PPPostMark: Private LLM Watermarking

Meenakshi Iyer
meenakshiiye@umass.edu
University of Massachusetts, Amherst
Amherst, Massachusetts, USA

Abstract

Generative AI offers powerful capabilities but also introduces risks like phishing and misinformation, highlighting the need for reliable detection. Existing ML-based detectors face issues such as inconsistency, ethical concerns, and lack of standardization, pointing to the need for robust, unforgeable watermarking. Similarly, search engines expose user data, raising privacy concerns. While tools like Tor help, they still leak metadata. Newer cryptographic solutions like Tiptoe aim to enable private web search. These issues converge in the challenge of private watermarking—prompted by features like Google's AI Overviews—raising the question: can we embed watermarks without seeing the generated text? Our response is PPPostMark: Privacy Preserving PostMark, a private watermarking scheme that uses multiple communication rounds between a client and server in order to watermark text without leaking much information. We show that PPPostMark satisfies completeness, unforgeability, and distortion-freeness/undetectability. We hope to open the door to new lines of work in private LLM watermarking schemes.

CCS Concepts

• **Social and professional topics** → **Censoring filters**; • **Information systems** → *Information retrieval*; **Personalization**; • **Security and privacy** → **Cryptography**; **Software security engineering**; **Usability in security and privacy**; **Privacy protections**.

Keywords

LLM, Privacy, Watermarking

ACM Reference Format:
Meenakshi Iyer. 2025. PPPostMark: Private LLM Watermarking. In . ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 Introduction

Generative AI, with its widespread use, offers incredible capabilities but also presents significant risks, such as enabling sophisticated phishing scams, spread of misinformation, and much more [9]. To address these concerns, a crucial need arises for reliable methods to detect AI-generated content. However, Machine Leaning Model based detection models have a number of issues, including in- consistent results, ethical concerns about relying on AI creators for detection, and the lack of standardized detection methods [9]. This highlights the urgent need for a robust, detectable, and unforgeable watermarking scheme to effectively identify and mitigate the risks associated with AI-generated content.

Similarly, Search Engines are used on a daily basis by individuals, specifically to search the web, and retrieve thousands to millions of search results, and then are ranked and displayed to the user. However, these queries can also be sold to companies as personal data, revealing information about the user. They can also result in more targeted ads. Given these issues, and many other issues that stem from lack of privacy, the need for a Private searching scheme exists. There are solutions such as using onion browsers like Tor, but these inaverdently can reveal other metadata about the user or the user's queries. Newer constructions such as Tiptoe [12] offer Private Web Searching backed by cryptographic hardness assumptions.

Combining these two problems poses a question of *private* watermarking. Motivated by Google's recent addition of an AI-Overview feature that passes the query into an Large Language Model [1] as well, we ask whether it would be possible to generate watermarked text without knowledge of the generated text.

2 Preliminaries

We have two distinct entities that interact in our watermarking scheme.

- **Server.** Encompassing the **SecTable**, **Embedder**, and **Inserter**, the server handles communication between all 3 constructions, as well as to the user. This entity has black-box access to the underlying Large Language Model used to insert and generate text.
- **Client.** This is the user who wants watermarked text. They should be able to query with an encrypted prompt, and after a few rounds of communication, be returned a encrypted watermarked text. They share a key with the **Inserter**, which the server cannot access.

A Large Language Model LLM is formally defined as an auto-regressive model over token vocabulary T is a deterministic algorithm that takes in a prompt $\mathbf{p} \in T^*$ and text previously outputted by the model $t \in T^*$ and outputs a new token t'.

A watermarking scheme is defined as a triple of algorithms (***Initialize, Watermark, Detect***) where:

- ***Initialize*** sets up any public or private key pairs.
- ***Watermark*** given a prompt, returns watermarked text using the secret key and the underlying Large Language Model.
- ***Detect*** given a candidate text, outputs whether or not the text was watermarked.

ACM ISBN 978-x-xxxx-xxxx-x/YYYY/MM
https://doi.org/10.1145/nnnnnnn.nnnnnnn

As with other watermarking schemes, we have a number of security definitions that we aim to satisfy, listed below. First, we define a few computational models for the security definitions.

- $\text{Watermark}_{\text{sk}}(\mathbf{p})$. Under a given secret key and given prompt $\mathbf{p}$, Watermark outputs a watermarked text. It is assumed that the text outputted is encrypted in this case.
- $\text{Detect}_{\text{sk}}(t)$. Under a given secret key and candidate text t, Detect outputs 0 or 1 depending on the presence of a watermark. Here, 0 corresponds with unwatermarked, and 1 corresponds with watermarked.
- $\text{CosineSimilarity}(u, v)$. Given two texts u and v, CosineSimilarity outputs a value between 0 and 1 that represents how semantically similar the texts are. Here, a lower value corresponds with dissimilar texts, and a higher value corresponds with similar texts.

The proofs of these are in the security analysis below.

