WordMark: A Tokenizer-agnostic Word-level Watermark for Large Language Models

Anonymous authors
Paper under double-blind review

ABSTRACT

Large language models are increasingly used by writing services to generate professional documents whose provenance and integrity may later need to be verified. Although multi-bit watermarking is able to embed a signed provenance record directly in the text, existing generation-time methods typically define bits over model-specific token sequences. Thus, verification requires the originating tokenizer, making these methods poorly suited to services that route requests among heterogeneous or evolving backend models. We present WordMark, a tokenizer-agnostic, word-level multi-bit watermark. During generation, a lightweight generator-specific token encoder predicts the word-level bit that each candidate token is likely to produce, enabling bit-guided autoregressive sampling. A shared word decoder then uses representations from an auxiliary language model to recover context-dependent bits directly from the resulting words. WordMark embeds each ECC codeword bit as a parity constraint over a small group of carrier words, reducing forced low-probability choices. For watermark detection, the payload is represented by repeated index-dependent ECC codewords and verified through designated-codeword matching, reducing false positives on non-watermarked text. Experiments across multiple auxiliary language models, generators, payload lengths, and text domains demonstrate reliable payload recovery and watermark detection while preserving generation quality without requiring the detector to know the generator tokenizer.

1 INTRODUCTION

Large language models (LLMs) are increasingly deployed as writing assistants for producing analyses, summaries, and other professional documents (OpenAI, 2022; Google, 2023; Anthropic, 2023; Copy.ai, 2026; Grammarly, 2026). Users can supply instructions or data, and the writing service generates reports. The service may internally route requests among different backend LLMs over time. However, a report may be copied, exported to PDF, posted online, or redistributed through other platforms, losing the metadata about its origin in the process. Text watermarking provides a way to keep provenance information associated with the content by embedding a recoverable signal directly into the generated text (Liu et al., 2024b; Ye et al., 2026a; Wu et al., 2025). In such a deployment, the embedded payload may be a URL or a provenance identifier that resolves to a provenance record containing a cryptographic digest of the document and the issuer's digital signature. A downstream recipient can recover this reference from the text, verify the signature, and compare the signed digest with the received document to authenticate its provenance and integrity.

However, most generation-time multi-bit watermarking approaches embed and decode bits over the token sequence of the originating language model (Wang et al., 2024; Fernandez et al., 2023; Yoo et al., 2024; Qu et al., 2025; Kordi Boroujeny et al., 2024; Jiang et al., 2025; 2026; Zhou et al., 2024; Fairoze et al., 2023). Token identities determine keyed vocabulary partitions or payload assignments. As a result, recovering the payload requires reconstructing the originating token sequence with the corresponding tokenizer. This is poorly matched to writing services that expose a service-level interface while using heterogeneous or evolving LLM backends.

Post-hoc neural watermarking methods avoid direct dependence on the originating tokenizer by transforming already generated text (Abdelnabi & Fritz, 2021; Zhang et al., 2024; Xu et al., 2024). For example, AWT (Abdelnabi & Fritz, 2021) and REMARK-LLM (Zhang et al., 2024) learn neu-

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.

ral encoder–decoder systems for multi-bit embedding, with the payload length fixed during training. Therefore, supporting a different payload length requires retraining the encoder–decoder. SAEMark (Yu et al., 2025) performs sentence-level selection using sparse-autoencoder (SAE) feature statistics extracted by an auxiliary model, matching these statistics to targets derived from a message-specific key. This enables detection independently of the originating LLM and its tokenizer. However, recovering an arbitrary unknown L-bit message through exhaustive key search requires checking 2^L possibilities, limiting scalability to longer payloads.

In this paper, we propose WordMark, which instead embeds a multi-bit message during ordinary autoregressive generation and provides a word-level decoding interface that does not require access to the originating LLM or its tokenizer. A key challenge is that WordMark decodes bits from words, whereas the generator selects tokens before knowing which word they will eventually form. WordMark bridges this mismatch with a word decoder and a generator-specific token encoder. To obtain a context-dependent word-to-bit mapping, an auxiliary LLM provides contextual hidden states for each word conditioned on its preceding text, and the word decoder learns to divide plausible words between bits 0 and 1 while retaining substantial probability mass for both bits. Because this mapping is defined independently of the generator, the same word decoder can be used across different backend models. During generation, the token encoder predicts which word-level bit each candidate token is likely to produce. It combines hidden states of the current prefix with the candidate token embedding, capturing both the generation context and the specific token being considered. The token encoder is trained from word-decoder outputs over the likely words resulting from each candidate, allowing WordMark to steer token sampling toward a desired word-level bit without explicitly generating multiple candidate words.

Another challenge is that not every generation step provides comparable support for both watermark bits. At some steps, most of the generator's next-token probability mass may favor candidates associated with the same word-level bit, leaving little natural support for the opposite bit. Therefore, forcing a predetermined message bit at every word would either reduce embedding reliability or require sampling unlikely tokens, thereby degrading text quality. WordMark addresses this problem with syndrome-based embedding. Instead of assigning one coded message bit directly to each word, WordMark represents each coded bit by a parity constraint over a small block of words. For a block of K words, the bits of the first $K-1$ words are not specified in advance; they are allowed to take the bit naturally induced by generation. Once these bits are known, the bit of the final carrier is uniquely determined so that the XOR of all K carrier bits equals the coded message bit. Thus, only one position per block must satisfy a predetermined bit. The forced position need not coincide with an imbalanced generation step, while any imbalanced step among the first $K-1$ positions can follow the bit naturally induced by generation. As a result, a low-support bit is much less likely to be forced at an unfavorable generation step, reducing distortion to the generator's original distribution.

To improve payload-recovery reliability, WordMark applies a BCH error-correcting code (Bose & Ray-Chaudhuri, 1960). However, successful error-correcting decoding alone does not establish watermark presence: an unrelated bit sequence extracted from non-watermarked text may also be decoded to a valid payload. Therefore, we combine WordMark's word-level bit channel with the designated-verification construction of BREW (Kim et al., 2026). The payload is repeatedly encoded into index-dependent ECC codewords. WordMark embeds and decodes the individual codeword bits, while designated-codeword verification estimates a common payload across the recovered codewords and measures how many of them agree with the codewords expected for that payload. This separates payload recovery from watermark detection and reduces false positives on non-watermarked text. In summary, our contributions are as follows:

- We propose WordMark, a word-level multi-bit watermarking framework for heterogeneous LLM backends. Unlike existing generation-time watermarks whose embedding and decoding rules depend on model-specific token sequences, WordMark defines watermark decoding over words. This allows a common decoding interface to be used across different generators without access to the originating LLM or its tokenizer.
- We develop a word-level watermarking architecture that bridges word-level decoding and token-level autoregressive generation using a shared word decoder and generator-specific token encoders. We further introduce syndrome-based embedding, which uses local parity constraints to reduce the need to force a predetermined bit when the generator strongly favors the opposite bit, thereby reducing generation distortion.

- We combine WordMark's word-level bit channel with repeated index-dependent ECC codewords and designated-codeword verification in BREW, separating payload recovery from watermark-presence detection and reducing false positives on non-watermarked text. We evaluate WordMark across heterogeneous base and instruction-tuned LLMs, payload lengths, and text domains in terms of bit and payload recovery, watermark detection, text quality, and generation overhead.

2 THREAT MODEL

System setting. An individual or organization, referred to as the *issuer*, uses the writing service to generate a document x. Before generation, a compact payload p, such as a URL, is allocated to the document. WordMark embeds p into the text during generation. After the final document has been generated, the issuer computes its cryptographic digest $d = \mathsf{Hash}(x)$, and signs the digest using its private key sk, obtaining the signature $\sigma = \mathsf{Sign}_{sk}(d)$. The URL p resolves to a provenance record $R = (d, \sigma)$, while the corresponding public key pk is made available to downstream verifiers through an authenticated public channel.

Given a received document x', a downstream recipient first applies the watermark detector to determine whether the text contains a watermark and to recover p. The recipient then retrieves R through p and verifies $\mathsf{Verify}_{pk}(d, \sigma) = 1$. Finally, the recipient computes $\mathsf{Hash}(x')$ and checks whether $\mathsf{Hash}(x') = d$. It will distinguish between three consequences:

- If the watermark is present, the signature is valid, and the two digests match, the document is accepted as authentic and intact.
- If the watermark remains detectable and the signature is valid but $\mathsf{Hash}(x') \neq d$, the document retains the issuer's provenance signal but fails the integrity check, indicating that it has been modified after issuance.
- If the watermark detection fails, the text is not attributed to the issuer.

Adversary's goals. The adversary seeks to cause a document that was not issued by the issuer in its received form to be accepted as authentic and intact. Of particular interest are piggyback attacks (Pang et al., 2024), where the adversary modifies a small number of semantically important elements, such as numbers, negations, or factual keywords, while attempting to preserve the embedded watermark. Such modifications may inject misinformation without substantially changing the remainder of the document. The adversary may also attempt to have independently created text falsely attributed to the issuer.

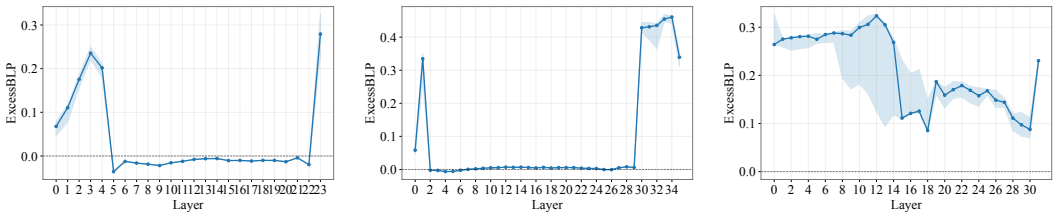
Adversary's capabilities. The adversary may obtain legitimately watermarked documents, inspect the public decoding and verification procedures, and arbitrarily modify or redistribute the text. The adversary may also create non-watermarked text and attempt to present it as content issued by the issuer. We assume that the adversary cannot compromise the issuer's private key, alter authenticated provenance records R, or break the security of the underlying digital-signature and cryptographic hash functions.

3 WORDMARK TRAINING

3.1 DESIGN INSIGHT

WordMark defines watermark bits over words, whereas an autoregressive generator selects tokens before the resulting word is fully known. A shared word decoder defines the word-to-bit mapping, and a generator-specific token encoder predicts this mapping during generation.

Word-level decoding. Given a prefix c and a word w, the word decoder outputs $D_{\theta_d}(b \mid c, w)$ for $b \in \{0, 1\}$. It uses the auxiliary model's hidden state at the final token of w, making the mapping context-dependent and reproducible from text alone without access to the originating generator or its tokenizer.



(a) Pythia-410M (b) Qwen2.5-3B (c) Mistral-7B

Figure 1: Layer-wise partitionability. Curves show median ExcessBLP across 50 independently initialized linear probes while shading shows the minimum–maximum range.

Token-level prediction. For a candidate token v following prefix c, let $\mathcal{W}(c, v)$ contain the possible complete words formed by starting or continuing a word with v. Under generator G, the corresponding bit distribution is

$$y_G(b \mid c, v) = \sum_{w \in \mathcal{W}(c,v)} p_G(w \mid c, v)\, D_{\theta_d}(b \mid c, w). \tag{1}$$

The token encoder predicts this distribution without enumerating possible words during generation. For a given auxiliary model, the word decoder can be shared across generators, while each generator uses its own token encoder.

