Publicly Detectable Watermarking for Language Models*

Jaiden Fairoze[1], Sanjam Garg[1], Somesh Jha[2],
Saeed Mahloujifar[3], Mohammad Mahmoody[4], and Mingyuan Wang[1]

[1]{fairoze,sanjamg,mingyuan}@berkeley.edu University of California, Berkeley
[2]jha@cs.wisc.edu University of Wisconsin, Madison
[3]saeedm@meta.com FAIR, Meta
[4]mohammad@virginia.edu University of Virginia

October 26, 2023

Abstract

We construct the first provable watermarking scheme for language models with public detectability or verifiability: we use a private key for watermarking and a public key for watermark detection. Our protocol is the first watermarking scheme that does not embed a statistical signal in generated text. Rather, we directly embed a publicly-verifiable cryptographic signature using a form of rejection sampling. We show that our construction meets strong formal security guarantees and preserves many desirable properties found in schemes in the private-key watermarking setting. In particular, our watermarking scheme retains distortion-freeness and model agnosticity. We implement our scheme and make empirical measurements over open models in the 7B parameter range. Our experiments suggest that our watermarking scheme meets our formal claims while preserving text quality.

1 Introduction

Generative AI (GenAI) technologies, such as large language models (LLMs) and diffusion models, have some impressive capabilities. These capabilities include in-context learning, code completion, text-to-image generation, and document and code chat. However, GenAI technologies are also being used for nefarious purposes (e.g., generating fake tweets, generating attacks, and harmful prose). To protect against such use cases, a large body of work has focused on detecting AI-generated content [4, 9, 10, 12, 21, 23, 34]. The problem is as follows: given content c, is c generated by some specific GenAI technique (e.g. GPT-4, Bard, or Stable Diffusion [25, 29, 30])? In other words, we want a "GenAI Turing Test."

At present, the main approach when trying to detect AI-generated text is to train yet another AI model to perform the detection [10, 12, 23, 34]. This method makes a critical assumption: that AI-generated text has features embedded within it that AI can identify. The key problem with this assumption is that generative models are explicitly designed to produce realistic content that is hard to distinguish from natural (generated by human or nature) content. As a result, any "black-box" detection scheme would suffer from high false positive and/or false negative rates as generative models become more realistic. Available detectors such as GPTZero [10] have no guarantee of correctness—for example, the authors state explicitly that detection results from their tool should not be used to punish students.

To circumvent this fundamental issue, a recent line of work [1, 7, 18, 20] has taken a different approach. By embedding a watermark into generated content at generation-time, they are able to get stronger detection criteria. Watermarking techniques alter the generation process to embed some "secret" in the generated

*Jaiden Fairoze, Sanjam Garg, and Mingyuan Wang are supported in part by DARPA under Agreement No. HR00112020026, AFOSR Award FA9550-19-1-0200, NSF CNS Award 1936826, and research grants by Visa Inc, BAIR Commons Meta Fund, Stellar Development Foundation, and a Bakar Fellows Spark Award. Somesh Jha is supported by Air Force Grant FA9550-18-1-0166, the National Science Foundation (NSF) Grants CCF-FMitF-1836978, IIS-2008559, SaTC-Frontiers-1804648, CCF-2046710, CCF-1652140, and 2039445, and ARO grant number W911NF-17-1-0405, and DARPA-GARD problem under agreement number 885000. Mohammad Mahmoody was supported by NSF grants CCF-1910681 and CNS1936799.

content. Such a secret is embedded into the content in a way that one who knows the secret can detect it, so long as the generated content is not altered significantly. In particular, the cryptographic approach of Christ et al. [7] achieves formal notions of completeness (any watermarked text will be detected), soundness (one cannot watermark a text without knowing the secret), and distortion-freeness (watermarking does not change the output distribution).

In summary, these watermarking techniques embed a "signal" in the output of the generative model, where the signal depends on a secret key. Note that detecting whether a piece of text is watermarked does not require access to the model. Once a text t is received, one can check whether it was watermarked or not using the secret key. Thus, the secret key is used for both embedding and detecting the watermark.

The aforementioned watermarking approaches have one problem in common: the generative model and the detector need to share a secret key. That is, the detector cannot detect AI-generated content unless it knows what secret was embedded in the content. This is acceptable in scenarios where the entity trying to detect the watermark is the same as the entity trying to generate the content. For example, an entity that provides a chat API may be able to provide a detection API as well. However, such a setup is not ideal due to the following reasons: 1. Privacy: The entity who wants to check the integrity of the content might not be willing to share it with the detector. 2. Efficiency: Providing a detection API is costly. 3. Conflict of interest: The entity providing the detection API might not be trusted in certain cases. For instance, consider a case where the entity is accused of generating a certain inappropriate text and is brought to a court of law. It is not reasonable to ask the same entity to tell if the text passes the watermark detection.

One solution could be sharing the secret with the world so everyone can run the detection. However, this raises another important challenge: anyone can now embed the secret to any content, AI-generated or not. This would not be acceptable because the watermarking is subject to "availability attacks." An attacker can create masses of watermarked content that is not generated by AI to question the dependability of the watermarking detection. It also defeats one of the main purposes of watermarking, in which an entity might want to use the watermark as a signature for their content. This would be useful when the generated content needs to come with proof that it is generated by a credible source. Such signatures are also useful to refute any accusations about a generated content; an entity would not be accountable for a content if it does not have their signature (watermark).

In this paper, we aim to solve these problems. We specifically focus on LLMs and the text modality. Our main question is:

Is it possible to construct a publicly-detectable watermarking scheme?

A publicly-detectable scheme would provably resolve this trust issue—users would be able cryptographically verify that a watermark is or is not present. Further, they would know that the only entity capable of embedding a watermark is the model provider. We state the full properties needed for public detectability as follows:

1. **Security of introducing watermark:** To guarantee a user is convinced that a watermark is detected, the watermarking scheme must be cryptographic in the sense that false positives are "essentially impossible" in a rigorous mathematical sense as opposed to empirical validation—it should be impossible for any probabilistic polynomial-time (PPT) adversary to attack the scheme.

2. **Resistance to simple edits (robustness):** It is likely that text obtained from language models is modified—to some extent—before publication. The watermark detector should be robust in the sense that it can still detect a watermark even if the original LM output has changed, but the text semantics are preserved.

3. **Distortion-freeness:** The watermarking scheme should not degrade the quality of the LM output. This is known as distortion-freeness.

4. **Model agnostic:** The watermarking scheme should be agnostic to the specific auto-regressive model and its configuration (e.g., the specific decoder algorithm, model parameters, etc). The watermarking scheme should use the auto-regressive model as a black box.

5. **Publicly detectable:** The detector should be able to determine if a candidate text is watermarked without access to the model weights or secret material pertaining to the watermarking scheme.

1.1 Paper Outline

In Section 2, we define our security model, assumptions, and related work. In Section 3, we define the necessary background: the auto-regressive model interface, public key signatures, and the random oracle model. In Section 4, we present our formal definitions. We describe our base construction and show how to extend it to our full protocol. We show that our protocol meets all definitions. In Section 5, we describe the implementations of our scheme and the Christ et al. [7] scheme. We empirically evaluate both schemes. Finally, in Section 6, we discuss related ideas and future work.

2 Security Model

2.1 Assumptions

We assume that any contiguous block of ℓ tokens contains at least α bits of min-entropy, i.e., no particular sample is more than $2^{-\alpha}$ likely to happen.[1] This assumption allows us to capture security properties and present our protocol cleanly. Recall that entropy is necessary: one can only hope to embed a watermark in text generated with high entropy. Prior work showed that entropy can be handled dynamically [7]—this optimization partially applies to our protocol (see Section 4.2). However, we present the protocol without the optimization for clarity and brevity. In addition, ℓ effectively serves as a parameter to tune the trade-off between robustness and distortion-freeness. Higher ℓ values lead to more distortion-free text at the cost of robustness and vice versa.

Assumption 2.1. *For any prompt $\boldsymbol{\rho}$ and tokens $\boldsymbol{t}$, the new tokens $\boldsymbol{t}' \leftarrow \mathsf{GenModel}_\ell(\boldsymbol{\rho}, \boldsymbol{t}) \in \mathcal{T}^\ell$ were sampled from distributions with min-entropy at least α.*

2.2 Entities

In our model, there are three distinct entities.

Model provider The model provider provides the LM service: given a prompt, it returns the LM output for that prompt for a given LM configuration. An honest model provider will run the watermarking protocol at text generation time. This entity has white-box access to the model weights in addition to any secret material specific to the watermarking protocol, e.g., a secret watermarking key.

Detector The detector computes if a candidate text is watermarked with respect to a specific model provider. It performs this job without access to the model weights or the secret watermarking key. In particular, it only has access to the public detection key.