- **Completeness.**

$$Pr[\text{Detect}_{\text{sk}}(t) => 1 : t \leftarrow \text{Watermark}_{\text{sk}}(\mathbf{p})] \leq \epsilon \quad (1)$$

- **Unforgeability.** For every computationally bounded adversary $\mathcal{A}$ with access to a watermarking oracle Wat_{sk}

$$Pr[\text{Detect}_{\text{sk}}(t^*) => 1 : t^* \leftarrow \mathcal{A}^{\text{Wat}_{\text{sk}}(.)}] \leq \epsilon \quad (2)$$

  Here, Wat_{sk} is an oracle that, under a given encrypted prompt, will watermark the generated text with a randomly sampled server key sk. To win, the adversary must return an unqueried watermarked text. Given the design of the scheme, we also require the the cosine similarity score between the texts the oracle outputs and the adversary returns be small.
- **Distortion-Freeness.** A watermarking scheme is distortion-free if no computationally bounded distinguishing adversary $\mathcal{D}$ can distinguish between Wat_{sk} and LLM, the underlying blackboxed Large Language Model.

$$Pr[\mathcal{D}^{\text{Wat}_{\text{sk}}(\mathbf{p})} => 1] - Pr[\mathcal{D}^{\text{LLM}(\mathbf{p})} => 1] \leq \epsilon \quad (3)$$

- **Privacy-Preservingness** The probability some computationally bounded adversary A can output a text that is semantically similar to a random watermarked text is negligible.

$$Pr[\text{CosineSimilarity}((\mathcal{A}(t)), t) \leq \epsilon : t \leftarrow \mathcal{A}^{\text{Watermark}_{\text{sk}}(\mathbf{p})}] \leq \epsilon \quad (4)$$

- **Robustness**

$$Pr[\text{Detect}_{\text{sk}}(\mathcal{A}(t)) => 0 : t \leftarrow \text{Watermark}_{\text{sk}}(\mathbf{p})] \leq \epsilon \quad (5)$$

Completeness requires that the scheme have a low rate of false negatives. This is standard for a practical watermarking scheme, and motivates a Large Language Model Provider to integrate watermarking into their product.

Unforgeability requires that an adversary cannot easily forge the watermark without knowledge of the watermarking key. In the real world setting, an adversary who is easily able to falsely generate a watermarked text can use the watermarked text in a harmful manner (eg. falsely claiming that an unsafe generation was provided by the model, due to the watermark). Similarly, to motivate the Large Language Model Provider to integrate the watermark into their product, there should not be a high rate of false positives.

Distortion-Freeness requires a certain quality to be maintained by the watermarked text, and is necessary in order to make the text outputted by the model usable. Downstream tasks that rely on the watermarked text generation looking identical to that of unwatermarked text motivate the necessity of this property. In this work, we only aim to show that the insertion procedure is as distortion-free as that of the original scheme. In the original scheme [4], the primary statement made is that insertion of any amount of text hurt quality, and thus is expected in most watermarking schemes. However, they support their intuition of their algorithm via empirical results that showcase a 0.95 cosine similarity score between unwatermarked and watermarked text, as well as detailed quality analysis using both LLM-as-a-judge, and human evaluators.

Privacy-Preservingness is the main contribution of this work. Users should be motivated to use watermarked text so long as the contents of their prompts and outputs are not leaked. This of course depends on the fact that the server is honest, and the user is honest. Maintaining privacy in a setting where the server may be compromised is an interesting direction for future work.

A note on Robustness. There are many works in the line of watermarking that aim to empirically or provably demonstrate high robustness, that is, that the watermark remains in the text after an adversary edits. Strong robustness has been shown to be impossible, but the threshold of how many edits can be handled in the weak robustness setting is ever-increasing. The 2 main attacks are the Edit Distance attack [11], in which the adversary randomly flips bits in the watermarked text, and the Paraphrase attack [4], in which the adversary paraphrases the text using a weak Large Language Model. PostMark [4]was designed to maximize robustness under the Paraphrase attack setting, but it is unclear how that attack would translate in a privacy-preserving setting. We hope future research investigates this further.

3 Existing Works

A number of existing works regarding watermarking have recently emerged, falling into two main categories: "NLP-based", or "Cryptography-based." In the former, works such as [13, 15] bias the tokens in the direction of a certain secret. In the latter, primitives such as Error Correcting Codes [6, 9] are used in order to lower an entropy requirement, and thus are be able to successfully embed the watermark in most generated texts.

We summarize one from each category.

3.1 Kirchenbauer et. Al

[13] The baseline method on sequences with *sufficient entropy* is to watermark as follows. Given a prompt and a series of previously generated tokens, the probability vector over the vocabulary for the next token is generated. The hash of the previous token is used to seed a random number generator, and using that seed, partitions the vocabulary into a Green and Red List G and R. The next token is then sampled from G.

Detection (without the consideration of an adversary) works with a null hypothesis H_0 – the probability that human generated text produces T tokens without sampling from the corresponding R, is $\frac{1}{2^T}$. A more robust approach uses the one proportion z-test to evaluate H_0. If it is true (not AI-generated text), then the analysis on the green list tokens ($|s|_G$) would follow an expected value of

$T/2$ and variance $T/4$. Given the formula: $z = 2(|s|_G - T/2)/\sqrt{T}$, we reject the null hypothesis when z is above a certain value.