3.2 Learning the Word Decoder

Discovering balanced partitions. For each prefix c, the auxiliary model $\mathcal{A}$ provides a contextual word cloud $\mathcal{W}_{\mathcal{A}}(c)$ of plausible next words with normalized probabilities $p_{\mathcal{A}}(w \mid c)$. We seek a partition in which words assigned to each bit collectively account for a substantial share of the auxiliary model's next-word probability, reducing the need to generate unlikely words during embedding.

We probe the auxiliary model's contextual word representations at different layers. A linear probe assigns each word a score

$$s_{\ell,r}(c, w) = \hat{\mathbf{v}}_{\ell,r}^{\top} h_{\mathcal{A}}^{\ell}(c \| w) + \beta_{\ell,r}, \tag{2}$$

where $h_{\mathcal{A}}^{\ell}(c \| w)$ is the hidden state at the final token of w, and r indexes probe initializations. Without predefined bit labels, we train probes to balance the probability mass assigned to the two bits while encouraging sufficient score variation to avoid a degenerate partition.

Figure 1 compares layers using ExcessBLP, which measures improvement in hard-partition balance over random probes. We select layers with high median ExcessBLP and low variation across initializations. The selected probe defines the training labels

$$y(c, w) = \mathbf{1}\left[s_{\ell^\star, r^\star}(c, w) \geq 0\right]. \tag{3}$$

The probe objectives and layer-selection details are provided in Appendix C.

Partition-guided decoder training. A balanced hard partition does not necessarily yield confident probe probabilities. Because these probabilities supervise the token encoder through Eq. (1), we train a word decoder on the fixed probe-assigned labels using probability-weighted binary cross-entropy. This encourages more confident predictions while retaining the learned partition. The trained decoder is frozen for token-encoder training and text decoding. Appendix C reports the objective and partition-agreement results. In Appendix F, we additionally compare linear and nonlinear partition–decoder designs. We observe that some auxiliary-model layers exhibit limited linear partitionability but support well-balanced partitions under nonlinear probes. Therefore, we compare linear and nonlinear probes and their associated word decoders in Appendix F.

3.3 Token Encoder Training

For each generator G, we estimate training targets $\widehat{y}_G(b \mid c, v)$ offline from plausible complete words that each candidate token may begin or continue. Inspired by EAGLE-3 (Li et al., 2025), the

token encoder combines hidden states of prefix c from multiple layers, denoted by $H_G(c)$, with the candidate token embedding $e_G(v)$:

$$E_{\theta_e^G}(\cdot \mid c, v) = F_G\left(H_G(c), e_G(v)\right), \tag{4}$$

where F_G is a lightweight fusion network. The prefix features are shared across candidate tokens, avoiding an additional generator forward pass for each candidate token. We train separate word-start and word-continuation encoders with cross-entropy against $\widehat{y}_G$, since continuation choices are constrained by the partially generated word. The training details are provided in Appendix D.

4 WordMark Generation and Decoding

4.1 Indexed ECC Codeword Construction

Successful ECC decoding alone does not establish watermark presence because bits extracted from non-watermarked text may also decode to a valid payload. Following BREW Kim et al. (2026), we encode the same payload into indexed codewords for subsequent consistency verification. We use BCH codes for error correction, although the framework is not restricted to this choice.

Let $\mathbf{p} \in \{0,1\}^L$ denote the payload. We use a BCH code with codeword length n and correction capability e_{BCH}, allowing correction of up to e_{BCH} bit errors per codeword. For codeword index u, we derive an index-specific mask

$$\mathbf{z}_u = \mathrm{PRF}_\kappa(u) \in \{0,1\}^L, \tag{5}$$

where κ is the masking key. The corresponding codeword is

$$\mathbf{c}_u = \mathrm{BCH}_{\mathrm{Encode}}(\mathbf{p} \oplus \mathbf{z}_u) = (c_{u,1}, \ldots, c_{u,n}). \tag{6}$$

These masks vary the encoded representation across indices while preserving the underlying payload. WordMark embeds the resulting codeword bits during generation.

4.2 Syndrome-Based Bit Embedding

A carrier word satisfies a deterministic character-level policy shared by generation and decoding, where the policy is introduced in Appendix E. Assigning a predetermined bit to every carrier can require unlikely word choices when the generator favors the opposite bit. WordMark instead represents each codeword bit $c_{u,j}$ through a syndrome block of K carrier bits:

$$\bigoplus_{k=1}^{K} b_{u,j,k} = c_{u,j}. \tag{7}$$

The first $K-1$ positions are *free* where the bits are determined during generation. The final position is *forced*, with target

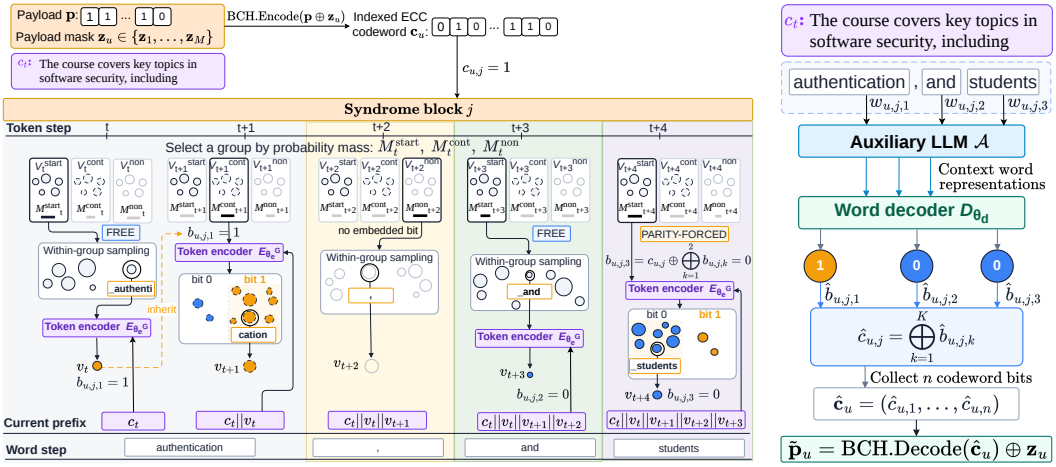
$$b_{u,j,K} = c_{u,j} \oplus \bigoplus_{k=1}^{K-1} b_{u,j,k}. \tag{8}$$

Thus, only one position per syndrome block receives a predetermined bit before the word starts. Each length-n codeword requires nK carrier words.

4.3 Token-Level Watermark Sampling

Figure 2 illustrates WordMark's generation and decoding procedures. During generation, the token encoder guides token sampling toward the word-level bits assigned by syndrome embedding. Because a carrier word may span multiple tokens, at each generation step, WordMark distinguishes tokens that start a word from those that continue it and applies the corresponding sampling rules.

Word-aware grouping. At step t, WordMark partitions the candidate token set $\mathcal{V}_t$ into three groups: $\mathcal{V}_t^{\mathrm{start}}$, $\mathcal{V}_t^{\mathrm{cont}}$, and $\mathcal{V}_t^{\mathrm{non}}$, containing tokens that start a carrier word, continue the active



(a) Generation (b) Decoding

Figure 2: Overview of WordMark generation and decoding.

carrier word, or do neither. Membership depends on the current text and the candidate token's appended text, rather than its token ID alone. Let $p_G(v \mid c_t)$ denote the generator distribution after truncation and normalization. The probability mass of group g is

$$M_t^g = \sum_{v \in \mathcal{V}_t^g} p_G(v \mid c_t), \qquad g \in \{\text{start}, \text{cont}, \text{non}\}. \tag{9}$$

WordMark selects group g with probability M_t^g. Within the selected group, the original conditional distribution is

$$\pi_t(v \mid g) = \frac{p_G(v \mid c_t)}{M_t^g}, \qquad v \in \mathcal{V}_t^g. \tag{10}$$

Embedding bits modifies this within-group distribution while preserving the group-selection probabilities. Non-carrier tokens are sampled directly from $\pi_t(\cdot \mid \text{non})$. Appendix G provides the detailed grouping rules.

Target assignment and support checks. Each carrier word is assigned a target bit $b_t^\star$, which remains fixed as the word is generated across one or more tokens. At a free position, the first token is sampled from the original start-group distribution, and the token encoder's predicted bit becomes the target. At a forced position, the target bit is determined before the word begins by Eq. (8).

For a forced word start, let $\mathcal{V}_t^{\text{start}}$ denote the word-start candidate set and let $b_t^\star$ denote the target bit. The target-supporting candidates and their total probability mass are

$$\mathcal{V}_{t,\text{hard}}^\star = \left\{ v \in \mathcal{V}_t^{\text{start}} : E_{\theta_e^G}(b_t^\star \mid c_t, v) > 0.5 \right\}, \tag{11}$$

$$\Pi_t^{\text{start}}(b_t^\star) = \sum_{v \in \mathcal{V}_{t,\text{hard}}^\star} \pi_t(v \mid \text{start}). \tag{12}$$

If $\Pi_t^{\text{start}}(b_t^\star) \geq \tau_0$, WordMark samples from $\pi_t(\cdot \mid \text{start})$ restricted to $\mathcal{V}_{t,\text{hard}}^\star$; otherwise, it uses the original word-start distribution.

For word continuations, WordMark instead applies a soft penalty. Define

$$\mathcal{V}_{t,\text{good}}^\star = \left\{ v \in \mathcal{V}_t^{\text{cont}} : E_{\theta_e^G}(b_t^\star \mid c_t, v) > \tau_1 \right\}, \tag{13}$$

and let the continuation distribution be

$$\tilde{\pi}_t(v \mid \text{cont}) = \frac{\pi_t(v \mid \text{cont}) \exp\left[-\lambda \mathbf{1}[v \notin \mathcal{V}_{t,\text{good}}^\star]\right]}{\sum_{u \in \mathcal{V}_t^{\text{cont}}} \pi_t(u \mid \text{cont}) \exp\left[-\lambda \mathbf{1}[u \notin \mathcal{V}_{t,\text{good}}^\star]\right]}. \tag{14}$$

Thus, word starts use a hard support gate, whereas continuations use a soft bias that discourages but does not exclude low-confidence tokens. The complete procedure is given in Appendix G and Algorithm 1.

4.4 Decoding and Watermark Detection

The first W eligible generated words are left unwatermarked to provide context for decoding without the original prompt. The recipient skips these words, applies the same carrier policy, and decodes each subsequent carrier word w_i from its preceding textual context q_i:

$$\hat{b}_i = \arg\max_{b \in \{0,1\}} D_{\theta_d}(b \mid q_i, w_i). \tag{15}$$

Each consecutive group of K carrier bits yields one codeword bit:

$$\hat{c}_{u,j} = \bigoplus_{k=1}^{K} \hat{b}_{u,j,k}. \tag{16}$$

Payload recovery and verification. Every group of n recovered codeword bits forms an extracted sequence $\hat{\mathbf{c}}_u$. Let N_{seq} denote the number of complete n-bit sequences recovered from the text x. The detector independently BCH-decodes these sequences, removes their index-specific masks, and selects the most frequently recovered payload as $\hat{\mathbf{p}}$.

