GuardEmb: Dynamic Watermark for Safeguarding Large Language Model Embedding Service Against Model Stealing Attack

Liaoyaqi Wang
Johns Hopkins University
lwang240@jh.edu

Minhao Cheng
Pennsylvania State University
mmc7149@psu.edu

Abstract

Large language model (LLM) companies provide Embedding as a Service (EaaS) to assist the individual in efficiently dealing with downstream tasks such as text classification and recommendation. However, recent works reveal the risk of the model stealing attack, posing a financial threat to EaaS providers. To protect the copyright of EaaS, we propose GuardEmb, a dynamic embedding watermarking method, striking a balance between enhancing watermark detectability and preserving embedding functionality. Our approach involves selecting special tokens and perturbing embeddings containing these tokens to inject watermarks. Simultaneously, we train a verifier to detect these watermarks. In the event of an attacker attempting to replicate our EaaS for profit, their model inherits our watermarks. For watermark verification, we construct verification texts to query the suspicious EaaS, and the verifier identifies our watermarks within the responses, effectively tracing copyright infringement. Extensive experiments across diverse datasets showcase the high detectability of our watermark method, even in out-of-distribution scenarios, without compromising embedding functionality. Our code is publicly available at https://github.com/Melodramass/Dynamic-Watermark.

1 Introduction

Text embeddings, generated by large language models (LLMs), are concise representations of text derived from high-dimensional spaces (Kashyap et al., 2023). These embeddings have proven essential for a variety of downstream natural language processing tasks, such as text classification, summarization, and translation (Neelakantan et al., 2022). Developing effective LLM embeddings requires substantial computational resources, model advancements, and streamlined training pipelines. Recognizing these challenges, industry leaders like OpenAI offer Embedding as a Service (EaaS) to provide high-quality LLM embeddings to a wider audience. This service enables individuals and organizations to leverage the power of LLMs for various applications without the need for extensive infrastructure or expertise (OpenAI, 2021). However, recent research has highlighted a growing threat: embedding stealing (Dziedzic et al., 2023). This illicit practice involves adversaries fine-tuning a pre-trained language model with a minimal amount of data, enabling them to offer a comparable EaaS to the public for profit, potentially infringing on the original provider's copyright. This raises serious concerns about protecting the intellectual property rights of those who invest in developing and deploying high-quality embedding models.

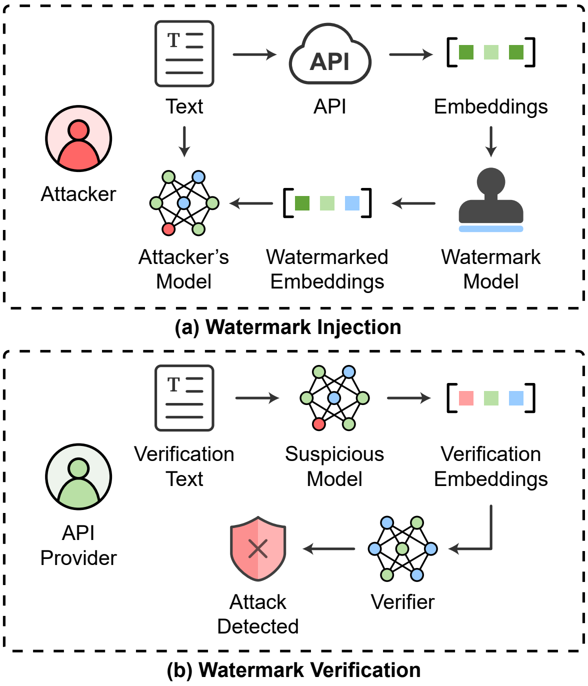


Figure 1: Our watermarking method, GuardEmb presents outstanding ability in infringement detection.

Watermarking has long been a popular technique for protecting the copyright of various content types (Cox et al., 2008). Researchers have also

Findings of the Association for Computational Linguistics: EACL 2024, pages 7518–7534
November 12-16, 2024 ©2024 Association for Computational Linguistics

explored its potential as a defense against model stealing attacks (Jia et al., 2021; Hitaj and Mancini, 2018). However, existing methods primarily focus on proving ownership of the model itself, rather than the specific content generated or encoded by it. While some approaches utilize fine-tuned generative decoders (Fernandez et al., 2023) or content encoders (Zhu et al., 2018; Luo et al., 2020) to embed watermarks directly into the content, these techniques typically require white-box access to the model. This poses a significant challenge for LLMs, which can have billions of parameters, making white-box watermarking impractical (Liu et al., 2023).