An example is with $z > 4$ (quite low), where the rate of false positives is 0.00003, but will result in successful detection of a watermarked sequence of 16 tokens or more.

With insufficient entropy, the algorithm changes slightly in order to minimize the impact on low entropy tokens (i.e Donald is likely followed with Duck, so instead of accidentally putting Duck in the Red List, we add constant δ to all the logits so it will be chosen). The z-statistic is computed with $z = 2(|s|_G - T\gamma)/\sqrt{T\gamma(1-\gamma)}$.

3.2 Christ and Gunn

[6] The intuition behind [6]'s watermarking scheme is to have the bits of the codeword directly correlated to every output of the text. To do this, they start by generating a Pseudorandom Code (PRC), which is defined as a codeword from the Error Correcting Codes whose codewords are both pseudorandom (to a computationally bounded adversary), and efficiently decodable with a key. This codeword is used as the random seed of the LLM, and there is a bitwise correleation between each bit of the output and each bit of the seed, thus making the PRC recoverable. To do this, they use a binary transform, so that the output space of the text goes from the size of the vocabulary, to just two. The entropy of the tokens and the bits of the PRC are mapped using a Bernoulli distribution in order to sample the associated token.

Motivation. However, these works are not feasible in the privacy preserving setting because they rely on creating/detection a relationship between the generated tokens, and bits of a codeword/secret. Many of these works are also not blackboxed, and rely on access to the Large Language Model logits. Thus, we turn to PostMark [4] as our underlying scheme to transform.

3.3 PostMark

The main foundation of our work is a post-hoc watermarking scheme called PostMark [4]. Postmark generates texts from the Large Language Model natively given a prompt, and then uses 3 main constructions for the watermarking. Firstly, an **Embedder** creates vector embeddings of the generated text. Then, a key of sorts is created using a random mapping between the vector embeddings of many random documents and the set of vocabulary, and this is the **SecTable**. Lastly, an **Inserter** is required, which is an Large Language Model with instruction following capabilities.

The process starts with an embedding of an already generated text created using the **Embedder**. Then, the cosine similarity [7] between this embedding and all the word embeddings in **SecTable** is computed. Then, top-k selection [20] and filtering is performed to create a list of watermark words from the **SecTable**. Then, the watermark is inserted using **Inserter**, with the instruction to rewrite the text to include words from the watermark word list. There exists a hyperparameter insertion ratio that determines the number of words to insert.

Detection occurs by embedding the text using the **Embedder**, and generating the list of watermark words, and seeing how many are present in the text. A word w is detected as present in the text if that word is present in the text, or there exists another word w' with an embedding cosine similarity greater than the threshold of 0.7, for added robustness to paraphrasing attacks. The process is described in the image below.

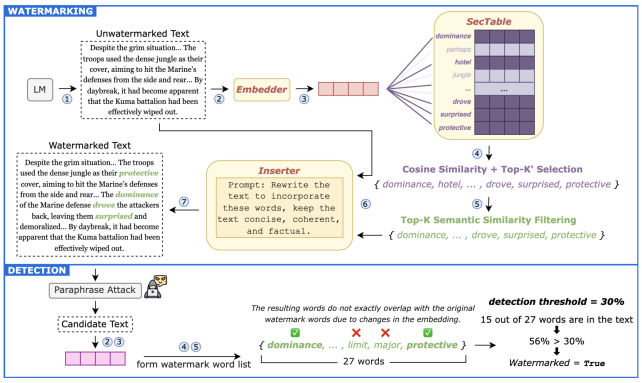

Figure 1: Figure 1

3.4 Comparison of Methods

We compare these 3 works in the context of robustness in the table below.

Scheme	Paraphrase Robustness	Edit Distance Robustness
[6]	Not analyzed	Constant rate of insertion and deletions
[13]	High, empirical results suggest it is not easy to introduce enough red tokens to remove the water- mark	Considered trivial, since it would significantly reduce quality
[4]	High	Not analyzed

Table 1: Comparison of Watermarking Scheme Robustness.

3.5 PostMark Leakage

There are several places of potential leakage. Primarily, the list of watermark words being public reveals information about the contents of the response. Secondly, the **SecTable**'s two sets (words and document embeddings) must be private so that information about the words selected is not revealed. Lastly, if the client can access the mapping of words, they can adversarially insert words into the text, and make a false positive, which breaks the notion of unforgeability.

4 PPPostMark

4.1 High Level Overview

Figure 2 describes the watermarking process of Privacy Preserving PostMark (PPPostMark). The intuition comes from a first attempt at making the scheme private: adding a layer of encryption to each