It then reconstructs the expected codeword $\mathbf{c}_u^\star = \text{BCH}_{\text{Encode}}(\hat{\mathbf{p}} \oplus \mathbf{z}_u)$ at each index and computes

$$S(x) = \frac{1}{N_{\text{seq}}} \sum_{u=1}^{N_{\text{seq}}} \mathbf{1}\left[d_{\text{H}}(\hat{\mathbf{c}}_u, \mathbf{c}_u^\star) \le e_{\text{BCH}}\right], \tag{17}$$

where d_{H} is Hamming distance. If $N_{\text{seq}} = 0$ or there is no valid payload estimate, the detector returns no watermark. Otherwise, watermark presence requires $S(x) \ge \gamma$, where γ is the detection threshold. This check measures consistency with the estimated payload $\hat{\mathbf{p}}$ across indexed codewords, beyond successful ECC decoding alone. Appendix H details the procedure.

5 Results

5.1 Experimental Setup

Models and data. We construct word decoders using three auxiliary LLMs: Pythia-410M (Biderman et al., 2023), Qwen2.5-3B (Qwen Team, 2024), and Mistral-7B (Jiang et al., 2023). WordMark is evaluated with 4 generators: Llama2-7B (Touvron et al., 2023), Gemma2-9B (Gemma Team, 2024), Mixtral-8×7B (Jiang et al., 2024), and Qwen2.5-14B-Instruct (Qwen Team, 2024). For each auxiliary LLM, its word decoder is shared across all generators, while a lightweight token encoder is trained separately for each generator. Both components are trained on OpenWebText. We evaluate on four datasets covering diverse text domains: OpenWebText (Gokaslan et al., 2019), BillSum (Kornilova & Eidelman, 2019), Wikipedia (Merity et al., 2017), and CCNews (Hamborg et al., 2017). We additionally evaluate the Qwen2.5-14B-Instruct on Alpaca (Taori et al., 2023).

Default settings and baselines. Unless otherwise stated, we use syndrome size $K = 1$, warmup length $W = 10$, sampling thresholds $\tau_0 = 0.2$, $\tau_1 = 0.55$, and $\lambda = 1$, temperature 1.0, and top-$p = 0.99$ sampling. When the auxiliary LLM is Pythia-410M or Mistral-7B, we use MLP decoder; when the auxiliary LLM is Qwen2.5-3B, we use linear decoder. On OBWC, we evaluate 11-bit and 19-bit payloads that are encoded with BCH($n = 31$, $L = 11$, $e_{\text{BCH}} = 5$) and BCH($n = 51$, $L = 19$, $e_{\text{BCH}} = 5$),respectively. On Alpaca, we evaluate 5-bit and 6-bit payloads with BCH($n = 15$, $L = 5$, $e_{\text{BCH}} = 3$) and BCH($n = 31$, $L = 6$, $e_{\text{BCH}} = 7$), respectively. Each example will generate $N_{\text{BCH}}K$ words unless otherwise stated. We compare WordMark with representative multi-bit watermarking methods, including MPAC (Yoo et al. (2024)), Qu et al. (2025), and BREW (Kim et al., 2026). For MPAC, Qu et al., and WordMark, we use the same BREW-style designated verification procedure to reduce false positives and to make detection scores comparable across methods. The details of training configurations and specific settings of baselines are provided in Appendix I.

Metrics. Carrier-bit accuracy (CBA) measures the fraction of carrier words whose decoded bits match their embedded bits. Syndrome-bit accuracy (SBA) measures the fraction of BCH codeword bits correctly recovered from the syndrome blocks. When $K = 1$, SBA is identical to CBA.

Payload-bit accuracy (PBA) measures the fraction of correctly recovered payload bits, while payload recovery accuracy (PRA) is the fraction of examples whose entire payload is recovered correctly. We use perplexity (PPL) to measure text quality. For watermark detection, we report AUC and TPR@1%FPR that represents the true-positive rate at 1% false-positive rate.

5.2 Embedding and Decoding Performance across Model Pairs

We evaluate WordMark's embedding and decoding performance across all 4×3 pairs of generators and auxiliary LLMs. For each auxiliary LLM, one word decoder is shared across all 4 generators, with a separate token encoder trained for each pair. This experiment examines how well these components work together to embed and recover payloads across different model pairs. Each payload is encoded into a single BCH codeword and embedded and decoded using WordMark, without BREW's repeated codeword construction for watermark detection. In Table 1 and Table 2, across the evaluated configurations, different pairs of generator and auxiliary LLM show reliable decoding performance. Statistics of CBA per data set from OBWC are reported in the Appendix J.

Table 1: WordMark decoding performance on OBWC. Each payload is encoded into a BCH codeword without BREW integration. OBWC reports mean over OpenWebText, BillSum, Wikipedia, and CC-News.

Auxiliary LM	Generator	11 bits				19 bits				Non-WM PPL
		CBA ↑	PRA ↑	PBA ↑	PPL ↓	CBA ↑	PRA ↑	PBA ↑	PPL ↓	↓
Pythia-410M	Gemma2-9B	0.9677	0.9990	0.9994	7.0903	0.9713	0.9994	0.9997	7.3985	6.5628
	Llama2-7B	0.9515	0.9948	0.9971	10.6828	0.9500	0.9945	0.9970	11.3720	8.7742
	Mixtral-8×7B	0.9255	0.9898	0.9937	9.4723	0.9388	1.0000	1.0000	9.9970	6.6737
	Qwen2.5-14B-Instruct	0.9466	0.9922	0.9957	3.3230	0.9493	0.9947	0.9972	3.5198	3.4540
Qwen2.5-3B	Gemma2-9B	0.9666	0.9990	0.9995	7.5185	0.9659	0.9988	0.9994	7.9138	6.5628
	Llama2-7B	0.9364	0.9848	0.9916	10.7395	0.9371	0.9879	0.9934	11.7265	8.7742
	Mixtral-8×7B	0.9370	0.9844	0.9915	8.3745	0.9462	0.9938	0.9967	8.8028	6.6737
	Qwen2.5-14B-Instruct	0.9515	0.9954	0.9975	3.5543	0.9549	0.9971	0.9985	3.7450	3.4540
Mistral-7B	Gemma2-9B	0.9625	0.9975	0.9987	7.4740	0.9641	0.9984	0.9991	7.7903	6.6737
	Llama2-7B	0.9399	0.9879	0.9934	10.9438	0.9411	0.9907	0.9950	11.3528	8.7742
	Mixtral-8×7B	0.9327	0.9827	0.9910	9.6005	0.9445	0.9912	0.9947	10.1640	6.6737
	Qwen2.5-14B-Instruct	0.9338	0.9816	0.9899	3.3330	0.9374	0.9883	0.9936	3.5483	3.4540

Table 2: WordMark decoding performance on Alpaca using 5-bit and 6-bit payloads. Each payload is encoded into a BCH codeword without BREW integration.

Auxiliary LM	Generator	5 bits				6 bits				Non-WM PPL
		CBA ↑	PRA ↑	PBA ↑	PPL ↓	CBA ↑	PRA ↑	PBA ↑	PPL ↓	↓
Pythia-410M	Qwen2.5-14B-Instruct	0.9492	0.9927	0.9956	2.2692	0.9523	0.9991	1.0000	2.3569	
Qwen2.5-3B	Qwen2.5-14B-Instruct	0.9411	0.9892	0.9935	2.4212	0.9453	0.9980	0.9992	2.6004	2.0742
Mistral-7B	Qwen2.5-14B-Instruct	0.9441	0.9905	0.9960	2.2665	0.9468	0.9992	0.9993	2.4421	

5.3 Comparison with Existing Watermarking Methods

We compare WordMark with existing multi-bit watermarking methods in terms of payload recovery, watermark detection, and text quality. For MPAC, Qu et al., and WordMark, we adopt the BREW-style repeated, index-specific codeword construction and designated-verification procedure, while retaining each method's native embedding and codeword-extraction mechanisms. WordMark, BREW, and MPAC use BCH coding, whereas Qu et al. retains its native Reed–Solomon coding. We evaluate Gemma2-9B on OBWC and Qwen2.5-14B-Instruct on Alpaca, using Qwen2.5-3B as WordMark's auxiliary LLM. All methods use a 5-bit payload and a 250-token generation budget. WordMark uses the default syndrome size $K = 1$.

Table 3 highlights the trade-off between embedding distortion and detection performance. MPAC uses a small bias of 2 on logits, resulting in the lowest PPL among the watermarking methods but weaker detection performance. BREW and Qu et al. use a larger bias of 6 to improve detection, at the cost of much higher PPL. With $K = 1$, WordMark achieves perfect payload recovery and a favorable quality–detection trade-off. Its PPL is higher than that in non-watermarked generation

Table 3: Comparison with token-level multi-bit watermarking baselines. All methods use a 5-bit payload and a 250-token generation budget.

	Gemma2-9B					Qwen2.5-14B-Instruct				
Metric	Non-WM	MPAC	Qu et al.	BREW	WordMark	Non-WM	MPAC	Qu et al.	BREW	WordMark
PBA ↑	–	0.9792	0.9612	0.9780	**0.9972**	–	0.7804	0.9484	0.9868	**1.0000**
PRA ↑	–	0.9520	0.9600	0.9780	**0.9861**	–	0.5660	0.9380	0.9760	**1.0000**
AUC ↑	–	0.9757	0.9965	**0.9988**	0.9963	–	0.6988	0.9569	0.9961	**0.9997**
TPR@1%FPR ↑	–	0.2244	0.9979	**1.0000**	0.9861	–	0.0300	0.8260	0.9660	**0.9900**
PPL ↓	9.297	**11.903**	20.752	20.493	15.912	2.087	**2.497**	6.352	6.246	6.258

because the carrier-bit constraint modifies sampling at selected carrier words. However, WordMark applies this constraint only to eligible carrier words and can retain natural sampling when the target-bit support is insufficient, resulting in lower PPL than Qu et al. and BREW. Note that the perplexity is higher that those of Table 1 and Table 2 because this evaluation adopts a BREW-style WordMark for detection, which will repeatedly embed codewords into the text and produce more forced positions. Increasing K provides more free carrier positions and may reduce distortion, while also changing the number of completed blocks and detection performance. We analyze this trade-off in Section 5.4.

5.4 Ablation of Syndrome Size K

We evaluate how the syndrome size K affects watermark detectability and text quality. Each BCH codeword bit is represented by a syndrome block of K carrier words. Within each block, one carrier position is assigned a forced target bit, while the other $K-1$ positions are free. A forced position may fall back to natural sampling due to insufficient probability mass assigned by the generator to tokens representing the target bit. Let FBR denote the fraction of forced positions that fall back in this way. We define $\mathrm{NSCR} = 1 - \frac{1-\mathrm{FBR}}{K}$, which measures the fraction of carrier positions that are not actively forced by the watermarking rule. We report the average number of complete indexed BCH codewords available to the detector under each token budget. Each codeword contains n bits and requires nK carrier words to embed. As shown in Figure 3, smaller K allows more complete codewords to be embedded within the same generation budget, improving AUC, TPR at 1% FPR, syndrome-bit accuracy, and payload recovery. At the same time, smaller K lowers NSCR, meaning that a larger fraction of carrier positions are actively forced that increases perplexity. Thus, K controls the trade-off between detection evidence and text quality.

