Lecture 7: Watermarking for Generative Models

Lecturer: Vinod Vaikuntanathan **Date:** February 26, 2026
Scribes: Arushi Mantri, Lucia Wang

1. Problem definition

The goal of this lecture is to explore methods for detecting AI-generated content; in particular, how to reliably differentiate it from human-generated content.

There are several important motivations for doing this. For one, detection is critical for preventing AI-generated misinformation, as well as addressing issues of copyright infringement and plagiarism. Additionally, it helps with mitigating model collapse, a phenomenon where AI systems experience progressive quality loss and performance degradation due to being trained on AI-generated data.

2. Overview of detection approaches

We consider two broad approaches to detecting AI-generated content.

2.1. Post-hoc detection

The first approach, *post-hoc detection*, does not make changes to the model itself. Instead, a separate detection algorithm is used after the content is generated. This approach relies on the assumption that there are intrinsic, detectable, and computable markers of AI-generated content. Examples include tools such as GPTZero [1], a deep learning model trained to identify AI-generated text, and DetectGPT [5], which performs zero-shot detection by leveraging structural properties of LLM probability functions. Post-hoc detection methods also include heuristic signals, such as the "em-dash effect", where frequent use of the em-dash is treated as an indicator of AI-generated content.

One of the main benefits of post-hoc detection is that there is no need for cooperation by the model owners. However, this approach is also prone to false negatives and false positives, which are especially concerning. For example, many human writers also use em-dashes, and we want to avoid falsely accusing their work of being AI-generated.

In general, intrinsic markers of AI-generated content tend to be brittle, and post-hoc detection is becoming increasingly unrealistic as AI models continue to improve and the distribution of AI-generated content becomes more and more similar to that of human-generated content. Thus, we turn to our second approach for detecting AI-generated content, *watermarking*.

2.2. Watermarking

Instead of relying on intrinsic markers in AI-generated content, the watermarking approach modifies the underlying model itself so that it intentionally embeds markers into its generated content. Typically, this is done using a secret key, with the same key also being used for detection. While publicly-detectable watermarking schemes have also been proposed [3], in this lecture we focus on private-key watermarking schemes. Note that in contrast to post-hoc detection, the watermarking approach requires cooperation from model owners.

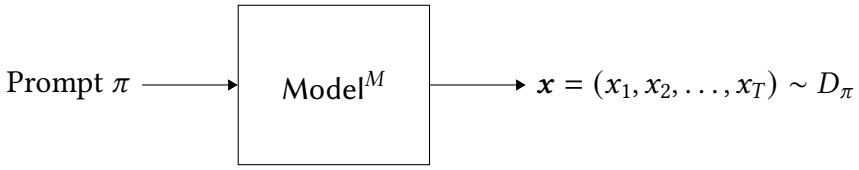
The idea of watermarking objects is not new in cryptography and security. In the traditional setting, static content, such as images or text, is taken as the input to be watermarked. This content must then be modified in some way while preserving the same qualities and meaning as the original. As one can imagine, this is an extremely difficult task.

For example, given an input image, we might attempt to add a watermark by changing certain pixels in the image. However, this involves a fundamental tradeoff. On one hand, we want the modifications to be substantial enough for the watermark to be effective and robust. On the other hand, we don't want to alter the image so much that its quality or content is degraded.

In the context of detecting AI-generated content, however, we have a critical advantage. In addition to the static input content itself, we also have access to the generative process or model that produced the content. This gives us much more flexibility and freedom for the watermarking task.

3. Generative models

Before describing the watermarking approach in more detail, we give a brief overview of *generative models*. Essentially, a generative model is a sampler. Given a prompt π, it produces an output $\boldsymbol{x} = (x_1, x_2, \ldots, x_T)$ of length T that is randomly drawn from a distribution, D_π, depending on π.



Generative models include autoregressive models, such as LLMs, as well as many other kinds of models, such as diffusion models. In this lecture, we focus on autoregressive models.

