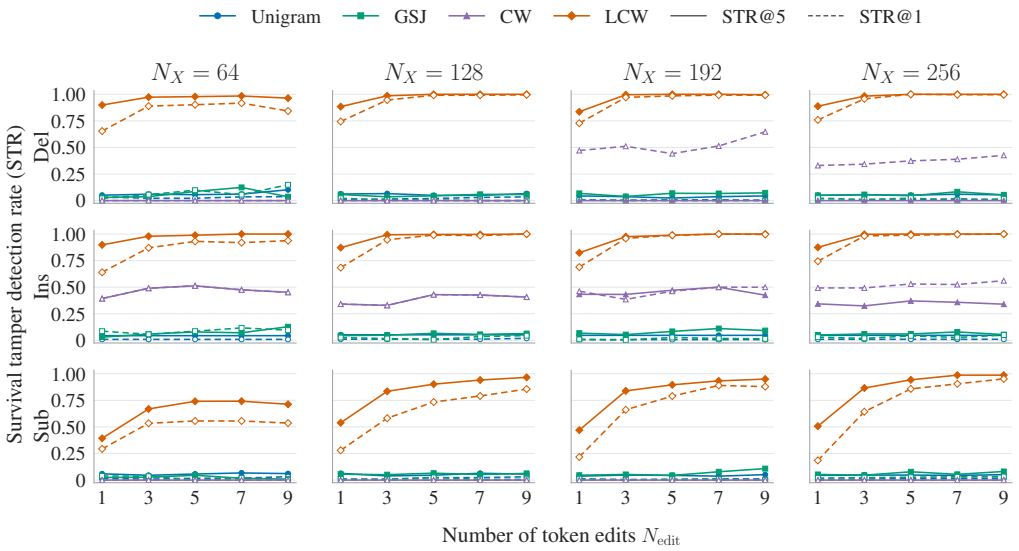



Figure 5: LLaDA tamper detection under token-level edits. Rows show deletion, insertion, and substitution; columns show target text lengths. Solid and dashed lines report STR@5 and STR@1, respectively, using matched watermark/tamper FPR targets.

Table 10: Watermark detectability and robustness under autoregressive and tree-based speculative decoding at $N_X = 128$. Robustness is evaluated under 10% token-level deletion, insertion, and substitution.

| Backend | Method | Detectability | | Del | | Ins | | Sub | |
|---|---|---|---|---|---|---|---|---|---|
| | | TPR@5 | TPR@1 | WSR@5 | WSR@1 | WSR@5 | WSR@1 | WSR@5 | WSR@1 |
| Qwen3.5-9B | Unigram | 1.000 | 0.997 | 1.000 | 0.990 | 1.000 | 0.997 | 1.000 | 0.997 |
| | KGW | 0.983 | 0.967 | 0.973 | 0.943 | 0.980 | 0.963 | 0.963 | 0.926 |
| | Bileve | 0.739 | 0.709 | 0.522 | 0.455 | 0.462 | 0.408 | 0.595 | 0.555 |
| | LCW | 0.990 | 0.990 | 0.812 | 0.723 | 0.802 | 0.683 | 0.960 | 0.950 |
| JetSpec | Unigram | 0.990 | 0.967 | 0.987 | 0.967 | 0.993 | 0.980 | 0.987 | 0.967 |
| | LCW | 0.960 | 0.911 | 0.525 | 0.188 | 0.614 | 0.396 | 0.931 | 0.871 |
| DDTree | Unigram | 0.993 | 0.983 | 0.997 | 0.983 | 0.997 | 0.977 | 0.990 | 0.970 |
| | LCW | 0.970 | 0.921 | 0.446 | 0.277 | 0.614 | 0.485 | 0.960 | 0.901 |

detection requires additional searches and takes 0.311, 1.199, 2.106, and 3.057 seconds per text at the corresponding lengths, respectively. Thus, the additional tamper-detection cost is moderate. All measurements use a single CPU. For longer texts, we can divide the text into shorter segments and apply detection to each segment separately. Overall, these results show that block-level beam search with within-block dynamic programming effectively controls the search cost.

### C.8 Additional Results Beyond Diffusion

**Experimental setup.** For autoregressive generation, we evaluate Qwen3.5-9B with temperature 0.8, top-$p = 0.95$, disabled thinking, and batch size 1. We compare LCW with Unigram (Zhao et al., 2023), KGW (Kirchenbauer et al., 2023), and Bileve (Zhou et al., 2024). For tree-based speculative decoding, we evaluate JetSpec (Hu et al., 2026) and DDTree (Ringel & Romano, 2026) using a Qwen3-8B-8bit target model. Both backends use target-sampling verification, temperature 1.0, top-$p = 0.95$, top-$k = 0$, a tree budget of 64, candidate top-$k = 8$, and depth 15. JetSpec reuses depth-wise predictions across alternative branches, whereas DDTree constructs candidate paths from the position-wise marginals of a DFlash head.

For the beyond-diffusion experiments, LCW uses $N_{\text{sk}} = 6$, $N_{\text{lc}} = 4$, and $N_{\text{bk}} = 2$. For Qwen3.5-9B autoregressive generation, we use a block width of $N_{\text{bw}} = 32$, matching the setting used in the

Table 11: Tamper evidence under autoregressive and tree-based speculative decoding at $N_X = 128$. Results are reported under $N_{\text{edit}} = 3$ token-level deletion, insertion, and substitution edits.

| Backend | Method | Del | | Ins | | Sub | |
|---|---|---|---|---|---|---|---|
| | | STR@5 | STR@1 | STR@5 | STR@1 | STR@5 | STR@1 |
| Qwen3.5-9B | Bileve | 0.464 | 0.464 | 0.471 | 0.467 | 0.234 | 0.236 |
| | LCW | 0.980 | 0.875 | 0.990 | 0.948 | 0.828 | 0.576 |
| JetSpec | LCW | 0.894 | 0.732 | 0.908 | 0.803 | 0.418 | 0.130 |
| DDTree | LCW | 0.929 | 0.797 | 0.963 | 0.840 | 0.510 | 0.117 |

dLLM experiments. For JetSpec and DDTree, we use $N_{\text{bw}} = 16$, following their default speculative-decoding configuration. Detection and calibration follow the same configuration as in Section 4.

**Full results.** Tables 10 and 11 report the complete results. LCW transfers effectively to autoregressive generation. On Qwen3.5-9B, LCW achieves TPR@5 of 0.990, compared with 0.739 for Bileve. Under 10% token-level substitution, its WSR@5 is 0.960 versus 0.595 for Bileve, while under three substitutions its STR@5 reaches 0.828 versus 0.234. LCW also improves substantially over Bileve for deletion and insertion in both robustness and tamper detection, showing that the latent-chain construction remains effective even in a standard autoregressive setting.

LCW also remains effective under tree-based speculative decoding. On JetSpec and DDTree, it achieves TPR@5 of 0.960 and 0.970, substitution WSR@5 of 0.931 and 0.960, and STR@5 of 0.894/0.908/0.418 and 0.929/0.963/0.510 for deletion, insertion, and substitution, respectively. These results show that LCW can support detectability, robustness, and tamper evidence across autoregressive and tree-based speculative generation.