User Users generate prompts which are sent to the model provider in exchange for the model output. Users may test text for the presence of a watermark by sending candidate text to the detector. The user should be convinced that the watermark is or is not present, i.e., the detector must provide a "proof of watermark" or "non-watermark" that can be verified without model weights or secret material pertaining to the watermarking protocol.

2.3 Related Work

Text distinguishers We discuss key approaches for detecting AI-generated text without introducing any changes to text generation. See [16] for a comprehensive survey.

Early approaches to detecting AI-generated text revolve around looking for features of AI-generated text that are not present in human-generated text—if you can or cannot identify such features, you can conclude the text was or was not AI-generated. Examples of features include relative entropy scoring [21], perplexity [4], and other statistical signals [9]. We refer the reader to [4] for a survey.

[1] Formally, the min-entropy $H_\infty(\mathcal{D})$ of a distribution D is defined as $-\log\left(\max_{\omega \in \mathsf{Supp}(\mathcal{D})} \Pr[D = \omega]\right)$.

Another common method is to train another model to automatically identify distinguishing features. Research of this nature [10, 12, 23, 34] uses deep learning as a binary classifer.

The problem with this idea is that it relies on AI-generated text being fundamentally different from human-generated text. This reliance is at odds with the core goal of LMs: to produce human-like text. As models get better, statistical features of AI-generated text will decay. In particular, GPT-4 [25] and other cutting edge models are quickly closing this gap. Chakraborty et al. [6] formally show that as AI-generated text approches human quality, text distinguishers demand longer text samples.

Beyond relying on an diminishing assumption, text distinguishers lack formal guarantees—the detector's correctness is only empirically validated, so any validation performed is only relevant to the exact model, its configuration, and prompting context during experimentation.

Other work has shown that it is possible to train models to transform text such that it fools text distinguishers [19, 31].

Watermarking schemes There is a recent line of work using ML to perform watermarking [2, 22, 24, 27, 33]. Notably, Liu et al. [22] address the same problem as this paper: their approach is to train two models—one for embedding a signal and one for detecting it. This is analogous to using asymmetric keys. Crucially, all schemes in this category are entirely empirical and have no formal guarantees such as correctness, soundness, or distortion-freeness.

Recently, Kirchenbauer et al. [18] gave the first watermarking scheme with formal guarantees. They showed that when model entropy is high, a watermark can be planted by hashing previous tokens to embed a watermark signal in the next token. Crucially, hashing tokens to zero or one effectively assigns a binary label to potential next tokens. By ensuring that only tokens with label "zero" appear in generated text, the watermark can be detected after text generation by recomputing the hash. Kirchenbauer et al. [18] bound the distortions introduced by the watermark by measuring perplexity: the difference between the distribution produced by the plain model and the one produced by the model with watermarking.

The Gumbel softmax scheme of Aaronson [1] is another approach to LM watermarking. The scheme uses exponential minimum sampling to sample from the model using randomness based on previous tokens (via hashing). This scheme is distortion-free so long as no two output texts that share a common substring are public [7]. This is unlikely for a widely-used LM.

Kuditipudi et al. [20] design a family of watermarking schemes that aim to maximize robustness. The main idea of their scheme is to use a key that is as large as the generated text output—this permits statistical distortion-freeness as opposed to the cryptographic distortion-freeness of this paper and Christ et al. [7] at the cost of computation that scales with the generated text output. The long key is then "aligned" with the generated text by computing an alignment cost—this alignment cost can be (re)computed at detection time with the detection key and the generated text. Text that has been watermarked will be statistically likely to have low alignment cost at detection time. Their scheme has the desirable property that the alignment cost is a measure of edit distance and thus a watermark may persist even if text is inserted or deleted from the original watermarked text. Their scheme is not provably complete or sound.

Linguistic steganography Linguistic steganography generalizes LM watermarking. The main goal is to embed a hidden message in natural language text. A steganographic protocol provides formal security if an adversary cannot determine whether a message is from the original distribution or the distorted distribution that embeds a hidden message [15]. The key difference in this setting compared to watermarking is that distortions to the distribution are permitted so long as some notion of semantic similarity is preserved. Furthermore, there are critical differences in the problem model between linguistic steganography and LM watermarking. In LM watermarking, prompts are adversarially chosen and the watermarking protocol should be agnostic to the plain text distribution. The focus of linguistic steganography is to achieve undetectability. In LM watermarking, undetectability is not important—what is important is that the text is of similar (ideally, the same) quality as unwatermarked text, i.e., it should be distortion-free. That is, watermarked text should still be usable for the same downstream tasks for which unwatermarked text is useful.

3 Background

3.1 Preliminaries

Let $a \parallel b$ denote the tail-to-head concatenation of a to b. We use $\log(\cdot)$ to take logarithms base 2. Let ϵ denote the empty list or empty string. Let a_i denote the i-th bit of vector $\mathbf{a}$. We use Python slicing notation throughout: $\mathbf{a}[-i]$ refers to the i-th last element of a list and $\mathbf{a}[j:k]$ extracts the elements a_i for $i \in [j,k)$. We use $\mathcal{U}$ to represent the uniform distribution and $\xleftarrow{\$}$ to denote a random sample, e.g., $r \xleftarrow{\$} \mathcal{U}$.

For the cryptographic primitives in this paper, we use λ for the security parameter. A negligible function $\mathsf{negl}(\lambda)$ in λ are those functions that decay faster than the inverse of any polynomials. That is, for all $\mathsf{poly}(\lambda)$, it holds that $\mathsf{negl}(\lambda) < \mathsf{poly}(\lambda)$ for all large enough λ. Cryptographic primitives typically require its security to be $\mathsf{negl}(\lambda)$.

3.2 Language Models

Definition 3.1 (Auto-regressive Model)**.** An auto-regressive model Model over token vocabulary $\mathcal{T}$ is a deterministic algorithm that takes as input a prompt $\boldsymbol{\rho} \in \mathcal{T}^*$ and tokens previously output by the model $\boldsymbol{t} \in \mathcal{T}^*$ and outputs a probability distribution $p = \mathsf{Model}(\boldsymbol{\rho}, \boldsymbol{t})$ over $\mathcal{T}$.

$\mathsf{GenModel}$ wraps around Model to implement a generative model. $\mathsf{GenModel}$ iteratively generates n tokens. Decode is the specific decoding method, e.g., `argmax`.

Algorithm 1 Generative Model

```
function GenModel_n(ρ ∈ T*, t ∈ T*)
    for i = 1, ..., n do
        t ← t ∥ Decode(Model(ρ, t))
    return t
```

3.3 Public-key Signatures

Our scheme uses a public-key signature scheme with the following properties.

Definition 3.2 (Public-key Signature Scheme)**.** A publicly-key signature scheme S is a tuple of algorithms $\mathsf{S} = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Verify})$ where:

- $\mathsf{Gen}(1^\lambda) \to (\mathsf{sk}, \mathsf{pk})$ outputs a public/secret key pair $(\mathsf{sk}, \mathsf{pk})$ with respect to the security parameter λ.
- $\mathsf{Sign}_{\mathsf{sk}}(m) \to \sigma$ produces a signature σ, given a message m, using the secret signing key sk.
- $\mathsf{Verify}_{\mathsf{pk}}(m, \sigma) \to \{\texttt{true}, \texttt{false}\}$ outputs `true` or `false`, given a candidate message m and signature σ, using the public verification key.

Definition 3.3 (Unforgeability)**.** For every adversary $\mathcal{A}$, we have

$$\Pr_{\substack{(\mathsf{pk},\mathsf{sk}) \leftarrow \mathsf{Gen}(1^\lambda) \\ (m^*, \sigma^*) \leftarrow \mathcal{A}^{\mathsf{Sign}_{\mathsf{sk}}(\cdot)}(\mathsf{pk})}} \left[\mathsf{Verify}_{\mathsf{pk}}(m^*, \sigma^*) = \texttt{true}\right] \leq \mathsf{negl}(\lambda).$$

Here, the adversary gets oracle access to the signing oracle $\mathsf{Sign}_{\mathsf{sk}}(\cdot)$, but the final forgery (m^*, σ^*) it outputs must be some message m^* that it never queries the signing oracle on.

As a signature scheme, we require the above unforgeability property. Looking ahead, this property guarantees that it is hard to forge a watermark. Note that we require a rather weak unforgeability here that $\mathcal{A}$ is not given access to the signing oracle, which suffices for our purpose.

Next, in order to preserve the distortion-freeness, we require our signature scheme to have a pseudorandom signature. In particular, for any list of messages $\{m_1, m_2, \ldots\}$, it should be hard for the adversary to distinguish whether a given string is the valid signature or a truly random string without the public verification key. As a prominent example, the well-known Boneh-Lynn-Shacham (BLS) signature scheme satisfies this property.