An *autoregressive model* samples a string $\boldsymbol{x} = (x_1, x_2, \ldots, x_T)$ from $\mathcal{T}^*$, where $\mathcal{T}$ is the set of all tokens, and each $x_i \in \mathcal{T}$. What makes it autoregressive is that given the first t tokens in the string, the model calculates a conditional probability distribution p_{t+1} for the next token in the sequence. This probability distribution should be succinctly specifiable and computable via some function M used by the model. The model then samples from this distribution to generate the next token, continuing to add tokens to the sequence until reaching some token that signifies the end of the string. Below is the pseudocode for an autoregressive model.

Algorithm 1: Autoregressive Model$^M(\pi)$

Input: Prompt π
Output: String $\boldsymbol{x} = (x_1, x_2, \ldots, x_T) \in \mathcal{T}^*$

$t \leftarrow 0$
$x_0 \leftarrow \perp$
while $x_t \neq$ *done* **do**
  $p_{t+1} \leftarrow M(\pi, (x_1, x_2, \ldots, x_t))$
  Sample $x_{t+1} \sim p_{t+1}$;
  $t \leftarrow t + 1$;
end
return $(x_1, x_2, \ldots, x_t)$;

4. Watermarking "API"

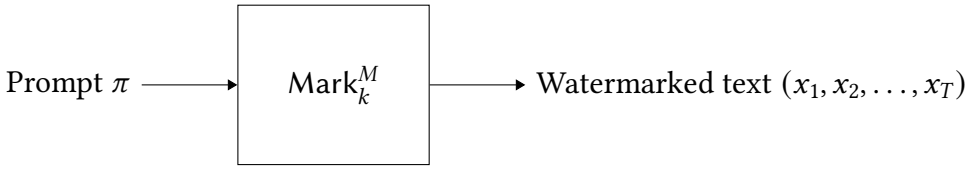
The watermarking API consists of two algorithms, Mark_k^M and Detect_k. Both algorithms share a private key k, and Mark_k^M additionally has access to the function M that specifies the unwatermarked autoregressive model.

4.1. Marking algorithm

The marking algorithm takes in a prompt π as input and then outputs the watermarked text:

$$\mathsf{Mark}_k^M(\pi) = (x_1, x_2, \ldots, x_T).$$

This is illustrated in the following diagram.

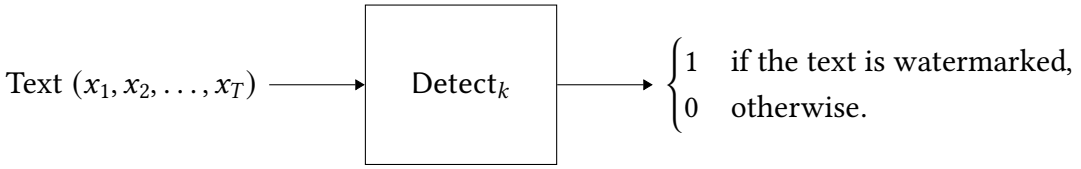


4.2. Detection algorithm

The detection algorithm takes in text as input and outputs either 0 (if the text is not watermarked) or 1 (if the text is watermarked):

$$\mathsf{Detect}_k(x_1, x_2, \ldots, x_T) = 0/1.$$

This is illustrated in the following diagram.



We note that Detect_k does not have access to the prompt π or the function M from the model, which makes sense given how AI-generated text must be identified in practice without additional information.

5. Strawman 1

Now, to better understand the properties we want in a strong watermarking scheme, we will examine strawman constructions. The first strawman scheme we consider is based on concepts from cryptography, so we begin by reviewing these concepts.

5.1. Preliminaries

First, we recall the definition of a *pseudorandom function*.