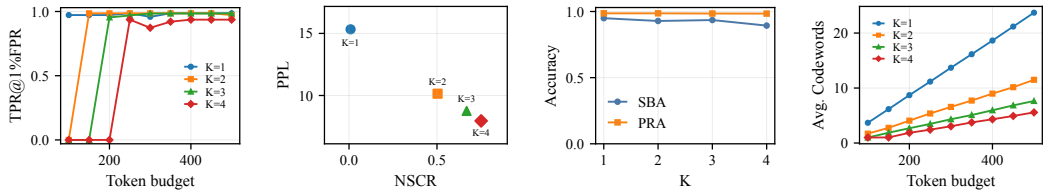


Figure 3: Ablation of syndrome size K on OpenWebText with Gemma-2-9B as generator and Qwen2.5-3B as auxiliary LLM, using BCH(15,5,3).

6 Conclusions

We propose WordMark, a tokenizer-agnostic watermark for embedding multi-bit payloads during LLM generation. By probing auxiliary-LLM representations, WordMark learns a context-dependent binary partition of words representing watermark bits 0 and 1, and trains a word decoder to infer the corresponding bit directly from text, without access to the originating generator or tokenizer. During generation, generator-specific token encoders predict decoder-aligned bits for candidate tokens, allowing one word decoder to support multiple generators. Syndrome-based embedding reduces forced decisions at low-support steps, while BREW-style repeated, index-dependent ECC codewords enable watermark-presence detection. Across auxiliary-model and generator pairs and multiple text domains, WordMark achieves reliable payload recovery and detection while maintaining a favorable trade-off between detection performance and text quality.

Ethical Statement

This work studies WordMark, a word-level watermarking method for large language models, with the goal of improving the attribution and accountability of AI-generated text. The work does not involve human subjects, sensitive personal data, or the collection of new personal information. Although watermarking technologies may have dual-use implications, WordMark is designed to support provenance verification and transparency rather than to facilitate harmful content generation, surveillance, or privacy violations. The method does not introduce additional content-generation capabilities beyond those of the underlying language models. We have conducted this research in accordance with the ICLR Code of Ethics. The authors are responsible for the claims, methods, and results presented in this paper.

Reproducibility Statement

We have made extensive efforts to support the reproducibility of WordMark. The paper and appendix provide detailed descriptions of the watermark construction, word decoder, generator-specific token encoder, carrier-selection policy, syndrome-based embedding procedure, BREW-style detection procedure, and decoding algorithms. We also specify the models, datasets, payload sizes, coding parameters, sampling settings, evaluation metrics, and baseline configurations used in our experiments. To promote transparency and reproducibility, we will release the implementation, model configurations, and experimental scripts upon publication, subject to the licensing requirements of the underlying models and datasets.

References

Scott Aaronson. My AI Safety Lecture for UT Effective Altruism, November 2022. URL https://scottaaronson.blog/?p=6823. Accessed May 5, 2023.

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

Anthropic. Introducing Claude. Website, 2023. https://www.anthropic.com/news/introducing-claude.

Mikhail J Atallah, Victor Raskin, Michael Crogan, Christian Hempelmann, Florian Kerschbaum, Dina Mohamed, and Sanket Naik. Natural language watermarking: Design, analysis, and a proof-of-concept implementation. In *Information Hiding: 4th International Workshop, IH 2001 Pittsburgh, PA, USA, April 25–27, 2001 Proceedings 4*. Springer, 2001.

Mikhail J Atallah, Victor Raskin, Christian F Hempelmann, Mercan Karahan, Radu Sion, Umut Topkara, and Katrina E Triezenberg. Natural language watermarking and tamperproofing. In *International workshop on information hiding*. Springer, 2002.

Stella Biderman, Hailey Schoelkopf, Quentin Gregory Anthony, Herbie Bradley, Kyle O'Brien, Eric Hallahan, Mohammad Aflah Khan, Shivanshu Purohit, USVSN Sai Prashanth, Edward Raff, et al. Pythia: A suite for analyzing large language models across training and scaling. In *International Conference on Machine Learning*, pp. 2397–2430. PMLR, 2023.

Raj C. Bose and Dwijendra K. Ray-Chaudhuri. On a class of error correcting binary group codes. *Information and Control*, 3(1):68–79, 1960. doi: 10.1016/S0019-9958(60)90287-4.

Copy.ai. Copy.ai. https://www.copy.ai/, 2026. Accessed: 2026-08-08.

Amirhossein Dabiriaghdam and Lele Wang. Simmark: A robust sentence-level similarity-based watermarking algorithm for large language models. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 30773–30794, 2025.

Sumanth Dathathri, Abigail See, Sumedh Ghaisas, Po-Sen Huang, Rob McAdam, Johannes Welbl, Vandana Bachani, Alex Kaskasoli, Robert Stanforth, Tatiana Matejovicova, Jamie Hayes, Nidhi

Vyas, Majd Al Merey, Jonah Brown-Cohen, Rudy Bunel, Borja Balle, Taylan Cemgil, Zahra Ahmed, Kitty Stacpoole, Ilia Shumailov, Ciprian Baetu, Sven Gowal, Demis Hassabis, and Pushmeet Kohli. Scalable watermarking for identifying large language model outputs. *Nature*, 634(8035):818–823, Oct 2024. ISSN 1476-4687. doi: 10.1038/s41586-024-08025-4. URL https://doi.org/10.1038/s41586-024-08025-4.

Jaiden Fairoze, Sanjam Garg, Somesh Jha, Saeed Mahloujifar, Mohammad Mahmoody, and Mingyuan Wang. Publicly-detectable watermarking for language models. *arXiv preprint arXiv:2310.18491*, 2023.

Xianheng Feng, Jian Liu, Kui Ren, and Chun Chen. A certified robust watermark for large language models, 2024. URL https://arxiv.org/abs/2409.19708.

Xiaoyan Feng, Yanjun Zhang, He Zhang, Leo Yu Zhang, and Shirui Pan. Tracing provenance and detecting tampering with complementary LLM watermarks, 2026. URL https://arxiv.org/abs/2608.12713.

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

Gemma Team. Gemma 2: Improving open language models at a practical size, 2024. URL https://arxiv.org/abs/2408.00118.

Aaron Gokaslan, Vanya Cohen, Ellie Pavlick, and Stefanie Tellex. Openwebtext corpus. http://Skylion007.github.io/OpenWebTextCorpus, 2019.

Google. Introducing Gemini: Our Largest and Most Capable AI Model. Website, 2023. https://blog.google/technology/ai/google-gemini-ai/.

Grammarly. Go: Your new personal ai assistant. https://www.grammarly.com/go-ai-assistant, 2026. Accessed: 2026-08-08.

Felix Hamborg, Norman Meuschke, Corinna Breitinger, and Bela Gipp. news-please: A generic news crawler and extractor. In *Proceedings of the 15th International Symposium of Information Science*, pp. 218–223, March 2017. doi: 10.5281/zenodo.4120316.

Abe Hou, Jingyu Zhang, Tianxing He, Yichen Wang, Yung-Sung Chuang, Hongwei Wang, Lingfeng Shen, Benjamin Van Durme, Daniel Khashabi, and Yulia Tsvetkov. Semstamp: A semantic watermark with paraphrastic robustness for text generation. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pp. 4067–4082, 2024.

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

A. Q. Jiang, A. Sablayrolles, A. Mensch, C. Bamford, D. S. Chaplot, D. de las Casas, F. Bressand, G. Lengyel, G. Lample, L. Saulnier, L. R. Lavaud, M.-A. Lachaux, P. Stock, T. L. Scao, T. Lavril, T. Wang, T. Lacroix, and W. E. Sayed. Mistral 7B, 2023. URL https://arxiv.org/abs/2310.06825.

Albert Q. Jiang, Alexandre Sablayrolles, Antoine Roux, Arthur Mensch, Blanche Savary, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Emma Bou Hanna, Florian Bressand, Gianna Lengyel, Guillaume Bour, Guillaume Lample, Lélio Renard Lavaud, Lucile Saulnier, Marie-Anne Lachaux, Pierre Stock, Sandeep Subramanian, Sophia Yang, Szymon Antoniak, Teven Le Scao, Théophile Gervet, Thibaut Lavril, Thomas Wang, Timothée Lacroix, and William El Sayed. Mixtral of experts, 2024. URL https://arxiv.org/abs/2401.04088.

Ya Jiang, Chuxiong Wu, Massieh Kordi Boroujeny, Brian Mark, and Kai Zeng. StealthInk: A Multi-bit and Stealthy Watermark for Large Language Models. In *Forty-second International Conference on Machine Learning*, 2025. URL https://openreview.net/forum?id=dktpDfUTtj.

Ya Jiang, Massieh Kordi Boroujeny, Surender Suresh Kumar, and Kai Zeng. MirrorMark: Generalizable Mirrored Sampling for Multi-bit LLM Watermarking, 2026. URL https://arxiv.org/abs/2601.22246.

Joeun Kim, HoEun Kim, Dongsup Jin, and Young-Sik Kim. Block-wise codeword embedding for reliable multi-bit text watermarking. In *Forty-third International Conference on Machine Learning*, 2026. URL https://openreview.net/forum?id=Ym1zKxtv8V.

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

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

Massieh Kordi Boroujeny, Ya Jiang, Kai Zeng, and Brian Mark. Multi-Bit Distortion-Free Watermarking for Large Language Models. *arXiv preprint arXiv:2402.16578*, 2024.

Anastassia Kornilova and Vladimir Eidelman. BillSum: A corpus for automatic summarization of US legislation. In Lu Wang, Jackie Chi Kit Cheung, Giuseppe Carenini, and Fei Liu (eds.), *Proceedings of the 2nd Workshop on New Frontiers in Summarization*, pp. 48–56, Hong Kong, China, November 2019. Association for Computational Linguistics. doi: 10.18653/v1/D19-5406. URL https://aclanthology.org/D19-5406.

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

Yuhui Li, Fangyun Wei, Chao Zhang, and Hongyang Zhang. EAGLE-3: Scaling up inference acceleration of large language models via training-time test. *Advances in Neural Information Processing Systems*, 38:136737–136756, 2025.

Aiwei Liu, Leyi Pan, Xuming Hu, Shiao Meng, and Lijie Wen. A semantic invariant robust watermark for large language models. In *International Conference on Learning Representations*, 2024a.

Aiwei Liu, Leyi Pan, Yijian Lu, Jingjing Li, Xuming Hu, Xi Zhang, Lijie Wen, Irwin King, Hui Xiong, and Philip Yu. A survey of text watermarking in the era of large language models. *ACM Comput. Surv.*, 57(2), November 2024b. ISSN 0360-0300. doi: 10.1145/3691626. URL https://doi.org/10.1145/3691626.

Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. Pointer sentinel mixture models. In *Proceedings of ICLR*, 2017.