Definition 3.4 (Pseudorandom Signature)**.** For any list of messages $\{m_1, m_2, \ldots\}$, it holds that

$$\left| \begin{array}{l} \Pr_{\substack{(\mathsf{sk},\,\mathsf{pk}) \leftarrow \mathsf{Gen}(1^\lambda) \\ \sigma_i \leftarrow \mathsf{Sign}_{\mathsf{sk}}(m_i)}} [\mathcal{A}(m_1, m_2, \ldots, \sigma_1, \sigma_2, \ldots) = 1] \\ \quad - \Pr_{\sigma_i \xleftarrow{\$} \mathcal{U}} [\mathcal{A}(m_1, m_2, \ldots, \sigma_1, \sigma_2, \ldots) = 1] \end{array} \right| \leq \mathsf{negl}(\lambda).$$

3.4 Random Oracle Model

Our construction uses random oracle $\mathcal{O}$ to model a cryptographic hash function H. A random oracle is a random function drawn uniformly randomly from the set of all possible functions (over specific input and output domains). Random oracle models are commonly used in cryptographic construction [3]. Constructions that are provably secure in the random oracle model are heuristically assumed to be also secure when one instantiates the random oracle $\mathcal{O}$ with a cryptographic hash function H. In the rest of the paper, we use $\mathcal{O}$ and H interchangeably.

4 Protocol

In this section, we present our key gadget and show how it can be optimized further.

4.1 Technical Overview

Let $\mathbf{t} := t_1, t_2, \ldots, t_\ell$ be bit samples from probability distributions $p_1, p_2, \ldots, p_\ell$ where each p_i is a probability distribution from an auto-regressive model. By Assumption 2.1, we know that $\sum_{i=1}^{\ell} -\ln p_i(t_i) \geq \alpha$ for some reasonably large α. That is, the ℓ tokens were sampled from distributions with at least α cumulative bits of entropy. Let $\mathbf{t}$ denote the first ℓ tokens sampled from the model (denote $\mathbf{t}$ as the message) and let $\boldsymbol{\sigma} := \mathsf{Sign}_{\mathsf{sk}}(\mathcal{O}(\mathbf{t}))$. We can embed the λ_σ-bit signature $\boldsymbol{\sigma} = \sigma_1, \sigma_2, \ldots, \sigma_{\lambda_\sigma}$ in a contiguous sequence of tokens from the auto-regressive model as follows: for each of the next $\ell \cdot \lambda_\sigma$ tokens sampled from the model, ensure that the i-th block of ℓ tokens hashes to the corresponding i-th bit in $\boldsymbol{\sigma}$, i.e., $\mathcal{O}(t_{i+1}, t_{i+2}, \ldots, t_{i+\ell}) = \sigma_i$ for $i \in [\lambda_\sigma]$. After this process, a complete message-signature pair is embedded into a contiguous sequence of generated tokens.

To detect the presence of a watermark, the detector needs to recover the message-signature pair. The detector first recovers the message $\mathbf{t}$ by looking at the first ℓ tokens. Next, the detector recovers each bit of the signature by computing $\sigma_i = \mathcal{O}(t_{i+1}, t_{i+2}, \ldots, t_{i+\ell})$ for $i \in \lambda_\sigma$ and let $\boldsymbol{\sigma} = (\sigma_1, \ldots, \sigma_{\lambda_\sigma})$. It can then verify the signature by computing $\mathsf{Verify}_{\mathsf{pk}}(\mathcal{O}(\mathbf{t}), \boldsymbol{\sigma})$ using the public verification key: if the signature verifies, it must be the case that the text was watermarked.

4.2 Optimizations

The technical overview above presents our key gadget. There are a number of optimizations one can add to the protocol to make well-defined improvements with no significant cost. We enumerate them as follows:

Arbitrary length output In the technical overview, the detector crucially relies on knowing the precise position in the text where the message and signature is embedded. We can relax this by searching over all windows of length ℓ for the message in linear time—it must be that the corresponding signature will be embedded in the next $\ell \cdot \lambda_\sigma$ tokens after the ℓ message tokens. Furthermore, in order to generate n tokens, we tile multiple message-signature pairs until enough tokens are generated. If n is not divisible by $\ell + \ell \cdot \lambda_\sigma$, we generate the remaining tokens using the native model sampler.

Embedding multiple watermarked blocks The scheme described above embeds one signature in a fixed number of output bits. To extend this scheme to support arbitrarily large output, we can tile the block structure defined above sequentially until the desired length is reached.

When n is large enough to permit multiple message-signature pairs, we can leverage the pseudorandomness of the signature to use significantly fewer tokens. Specifically, to embed k message-signature segments,

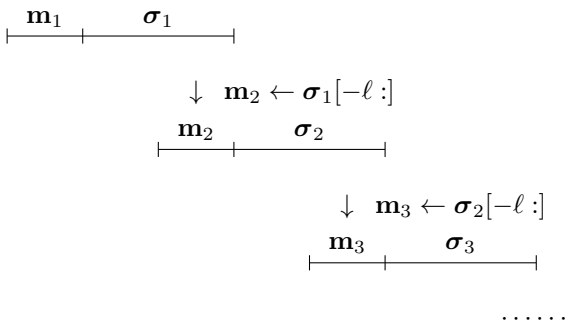

Figure 1: Tiling structure to compress multiple message-signature pairs. This is possible due to the pseudorandomness property of the signature.

we only need $k \cdot (\ell + \ell \cdot \lambda_\sigma) - (k-1) \cdot \ell$ tokens given $\lambda_\sigma \geq \ell$. Note that in practice, we set $\lambda_\sigma = 768$ (inherited from BLS signatures [5]) and $\ell \simeq 10$. The idea is to use the last ℓ bits of the signature from the previous message-signature pair as the message for the next segment. This preserves full distortion-freeness due to the pseudorandomness of the signature scheme. See Figure 1 for a visual description of this process.

Tuning the robustness vs. distortion-freeness trade-off We assume that each block of ℓ tokens has sufficient entropy (as defined by the parameter α) to ensure our formal notion of distortion-freeness is met. However, in practice, one can set ℓ to a concrete value (e.g., determined empirically for a specific model and hyperparameters) to tweak robustness—ℓ should be set as low as possible such that the text quality remains similar[2] to non-watermarked text in order to get more robustness. When ℓ is low, each message-signature segment requires fewer total tokens, meaning more segments can be embedded in n tokens. So long as one message-signature pair remains after text edits, the watermark is detectable.

Chaining embedded bits The plain gadget embeds one bit of the signature into a block of ℓ tokens. Observe that the scheme is susceptible to the following attack. If an adversary knows that a specific portion of text corresponds to a message-signature pair, she can construct a different message that still verifies. By definition of the random oracle, any freshly sampled token has $1/2$ probability of hashing to 0 or 1. The adversary replaces a signature $\boldsymbol{\sigma} \in \mathcal{T}^{\ell \cdot \lambda_\sigma}$ with a new message $\boldsymbol{\sigma}' \in \mathcal{T}^{\ell \cdot \lambda_\sigma}$ by changing ℓ tokens at a time and checking that the new block still hashes to the same value as the old block. If the hash is inconsistent, sample a new block of ℓ tokens and try again. This process can be repeated for all λ_σ bits of the signature. In addition, since the blocks are independent, the adversary can replace each block with adversarial text in a modular fashion.

We make this attack more difficult by introducing dependencies across adjacent blocks. For each $i \in \lambda_\sigma$ success condition for rejection hashing changes from

$$H\left(s_{i,1}^{\sigma_{i,1}} \cdots s_{i,\ell}^{\sigma_{i,\ell}}\right) = \sigma_i$$

to

$$H\left(\bigoplus_{j=1}^{i} s_{j,1}^{\sigma_{j,1}} \cdots s_{j,\ell}^{\sigma_{j,\ell}}\right) = \sigma_i$$

where $\bigoplus_{j=1}^{i} v_j$ stands for concatenation as $\bigoplus_{j=1}^{i} v_j := v_1 \parallel v_2 \parallel \ldots \parallel v_i$. Now, for the adversary to perform the same signature replacement attack, she can no longer change each ℓ-length token block independently.

[2]Our Theorem 4.1 proves that this similarity is closely related to how much entropy each ℓ tokens carry.

Dynamic entropy Through this paper, we assume that there is sufficient entropy in every ℓ tokens sampled from the LM in order to simplify presentation and analysis. We can relax this assumption in the real world—rather than setting a global length ℓ that is expected to be of sufficient entropy, we can empirically measure how much entropy is available from the LM at generation time. This optimization transforms the symmetric scheme of Christ et al. [7] into a substring-complete version, i.e., a version that embeds independent detectable segments to gain robustness.