Backdoor watermarking can effectively protect model intellectual property by establishing a strong association between specific triggers and target outputs. However, directly using the target embedding as a watermark in the context of EaaS protection presents the challenge of balancing detectability with stealth. Peng et al. (2023) proposed a method to address this by inserting a target embedding into the embeddings of texts containing special tokens, effectively creating a watermark. However, their approach relies on p-value, cosine, and L2 distance differences for watermark detection, introducing instability in real-world scenarios. Furthermore, maintaining the intended functionality of the embeddings becomes difficult as the number of special tokens increases.

In this paper, we address these challenges by introducing GuardEmb, a dynamic watermarking method designed to protect LLM embeddings from theft. We focus on a realistic yet challenging scenario where attackers utilize a stealing dataset that differs in distribution from the watermark training data. GuardEmb achieves two primary objectives: high accuracy in detecting EaaS stealers and maintaining the functionality of watermarked embeddings.

To balance watermark inheritability, detectability, and confidentiality, GuardEmb carefully selects moderately frequent special tokens from a general dataset. A deep network then learns dynamic watermarks that subtly modify embeddings containing these special tokens. Simultaneously, a verifier is trained to detect these watermarks. To ensure effective watermark injection, the process is formulated as an optimization problem, minimizing both the difference between clean and watermarked embeddings and the verifier's error rate. Augmentation techniques further enhance the robustness and generalizability of the watermarks.

If an attacker uses our EaaS to train a stolen model, their service will inadvertently inherit our watermarks. To verify this, we query the suspect EaaS with data from the watermark training distribution and analyze the responses. Using the trained verifier, we compute a watermark likelihood score; a high score indicates a high probability of model stealing.

Extensive experiments demonstrate GuardEmb's effectiveness. We evaluate the utility of watermarked embeddings on downstream tasks and measure detection performance on stolen models. The results show that our watermarks are highly detectable while preserving embedding functionality, even when facing embeddings outside the training distribution. GuardEmb effectively traces copyright infringement with an accuracy exceeding 96% and resists watermark removal attacks, ensuring its safety and effectiveness.

2 Related Work

2.1 Model Stealing Attack

The model stealing attack, also known as a model extraction attack, operates by querying the target model using a surrogate dataset and utilizing the responses as labels to train a similar model (Tramèr et al., 2016; Correia-Silva et al., 2018). This black-box attack poses several potential threats to cloud service providers (Gong et al., 2020). The process of training counterfeit models through extraction is not only cost-efficient compared to starting from scratch but also opens the door to subsequent activities such as membership inference, model inversion, or adversarial attacks (Sanyal et al., 2022; Oliynyk et al., 2023). Researchers have also focused on stealing pretrained encoders (Liu et al., 2022). Furthermore, the stealing of encoders has evolved from supervised learning to self-supervised learning (Dziedzic et al., 2022), increasing the difficulty of defending against model stealing attacks.

2.2 Backdoor Watermarking

Instead of considering the backdoor attack (Gu et al., 2019; Severi et al., 2021) as a threat, backdoors are introduced as an effective watermarking method. The protector incorporates predetermined trigger-target pairs during the training process, and the protected model produces specific outcomes if triggers are activated. This commitment scheme can be employed for ownership verification to safe-