Definition 5.1 (Pseudorandom function)**.** A *pseudorandom function (PRF)* is a collection of functions

$$\mathcal{F} = \{F_k : \{0,1\}^n \to \{0,1\}^m\},$$

each specified by a key k, from n bits to m bits, such that for all probabilistic polynomial time (PPT) algorithms $\mathcal{A}$, we have

$$\Pr\left[\mathcal{A}^{F_k} \to 1\right] - \Pr\left[\mathcal{A}^{R} \to 1\right] \leq \epsilon,$$

where R is a truly random function. For this lecture, we will consider the case where $\epsilon = 2^{-\lambda}$, where λ is the security parameter.

Informally, PRFs are functions such that no PPT adversary can distinguish between the outputs of the PRF versus a truly random function with probability greater than $2^{-\lambda}$. Additionally, PRFs must be efficient to compute given the key k.

It turns out that the length of the key depends on the value of λ, our security parameter. In particular, the key must be at least λ bits long; otherwise the adversary could simply guess the key with probability greater than $2^{-\lambda}$ and obtain the F_k values.

As an aside, recall that we can use PRFs for secure secret-key encryption schemes with the following encryption and decryption algorithms:

$$\mathsf{Enc}_k(m; r) = (r, \mathsf{PRF}_k(r) \oplus m),$$

$$\mathsf{Dec}_k(r, c) = c \oplus \mathsf{PRF}_k(r),$$

where k is the secret key, m is a plaintext message, r is a random string, and c is the ciphertext.

For the Strawman 1 scheme, the main use case for PRFs will be in the context of message authentication. In this context, we compute the message authentication code (MAC) for a message m and key k as

$$\mathsf{MAC}_k(m) = \mathsf{PRF}_k(m).$$

This is then used as a tag t which is appended to the original message before being sent to the receiver.

In order to verify the message on the other end, the receiver simply recomputes the tag and checks that it matches what was sent:

$$\mathsf{Verify}(m, t) = (\text{YES iff } t = \mathsf{PRF}_k(m)) .$$

The idea behind this message authentication scheme is that even after observing polynomially many messages and their tags, an adversary would still not be able to compute the correct tag for an unseen message, therefore protecting the integrity of the original message being sent.

5.2. Watermarking scheme

Using these preliminaries, we present our first strawman watermarking scheme. The marking algorithm for this scheme will compute the MAC for a generated string using the secret key, then append it as a tag to the end of the input.

Algorithm 2: Strawman 1 $\mathsf{Mark}_k^M(\pi)$

Input: Prompt π
Output: Watermarked string
1 Generate string $\boldsymbol{x} = (x_1, x_2, \ldots, x_T)$ using $\mathsf{Model}^M(\pi)$
2 Tag $t \leftarrow \mathsf{MAC}_k(\boldsymbol{x})$
3 **return** (x, t);

Then, the detect algorithm simply checks whether the tag provided with an input text matches the MAC for that input.

Algorithm 3: Strawman 1 $\mathsf{Detect}_k(\boldsymbol{x}, t)$

Input: String $\boldsymbol{x} = x_1, x_2, \ldots, x_T$ with tag t
Output: 0/1
1 **if** $t = \mathsf{MAC}_k(\boldsymbol{x})$ **then**
2 | **return** 1;
3 **end**
4 **return** 0;

This strawman has a few merits. First, it satisfies completeness, in that any model-generated, watermarked text will be detected as such. Second, it satisfies soundness, because for any arbitrary text x and tag t, the probability that $t = \mathsf{MAC}_k(x)$ (i.e. the tag accidentally matches the true authentication code from a random k) is less than or equal to $2^{-\lambda}$. That is, the probability of a false positive is very small.

However, the obvious flaw with this scheme is that the tag is trivially removable. In order for watermarking to be effective, we want to get as close as possible to unremovability. Thus, the takeaway from this strawman is that we need the watermark to be intrinsic, not just something that is trivially appended.

6. Strawman 2

6.1. Watermarking scheme

The next strawman we examine is the Red/Green Scheme. This watermarking scheme works by randomly partitioning the token space $\mathcal{T}$ into two equal parts $R \sqcup G$, which are colored red and green. The Mark_k^M algorithm generates the watermarked string by biasing the autoregressive model toward outputting green tokens rather than red tokens, as shown in the following pseudocode.