Given a distribution $p \leftarrow \mathsf{Model}(\cdot,\cdot)$, one can measure the entropy of sampling token t from p as $-\log(p(t))$. Thus, each time a new token is sampled, the generation algorithm can keep track of how much collective entropy has been seen up to that point, i.e., cumulative entropy can be measured as $\sum_{i=1}^{j} -\log(p_i(t_i))$ for a contiguous block of j tokens—this sum can be updated incrementally as more tokens are sampled. To use this optimization, the generation algorithm simply waits for sufficiently many token samples such that the cumulative entropy sum is large enough. Once the threshold is reached, all sampled tokens are taken as the message, and the protocol proceeds as before[3]. At detection time, the message is no longer of fixed size, so it no longer suffices to iterate all windows of fixed length in the text. Thus, the detector will incur quadratic performance costs by searching over all possible message strings from the input text.

4.3 Definitions

In this section, we formally define our publicly detectable watermarking scheme, which should satisfy (1) completeness, (2) soundness, (3) distortion-freeness, and (4) robustness.

Definition 4.1 (Publicly-Detectable Watermarking Scheme)**.** A publicly-detectable watermarking scheme PDWS for an auto-regressive model Model over token vocabulary $\mathcal{T}$ is a tuple of algorithms $\mathsf{PDWS} = (\mathsf{Setup}, \mathsf{Watermark}, \mathsf{Detect})$ where:

- $\mathsf{Setup}(1^\lambda) \to (\mathsf{sk}, \mathsf{pk})$ outputs a public key pair $(\mathsf{sk}, \mathsf{pk})$ with respect to the security parameter λ.
- $\mathsf{Watermark}_{\mathsf{sk}}(\boldsymbol{\rho}) \xrightarrow{\$} \boldsymbol{t}$ produces response text $\boldsymbol{t} \in \mathcal{T}^*$ given a prompt $\boldsymbol{\rho} \in \mathcal{T}^*$ using the secret key sk.
- $\mathsf{Detect}_{\mathsf{pk}}(\boldsymbol{t}^*) \to \{\mathtt{true}, \mathtt{false}\}$ outputs $\mathtt{true}$ or $\mathtt{false}$ given a candidate watermarked text $\boldsymbol{t}^*$.

A PDWS scheme must meet the following security definitions.

Definition 4.2 (Completeness)**.** A PDWS is δ-complete if for every prompt ρ and token sequence $\boldsymbol{t} \in \mathcal{T}^*$ of length $|\boldsymbol{t}| \geq \delta$, it holds that

$$\Pr_{\substack{(\mathsf{sk},\,\mathsf{pk}) \leftarrow \mathsf{Setup}(1^\lambda) \\ \boldsymbol{t} \leftarrow \mathsf{Watermark}_{\mathsf{sk}}(\boldsymbol{\rho})}} [\mathsf{Detect}_{\mathsf{pk}}(\boldsymbol{t}) = \mathtt{false}] \leq \mathsf{negl}(\lambda).$$

δ-Completeness ensures that text of sufficient length that was watermarked with the honest protocol results in non-detection with only negligible probability.

Definition 4.3 (Soundness)**.** A PDWS is k-sound if any adversary $\mathcal{A}$ cannot generate a watermarked text given the public detection key and any polynomial number of genuinely-watermarked texts. Formally,

$$\Pr_{\substack{(\mathsf{sk},\,\mathsf{pk}) \leftarrow \mathsf{Setup}(1^\lambda) \\ \boldsymbol{t}^* \leftarrow \mathcal{A}^{\mathsf{Watermark}_{\mathsf{sk}}(\cdot)}(\mathsf{pk})}} \begin{bmatrix} \mathsf{Detect}_{\mathsf{pk}}(\boldsymbol{t}^*) = \mathtt{true} \\ \wedge\ \mathsf{nonoverlapping}(\boldsymbol{t}^*, \boldsymbol{t}_1, \boldsymbol{t}_2, \ldots) = \mathtt{true} \end{bmatrix} \leq \mathsf{negl}(\lambda).$$

Here, the adversary is given oracle access to the watermarked model, namely $\mathsf{Watermark}_{\mathsf{sk}}(\cdot)$, and we use $\boldsymbol{t}_1, \boldsymbol{t}_2, \ldots$ to denote the watermarked text that the adversary receives as output when she queries the model $\mathsf{Watermark}_{\mathsf{sk}}(\cdot)$. Furthermore, the predicate $\mathsf{nonoverlapping}_k(\boldsymbol{t}^*, \boldsymbol{t}_1, \boldsymbol{t}_2, \ldots)$ outputs $\mathtt{true}$ if $\boldsymbol{t}^*$ does not share a k-length window of tokens with any of the genuinely-watermarked texts $\boldsymbol{t}_1, \boldsymbol{t}_2, \ldots$ and outputs $\mathtt{false}$ otherwise.

[3]Note that this optimization can only apply to the message portion of the embedding—applying it to the signature component would result in an exponential blowup of the detector's runtime.

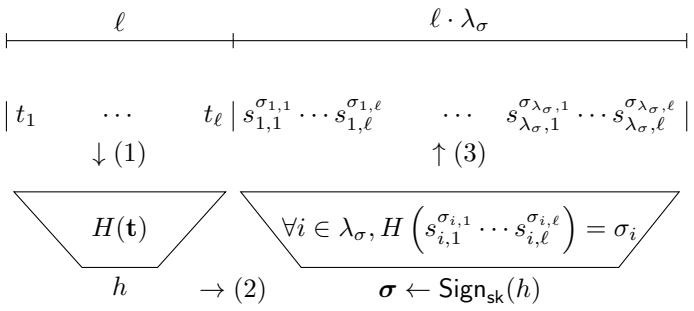

Figure 2: Our core gadget. This diagram presents the generation procedure—to perform detection, the same steps are performed on input tokens. Our watermarking scheme relies on planting $\ell + \ell \cdot \lambda_\sigma$ token segments with specific structures. There are three key steps to embedding the gadget. First (1), ℓ tokens are sampled natively from the LM. These tokens $\boldsymbol{t}$ are hashed to produce $h \leftarrow H(\boldsymbol{t})$. Second (2), the hash is signed using the secret signing key, producing the signature $\boldsymbol{\sigma} \leftarrow \mathsf{Sign}_{\mathsf{sk}}(h)$. Lastly (3), each bit of σ_i the signature is embedded into the next ℓ tokens by rejection sampling. That is, the i-th block of ℓ tokens are sampled such that the hash of the block yields the i-th bit of the signature, i.e., $\forall i \in \lambda_\sigma, H\left(s_{i,1}^{\sigma_{i,1}} \cdots s_{i,\ell}^{\sigma_{i,\ell}}\right) = \sigma_i$ where each s is one token.

Intuitively, our soundness definition requires the following. If the adversary manages to output a text $\boldsymbol{t}^*$ that is labeled as watermarked, it must be the case that she copied a long enough sequence of tokens from the genuinely-watermarked texts she receives from the watermarked model.

Definition 4.4 (Distortion-freeness)**.** A PDWS is (computationally) ε-distortion-free if for all polynomial-time distinguishers D, we have

$$\left|\Pr\left[D^{\mathsf{Model},\mathsf{GenModel}}(1^\lambda) \to 1\right] - \Pr_{(\mathsf{sk},\mathsf{pk}) \leftarrow \mathsf{Setup}(1^\lambda)}\left[D^{\mathsf{Model},\mathsf{Watermark}_{\mathsf{sk},\mathsf{pk}}}(1^\lambda) \to 1\right]\right| \leq \varepsilon.$$

Intuitively, distortion-freeness ensures that the watermarking algorithm does not noticeably change the quality of the model output, i.e., without the secret watermarking key, no PPT machine can distinguish plain LM output from watermarked LM output.

Definition 4.5 (Robustness)**.** A publicly-detectable watermarking scheme is δ-robust if for every prompt $\boldsymbol{\rho}$, security parameter λ,

$$\Pr_{\substack{(\mathsf{sk},\mathsf{pk}) \leftarrow \mathsf{Setup}(1^\lambda) \\ \boldsymbol{t} \leftarrow \mathsf{Watermark}_{\mathsf{sk}}(\boldsymbol{\rho})}}[\mathsf{Detect}_{\mathsf{pk}}(\mathcal{A}(\boldsymbol{t})) = \mathtt{false}] \leq \mathsf{negl}(\lambda)$$

where the adversary is allowed to transform the text $\boldsymbol{t}$ however she pleases so long as a δ-length segment that does verify remains.

4.4 Construction

Our main watermarking algorithm is presented in Algorithm 2. The core idea is to embed a message and a corresponding publicly-verifiable signature in the generated text. This message-signature pair should be extractable during detection. Once extracted, it can be verified using the public key.

To explain our scheme, we describe how to embed one message-signature pair in LM output—the construction can be applied repeatedly to generate arbitrarily long LM output (i.e., Line 3 in Algorithm 2). The construction is presented visually in Figure 2.