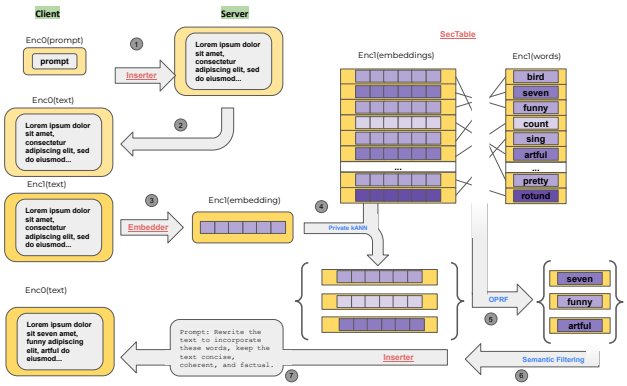

Figure 2: This figure summarizes the high level watermarking scheme. Firstly, the client passes in an encrypted prompt, under the first key, which is denoted as Enc0(prompt). Then, the encrypted generated text is returned, and re-encrypted under key 1 to get Enc1(text). The encrypted embedding of this is generated. Using Private Approximate Nearest Neighbors, k' semantically similar embeddings are selected. Using Oblivious Psueodorandom Function, the corresponding encrypted words are selected, and then filtered using cosine similarity. These words are then inserted into the text using the Private Large Language Model Inserter.

and every step. However, simply this does not suffice, because pseudorandom mappings and indexes of document embeddings may reveal information about the text being watermarked. Thus, we rely on protocols such as Private Approximate Nearest Neighbor Search (PANN) [**?**], Oblivious Pseudorandom Function (OPRF) [18], and Private Set Intersection (PSI) [14, 19] in order to maintain security. Figure 3 describes the detection process, in which the candidate text embeddings are generated, and then the watermark list is reconstructed. If a significant amount of watermark words remain, the text is considered watermarked. A detailed security analysis follows.

4.2 Private LLM

We recognize that they are no existing state of the art schemes for Private Large Language Models, that given an encrypted prompt. However, it is possible to do so [22] in a theoretic manner. Likely, this is difficult to implement, and would require a lot of server side computation. Moreover, the solution of simply applying Fully Homomorphic Encryption to a large computational model is extremely inefficient. However, the paper proposes speeding up computation by approximating building blocks in neural networks (such as activation functions in the ReLu function) using polynomial function techniques such as Chebyshev polynomials [16]. There are existing solutions for this [21]. Compression can also be used to shrink the size of the model. Thus, we rely on the assumption that a Private Large Language Model already exists, and aim to prove a theoretical result.

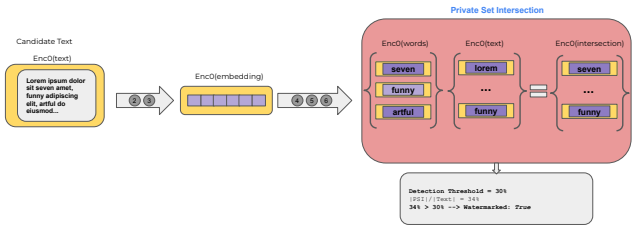

Figure 3: This figure summarizes the detection process. Firstly, the candidate text's embeddings are generated, and the watermark list is reconstructed. Private Set Intersection is used to find if the watermark words are present in the text. Then, if the ratio is higher than the threshold, the text is said to be watermarked.

4.3 Transformations

We summarize the technical transformation of the three main constructions of Postmark.

4.4 Inserter

The construction of **Inserter** is trivial, since it reduces down to the same assumption we made earlier – that we can generate text privately.

4.5 Embedder

Given a text, the **Embedder** outputs the semantic embedding of that text in the format of a plaintext vector. Even though this computation occurs once per generation, it is unreasonable to apply the trivial solution of Fully Homomorphic Encryption [3] here, since the models used to generate semantic embeddings are extremely large, making it extremely inefficient and computationally expensive. Thus, the **Embedder** will mostly remain the same as before, except it will decrypt the client's text first under a shared secret key, then generate a semantic embedding, and return the encrypted embedding under the same key to the server. Note that the server does not have access to this key.

4.6 SecTable

In the original scheme, the **SecTable** is a mapping between a set of words and a set of embeddings of documents, the latter represented as plaintext vectors. A natural step to take is to encrypt both sets. Now, we have a set of encrypted documented embeddings, and encrypted words.

4.7 PANN: Private Approximate Nearest Neighbor

Rather than using a Private Information Retrieval based scheme (we do not know the exact index of the target embeddings), we turn to a Private Private Approximate Nearest Neighbor scheme.

Private Approximate Nearest Neighbor (PANN) search allows a client to find similar items in a server's database without revealing their query or learning anything beyond the approximate nearest neighbors. It typically combines cryptographic techniques like PIR [10], homomorphic encryption [3], or secure multiparty computation [8] to compute similarity securely and efficiently over large datasets. We do not select an existing scheme, but numerous schemes exist [5, 17, 24], each offering a different benefit in regards to efficiency, privacy, and simplicity.

At a high level, the client will send the secret shared/encrypted query to the server, and the server will return the top-k matches securely, relying on Homomorphic Encryption or Garbled Circuits [2]. Using batching speeds up the process.

4.8 Oblivious Psueodorandom Function

In order to maintain the pseudorandom mapping between the set of embeddings and set of words, we need to utilize an Oblivious Pseudorandom Function (OPRF) [18]. In OPRF, the server has a key, and the client has an input. Using the server's key, the client learns the output of the PRF on the given key without learning the mapping, and the server does not learn the input or output.

4.9 Insertion

We summarize the communication between the client and server below.