OpenAI. ChatGPT: Optimizing language models for dialogue. Website, 2022. https://openai.com/blog/chatgpt.

Qi Pang, Shengyuan Hu, Wenting Zheng, and Virginia Smith. No free lunch in LLM watermarking: Trade-offs in watermarking design choices. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024. URL https://openreview.net/forum?id=rIOl7KbSkv.

Wenjie Qu, Wengrui Zheng, Tianyang Tao, Dong Yin, Yanze Jiang, Zhihua Tian, Wei Zou, Jinyuan Jia, and Jiaheng Zhang. Provably robust multi-bit watermarking for AI-generated text. In *34th USENIX Security Symposium (USENIX Security 25)*, pp. 201–220, Seattle, WA, August 2025. USENIX Association. ISBN 978-1-939133-52-6. URL https://www.usenix.org/conference/usenixsecurity25/presentation/qu-watermarking.

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

Rohan Taori, Ishaan Gulrajani, Tianyi Zhang, Yann Dubois, Xuechen Li, Carlos Guestrin, Percy Liang, and Tatsunori B. Hashimoto. Stanford alpaca: An instruction-following llama model. https://github.com/tatsu-lab/stanford_alpaca, 2023.

Umut Topkara, Mercan Topkara, and Mikhail J Atallah. The hiding virtues of ambiguity: quantifiably resilient watermarking of natural language text through synonym substitutions. In *Proceedings of the 8th workshop on Multimedia and security*, 2006.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*, 2023.

Lean Wang, Wenkai Yang, Deli Chen, Hao Zhou, Yankai Lin, Fandong Meng, Jie Zhou, and Xu Sun. Towards codable watermarking for injecting multi-bits information to llms. In B. Kim, Y. Yue, S. Chaudhuri, K. Fragkiadaki, M. Khan, and Y. Sun (eds.), *International Conference on Learning Representations*, volume 2024, pp. 39375–39401, 2024. URL https://proceedings.iclr.cc/paper_files/paper/2024/file/abdc8c031aa6c6917c3b593166e5e340-Paper-Conference.pdf.

Junchao Wu, Shu Yang, Runzhe Zhan, Yulin Yuan, Lidia Sam Chao, and Derek Fai Wong. A survey on LLM-generated text detection: Necessity, methods, and future directions. *Computational Linguistics*, 51(1):275–338, 2025.

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

Xiaojun Xu, Jinghan Jia, Yuanshun Yao, Yang Liu, and Hang Li. Robust multi-bit text watermark with LLM-based paraphrasers. *arXiv preprint arXiv:2412.03123*, 2024.

Peigen Ye, Huali Ren, Zhengdao Li, Anli Yan, Hongyang Yan, Shaowei Wang, and Jin Li. Securing large language models: A survey of watermarking and fingerprinting techniques. *ACM Computing Surveys*, 58(7):1–35, 2026a.

Ziping Ye, Gourab Dey, Christos Christodoulopoulos, Charith Peris, Anil Ramakrishna, Weitong Ruan, Aram Galstyan, Kai-Wei Chang, Rahul Gupta, and Ninareh Mehrabi. SWAN: Semantic Watermarking with Abstract Meaning Representation. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics*, pp. 36304–36315, San Diego, California, USA, 2026b. Association for Computational Linguistics. doi: 10.18653/v1/2026.acl-long.1681.

KiYoon Yoo, Wonhyuk Ahn, and Nojun Kwak. Advancing beyond identification: Multi-bit watermark for large language models. In Kevin Duh, Helena Gomez, and Steven Bethard (eds.), *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pp. 4031–4055, Mexico City, Mexico, June 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.naacl-long.224. URL https://aclanthology.org/2024.naacl-long.224/.

Zhuohao Yu, Xingru Jiang, Weizheng Gu, Yidong Wang, Qingsong Wen, Shikun Zhang, and Wei Ye. SAEMark: Steering personalized multilingual LLM watermarks with sparse autoencoders. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025. URL https://openreview.net/forum?id=tXnyVPNOfa.

Ruisi Zhang, Shehzeen Samarah Hussain, Paarth Neekhara, and Farinaz Koushanfar. REMARK-LLM: A Robust and Efficient Watermarking Framework for Generative Large Language Models. In *33rd USENIX Security Symposium (USENIX Security 24)*, 2024.

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 THE USE OF LARGE LANGUAGE MODELS AND AI TOOLS

In preparing this manuscript, we used large language models as writing assistants to improve the grammar, clarity, and organization of text drafted by the authors. AI tools were also used to assist with parts of the experimental workflow, including code development, debugging, experiment configuration, result organization, and preliminary analysis of experimental outputs. The authors designed the method and experiments, selected the models, datasets, metrics, and baselines, verified the implementation and reported results, interpreted the findings, and made all final decisions regarding the content of this paper. No AI tool was used as a substitute for scientific judgment or authorship, and the authors remain fully responsible for the accuracy, integrity, and originality of the work.

B RELATED WORK

B.1 GENERATION-TIME LLM WATERMARKING

Generation-time LLM watermarks embed a signal by modifying how the next token is sampled from the model distribution. Zero-bit watermarking approaches primarily test whether generated text is statistically consistent with a keyed watermarking process. Reweighting-based approaches bias or reweight key-dependent subsets of the vocabulary (Kirchenbauer et al., 2023b;a; Hu et al., 2024; Wu et al., 2024; Liu et al., 2024a; Qu et al., 2025; Feng et al., 2024), while sampling-based approaches use detector-reproducible randomness to control token selection, including inverse sampling (Kuditipudi et al., 2023), Gumbel-max sampling (Aaronson, 2022), and tournament sampling (Dathathri et al., 2024). Multi-bit watermarking methods extend this channel by associating different payload values with message-dependent vocabulary partitions, keys, or sampling randomness transformations (Wang et al., 2024; Fernandez et al., 2023; Yoo et al., 2024; Qu et al., 2025; Kordi Boroujeny et al., 2024; Jiang et al., 2025; 2026).

Despite their different embedding mechanisms, these approaches generally define watermark evidence over the token sequence of the originating model. Token identities and token contexts determine vocabulary partitions, message assignments, pseudorandom seeds, or sampling transformations. Consequently, detection or payload decoding typically requires reconstructing the originating token sequence with the corresponding tokenizer. WordMark instead defines watermark decoding over words, allowing heterogeneous generators to use their native tokenizations while sharing a common word-level decoding interface.

B.2 TOKENIZER-INDEPENDENT TEXT WATERMARKING

Several lines of work seek to embed and recover watermark information directly from the text without reconstructing the token sequence used by the originating language model. Classical linguistic watermarks encode information through handcrafted lexical or syntactic transformations, such as synonym substitution and meaning-preserving syntactic rewrites (Atallah et al., 2001; 2002; Topkara et al., 2006). Because the watermark is defined over visible linguistic structures rather than generator token IDs, these methods do not depend on the tokenizer that originally produced the text. However, their embedding opportunities are restricted to a predefined set of linguistic alternatives that must be identified and designed in advance. WordMark instead learns a context-dependent word-to-bit mapping from naturally occurring word choices and uses these learned choices as watermark carriers during autoregressive generation.

Learned post-hoc approaches such as AWT (Abdelnabi & Fritz, 2021) and REMARK-LLM (Zhang et al., 2024) replace handcrafted rules with message-conditioned neural encoder–decoder models that transform an existing text and recover the embedded payload. Their message dimension is tied to the training configuration, so supporting a different payload length generally requires training a new encoder–decoder pair. Xu et al. (2024) divide an existing text into sentence-level segments and use learned paraphrasers to encode one bit in each segment, making the payload length depend on the number of available sentence segments. In contrast, WordMark supports variable-length multi-bit payloads through its word-level watermark channel, without requiring the word decoder or token encoder to be retrained for a different payload length.

A separate line of tokenizer-independent methods embeds watermarks by rejecting or selecting among complete generated sentences (Hou et al., 2024; Dabiriaghdam & Wang, 2025; Ye et al., 2026b; Yu et al., 2025). For example, SemStamp (Hou et al., 2024) repeatedly generates complete candidate sentences until one satisfies a key-dependent semantic constraint, whereas SAEMark (Yu et al., 2025) generates multiple candidate sentences, evaluates their feature statistics, and selects the one that best matches the target bit. Related methods similarly accept, reject, or resample complete generated units according to semantic watermark conditions (Dabiriaghdam & Wang, 2025; Ye et al., 2026b). These methods first generate complete candidate sentences and then test whether they satisfy the watermark condition. As a result, several candidates may need to be generated before one is accepted, causing substantial generation overhead. WordMark instead embeds bits directly during autoregressive generation, without generating multiple candidate sentences or rewriting the text afterward.

B.3 Watermark Detection and Provenance Authentication

Payload decoding and watermark detection answer different questions. A multi-bit decoder attempts to recover an embedded message, often under the assumption that a watermark is present. However, for multi-bit watermarks, payload decoding alone does not provide a reliable watermark detector. BREW (Kim et al., 2026) observes that an error-correcting decoder may map bits extracted from non-watermarked text to a valid codeword, leading to false positives if successful decoding is treated as evidence of watermark presence. BREW therefore represents the same payload through repeated index-dependent codewords, estimates a common payload from the recovered codewords, and checks whether they match the codewords designated by that estimate. We adopt this construction for watermark-presence detection while replacing BREW's token-level bit channel with WordMark's word-level embedding and decoding channel.

Watermark presence and content integrity are also distinct. An adversary may modify a small number of semantically important elements in a watermarked document, such as numbers, negations, or factual keywords, while leaving enough of the watermark intact for its payload to remain recoverable. Such piggyback attacks can preserve the apparent provenance signal while changing the meaning of the document (Pang et al., 2024; Feng et al., 2026). Bileve (Zhou et al., 2024) addresses a related distinction by combining a coarse-grained signal for source tracing with content-dependent signature bits for integrity verification.

WordMark separates these checks in a different way. It embeds a compact payload, such as a URL, that identifies an external provenance record containing a cryptographic digest and its signature. A downstream recipient first uses block-wise verification to determine whether the watermark is present and recover the payload. The recipient then verifies the signed digest and compares it with the digest of the received text. Consequently, a modified document may still be classified as watermarked and retain a recoverable provenance reference, while failing the subsequent integrity check. The corresponding system and adversary models are described in Section 2.

C Word Decoder Training Details

In this section, we introduce the probe objectives, layer-selection criterion, and partition-guided decoder training used in Section 3.2.