The first step is to natively sample a fixed number of tokens such that the entropy used at generation time to produce those tokens is sufficient for watermarking. This is captured in line 4. By Assumption 2.1, we know that ℓ tokens were sampled from distributions with at least α bits of entropy. Denote these ℓ tokens

Algorithm 2 Private Watermarking of Generative Models

1: **function** $\mathsf{EmbedAsymmetric}_{\mathsf{sk},n,\ell,\lambda_\sigma}(\boldsymbol{\rho} \in \mathcal{T}^*, \boldsymbol{t} \in \mathcal{T}^*)$:
2: $\quad \boldsymbol{t} \leftarrow \epsilon$
3: $\quad$ **while** $|\boldsymbol{t}| + \ell + \lambda_\sigma < n$ **do**
4: $\quad\quad \boldsymbol{t} \leftarrow \boldsymbol{t} \parallel \mathsf{GenModel}_\ell(\boldsymbol{\rho}, \boldsymbol{t})$
5: $\quad\quad \boldsymbol{\sigma} \leftarrow \mathsf{Sign}_{\mathsf{sk}}(H(\boldsymbol{t}[-\ell:]))$
6: $\quad\quad \boldsymbol{\sigma}_{\mathsf{prev}} \leftarrow \epsilon$
7: $\quad\quad \mathbf{m} \leftarrow \epsilon$
8: $\quad\quad$ **while** $\boldsymbol{\sigma} \neq \epsilon$ **do**
9: $\quad\quad\quad \sigma, \boldsymbol{\sigma} \leftarrow \boldsymbol{\sigma}[0], \boldsymbol{\sigma}[1:]$
10: $\quad\quad\quad \mathbf{x} \leftarrow \mathsf{GenModel}_\ell(\boldsymbol{\rho}, \boldsymbol{t})$
11: $\quad\quad\quad$ **while** $H(\mathbf{m} \parallel \mathbf{x} \parallel \boldsymbol{\sigma}_{\mathsf{prev}})) \neq \sigma$ **do**
12: $\quad\quad\quad\quad \mathbf{x} \leftarrow \mathsf{GenModel}_\ell(\boldsymbol{\rho}, \boldsymbol{t})$
13: $\quad\quad\quad \mathbf{m} \leftarrow \mathbf{m} \parallel \mathbf{x}$
14: $\quad\quad\quad \boldsymbol{t} \leftarrow \boldsymbol{t} \parallel \mathbf{x}$
15: $\quad\quad\quad \boldsymbol{\sigma}_{\mathsf{prev}} \leftarrow \boldsymbol{\sigma}_{\mathsf{prev}} \parallel \sigma$
16: $\quad$ **if** $|\boldsymbol{t}| < n$ **then**
17: $\quad\quad \boldsymbol{t} \leftarrow \boldsymbol{t} \parallel \mathsf{GenModel}_{n-|\boldsymbol{t}|}(\boldsymbol{\rho}, \boldsymbol{t})$
$\quad$ **return** $\boldsymbol{t}$

as the message $\boldsymbol{t}$. Once $\boldsymbol{t}$ has been sampled, it is hashed and signed (line 5). Crucially, the signature itself must be pseudorandom, so the signature scheme must be chosen to support this (See Section 3.3).

Once the signature is computed, the next step is to embed it into natural language. The key idea is to embed each bit of the signature into a block of tokens such that the block of tokens hashes to the signature bit. In lines 8 to 12, we sample ℓ more tokens using the native LM decoder and check if the hash matches the next signature bit. Note that the hash depends on all previous inputs to hashes for the current signature. If we have a match, we accept the token block and move on to the next bit of the signature. Otherwise, reject the tokens and freshly sample a new block of length ℓ. At the end of the rejection sampling process, the signature will be embedded in $\ell \cdot \lambda_\sigma$ tokens where λ_σ is the length of the signature—one message-signature pair is embedded in generated text. This process can be repeated to embed multiple pairs for added resilience.

Algorithm 3 Public Watermark Detector

1: **function** $\mathsf{Detect}_{\mathsf{pk},n,\ell,\lambda_\sigma}(\boldsymbol{t}' \in \mathcal{T}^*)$
2: $\quad$ **for** $i \in [n - (\ell + \lambda_\sigma)]$ **do**
3: $\quad\quad \boldsymbol{t} \leftarrow H(\boldsymbol{t}'[i : i + \ell])$
4: $\quad\quad \boldsymbol{\sigma} \leftarrow \epsilon$
5: $\quad\quad \mathbf{m} \leftarrow \epsilon$
6: $\quad\quad$ **for** $j \in [\lambda_\sigma]$ **do**
7: $\quad\quad\quad \mathbf{m} \leftarrow \mathbf{m} \parallel \boldsymbol{t}'[j \cdot (i + \ell) : j \cdot (i + \ell) + \lambda_\sigma]$
8: $\quad\quad\quad \boldsymbol{\sigma} \leftarrow \boldsymbol{\sigma} \parallel H(\mathbf{m} \parallel \boldsymbol{\sigma}))$
9: $\quad\quad$ **if** $\mathsf{Verify}_{\mathsf{pk}}(\mathbf{m}, \boldsymbol{\sigma}) =$ `true` **then return** `true`
$\quad$ **return** `false`

To detect if a watermark is present in candidate text, it suffices to extract one message-signature pair and verify it using the public key. In line 2, we iterate over all potential token blocks of length ℓ (adjusting by λ_σ indices to account for the signature). Once the message $\mathbf{m}$ is assigned, the signature is iteratively reconstructed in lines 6 to 8. If the signature verifies, we know with an overwhelming probability that the text was watermarked (See Theorem 4.2). Otherwise, move on to the next candidate block and try again. If no message-signature pair is verified, we conclude that the text was not watermarked (See Theorem 4.3)

4.5 Proofs

Assume that each block of ℓ tokens has at least α bits of entropy. We model the cryptographic hash $H(\cdot)$ as a random oracle, denoted $\mathcal{O}(\cdot)$. Without loss of generality, the proof sketches below are with respect to a single embedded message-signature pair, as presented in Figure 2.

Theorem 4.1 (Distortion-freeness)**.** *Let H be a random oracle and* Sig *be a secure signature scheme with pseudorandom signatures, then $\mathcal{PDWS}$ is a computationally distortion-free publicly detectable watermarking scheme.*

Proof. The only difference between our sampling algorithm and the original sampling algorithm is the following. The original sampling algorithm samples the next ℓ tokens directly from some distribution $\mathcal{D}$. Our sampling algorithm first samples a bit b and then samples the next batch of ℓ tokens according to $\mathcal{D}$, but conditioned on that its hash is consistent with b. We just need to prove that these two sampling process is computationally indistinguishable.

By the pseudorandomness of the signature scheme, the bit b is computationally indistinguishable from a truly random bit. Therefore, it suffices to prove that $\mathcal{D}$ is close to first sample a truly random bit b and then sample from $\mathcal{D}$ conditioned on the hash being b. By our assumption, $\mathcal{D}$ contains at least α bits of min-entropy. Therefore the probability

$$\Pr_H\left[\left|\Pr_D[H(D)=1]-1/2\right| \geq O(2^{-\alpha/2})\right] \leq 2^{-\alpha}.$$

This can be proven by any standard tail bounds, e.g., Hoeffding's inequality [13]. In other words, for any distribution D, over the choice of the random oracle H, the probability mass (according to $\mathcal{D}$) of samples with hash value 1 will be exponentially close to $1/2$. This shows that our sampling process is computationally indistinguishable from the original sampling process; hence, our scheme is computationally ε-distortion-free, where $\varepsilon = \exp(-\alpha)$. □

Theorem 4.2 (Completeness)**.** *$\mathcal{PDWS}$ is a ℓ-complete publicly detectable watermarking scheme.*

Proof. By Assumption 2.1, any ℓ consecutive tokens contains α bits of entropy. For any long enough output $\boldsymbol{t}$, there is enough entropy to embed a message/signature pair. It is easy to see that if the watermarking scheme successfully embeds in a message/signature pair, the detection algorithm will mark the text as "watermarked". The only possibility that the watermarking fails is if the rejection sampling algorithm fails to find the next batch of tokens whose hash is consistent with the target bit. By our analysis of the distortion-freeness, for each sampling of the next batch of tokens, its hash value will be uniformly random. Consequently, each sampling attempt will succeed in finding a consistent hash with probability $1/2$. After λ attempts, our rejection sampling will find the next batch of tokens with probability $1 - 2^{-\lambda}$. □

Theorem 4.3 (Soundness)**.** *$\mathcal{PDWS}$ is a complete publicly detectable watermarking scheme.*

Proof. The soundness of our watermarking scheme is based on the unforgeability of the signature scheme by a simple reduction.