- **Round 1: Text Generation.**
  - The client sends the server the encrypted prompt $Enc_{skInserter}(\mathbf{p})$, and the server runs the privacy preserving Large Language Model **Inserter**, and gets back $Enc_{skInserter}(\mathbf{t})$.
  - This is sent back to the client, who then decrypts it under $skInserter$ and encrypts it under $skEmbedder$ to get $Enc_{skEmbedder}(\mathbf{t})$.
- **Round 2: Generating an Embedding.**
  - Then, the **Embedder**, under the shared key $skEmbedder$, generates an embedding of the text to get $Enc_{skEmbedder}(\mathbf{e})$.
  - Now, we use a PANN scheme described above in order to retrieve a set of several close embeddings, which we call **E'**.
- **Round 3: Selecting k Words.**
  - Given $Enc_{skEmbedder}(\mathbf{e'})$, $\mathbf{e'} \in \mathbf{E'}$, we run the OPRF protocol described above, and are returned with k' encrypted words.
  - Then, we compute the cosine similarity between the encrypted words and the generated text, till we are left with k encrypted words, which we return to the client.
- **Round 4: Insertion.**
  - The client re-encrypts these words under the $skInserter$ and sends them to the server, and then the server runs the privacy preserving Large Language Model **Inserter** with these encrypted words as well, and returns $Enc_{skInserter}(\mathbf{t})$, the watermarked text, to the client.

4.10 Detection

To detect, the same process as above is run in order to find the watermark word list. In order to determine if a word is present in the text, Private Set Intersection [19] is used. The client encrypts each word of the text, and if any of them match the encryption of a word in the watermark word list, that word is marked as present. There are other versions of Private Set intersection that can be used, such as n OPRF-based Private Set Intersection protocol, where the client uses an Oblivious PRF to obtain pseudorandom encodings of its elements, while the server computes the same PRF on its own set using a secret key. The client then compares the outputs to find common elements, achieving PSI efficiently without relying on fully homomorphic encryption. If a significant number of words are marked as present, the text is determined to be watermarked.

5 Security Analysis

First and foremost, a robust set of parameters must be chosen for each of the constructions. Key lengths may vary, and the setup that allows for a shared key between the server and client, as well as the different constructions is not specified in this work. Secondly, the following security definitions do not encompass any and all scenarios that this watermarking scheme may be used in, and we hope that future work further improves upon this.

5.1 Completness

Recall the definition for completeness, where a genuinely watermarked text should fail to be detected with low probability. We aim show that we can embed as well as the original scheme.

$$Pr[\mathsf{Detect}_{\mathsf{sk}}(t) => 1 : t \leftarrow \mathsf{Watermark}_{\mathsf{sk}}(\mathbf{p})] \leq \epsilon \tag{6}$$

We step through the construction of the scheme. First, the embedding generated by the private Embedder will not fail to output the same embedding as the original Embedder, since the only difference is a layer of encryption before and after generating the embeddings. Second, by the cryptographic guarantees of an Oblivious Psueodorandom Function, we know the watermark words sampled are still pseudorandom, and thus, not necessarily likely to already be in the text. Therefore, similar to Postmarks assumption that the Inserter is an Large Language Model with good instruction following capabilities, as long as our Private Large Language Model has good instruction following capabilities, we are able to successfully embed the watermark words in the text. If we know the watermark words exist in the text, in the absence of an adversarial edits, we will be able to successfully detect as well. Further work can empirically assert this.

5.2 Distortion-Freeness/Undetectability

In order to show distortion-freeness, we rely on the pseudorandomness of the choice of watermark words being inserted. We aim to show that pseudorandomness implies undetectability, and do this by showing a scheme that is detectable breaks pseudorandomness. Consider the adversary $\mathcal{D}$ that plays the undetectability game on the watermarking scheme. In the game is shown, the $K_{Watermark}$ encompasses all keys required to watermark.

Inputting two distinct prompts, the adversary should not be able to distinguish between the watermarked and unwatermarked text. The adversary's advantage is the probability that they are able to distinguish between the unwatermarked and watermarked text, as shown below.

$$\underline{\textbf{proc Initialize}}$$
$$K_{Watermark} \xleftarrow{\$} \mathcal{K}$$
$$b \xleftarrow{\$} \{0,1\}$$
$$\underline{\textbf{proc Wat}(\mathcal{E}(p))}$$
$$\text{If } b == 0:$$
$$\text{return } \textbf{RandomWords}(\mathcal{E}(p))$$
$$\text{Else}:$$
$$\text{return } \textbf{WatermarkWords}(\mathcal{E}(p))$$
$$\underline{\textbf{proc Finalize}(b')}$$
$$\text{return } b == b'$$

$$Pr[\mathcal{D}^{\text{Wat}_{\text{sk}}(\mathbf{p})} => 1] - Pr[\mathcal{D}^{\text{LLM}(\mathbf{p})} => 1] \leq \epsilon \tag{7}$$

Now, consider the adversary $\mathcal{A}$ playing the Pseudorandom Function game that simulates the Undetectability game for adversary $\mathcal{D}$. In this, if the adversary $\mathcal{D}$ returns unwatermarked, the adversary $\mathcal{A}$ returns random. If $\mathcal{D}$ returns watermarked, then, the adversary $\mathcal{A}$ returns real.