C.1 Probe Training Objectives

For each prefix c, let $\mathcal{W}_{\mathcal{A}}(c) = \{w_1, \ldots, w_{N_c}\}$ denote the contextual word cloud, with auxiliary-model probabilities normalized over this set:

$$\sum_{w \in \mathcal{W}_{\mathcal{A}}(c)} p_{\mathcal{A}}(w \mid c) = 1. \tag{18}$$

The linear probe score is defined in Eq. (2), where $\hat{\mathbf{v}}_{\ell,r}$ is a learned unit-norm direction and $\beta_{\ell,r}$ is a learned scalar bias. The probe converts its score into soft bit probabilities:

$$\rho_{\ell,r}(1 \mid c, w) = \phi\left(\frac{s_{\ell,r}(c,w)}{T_{\ell,r}(c)}\right), \qquad \rho_{\ell,r}(0 \mid c, w) = 1 - \rho_{\ell,r}(1 \mid c, w), \tag{19}$$

where ϕ is the sigmoid function. The adaptive temperature $T_{\ell,r}(c) > 0$ adjusts for score scale, reducing the tendency of small scores to appear balanced simply because their sigmoid outputs are close to $\frac{1}{2}$. We define a balance objective that encourages equal allocation of auxiliary-model probability mass:

$$\mathcal{L}_{\text{balance}} = \mathbb{E}_c \left[\left(\sum_{w \in \mathcal{W}_{\mathcal{A}}(c)} p_{\mathcal{A}}(w \mid c)\, \rho_{\ell,r}(1 \mid c, w) - \frac{1}{2} \right)^2 \right]. \tag{20}$$

To discourage a degenerate solution with nearly identical scores, we also encourage sufficient within-cloud score spread. Let

$$\Delta_{\ell,r}(c) = \sqrt{\text{Var}_{w \sim p_{\mathcal{A}}(\cdot|c)} \left[s_{\ell,r}(c, w) \right]} \tag{21}$$

be the probability-weighted standard deviation of the scores. The spread loss is

$$\mathcal{L}_{\text{spread}} = \mathbb{E}_c \left[\max\left(0, \tau_{\text{spread}} - \Delta_{\ell,r}(c)\right)^2 \right]. \tag{22}$$

The overall objective is

$$\mathcal{L}_{\text{probe}} = \mathcal{L}_{\text{balance}} + \lambda_{\text{spread}} \mathcal{L}_{\text{spread}}, \tag{23}$$

where τ_{spread} specifies the target minimum spread and λ_{spread} controls its contribution.

C.2 Partitionability and Layer Selection

Probe r at layer ℓ defines the hard assignment

$$\widehat{b}_{\ell,r}(c, w) = \mathbf{1}[s_{\ell,r}(c, w) \geq 0]. \tag{24}$$

The probability mass assigned to bit b is

$$q^{(b)}_{\ell,r}(c) = \sum_{w \in \mathcal{W}_{\mathcal{A}}(c)} p_{\mathcal{A}}(w \mid c)\, \mathbf{1}\left[\widehat{b}_{\ell,r}(c, w) = b \right]. \tag{25}$$

We measure balanced partitionability as

$$\text{BLP}_{\ell,r}(c) = 2 \min\left\{ q^{(0)}_{\ell,r}(c), q^{(1)}_{\ell,r}(c) \right\}. \tag{26}$$

A value of 1 indicates equal probability mass on both sides, whereas a value near 0 indicates that almost all mass lies on one side. We average over prefixes:

$$\overline{\text{BLP}}_{\ell,r} = \mathbb{E}_c[\text{BLP}_{\ell,r}(c)].$$

Some layers may support balanced partitions even under random directions. Therefore, we estimate a layer-specific random-probe baseline RandomBLP_ℓ, using random unit directions with the same bias-balancing procedure, and define

$$\text{ExcessBLP}_{\ell,r} = \overline{\text{BLP}}_{\ell,r} - \text{RandomBLP}_\ell. \tag{27}$$

We train 50 independently initialized probes per layer. Figure 1 reports the median and minimum–maximum range of ExcessBLP. We select layers with high median ExcessBLP and low variation across initializations: layer 14 for Mistral-7B, layer 23 for Pythia-410M, and layer 33 for Qwen2.5-3B. The selected probe provides the fixed labels $y(c, w)$ in Eq. (3).

C.3 Partition-Guided Decoder Objective

The word decoder outputs

$$D_{\theta_d}(b \mid c, w), \qquad b \in \{0, 1\}, \tag{28}$$

using the selected auxiliary-model representation. For decoder logit function g_{θ_d},

$$D_{\theta_d}(1 \mid c, w) = \phi\left(g_{\theta_d}\left(h^{\ell^\star}_{\mathcal{A}}(c \| w) \right) \right), \tag{29}$$

and $D_{\theta_d}(0 \mid c, w) = 1 - D_{\theta_d}(1 \mid c, w)$.

We train the decoder from random initialization on the frozen probe-assigned labels using probability-weighted binary cross-entropy:

$$\mathcal{L}_D = -\mathbb{E}_c \left[\sum_{w \in \mathcal{W}_{\mathcal{A}}(c)} p_{\mathcal{A}}(w \mid c) \log D_{\theta_d}(y(c, w) \mid c, w) \right]. \tag{30}$$

The probability weights emphasize plausible words under each prefix. After training, D_{θ_d} is frozen and used both to construct token-encoder targets and to decode text.

C.4 Partition Agreement and Prediction Confidence

We evaluate the decoder on held-out word clouds. Weighted assignment accuracy (WAG) measures agreement with the probe-assigned labels:

$$\mathrm{WAG} = \mathbb{E}_c \left[\sum_{w \in \mathcal{W}_{\mathcal{A}}(c)} p_{\mathcal{A}}(w \mid c) \mathbf{1} \left[\arg\max_b D_{\theta_d}(b \mid c, w) = y(c, w) \right] \right]. \tag{31}$$

We also report cross-entropy against these labels. Probe CE is computed using the original probe probabilities $\rho_{\ell^\star, r^\star}$, while decoder CE uses D_{θ_d}, with the same probability weighting.

Table 4 reports results for linear decoders. Decoder training reduces CE while retaining high agreement with the probe-assigned partition. Comparisons with nonlinear probes and MLP decoders are provided in Appendix F.

Table 4: Partition-guided decoder performance on test word clouds. Probe and decoder CE use the probe-assigned labels.

Auxiliary LM	Probe layer	Probe CE ↓	Decoder WAG ↑ / CE ↓
Pythia-410M	23	0.380	0.985 / 0.067
Qwen2.5-3B	33	0.174	0.970 / 0.094
Mistral-7B	14	0.641	0.994 / 0.054

D Token Encoder Training Details

For each generator G, the token encoder approximates the word-decoder-induced distribution in Eq. (1). Training targets $\widehat{y}_G(b \mid c, v)$ are estimated offline by evaluating plausible complete words that each candidate token may begin or continue.

Input representations. The encoder uses prefix features from three generator layers:

$$H_G(c) = \left(h_G^{\ell_1}(c), h_G^{\ell_2}(c), h_G^{\ell_3}(c) \right), \tag{32}$$

where $h_G^{\ell}(c)$ is the hidden state at the final token position of prefix c from layer ℓ. Together with candidate token embedding $e_G(v)$, these features are passed to the fusion network F_G, as defined in Eq. (4). All candidate tokens at the same generation step share $H_G(c)$.

Training objective. We minimize cross-entropy against the estimated bit distribution:

$$\mathcal{L}_{E_G} = -\mathbb{E}_{(c,v)} \left[\sum_{b \in \{0,1\}} \widehat{y}_G(b \mid c, v) \log E_{\theta_e^G}(b \mid c, v) \right]. \tag{33}$$

Separate encoders are trained for word-start and word-continuation steps. Word-start steps typically allow a broader range of possible words, whereas continuation steps are constrained by the partially generated word.

E CARRIER POLICY AND CARRIER STATE MACHINE

WordMark defines a deterministic carrier policy to identify the carrier words that can be used during training, generation, and detection.

E.1 CARRIER POLICY

A *carrier word* is a generated word that satisfies the carrier policy and therefore can carry 1 bit. In our implementation, a basic carrier is an ASCII alphabetic word composed of the characters `A--Z` and `a--z`. Surrounding whitespace and punctuation are not part of the carrier. For example, in `The model, however, failed to`, the words `The`, `model`, `however`, `failed`, and `to` are potential carriers, whereas the spaces and punctuation are not.

The carrier policy deliberately uses a conservative subset of word forms to ensure that generation and detection identify the same carrier positions. We exclude alphabetic sequences that form part of structured identifiers, including alphanumeric forms, such as `GPT4`, underscore-containing identifiers, email addresses, URLs, and domains, such as `openai.com`. These forms provide limited lexical flexibility, and watermark steering could alter their intended content. For forms containing a hyphen or apostrophe, we use only the alphabetic portion preceding the connector as a carrier and suppress the remaining suffix. Thus, `zero-day` contributes `zero`, and `company's` contributes `company`.

E.2 CARRIER STATE MACHINE

Generation proceeds token by token, but a carrier word may span multiple tokens, and its eligibility may change as additional characters are generated. Therefore, WordMark maintains a *carrier state* s_t among generation steps. The state records the text currently being formed at the end of the output, its status under the carrier policy, and its provisionally assigned bit if WordMark is controlling it as a carrier. The status indicates whether the current form remains eligible, has become ineligible, or whether the remainder of a hyphenated or apostrophe-containing form should be ignored. If there is no unresolved carrier-relevant text at the end of the output, such as a punctuation, the state is empty.

A bit assigned to an active word remains provisional until the word ends. If the formed word satisfies the carrier policy, WordMark commits the word and its bit to the carrier-bit sequence. If later characters make the word ineligible, the provisional assignment is discarded, and WordMark will try to embed the same assigned bit at the next carrier position.

For each candidate token v, let $a_t(v)$ denote the exact text that v would append to the current output. Applying this text to the current carrier state produces the candidate token's *carrier transition*:

$$\mathcal{T}\big(s_t, a_t(v)\big) = \Big(g_t(v), C_t(v), s_{t+1}^{(v)}\Big), \tag{34}$$

where $g_t(v) \in \{\mathrm{start}, \mathrm{cont}, \mathrm{non}\}$ is the candidate token's sampling group, $C_t(v)$ is the ordered list of eligible carrier words completed by selecting the candidate token, together with their assigned bits, and $s_{t+1}^{(v)}$ is the resulting carrier state. Each carrier event contains a completed eligible word and its assigned bit. If the candidate token does not complete an eligible carrier word, then $C_t(v) = \varnothing$.

The sampling group and the carrier events serve different purposes. The group determines how the candidate token participates in the current sampling decision, whereas $C_t(v)$ determines which carrier positions are committed after the candidate token is selected. For example, suppose the active word is `model`. A candidate token that appends `s` belongs to the continuation group, produces no carrier event, and changes the active word to `models`. A candidate token that appends a space belongs to the non-carrier group but completes and commits `model`. A candidate token that appends `4` also belongs to the non-carrier group, but it makes the form `model4` ineligible and therefore produces no carrier event. In the last case, `model4` is ineligible and is not counted as a carrier during either generation or decoding. The next eligible word continues the current syndrome block.

The transitions of all candidate tokens at a step are evaluated without changing s_t. After token v_t is sampled, WordMark applies only its transition, records the carrier events in $C_t(v_t)$ in textual order, and updates the carrier state as

$$s_{t+1} = s_{t+1}^{(v_t)}. \tag{35}$$

E.3 CANDIDATE DETOKENIZATION

Computing the carrier transition requires the exact text $a_t(v)$ produced by each candidate token. A straightforward implementation obtains this text by detokenizing the full token prefix before and after appending v. Repeating this operation for every candidate token repeatedly processes the same prefix, so its cost grows with both the number of candidate tokens and the prefix length.