Algorithm 4: Strawman 2 $\mathsf{Mark}_k^M(\pi)$

Input: Prompt π
Output: Watermarked string $\boldsymbol{x} \in \mathcal{T}^*$

$t \leftarrow 0$
$x_0 \leftarrow \perp$
while $x_t \neq$ *done* **do**
  $p_{t+1} \leftarrow M(\pi, (x_1, x_2, \ldots, x_t))$
  $q_{t+1} \leftarrow p_{t+1}$ with increased probability over G and reduced probability over R
  Sample $x_{t+1} \sim q_{t+1}$;
  $t \leftarrow t + 1$;
end
return $(x_1, x_2, \ldots, x_t)$;

More details on how to determine the partition $\mathcal{T} = R \sqcup G$ for each t, how to get the biased probability q_t from the corresponding p_t, and how to use key k to make the watermarking scheme private can be found in Kirchenbauer et. al.'s work [4].

Since watermarked outputs have higher proportions of green tokens, the Detect_k algorithm works by simply counting the number of green tokens, as illustrated in the following pseudocode.

Algorithm 5: Strawman 2 $\mathsf{Detect}_k(x)$

Input: String $\boldsymbol{x} = (x_1, x_2, \ldots, x_T) \in \mathcal{T}^T$
Output: 0/1

$c \leftarrow |\{t \in [T] : x_t \in G\}|$;
if $c/[T]$ *is sufficiently large* **then**
  return 1;
end
return 0;

The specifics on what counts as *sufficiently large* and how the key is used in the private watermarking case are also covered by Kirchenbauer et. al. [4].

6.2. Problems

Though the Red/Green Scheme avoids the removability issue from Strawman 1 by generating the output string in a way that is intrinsically watermarked, it has several issues that make it undesirable as a watermarking scheme.

The first problem is that the quality of the output degrades as the model avoids red tokens. This is more obvious when we consider the extreme case where $\text{Mark}_k^M(\pi)$ only returns green tokens, as this would greatly limit the expressive capability of the model. Reduced quality makes watermarking useless in practice since the unwatermarked model would be preferable.

The next problem is that it is easy to falsely identify text as watermarked under this scheme. For example, green tokens may be particularly common in human-generated text that is repetitive or covers a topic with green keywords, and this could result in false accusations of being AI-generated. We want to completely avoid such accusations from our watermarking scheme.

Finally, we consider the situation of prompts π for which there is a deterministic output. For such π, applying the Red/Green Scheme to get a new watermarked output results in a loss of quality because the correct output tokens are fixed. However, not watermarking the model output for π would make the watermarking scheme incomplete. Since it is impossible to detect whether deterministic text is AI-generated or human-generated, it makes sense to choose quality over completeness in this tradeoff. In other words, it makes sense to disregard completeness when there is high determinism and only watermark when there is high enough entropy in the generated text.

7. Properties

Strawman 2 helps us identify that, in addition to being intrinsic, there are three general properties that we want a watermarking scheme to satisfy: soundness, completeness assuming entropy, and preservation of quality. We now give the technical definitions that we will use for each of these properties.

7.1. Soundness

The first property we consider is *soundness*, which means that there are no false positives. Soundness must hold for our watermarking scheme, as we do not want Detect_k to mislabel human-generated content as AI-generated in any circumstances. We have the following definition.

Definition 7.1 (Soundness)**.** For all fixed strings $x \in T^*$,

$$\Pr_k\left[\text{Detect}_k(x) = 1\right] \leq 2^{-\lambda}.$$

There are stronger conditions we could consider in lieu of soundness, such as *unforgeability*. This property essentially means that even given access to examples of watermarked text, new text cannot be produced that Detect_k will identify as watermarked. However, the property of soundness as per our definition is sufficient for this lecture.