Adversary A^{PRF}**:**

run $\mathcal{D}$ till halt.

when $\mathcal{D}$ makes query $E(p)$

return Watermark(**Wat**$(E(p))$) to $\mathcal{D}$ ▷ i.e., watermark using the watermark words returned by $\mathcal{A}$'s oracle.

return D's output.

If we assume that the adversary $\mathcal{D}$ *does* win the Undetectability game with a non-negligable advantage, then, we can also win the PRF game with a non-negligable advantage. However, by the pseudorandomness of the OPRF, it is not possible to have a high advantage, thus, the scheme is undetectable.

Remark. Note that there may be a significant computational difference between the Unwatermaked and Watermarked game in the case of a non-private **Inserter**. However, because the construction of the private LLM relies on computationally expensive operations, such as FHE, this is likely to make the difference in computation negligable. Thus, an adversary using a timing attack in order to break Undetectability will not win the game with high advantage.

5.3 Unforgeability

Recall the definition of Unforgeability necessitates that the forged text that the adversary returns shares a low semantic similarity with any of the previously queried texts. For every computationally bounded adversary $\mathcal{A}$ with access to a watermarking oracle Wat_{sk}

$$Pr[\text{Detect}_{\text{sk}}(t^*) => 1 : t^* \leftarrow \mathcal{A}^{\text{Wat}_{\text{sk}}(.)}] \leq \epsilon \tag{8}$$

Here, Wat_{sk} is an oracle that, under a given encrypted prompt, will watermark the generated text with a randomly sampled server key sk. To win, the adversary must return an unqueried watermarked text. Given the design of the scheme, we also require the the cosine similarity score between the texts the oracle outputs and the adversary returns be small.

If an adversary was able to generate a watermarked text *without* knowledge of any of the keys, then it means the watermark words are present in the text. However, it is not likely for the adversary to simply *guess* which words are part of the native generation of the text, since these words are chosen pseudorandomly, and then semantically filtered. By a union bound [6] over the space of keys, it is easy to see that likelihood of the adversary winning the Unforgeability game is negligible.

5.4 Privacy Preservingness.

To prove Privacy Preservingness, we step through the scheme at each round.

- **Round 1: Text Generation.** The client sends the server the encrypted prompt $Enc_{skInserter}(\mathbf{p})$, and the server runs the privacy preserving Large Language Model **Inserter**, and gets back $Enc_{skInserter}(\mathbf{t})$. In this step, there is no information leaked besides the size of the ciphertext. Advanced encryption schemes can handle this via padding, or maybe format preserving encryption. In the next step, it is sent back to the client, who then decrypts it under *skInserter* and encrypts it under *skEmbedder* to get $Enc_{skEmbedder}(\mathbf{t})$. No information is leaked in this step either, since we assume we have trustworthy processes of sending information.
- **Round 2: Generating an Embedding.** Then, the **Embedder**, under the shared key *skEmbedder*, generates an embedding of the text to get $Enc_{skEmbedder}(\mathbf{e})$. Since all the embedder does is add a layer of encryption at each step, no information is leaked, besides to the embedder. Now, we use a PANN scheme described above in order to retrieve a set of several close embeddings, which we call **E'**. By the privacy of PANN, no information is being leaked.
- **Round 3: Selecting *k* Words.** Given $Enc_{skEmbedder}(\mathbf{e'}), \mathbf{e'} \in \mathbf{E'}$, we run the OPRF protocol described above, and are returned with k' encrypted words. OPRF by construction does not reveal the pseudorandom mapping. Then, we compute the cosine similarity between the encrypted words and the generated text, till we are left with k encrypted words, which we return to the client. Since this is a polynomial function on embeddings, it can be done using Fully Homomorphic Encryption, which doesn't leak information about the ciphertexts.
- **Round 4: Insertion.** The client re-encrypts these words under the *skInserter* and sends them to the server, and then the server runs the privacy preserving Large Language Model **Inserter** with these encrypted words as well, and returns $Enc_{skInserter}(\mathbf{t})$, the watermarked text, to the client.

Therefore, the scheme must be privacy preserving, as no information besides ciphertext length is leaked.

5.5 Impossibly of Strong Robustness

One important piece of literature is an impossibility result regarding the inability to form strong watermarking schemes, in both the public and private setting. As defined by [25], a strong watermarking scheme is one where a computationally bounded adversary cannot remove the watermark without significant quality degradation. A weak watermarking scheme is where the adversary is limited in some regard. Some existing schemes use edit distances as a way to

limit the adversary, i.e. the adversary can only edit up to **t** bits of the output.

The attack relies on the following assumptions:

- The adversary has access to a quality oracle, which allows it to evaluate the quality of a response y on a prompt x.
- The adversary has access to a perturbation oracle such that given an input of a prompt x and response y, will return a randomly perturbed response y' such that:
  - The probably that the quality of y' is greater than that of y is bounded away from 0.
  - The random walk that the oracle takes on the space of outputs has good mixing properties. This means that the walk will eventually reach a steady state distribution.