WordMark instead analyzes the tokenizer's vocabulary and decoding rules once and compiles them into a tokenizer-specific lookup structure. During generation, WordMark retains a small *detokenization state* d_t containing only the prior decoding information that may affect the next token. The required summary depends on the tokenizer. For example, in a byte-level tokenizer, d_t may retain trailing bytes that cannot be decoded until they are combined with bytes from the next token. It may also record whether generation is at the beginning of the output when the tokenizer applies different whitespace rules there. The detokenization state does not contain the full token prefix or any watermark information.

Given d_t and candidate token v, the compiled lookup returns both the exact text produced by the candidate token and the detokenization state that would result if it were selected:

$$\mathrm{LOOKUP}(d_t, v) = \left(a_t(v), d_{t+1}^{(v)}\right). \tag{36}$$

The resulting text $a_t(v)$ is then passed to the carrier state machine:

$$\begin{aligned} (d_t, v) &\xrightarrow{\mathrm{LOOKUP}} \left(a_t(v), d_{t+1}^{(v)}\right), \\ \left(s_t, a_t(v)\right) &\xrightarrow{\mathcal{T}} \left(g_t(v), C_t(v), s_{t+1}^{(v)}\right). \end{aligned} \tag{37}$$

Thus, d_t determines what text a candidate token would produce, whereas s_t determines how that text would affect carrier words and watermark positions.

Because each compiled lookup depends only on the small state d_t and candidate token v, it does not process the full token prefix, and its cost does not grow with the prefix length. After token v_t is sampled, it retains only the corresponding next states:

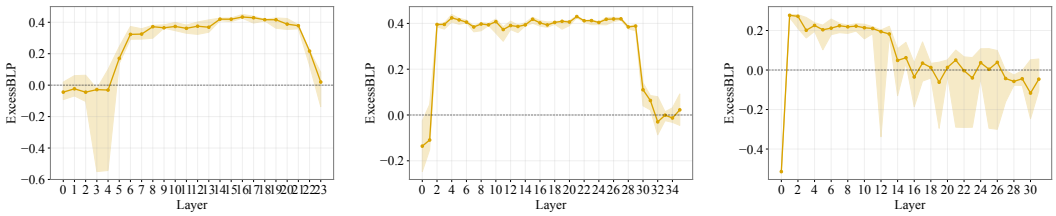
$$d_{t+1} = d_{t+1}^{(v_t)}. \tag{38}$$

F ABLATIONS OF REPRESENTATION PROBING FOR WORD DECODER

In Section 3.2, we use a linear probe to discover the word partition and a linear partition-guided decoder to produce the final word-decoder probabilities. Here we examine nonlinear (MLP) representation probing and MLP-based decoder training as alternatives to the linear design used in the main method.

The motivation for considering nonlinear probes is that some auxiliary representations show little linear partitionability at certain layers. By applying nonlinear probes to the same contextual word representations, we find that some of these layers still support strong balanced partitions. For example, Fig. 4 shows that the nonlinear probe obtains high ExcessBLP across a broad middle-layer range for Pythia-410M, whereas Fig. 1 shows that the linear probe has ExcessBLP close to zero across many of those same layers. Similarly, for Qwen2.5-3B, several middle layers have weak linear ExcessBLP in Fig. 1 but strong nonlinear ExcessBLP in Fig. 4. This suggests that different layers may favor different word-level partition structures.

We select the best nonlinear-probe layer for each auxiliary model and repeat the partition-assigned decoder training procedure. Specifically, we use nonlinear layer 1 for Mistral-7B, nonlinear layer 16 for Pythia-410M, and nonlinear layer 21 for Qwen2.5-3B. We also compare two decoder variants: linear-probe labels with a linear decoder and nonlinear-probe labels with an MLP decoder. Note that the MLP decoder and the nonlinear probe use the same one-hidden-layer architecture.



(a) Pythia-410M (b) Qwen2.5-3B (c) Mistral-7B

Figure 4: Layer-wise ExcessBLP for nonlinear probes. The solid line reports the median across independent probe initializations, and the shaded region shows the minimum-to-maximum range. Nonlinear probes reveal additional partitionable structure in some layers where the linear probe has low ExcessBLP.

We evaluate word decoders using weighted assignment accuracy (WAA) and cross-entropy (CE):

$$\mathrm{WAA} = \mathbb{E}_c \left[\sum_{w \in \mathcal{W}_{\mathcal{A}}(c)} p_{\mathcal{A}}(w \mid c)\, \mathbf{1} \left[\arg\max_{b \in \{0,1\}} D_{\theta_d}(b \mid c, w) = y(c, w) \right] \right], \tag{39}$$

$$\mathrm{CE} = -\mathbb{E}_c \left[\sum_{w \in \mathcal{W}_{\mathcal{A}}(c)} p_{\mathcal{A}}(w \mid c)\, \log D_{\theta_d}(y(c, w) \mid c, w) \right], \tag{40}$$

where $y(c, w)$ is the probe-assigned label and the expectation is over test prefixes. WAA measures agreement with these labels, while CE measures the probability assigned to them. Probe CE replaces D_{θ_d} with the selected probe's probabilities $\rho_{\ell^\star, r^\star}$ in Eq. (40). In Table 5, we observe that MLP decoders achieve higher WAA and lower CE than linear decoders, indicating closer agreement with the probe-assigned labels and higher confidence in those labels. However, these results alone do not establish better watermarking performance because the resulting word-level bit distributions must also be predictable by the generator-specific token encoders.

Table 5: Partition-assigned word decoder on test set. Decoder cells report WAA (↑) / CE (↓).

Aux. LM	Partition	Probe CE	Linear decoder	MLP decoder
Pythia-410M	Linear L23	0.380	0.985 / 0.067	0.999 / 0.015
	Nonlinear L16	0.597	0.886 / 0.294	0.990 / 0.042
Qwen2.5-3B	Linear L33	0.174	0.970 / 0.094	0.996 / 0.020
	Nonlinear L21	0.200	0.973 / 0.078	0.993 / 0.034
Mistral-7B	Linear L14	0.641	0.994 / 0.054	0.999 / 0.013
	Nonlinear L1	0.648	0.866 / 0.312	0.997 / 0.018

We next train token encoders for these decoder variants. Table 6 reports WAA and CE separately for word-start and word-continuation steps. Here, WAA measures agreement between the token encoder's predicted bit and the most probable bit in the word-decoder-induced target distribution, weighted by the generator's candidate-token probabilities. CE compares the predicted bit probabilities with this target distribution. Together, these metrics evaluate how well the token encoder approximates the word-level bit mapping.

We further evaluate the carrier bit accuracy in real generation and detection pipeline across different auxliary LLM and decoder variants for various generators. Table 7 and Table 8 shows that the preferred partition–decoder design for each generator depends on the auxiliary LLM. Qwen2.5-3B achieves similar performance on both linear decoder and MLP decoder. In contrast, Pythia-410M and Mistral-7B perform obviously worse with linear decoder. Therefore, in our main experimentsm we select linear decoder from Qwen2.5-3B, and MLP decoder from Pythia-410M and Mistral-7B.

Table 6: Token encoder training performance for decoder variants using Qwen2.5-14B-Instruct as the generator. Start and continuation columns report WAA (↑) / CE (↓) on the test set. Each auxiliary model is evaluated with two decoder variants: linear partition with linear decoder and nonlinear partition with MLP decoder.

Auxiliary LM	Decoder variant	Start encoder	Continuation encoder
Pythia-410M	Linear partition + linear decoder	0.951 / 0.262	0.946 / 0.254
	Nonlinear partition + MLP decoder	0.969 / 0.199	0.975 / 0.170
Qwen2.5-3B	Linear partition + linear decoder	0.965 / 0.223	0.979 / 0.170
	Nonlinear partition + MLP decoder	0.977 / 0.163	0.987 / 0.129
Mistral-7B	Linear partition + linear decoder	0.967 / 0.202	0.974 / 0.179
	Nonlinear partition + MLP decoder	0.960 / 0.163	0.958 / 0.172

Table 7: Carrier bit accuracy (CBA) of linear and nonlinear decoder variants across generators and auxiliary language models on OpenWebText prompts. Each entry reports the CBA of the 'linear decoder / MLP decoder'.

	Auxiliary LM		
Generator	Pythia-410M	Qwen2.5-3B	Mistral-7B
Gemma2-9B	0.795 / 0.976	0.970 / 0.980	0.920 / 0.975
Llama2-7B	0.784 / 0.955	0.945 / 0.954	0.897 / 0.947
Mixtral-8×7B	0.812 / 0.921	0.947 / 0.949	0.907 / 0.942
Qwen2.5-14B-Instruct	0.825 / 0.959	0.953 / 0.958	0.958 / 0.959

Table 8: Carrier bit accuracy (CBA) of linear and nonlinear decoder variants across generators and auxiliary language models on the Alpaca dataset. Each entry reports the CBA of 'linear decoder / MLP decoder'.

	Auxiliary LM		
Generator	Pythia-410M	Qwen2.5-3B	Mistral-7B
Qwen2.5-14B-Instruct	0.801 / 0.949	0.941 / 0.952	0.868 / 0.944

G WordMark Sampling Details

Algorithm 1 implements the sampling rules described in Section 4.3. Token grouping and carrier-state updates follow Appendix E.

For a free word-start position, WordMark samples the first token from the original word-start distribution and assigns its predicted bit as the target bit. For a forced word-start position, it determines the target bit using Eq. (8) and applies the hard support rule defined in Eqs. (11)–(12). Specifically, the restricted distribution is used only when the target-supporting mass is at least τ_0; otherwise, sampling falls back to the original word-start distribution.

For a word continuation, WordMark retains the active carrier word's target bit and samples from the softly penalized distribution in Eqs. (13)–(14). Unlike word starts, continuation sampling does not require a minimum target-supporting mass.

Algorithm 1 WordMark Generation with Indexed BCH Codewords

Input: Prompt, payload $\mathbf{p}$, masking key κ, generator G, token encoder $E_{\theta_e^G}$, syndrome size K, warmup length W, thresholds τ_0, τ_1, penalty λ, BCH encoder
Output: Generated text x

Initialize the output, carrier state, warmup counter, and codeword index $u = 1$
Construct $\mathbf{c}_u$ using Eq. (6)
while generation continues **do**
  Obtain the truncated and normalized distribution $p_G(\cdot \mid c_t)$
  Classify candidate tokens using the carrier transitions in Appendix E
  Sample group g with probability M_t^g from Eq. (9)
  if warmup is active or $g = \mathrm{non}$ **then**
    Sample $v \sim \pi_t(\cdot \mid g)$
  else if $g = \mathrm{start}$ and the position is free **then**
    Sample $v \sim \pi_t(\cdot \mid \mathrm{start})$
    Assign $b_t^\star \leftarrow \arg\max_{b \in \{0,1\}} E_{\theta_e^G}(b \mid c_t, v)$
  else if $g = \mathrm{start}$ **then**
    Set $b_t^\star$ using Eq. (8)
    Construct $\mathcal{V}_{t,\mathrm{hard}}^\star$ and compute $\Pi_t^{\mathrm{start}}(b_t^\star)$ using Eqs. (11) and (12)
    If $\Pi_t^{\mathrm{start}}(b_t^\star) \geq \tau_0$, sample from $\pi_t(\cdot \mid \mathrm{start})$ restricted to $\mathcal{V}_{t,\mathrm{hard}}^\star$; otherwise sample from $\pi_t(\cdot \mid \mathrm{start})$
  else
    Use the active carrier word's target bit $b_t^\star$
    Construct $\mathcal{V}_{t,\mathrm{good}}^\star$ and $\tilde{\pi}_t(\cdot \mid \mathrm{cont})$ using Eqs. (13) and (14)
    Sample $v \sim \tilde{\pi}_t(\cdot \mid \mathrm{cont})$
  end if
  Append v and apply its carrier-state transition
  Update the warmup counter and syndrome position from completed eligible carrier words
  if all n syndrome blocks of $\mathbf{c}_u$ have been completed **then**
    $u \leftarrow u + 1$
    Construct $\mathbf{c}_u$ using Eq. (6)
  end if
end while
return x

H Payload Recovery and Detection Details

Algorithm 2 implements Section 4.4. Let N_{car} denote the number of carrier words after warmup. The detector forms

$$N_{\mathrm{seq}} = \left\lfloor \frac{N_{\mathrm{car}}}{nK} \right\rfloor$$

complete extracted codeword sequences. Any trailing carrier bits outside these sequences are discarded.