If there exists a PPT adversary that can find a text labeled as watermarked, it must mean that this watermarked text has a valid message/signature pair embedded inside. Then, one may extract this pair, which constitutes a forgery attack against the underlying signature scheme. □

Theorem 4.4 (Robustness)**.** *$\mathcal{PDWS}$ is an ℓ-robust publicly detectable watermarking scheme.*

Proof. The robustness of our scheme is rather easy to see. Let $\boldsymbol{t}$ be the output of the LLM. If the adversary's output $\mathcal{A}(\boldsymbol{t})$ contains 2δ consecutive tokens from the original $\boldsymbol{t}$, it must mean that there is a δ consecutive tokens, which embeds a message/signature pair, is preserved in $\mathcal{A}(\boldsymbol{t})$. The detection algorithm will recover this consecutive sequence by an exhaustive search, resulting in a successful detection output. □

5 Evaluation

Prior work [18, 20] often evaluated correctness or robustness of the watermarking detection algorithm through implementations. Such analysis is not necessary for this work and the Christ et al. [7] scheme—the level of robustness, distortion-freeness, and detectability is immediate from formal proofs. What is not immediate is the level of real-world performance and utility—we use benchmarks to evaluate the practicality of using the proposed watermarking schemes. That is, we quantify the delay introduced at generation time and detection time, compare generated text examples, and measure empirical entropy from open models. Entropy measurements are particularly important for our scheme and that of [7] since security increases with higher entropy.

Following prior work with implementations [18, 20], we use samples from the news-like subset of the C4 dateset [28] as prompts for all evaluation. We use the 1.3B parameter OPT model [35] for performance analysis and the 7B parameter Llama 2 model [32] for text quality analysis.

We implement our asymmetric protocol in addition to the base (non-substring-complete) version of the Christ et al. [7] symmetric protocol. For all experiments, we embed a single message-signature pair into the generated text. Our implementation is entirely written in Python 3 with PyTorch [26] and wraps around the Hugging Face `transformers` interface for transformer models [8].

5.1 On Distortion-Freeness

#	Prompt	Plain (tokens)	Plain (bits)	Symmetric	Asymmetric
1	...may never know, but they will certainly know how much I loved and enjoyed this year.""\nBoyd will play Shallowater in the Region I-3A semifinals Friday at 6 p.m. Friday at Lubbock Cooper High School.	\nBROWNWOOD — The Vernon Lady Lions' Tigers bowling team lost its first game of the season, but prevailed 1,521-1486 to pick up its first win of the year.\nVernon's Ella Bland finished low series at 3...	</s><s>Visit Mexico's beautiful Yucatán Península\nWritten by Carol DuBos on March 19, 2015</s><s>Heritage, Ginseng, Art and Theaters\n Don't miss the Selliott African American Heritage Museum\nin Wi...	\nCollinsville (13-3) — BAdgett 13, W. Watson 3, Ben. Watson 2, Selby 12, McCalip 2.\nKills — Collinsville 38 (Selby 12), Windthorst 37 (Kirk 23).\nECU 14-13 Bradley 13-13 Chico 15-15 Collinsville 15-...	\nWith Vols win at Texas A&M, Coleman leads SEC since start of '18\nKNOXVILLE — Who made the most sense to chase the school scoring record?\nIf karma's a thing, then Tennessee forward Grant Williams p...
2	...you will be cited for the violation.""\nThe Click It or Ticket effort was made possible with federal highway safety funds administered by the Illinois Department of Transportation, the release stated.	More information on Click It or Ticket is available by calling 1-800-492-9243 or visiting www.illinoisclickittocli- ckit.org, the release stated.\nThe St. Charles Police Department is focusing on the ov...	\nLocally, more the 31 lives were lost last year as a result of impaired driving, Majewski says. This fall, DuPage County recorded its highest number of fatal crashes in the last decade.\nAs part of t...	\nFor more information about the Click It or Ticket/Drive Sober or Get Pulled Over campaign, visit IDOT's campaign website, cite.it/enforcement, or www.trafficsafety.illinois- .gov/ buzz/stcharles.</s><s...	The nationwide campaign is coordinated annually by the National Highway Traffic Safety Administration with the work group summited this week. \nFor more information about the statewide enforcement eff...

Table 1: Example text completions based on Llama 2-7B [32]. The columns, from left to right, denote: the prompt, completion from the plain model, completion from the plain model over a bit-level vocabulary, completion with an embedded symmetric watermark [7], and completion with an embedded asymmetric watermark (this work). Prompts were selected randomly from the news-like portion of the C4 dataset [28]. Cell entries are truncated to 200 characters—the truncation point is denoted by ellipses. All text completions were generated with multinomial sampling for 1930 tokens. In the symmetric case, the security parameter λ was set to 16. In the asymmetric case, the token segment length ℓ was set to 10 and the collision parameter b was set to 4. Note Plain (bits) in Row 1 exhibits examples of degenerate output in the "</s>" tags and the discussion of "Yucatán Peninsula," which is off-topic.

Example text completions Section 5.1 presents a random selection of prompts from the C4 dataset and both unwatermarked and watermarked text completions. We remark that these prompts are likely not representative of common prompts, nor are the generations optimized. For example, using a better sampling

method such as nucleus sampling [14] instead of multinomial sampling would likely yield better quality output. The completions were generated ceteris paribus and we compare the text quality of each output with respect to the others.

We observe portions of degenerate output across all forms of text generation, most likely due to naively sampling from the model without any additional optimization. Whereas we sample directly from the model in our implementation, real-world implementations (e.g. Hugging Face `transformers`) use heuristics such as automatic stopping, advanced decoding, and beam search to maximize text quality.

Empirical segment size and hash size trade-off Aggressively setting ℓ to low values (e.g., near 1) leads to text distortions. The root problem is that even if most segments of length ℓ are of high entropy, a single low entropy section leads to either distorted text for that segment, or stalls text generation completely when a completion that satisfies the hash criteria cannot be found. Kaptchuk et al. [17] provide an illustrative example of such a low entropy scenario: given the inputs "The largest carnivore of the Cretaceous period was the Tyrannosaurus," the next token is almost certainly going to be "Rex." More specifically, we use a second parameter b, denoted the collision parameter, to control how many bits of the signature to hash per ℓ-length token segment: higher b packs more bits of the signature into the same amount of text at the expense of more rejections (in expectation) during rejection sampling. There is clear degradation when the ratio $\frac{\ell}{b}$ is small. If b is high, the watermarking protocol tries to embed *more* bits of the signature into a single segment—if the segment does not have enough entropy to support this, distortions are induced. Similarly, when the number of bits compared in hashes is large, it quickly becomes unlikely that there is a high-quality sequence of tokens that satisfies the hash. We found that $\ell = 10$ and $b = 4$ produced good text quality without increasing the generated text length by a large factor.

Bit-level reduction introduces distortions In order to implement the bit-level watermarking scheme of Christ et al. [7], we additionally implemented the arbitrary-to-binary vocabulary reduction as specified in their paper. We noticed that our straightforward implementation of the reduction distorts text even before any watermarking is applied. This is visible in Section 5.1 by comparing the Plain (tokens) and Plain (bits) columns. Especially for long text completions (e.g., above 1,000 tokens), multinomial sampling over bits often loses the initial context. We hypothesize that this degradation is due to imprecise (floating point) arithmetic losing granularity when probabilities are very close to 0 or 1—a common situation especially as model size increases. It may be possible to overcome this degradation by trading off computation time for higher precision arithmetic. Moreover, a native implementation of the reduction within an ML framework (e.g., PyTorch [26]) would likely help.

5.2 Computational Costs

Text generation Plain text generation without any watermarking or bit reduction is, as expected, the fastest—we use this setting as our control against which we compare the performance of the watermarking schemes.

Observe that the time it takes to generate text over bits is nearly the same as the time it takes to generate text with the symmetric watermarking scheme. This tells us that the dominating cost in the implementation of the Christ et al. [7] protocol is the reduction to a binary vocabulary from an arbitrary vocabulary. For reference, OPT-1.3B's tokenizer's vocabulary has 50,272 tokens.

While generation times for the plain, plain with bits, and symmetric settings are stable for repeated samples, the runtimes for our asymmetric generation algorithm exhibit high variance. This is due to the probabilistic construction of asymmetric generation: we either accept or reject each sampled token based on whether it hashes to the desired signature bits. The acceptance criteria has two key components: b, the number of comparison bits in each hash, and λ, the min-entropy of an ℓ-length token segment. High runtimes surface when the algorithm encounters a low-entropy section of sampling or a missed hash, or in the worst case, both. On the contrary, if entropy remains relatively high throughout sampling or we have a high collision rate for hashes, then performance is at parity or better than that of the bit-level generation schemes. We believe this problem can be ameliorated by applying the optimizations discussed in Section 6.2.