  Note it is true that it doesn't require the text chosen by the adversary be the same level of quality, but if the quality oracle can make results that are comparable in quality, then they would just use that to generate the texts, and the attack would be trivial.

In practice, the quality oracle can be the model itself. Given these assumptions, they define the attack as follows.

5.5.1 The Attack: The attack relies on the intuition that the adversary's goal is to not generate a semantically similar text, but rather a text that is quality preserving. The adversary would use rejection sampling to run a random walk on the set of outputs of a prompt x, such that they only consider the outputs that retain a high level of quality. This attack works in both the public and private setting.

In regards to our scheme, we do not analyze robustness, but the intuition is that the paraphrase robustnessess will still be bounded in some regard, especially because external quality checking is harder.

6 Limitations

As mentioned previously, the assumption of an efficient Private Large Language Model [21] is the primary bottleneck in this scheme.

Secondly, a common hurdle in watermarking schemes is overcoming low entropy. Entropy is defined as the randomness in a probability distributions. In large language models, low entropy sequences are when defined as follows:

Given previous tokens $t_1, t_2, ...t_n$,

$$\exists i \in t_1, t_2, ..., t_v \forall j \in t_1, t_2, ..., t_v, j \neq i : p(t_i|t_1, t_2, ..., t_n) >> p(t_j|t_1, t_2, ..., t_n)$$

where $v = |V|$, and $>>$ means significantly greater.

Informally, it means that given the previous token(s), there is one extremely likely token. Consider a generation regarding U.S. Presidents, where the most recent token sampled is "Barack". With high probability, the next token will be "Obama". In the case of watermarking, it would be hard to embed a signal that changes the distribution of the next token to anything other than "Obama". Therefore, schemes often have some threshold of errors that they tolerate during generation. Therefore, it is hard to make improvements on completeness without harming distortion freeness, and vice versa.

Thirdly, there is a reliance that the server is honest, and the client is honest. In the real world setting, this may not necessarily hold.

Lastly, the scheme is not efficient. We are limited by the computational complexity of schemes that rely on homomorphic encryption, such as Private Approximate Nearest Neighbor search. Even with the fastest of underlying primitives, the scheme itself composes large embedding models and generation models, which drive up the complexity.

We also leak ciphertext sizes throughout the scheme, which may be able to be circumvented via padding.

7 Further Directions

We are interested in seeing how the private watermark performs empirically, both in an adversarial and non adversarial setting. A proof of concept will open the door to many new private watermarking schemes.

We also plan on analyzing this scheme in the context of the training of an Privacy Preserving Large Language Model. [23] showed that LLMs that were trained on watermarked texts were *radioactive*. That is, without knowledge of the secret key under which the training text was watermarked, the trained LLM generated texts that were also partially watermarked under the same key. A next step would be to see if this holds in a privacy preserving setting. If it did hold, it would result in information leakage, an interesting analysis to consider.

8 Conclusion

In conclusion, Private LLM Watermarking is a cryptographic approach to embedding identifiable signals into AI-generated content without exposing sensitive inputs or revealing the watermarking process itself. We aim to provide provable model traceability while preserving user privacy and query confidentiality. This is especially relevant in settings where generated outputs such as text from Large Language Models are based on private prompts, as in systems like Google's AI Overviews. A number of existing watermarking schemes often assume access to the plaintext output, but private watermarking explores whether it's possible to embed robust, unforgeable watermarks without ever revealing or observing the generated content directly. By combining tools like FHE, PANN, or OPRF, PPPostMark seeks to balance detection, integrity, and privacy in high-risk generative AI deployments.

References

[1] [n. d.]. Google scrambles to manually remove weird AI answers in search.

[2] Mihir Bellare, Viet Tung Hoang, and Phillip Rogaway. 2012. Foundations of garbled circuits. In *Proceedings of the 2012 ACM Conference on Computer and Communications Security* (Raleigh, North Carolina, USA) *(CCS '12)*. Association for Computing Machinery, New York, NY, USA, 784–796. doi:10.1145/2382196.2382279

[3] Zvika Brakerski and Vinod Vaikuntanathan. 2011. Fully Homomorphic Encryption from Ring-LWE and Security for Key Dependent Messages. In *Advances in Cryptology – CRYPTO 2011*, Phillip Rogaway (Ed.). Springer Berlin Heidelberg, Berlin, Heidelberg, 505–524.

[4] Yapei Chang, Kalpesh Krishna, Amir Houmansadr, John Wieting, and Mohit Iyyer. 2024. PostMark: A Robust Blackbox Watermark for Large Language Models. arXiv:2406.14517 [cs.LG] https://arxiv.org/abs/2406.14517

[5] Hao Chen, Ilaria Chillotti, Yihe Dong, Oxana Poburinnaya, Ilya Razenshteyn, and M. Sadegh Riazi. 2020. SANNS: Scaling Up Secure Approximate k-Nearest Neighbors Search. In *29th USENIX Security Symposium (USENIX Security 20)*. USENIX Association, 2111–2128. https://www.usenix.org/conference/usenixsecurity20/presentation/chen-hao