Algorithm 2 WordMark Payload Recovery and Watermark Detection with Indexed BCH Codewords

Input: Received text x, word decoder D_{θ_d}, masking key κ, syndrome size K, warmup length W, BCH encoder and decoder, codeword length n, correction capability e_{BCH}, detection threshold γ

Output: Watermark decision $\mathrm{is_wm}$, payload estimate $\hat{\mathbf{p}}$, score S

Identify eligible carrier words and skip the first W
Decode the remaining carrier bits using Eq. (15)
XOR each consecutive group of K bits using Eq. (16)
Form N_{seq} complete length-n sequences $\hat{\mathbf{c}}_1, \ldots, \hat{\mathbf{c}}_{N_{\mathrm{seq}}}$; discard trailing bits
if $N_{\mathrm{seq}} = 0$ **then**
    return $\mathrm{false}, \perp, 0$
end if
Initialize an empty list $\mathcal{P}$
for $u = 1, \ldots, N_{\mathrm{seq}}$ **do**
    $\mathbf{z}_u \leftarrow \mathrm{PRF}_{\kappa}(u)$
    $\mathbf{m}_u \leftarrow \mathrm{BCH}_{\mathrm{Decode}}(\hat{\mathbf{c}}_u)$
    if $\mathbf{m}_u \neq \perp$ **then**
        if $d_{\mathrm{H}}(\hat{\mathbf{c}}_u, \mathrm{BCH}_{\mathrm{Encode}}(\mathbf{m}_u)) \leq e_{\mathrm{BCH}}$ **then**
            Append $\mathbf{m}_u \oplus \mathbf{z}_u$ to $\mathcal{P}$
        end if
    end if
end for
if $\mathcal{P}$ is empty **then**
    return $\mathrm{false}, \perp, 0$
end if
Select the most frequent payload in $\mathcal{P}$ as $\hat{\mathbf{p}}$, resolving ties deterministically
for $u = 1, \ldots, N_{\mathrm{seq}}$ **do**
    $\mathbf{c}_u^{\star} \leftarrow \mathrm{BCH}_{\mathrm{Encode}}(\hat{\mathbf{p}} \oplus \mathbf{z}_u)$
end for
Compute S using Eq. (17)
$\mathrm{is_wm} \leftarrow [S \geq \gamma]$
return $\mathrm{is_wm}, \hat{\mathbf{p}}, S$

Decoding failures and payload selection. Each extracted sequence $\hat{\mathbf{c}}_u$ is independently BCH-decoded. If decoding succeeds within the correction radius e_{BCH}, the detector removes the index-specific mask to recover a payload candidate:

$$\tilde{\mathbf{p}}_u = \mathrm{BCH}_{\mathrm{Decode}}(\hat{\mathbf{c}}_u) \oplus \mathbf{z}_u. \tag{41}$$

Failed decoding attempts are excluded from voting. The detector selects the most frequently recovered payload as $\hat{\mathbf{p}}$.

Verification. Verification compares the original extracted sequences, before BCH correction, with the expected codewords for $\hat{\mathbf{p}}$. The denominator in Eq. (17) includes all N_{seq} complete sequences, including those that failed decoding. If $N_{\mathrm{seq}} = 0$ or no valid payload candidate exists, the detector returns $(\mathrm{false}, \perp, 0)$. Otherwise, it returns the payload estimate and score together with the watermark decision.

I Implementation Details

I.1 Training Details

We train the word decoder and the generator-specific token encoders on OpenWebText. The training split contains 30,000 examples and the held-out split contains 5,000 examples. Each example consists of a prefix and a set of plausible next-word candidates.

The word decoder is auxiliary-model specific. For each auxiliary LM, we train a separate decoder, and then share it across all generators evaluated with that auxiliary LM. The linear decoder is a

single affine scalar head, while the MLP variant uses a LayerNorm–Linear(512)–GELU–Linear scalar head. Decoder training uses AdamW. In partition-guided decoder training, we use learning rate 10^{-3}, batch size 16, and 5 epochs with a detector context length of 512 tokens.

The token encoder is trained separately for each auxiliary-model–generator pair, because it depends on both the auxiliary decoder labels and the generator's tokenization and hidden states. We train separate token encoders for word-start and word-continuation decisions. For each auxiliary-model–generator pair, the token encoder uses generator features from three relative depths: one layer near 25% of the depth, one near 50%, and one near 90%. These prefix features are combined with the frozen generator input embedding of the candidate token. Each feature source is first mapped to a shared 512-dimensional representation; the three prefix features are then fused using candidate-conditioned modulation and attention, and the resulting representation is used to predict a binary distribution. Token-encoder training uses AdamW with learning rate 10^{-3}, batch size 1024, and 5 epochs. The generator-side context length used when materializing training features is 512 tokens.

For instruction-tuned generators, we still use the model's native chat template. Since OpenWebText is a continuation-style corpus rather than an instruction dataset, we wrap the OpenWebText prefix as the assistant prefix under a small bank of continuation-style system/user prompts, e.g., "You are a helpful assistant" / "Continue the visible assistant text naturally." The auxiliary word decoder still only observes the text prefix without the system prompt. During evaluation, for instruction datasets such as Alpaca, we instead render the dataset instruction and input as the user message under the same model chat template.

I.2 Baseline Configurations

For token-level baselines, we follow the official implementations when available and preserve the recommended watermarking hyperparameters whenever possible. For BREW, we use BCH(15,5) with correction radius $e_{\mathrm{BCH}} = 3$. Each repeated block carries a 5-bit payload masked by a keyed block-specific mask. One generated token carries one BCH coded bit, giving 15 tokens per block. We use soft greenlist watermarking with logit bias $\delta = 6$.

For MPAC, as in its original setting, each token carries one 2-bit symbol. To match the controlled 5-bit payload setting, we combine MPAC with the same BREW-style BCH(15,5) repeated-block layer: each 15-bit BCH codeword is padded with one ignored bit to form 16 bits. The main MPAC setting uses the recommended bias $\delta = 2$.

For Qu et al., our controlled comparison uses a payload-matched RS(3,1,1) code over GF(2^5). A 5-bit masked payload symbol is encoded into three GF(32) symbols. We use $\delta = 6$.

I.3 Metrics Definition

Payload recovery accuracy (PRA) and payload bit accuracy (PBA) are computed only for examples that contain at least one complete payload-coding block. The definition of a complete block depends on the watermarking method. For WordMark without BREW, a BCH$(15, 5)$ codeword contains 15 coded bits. Because each syndrome bit is encoded using a group of K carrier words, one complete codeword requires $15K$ carrier words. For token-level BREW, each token carries one bit, so one BCH$(15, 5)$ codeword requires 15 tokens. For WordMark+BREW, the same BCH codeword is repeatedly embedded, with one complete block formed every $15K$ carrier words. For MPAC+BREW, one complete block consists of 8 tokens encoding 16 packed bits, including 15 BCH-coded bits. For Qu+BREW, one complete block consists of 30 tokens encoding one RS$(3, 1, 1)$ codeword. Examples that stop before completing the method-specific block are excluded from PRA and PBA. PRA counts an example as recovered only when the decoded payload is exactly equal to the embedded payload. PBA is the bit accuracy between the decoded and embedded payloads.

For WordMark, carrier-bit accuracy (CBA) is computed over all committed carrier words. If a word falls back to natural sampling but is still identified as the detector-visible valid carrier under the runtime carrier lifecycle, it consumes one carrier position and is included in CBA. Thus, CBA is measured on the actual carrier stream agreed between generation and detection.

Syndrome-bit accuracy (SBA) is computed over complete syndrome blocks of size K. For each complete group, the recovered syndrome bit is the XOR of its K carrier bits. When $K = 1$, SBA is identical to CBA.

AUC and TPR@1%FPR are computed from per-example detection scores, and they are computed only with at least one complete block under the corresponding method.

J Per-Dataset Embedding and Decoding Performance

In Table 9, we report CBA and PRA separately for OpenWebText, BillSum, Wikipedia, and CC-News, complementing the OBWC averages in Table 1.

Table 9: Carrier-bit accuracy across datasets for selected generator–auxiliary-model pairs. The results in Table 1 are obtained by macro-averaging these four datasets.

Auxiliary LM	Generator	11 bits				19 bits			
		OpenWebText	BillSum	Wikipedia	CC-News	OpenWebText	BillSum	Wikipedia	CC-News
Pythia-410M	Gemma2-9B	0.9761	0.9648	0.9592	0.9709	0.9757	0.9691	0.9669	0.9735
	Llama2-7B	0.9554	0.9472	0.9465	0.9568	0.9561	0.9452	0.9404	0.9583
	Mixtral-8×7B	0.9213	0.9249	0.9206	0.9353	0.9333	0.9425	0.9334	0.9459
	Qwen2.5-14B-Instruct	0.9551	0.9386	0.9412	0.9513	0.9581	0.9462	0.9413	0.9516
Qwen2.5-3B	Gemma2-9B	0.9702	0.9713	0.9601	0.9648	0.9679	0.9714	0.9594	0.9649
	Llama2-7B	0.9451	0.9332	0.9285	0.9387	0.9432	0.9346	0.9291	0.9413
	Mixtral-8×7B	0.9472	0.9259	0.9293	0.9457	0.9472	0.9454	0.9461	0.9462
	Qwen2.5-14B-Instruct	0.9530	0.9482	0.9491	0.9558	0.9608	0.9511	0.9503	0.9576
Mistral-7B	Gemma2-9B	0.9751	0.9622	0.9473	0.9652	0.9745	0.9628	0.9550	0.9642
	Llama2-7B	0.9468	0.9321	0.9329	0.9478	0.9479	0.9365	0.9334	0.9468
	Mixtral-8×7B	0.9421	0.9218	0.9254	0.9416	0.9443	0.9417	0.9448	0.9472
	Qwen2.5-14B-Instruct	0.9442	0.9251	0.9287	0.9370	0.9446	0.9301	0.9318	0.9432