7.2. Completeness

The next property we consider is *completeness*. Completeness means that there are no false negatives, i.e. given k, Detect_k will identify any AI-generated text. However, we cannot simultaneously enforce soundness and completeness for a watermarking scheme because of prompts with deterministic outputs. In the case where a human-generated response would be the same as

the AI-generated output x, if $\text{Detect}_k(x) = 1$ then soundness is violated, and if $\text{Detect}_k(x) = 0$ then completeness is violated.

To avoid this issue, we only enforce completeness when there is high enough entropy in the output generation process, operating under the assumption that the outputs generated by AI through many random choices are actually distinguishable from human-generated text. This motivates our definition of b-completeness, parametrized by b.

Definition 7.2 (b-Completeness)**.** For all prompts π,

$$\Pr_{\substack{k \\ x \sim \text{Mark}_k^M(\pi)}} \left[\text{Detect}_k(x) = 0 \text{ AND } \text{Entropy}_M(\pi) \geq b\right] \leq 2^{-\lambda},$$

where λ is the security parameter.

Here, $\text{Entropy}_M(\pi)$ is an entropy function, which we will refine the definition of in a later lecture. For now, we can think of this as similar to the Shannon entropy function.

This notion of completeness is still too weak for practical use because of how easily AI-generated text can be manipulated. If simple modifications allow AI-generated text to avoid detection, then the watermarking scheme is not very effective. Instead of completeness, the stronger property we want to satisfy is *robustness*, which means that AI-generated text will be identified by Detect_k even after transformations in $\mathcal{E}$. Conditioning on high entropy as before, we get b-robustness, which is defined with respect to a class of channels $\mathcal{E}$ of modifications.

Definition 7.3 (b-Robustness)**.** For all prompts π and for all $E \in \mathcal{E}$,

$$\Pr_{\substack{k \\ x \sim \text{Mark}_k^M(\pi)}} \left[\text{Detect}_k(E(x)) = 0 \text{ AND } \text{Entropy}_M(\pi) \geq b\right] \leq 2^{-\lambda},$$

where λ is the security parameter.

The property of b-robustness is difficult to achieve in general, so we will revisit it in future lectures, instead focusing on b-completeness for now.

7.3. Quality

The final property we want the watermarking scheme to satisfy is preservation of the quality of the model outputs, since otherwise there is incentive to use the unwatermarked model rather than the watermarked version. Quality preservation is hard to quantify, so we use *undetectability* as a proxy. Undetectability is formally defined as follows.

Definition 7.4 (Undetectability)**.** For all models and for all PPT algorithms $\mathcal{A}$,

$$\Pr\left[\mathcal{A}^{\text{Model}^M} \to 1\right] - \Pr\left[\mathcal{A}^{\text{Mark}_k^M} \to 1\right] \leq 2^{-\lambda},$$

where λ is the security parameter.

Using undetectability as a gauge of quality makes sense because if it is not possible to determine in polynomial time whether an output came from the watermarked or unwatermarked version of the model, then the output of the watermarked model is "as good as" the output of the unwatermarked model for all polynomial time applications.

8. Undetectable watermarking scheme

We now present the main ideas underlying the undetectable watermarking scheme of Christ, Gunn, and Zamir [2], which utilizes correlated sampling.

In this scheme, we consider the tokens to be bits, so the token space is $\mathcal{T} = \{0, 1\}$. Given prompt π and previous outputs $x_1, \ldots, x_{t-1}$, the function M returns a probability $p_t \in [0, 1]$ such that the Model^M generates the next bit by sampling $x_t \sim \mathsf{Ber}(p_t)$. The key shared by the marking and detection algorithms is a tuple of T uniform random real numbers in $[0, 1]$, which we write as $k = (u_1, \ldots, u_T)$. We have the following marking algorithm $\mathsf{Mark}_k^M(\pi)$.