[6] Miranda Christ and Sam Gunn. 2024. Pseudorandom Error-Correcting Codes. arXiv:2402.09370 [cs.CR] https://arxiv.org/abs/2402.09370

[7] Giancarlo Crocetti. 2015. Textual Spatial Cosine Similarity. arXiv:1505.03934 [cs.IR] https://arxiv.org/abs/1505.03934
[8] Anders Dalskov, Daniel Escudero, and Ariel Nof. 2022. Fast Fully Secure Multi-Party Computation over Any Ring with Two-Thirds Honest Majority. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security* (Los Angeles, CA, USA) *(CCS '22)*. Association for Computing Machinery, New York, NY, USA, 653–666. doi:10.1145/3548606.3559389
[9] Jaiden Fairoze, Sanjam Garg, Somesh Jha, Saeed Mahloujifar, Mohammad Mahmoody, and Mingyuan Wang. 2025. Publicly-Detectable Watermarking for Language Models. arXiv:2310.18491 [cs.LG] https://arxiv.org/abs/2310.18491
[10] Yael Gertner, Yuval Ishai, Eyal Kushilevitz, and Tal Malkin. 2000. Protecting Data Privacy in Private Information Retrieval Schemes. *J. Comput. System Sci.* 60, 3 (2000), 592–629. doi:10.1006/jcss.1999.1689
[11] Noah Golowich and Ankur Moitra. 2024. Edit Distance Robust Watermarks for Language Models. arXiv:2406.02633 [cs.CR] https://arxiv.org/abs/2406.02633
[12] Alexandra Henzinger, Emma Dauterman, Henry Corrigan-Gibbs, and Nickolai Zeldovich. 2023. Private Web Search with Tiptoe. Cryptology ePrint Archive, Paper 2023/1438. doi:10.1145/3600006.3613134
[13] John Kirchenbauer, Jonas Geiping, Yuxin Wen, Jonathan Katz, Ian Miers, and Tom Goldstein. 2024. A Watermark for Large Language Models. arXiv:2301.10226 [cs.LG] https://arxiv.org/abs/2301.10226
[14] Vladimir Kolesnikov, Mike Rosulek, Ni Trieu, and Xiao Wang. 2019. Scalable Private Set Union from Symmetric-Key Techniques. Cryptology ePrint Archive, Paper 2019/776. https://eprint.iacr.org/2019/776
[15] Rohith Kuditipudi, John Thickstun, Tatsunori Hashimoto, and Percy Liang. 2023. Robust Distortion-free Watermarks for Language Models. *arXiv preprint arXiv:2307.15593* (2023).
[16] Tsu-Tian Lee and Jin-Tsong Jeng. 1998. The Chebyshev-polynomials-based unified model neural networks for function approximation. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)* 28, 6 (1998), 925–935. doi:10.1109/3477.735405
[17] Jingyu Li, Zhicong Huang, Min Zhang, Jian Liu, Cheng Hong, Tao Wei, and Wenguang Chen. 2024. PANTHER: Private Approximate Nearest Neighbor Search in the Single Server Setting. Cryptology ePrint Archive, Paper 2024/1774. https://eprint.iacr.org/2024/1774
[18] Ryan Little, Lucy Qin, and Mayank Varia. 2024. Secure account recovery for a privacy-preserving web service. In *Proceedings of the 33rd USENIX Conference on Security Symposium* (Philadelphia, PA, USA) *(SEC '24)*. USENIX Association, USA, Article 112, 18 pages.
[19] Daniel Morales, Isaac Agudo, and Javier Lopez. 2023. Private set intersection: A systematic literature review. *Computer Science Review* 49 (2023), 100567. doi:10.1016/j.cosrev.2023.100567
[20] Pranay Mundra, Jianhao Zhang, Fatemeh Nargesian, and Nikolaus Augsten. 2023. KOIOS: Top-k Semantic Overlap Set Search. arXiv:2304.10572 [cs.DB] https://arxiv.org/abs/2304.10572
[21] Srinath Obla, Xinghan Gong, Asma Aloufi, Peizhao Hu, and Daniel Takabi. 2020. Effective Activation Functions for Homomorphic Evaluation of Deep Neural Networks. 153098-153112 pages. doi:10.1109/ACCESS.2020.3017436
[22] Mohammad Raeini. July. Privacy-Preserving Large Language Models (PPLLMs). https://ssrn.com/abstract=4512071
[23] Tom Sander, Pierre Fernandez, Alain Durmus, Matthijs Douze, and Teddy Furon. 2024. Watermarking Makes Language Models Radioactive. arXiv:2402.14904 [cs.CR] https://arxiv.org/abs/2402.14904
[24] Sacha Servan-Schreiber, Simon Langowski, and Srinivas Devadas. 2022. Private Approximate Nearest Neighbor Search with Sublinear Communication. 911–929. doi:10.1109/SP46214.2022.9833702
[25] Hanlin Zhang, Benjamin L. Edelman, Danilo Francati, Daniele Venturi, Giuseppe Ateniese, and Boaz Barak. 2024. Watermarks in the Sand: Impossibility of Strong Watermarking for Generative Models. arXiv:2311.04378 [cs.LG] https://arxiv.org/abs/2311.04378