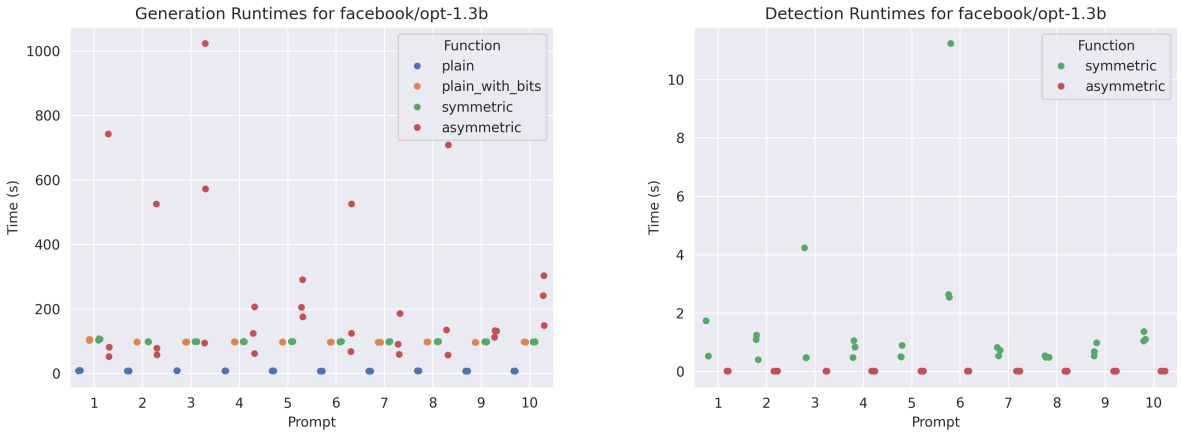

Figure 3: Generation and detection runtimes for each generation method over the OPT-1.3B model [35]. All generations used multinomial sampling to generate 816 tokens for 10 random prompts from the C4 dataset [28] with 3 runs per prompt.

Watermark detection For each of the prompts, we run the respective detection algorithms on the watermarked texts. Note that detecting an asymmetric watermark takes linear time in the generated text length n. Detecting a symmetric watermark, on the other hand, can take longer as it runs in quadratic time based on the generated text length n. Empirical results for the detection time of watermarked texts are shown in Figure 3 and support the aforementioned performance expectations. We see that detection times for the symmetric protocol are varied—they depend on how quickly the algorithm detects λ bits of entropy to begin the watermarking process. If most of the text forms the hash input, the symmetric detection algorithm will use significant computation time iterating over earlier text segments. On the other hand, asymmetric detection times are tightly clustered: extracting the message-signature gadget from text takes exactly one pass over the input, resulting in consistent detection times. For a text length of 816 tokens, asymmetric detection consistently runs near 10ms, whereas symmetric mostly varies between 0.4s and 2s.

5.3 Entropy Measurements

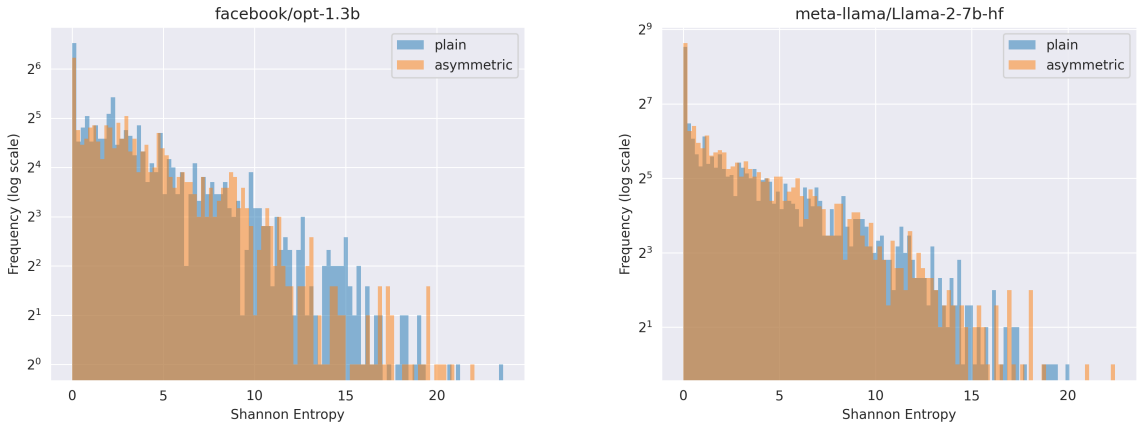

Figure 4: Entropy histograms from the OPT-1.3B model [35] and Llama 2-7B model [32] respectively with all else equal. These figures assert that entropy is distributed exponentially—there are many cases where tokens have little-to-no entropy, posing a challenge for watermarking schemes.

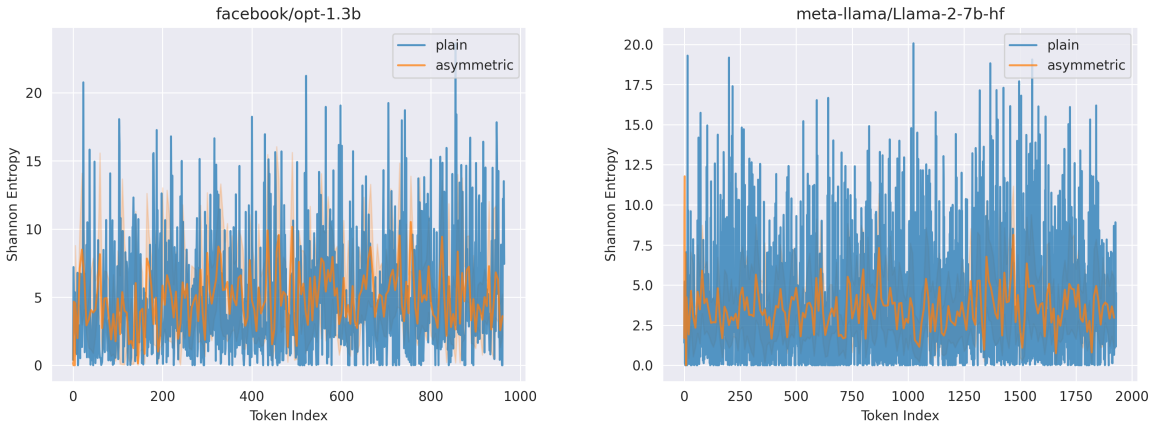

Figure 5: Entropy for each token over time from the OPT-1.3B model [35] and Llama 2-7B model [32] respectively with all else equal. For both models, the asymmetric sampling scheme results in lower overall entropy than the plain scheme, suggesting the rejection sampling discards many higher entropy tokens.

Any watermarking scheme fundamentally demands high entropy token distributions. In particular, we make empirical measurements of Shannon entropy to better constrain what ℓ values are reasonable in practice. In Figure 4, we present entropy histograms for the OPT-1.3B model and the Llama 2-7B model. For each token t sampled from the model, its empirical Shannon entropy was taken as $-\log(p(t))$ where $p(t)$ denotes the probability of sampling t from distribution p. The histograms in Figure 4 tell us that many tokens are sampled with little-to-no entropy—token entropy is distributed exponentially. This is a challenge to watermarking schemes. The Christ et al. [7] scheme is resilient to this problem by dynamically embedding a watermark such that entropy is taken into account. This approach cannot apply to our scheme as it would result in high (exponential) computation cost to the detector. Instead, we rely on token *segments*, rather than individual tokens, to have enough collective entropy to permit watermarking. We also see that the larger 7 billion parameter model has more entropy: on average, OPT-1.3B yields ~ 3.5 bits of entropy per token and Llama 2-7B yields ~ 3.8 bits of entropy per token. This trend is likely to continue to larger models in the 10 or 100 billion parameter range. Lastly, we see the time-series impact of our watermarking protocol on entropy in Figure 5—the rejection sampling process discards high entropy tokens that do not satisfy the hash condition.

6 Discussion

In summary, we present the first publicly-detectable watermarking scheme using asymmetric keys and proved its security against our definitions. We implement our scheme and the symmetric Christ et al. [7] scheme and make empirical evaluations. We make the following observations which may lend themselves naturally to future work:

6.1 Outsourcing Watermarking Itself

One of the main benefits of a publicly detectable watermarking scheme is that the watermark detector is outsourcable—the entity providing a "detection service" is different from the one providing the model. Besides outsourcing detection, we remark that our protocol also naturally supports outsourcing the watermarking process itself, i.e., an *external entity* can embed a publicly detectable watermark in text generated by a *private model*. In contrast with all known prior watermarking schemes, our protocol does not necessarily need to know the distribution from which to sample each token—it suffices to obtain a list of "next best tokens" given by the prompt and prior generated tokens. This means that our watermarking scheme can operate over API access to private LMs.