Algorithm 6: CGZ $\mathsf{Mark}_k^M(\pi)$

Input: Prompt π
Output: Watermarked string $\boldsymbol{x} \in \{0, 1\}^T$

$t \leftarrow 0$
$x_0 \leftarrow \perp$
while $t \leq T$ **do**
  $p_t \leftarrow M(\pi, (x_1, x_2, \ldots, x_{t-1}))$;
  if $u_t \leq p_t$ **then**
    $x_t \leftarrow 1$
  else
    $x_t \leftarrow 0$
  end
  $t \leftarrow t + 1$;
end
return $(x_1, x_2, \ldots, x_T)$;

For each $t \in [T]$, since $u_t \sim \mathsf{Unif}[0, 1]$ and

$$x_t = \begin{cases} 1 & \text{if } u_t \leq p_t, \\ 0 & \text{otherwise,} \end{cases}$$

we have that $\Pr[x_t = 1] = p_t$ and $\Pr[x_t = 0] = 1 - p_t$. This means that the marginal distribution of x_t generated by Mark_k^M is $\mathsf{Ber}(p_t)$, which is same as the unwatermarked model, indicating that the output quality is preserved. Note that this "distribution freeness" is a weaker property than undetectability, and we show how to actually achieve undetectability using PRFs next lecture.

Moreover, x_t and u_t are correlated, which is what allows for detection given the key. In particular, as $u_t \to 0$, the value of x_t is more likely to be 1, and as $u_t \to 1$, the value x_t is more likely to be 0. Based on this correlation, we have the following detection algorithm $\mathsf{Detect}_k(\boldsymbol{x})$.

Algorithm 7: CGZ Detect$_k(\boldsymbol{x})$

Input: String $\boldsymbol{x} = (x_1, x_2, \ldots, x_T) \in \{0,1\}^T$
Output: 0/1

$t \leftarrow 1$
while $t \leq T$ **do**
  if $x_t = 1$ **then**
    $s(x_t, u_t) \leftarrow -\ln u_t$
  else if $x_t = 0$ **then**
    $s(x_t, u_t) \leftarrow -\ln(1 - u_t)$
  end
  $t \leftarrow t + 1$;
end
$c(\boldsymbol{x}, \boldsymbol{u}) \leftarrow \sum_{t=1}^{T} s(x_t, u_t)$
if $c(\boldsymbol{x}, \boldsymbol{u})$ *is sufficiently large* **then**
  return 1;
end
return 0;

For each $t \in [T]$, the score

$$s(x_t, u_t) = \begin{cases} -\ln u_t & \text{if } x_t = 1, \\ -\ln(1 - u_t) & \text{if } x_t = 0 \end{cases}$$

quantifies the correlation between x_t and u_t, and the natural log relates to the entropy function for completeness. The soundness and completeness properties of the undetectable Christ-Gunn-Zamir watermarking scheme will be proven in the next lecture.

References

[1] George Alexandru Adam, Alexander Cui, Edwin Thomas, Emily Napier, Nazar Shmatko, Jacob Schnell, Jacob Junqi Tian, Alekhya Dronavalli, Edward Tian, and Dongwon Lee. Gptzero: Robust detection of llm-generated texts, 2026. URL https://arxiv.org/abs/2602.13042.

[2] Miranda Christ, Sam Gunn, and Or Zamir. Undetectable watermarks for language models, 2023. URL https://arxiv.org/abs/2306.09194.

[3] Jaiden Fairoze, Sanjam Garg, Somesh Jha, Saeed Mahloujifar, Mohammad Mahmoody, and Mingyuan Wang. Publicly-detectable watermarking for language models. Cryptology ePrint Archive, Paper 2023/1661, 2023. URL https://eprint.iacr.org/2023/1661.

[4] John Kirchenbauer, Jonas Geiping, Yuxin Wen, Jonathan Katz, Ian Miers, and Tom Goldstein. A watermark for large language models, 2024. URL https://arxiv.org/abs/2301.10226.

[5] Eric Mitchell, Yoonho Lee, Alexander Khazatsky, Christopher D. Manning, and Chelsea Finn. Detectgpt: Zero-shot machine-generated text detection using probability curvature, 2023. URL https://arxiv.org/abs/2301.11305.