Assume that an LM provider supports an API that returns the top ℓ next tokens for a given prompt. The watermarking process itself can be outsourced by replacing line 10 and line 12 with API calls. This has the downside of requiring linearly many requests in the output length—we acknowledge that this is likely a preventative expense for many use cases.[4]

6.2 Performance Optimizations

We remark that a number of performance optimizations can be applied to improve the generation speed of both our scheme and the Christ et al. [7] scheme. For the symmetric case, we refer to the binary reduction discussion in Section 5.1.

For the asymmetric scheme, the main computational bottleneck is rejection sampling. There are two straightforward optimizations to this process that would greatly improve concrete performance. First, rejection sampling naturally lends itself to parallelism. Instead of sequentially searching for a signature collision (on the CPU), this process can be performed in parallel on the GPU to take advantage of (a) faster inference and (b) faster hashing. Second, consider the case where $\ell = 1$ for simplicity. Whenever a token is rejected (i.e., the token hash did not match the signature hash), this information can be used to modify model logits to prevent sampling the same token repeatedly. This has a large impact on the expected running time of rejection sampling since if the token was sampled, it is likely to be sampled again. Preventing this situation drastically improves the time needed to find a matching token. This idea can be generalized to $\ell > 1$, however, care would be necessary to ensure that only the specific token sequence becomes improbable after each rejection.

6.3 Embedding Codewords

Unlike prior watermarking schemes [7, 18, 20], our framework of *embedding extractable bits into the generated text* readily allows for further improvement in robustness.

Instead of embedding a message-signature pair (which contains no redundancy) into the generated text, one may embed an "error-corrected encoding" of the message and signature into the generated text. Given an input text, the detection algorithm will first extract the potentially-erroneous codeword from the text and then apply error correction to recover the original message-signature pair. Apparently, the robustness of this new watermarking scheme will inherit the robustness of the error-correction scheme. For instance, if the error-correcting algorithm allows for 10% of errors, our watermarking scheme will be resilient to 10% of word replacements.

We emphasize that the robustness guarantee provided in this scheme can be *formally proven*, unlike the robustness claims in prior works, which are based on experimental results and heuristics.

One potential barrier to this proposed scheme is efficiency. The efficiency of the scheme depends on the efficiency of the error-correcting encoding. Normally, in the context of text editing, one aims for resilience against insertions and deletions. However, existing state-of-the-art error-correcting codes against insertions and deletions [11] have worse efficiency compared to their counterparts for Hamming error correction. However, we emphasize that our framework is modular. Any improvement in the construction of error-correcting codes will directly give improvement to the efficiency of this proposed scheme.

6.4 Error Correction to Handle Low Entropy

There is another novel way to use error correction to improve our scheme. Recall that the segment length ℓ needs to be sufficiently large such that no segment is of low entropy. We can use error correction to weaken this requirement as follows. First, replace the message and signature with error-corrected versions of the latter. During the text generation process, whenever a low entropy scenario is encountered, we can avoid searching for a low-quality block of tokens that matches by instead using the initial (i.e., "best") token block that does not match. So long as the error correction scheme can tolerate a small number of erroneous tokens, the original signature is still recoverable at detection time. We expect this protocol improvement to significantly reduce the ℓ parameter necessary for practical distortion-freeness.

[4] At the time of writing, OpenAI charges US$0.03 per 1000 input tokens and US$0.06 per 1000 output tokens for GPT-4 API access with 8k token context.

References

[1] Scott Aaronson. Neurocryptography. Invited Plenary Talk at Crypto'2023, 2023.

[2] 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.

[3] Mihir Bellare and Phillip Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In *Proceedings of the 1st ACM Conference on Computer and Communications Security*, pages 62–73, 1993.

[4] Daria Beresneva. Computer-generated text detection using machine learning: A systematic review. In *Natural Language Processing and Information Systems: 21st International Conference on Applications of Natural Language to Information Systems, NLDB 2016, Salford, UK, June 22-24, 2016, Proceedings 21*. Springer, 2016.

[5] Dan Boneh, Ben Lynn, and Hovav Shacham. Short signatures from the weil pairing. In *International conference on the theory and application of cryptology and information security*. Springer, 2001.

[6] Souradip Chakraborty, Amrit Singh Bedi, Sicheng Zhu, Bang An, Dinesh Manocha, and Furong Huang. On the possibilities of ai-generated text detection. *arXiv preprint arXiv:2304.04736*, 2023.

[7] Miranda Christ, Sam Gunn, and Or Zamir. Undetectable watermarks for language models. *arXiv preprint arXiv:2306.09194*, 2023.

[8] Hugging Face. Hugging face transformers. `https://huggingface.co/docs/transformers/index`, 2023. Accessed: 2023-10-08.

[9] Sebastian Gehrmann, Hendrik Strobelt, and Alexander M Rush. Gltr: Statistical detection and visualization of generated text. *arXiv preprint arXiv:1906.04043*, 2019.

[10] GPTZero. Gptzero — the trusted ai detector for chatgpt, gpt-4, & more. `https://gptzero.me/`, 2023. Accessed: 2023-10-05.

[11] Venkatesan Guruswami and Carol Wang. Deletion codes in the high-noise and high-rate regimes. *IEEE Transactions on Information Theory*, 2017.

[12] Jan Hendrik Kirchner, Lama Ahmad, Scott Aaronson, and Jan Leike. New ai classifier for indicating ai-written text. `https://openai.com/blog/new-ai-classifier-for-indicating-ai-written-text`, 2023. Accessed: 2023-10-05.

[13] Wassily Hoeffding. Probability inequalities for sums of bounded random variables. *The collected works of Wassily Hoeffding*, 1994.

[14] Ari Holtzman, Jan Buys, Li Du, Maxwell Forbes, and Yejin Choi. The curious case of neural text degeneration. *arXiv preprint arXiv:1904.09751*, 2019.

[15] Nicholas J Hopper, John Langford, and Luis Von Ahn. Provably secure steganography. In *Advances in Cryptology—CRYPTO 2002: 22nd Annual International Cryptology Conference Santa Barbara, California, USA, August 18–22, 2002 Proceedings 22*, pages 77–92. Springer, 2002.

[16] Ganesh Jawahar, Muhammad Abdul-Mageed, and Laks VS Lakshmanan. Automatic detection of machine generated text: A critical survey. *arXiv preprint arXiv:2011.01314*, 2020.

[17] Gabriel Kaptchuk, Tushar M Jois, Matthew Green, and Aviel D Rubin. Meteor: Cryptographically secure steganography for realistic distributions. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, pages 1529–1548, 2021.

[18] John Kirchenbauer, Jonas Geiping, Yuxin Wen, Jonathan Katz, Ian Miers, and Tom Goldstein. A watermark for large language models. In *Proceedings of the 40th International Conference on Machine Learning*, 2023.

[19] Kalpesh Krishna, Yixiao Song, Marzena Karpinska, John Wieting, and Mohit Iyyer. Paraphrasing evades detectors of ai-generated text, but retrieval is an effective defense. *arXiv preprint arXiv:2303.13408*, 2023.

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

[21] Thomas Lavergne, Tanguy Urvoy, and François Yvon. Detecting fake content with relative entropy scoring. *Pan*, 2008.

[22] Aiwei Liu, Leyi Pan, Xuming Hu, Shu'ang Li, Lijie Wen, Irwin King, and Philip S Yu. A private watermark for large language models. *arXiv preprint arXiv:2307.16230*, 2023.

[23] Eric Mitchell, Yoonho Lee, Alexander Khazatsky, Christopher D Manning, and Chelsea Finn. Detectgpt: Zero-shot machine-generated text detection using probability curvature. *arXiv preprint arXiv:2301.11305*, 2023.

[24] Travis Munyer and Xin Zhong. Deeptextmark: Deep learning based text watermarking for detection of large language model generated text. *arXiv preprint arXiv:2305.05773*, 2023.

[25] OpenAI. Gpt-4 technical report, 2023.

[26] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 2019.

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

[28] 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. *The Journal of Machine Learning Research*, 2020.

[29] Google Research. An overview of bard: an early experiment with generative ai. `https://ai.google/static/documents/google-about-bard.pdf`, 2023. Accessed: 2023-10-11.

[30] Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2022.

[31] Vinu Sankar Sadasivan, Aounon Kumar, Sriram Balasubramanian, Wenxiao Wang, and Soheil Feizi. Can ai-generated text be reliably detected? *arXiv preprint arXiv:2303.11156*, 2023.

[32] 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.

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

[34] Rowan Zellers, Ari Holtzman, Hannah Rashkin, Yonatan Bisk, Ali Farhadi, Franziska Roesner, and Yejin Choi. Defending against neural fake news. *Advances in neural information processing systems*, 2019.

[35] Susan Zhang, Stephen Roller, Naman Goyal, Mikel Artetxe, Moya Chen, Shuohui Chen, Christopher Dewan, Mona Diab, Xian Li, Xi Victoria Lin, et al. Opt: Open pre-trained transformer language models. *arXiv preprint arXiv:2205.01068*, 2022.