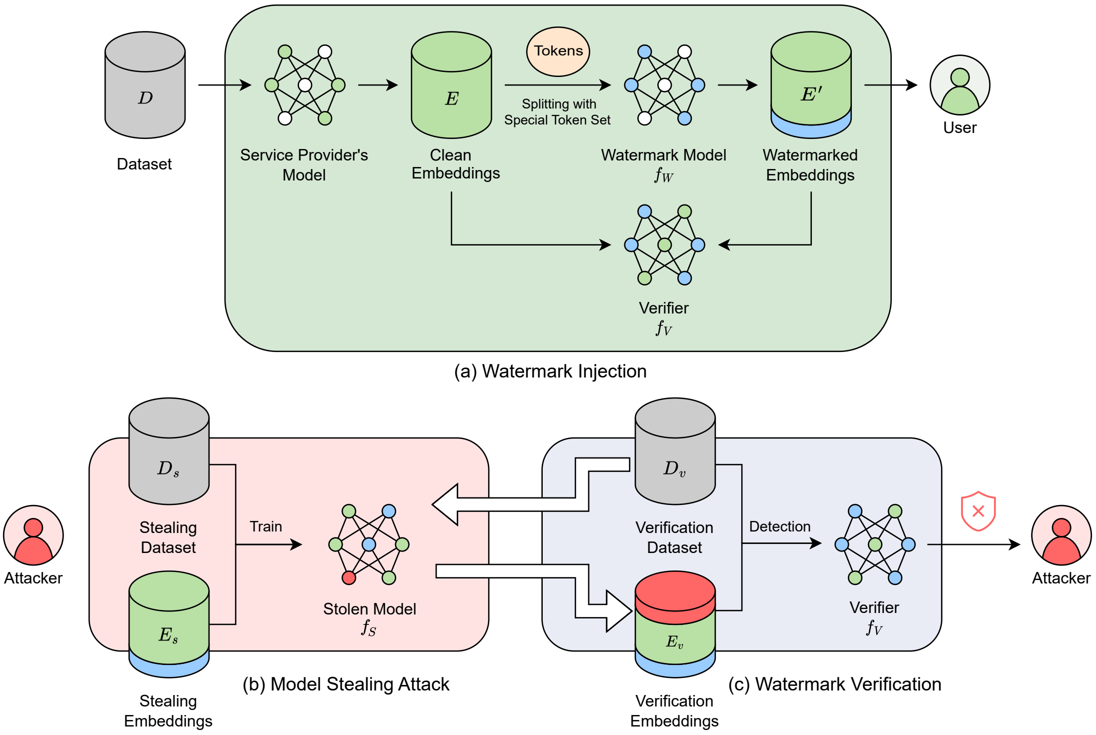


Figure 2: The detailed framework of GuardEmb.

guard model intellectual property (Adi et al., 2018; Zhang et al., 2018). Gu et al. (2023) protect the ownership of Pretrained Language Models (PLM) by backdooring the word embeddings, which cannot be mitigated by fine-tuning. Li et al. (2023) propose a robust watermarking framework for PLM by establishing a strong link between a digital signature and trigger words in the pretrained vocabulary table. Li et al. (2022) explore the untargeted backdoor watermarking scheme for dataset ownership verification. However, these works primarily concentrate on model or dataset protection, disregarding the importance of safeguarding embeddings from the content encoder against model stealing attacks (Liu et al., 2022). Peng et al. (2023) first proposes a method to watermark embeddings by incorporating a target embedding into clean embeddings based on the number of triggers in the text. Our proposed method GuardEmb, however, inject dynamic watermarks into embeddings to preserve the embeddings' utility and jointly train a verifier to accurately trace the watermarks and detect the stealer.

3 Threat model

Consider an Embedding as a Service (EaaS) provider that generates high-quality embeddings, denoted as E, in response to customer queries containing text data D. An adversary aims to replicate the underlying model powering this service by exploiting the relationship between query texts and their corresponding embeddings. Their goal is to launch a model stealing attack, ultimately creating a competing service for profit.

The adversary operates in a challenging environment where they have no direct access to the target model's architecture, training data, or specific watermarking techniques. Forced to operate in a black-box setting, they rely on publicly available information and inferences drawn from querying the EaaS. While possessing the financial means to query the EaaS, the attacker likely has limited computational resources compared to the EaaS provider, restricting their ability to train large-scale models or conduct exhaustive attacks. Crucially, the attacker's training data differs significantly from the data used to train the target model and any associated watermarking scheme. This data mismatch creates a critical vulnerability that the defender can exploit. Furthermore, this motivated attacker is capable of employing various techniques to identify and circumvent watermarking or other protective measures in their pursuit of stealing the EaaS model.

4 GuardEmb

4.1 Overview

In response to the imminent threat of model stealing attacks, we present our robust copyright protection method, named GuardEmb. The watermarks we added only slightly shift the original embedding distribution with a verifier trained to learn this distribution shift pattern. We present more quantitative explanations in Appendix C. As illustrated in Figure 2, GuardEmb comprises two pivotal procedures: Watermark Injection and Watermark Verification, designed to fortify EaaS against potential model theft. In the Watermark Injection process, we initially extract embeddings containing specific tokens earmarked for watermarking. A model f_W is then trained to learn a custom watermark tailored for clean embeddings E. The resulting watermarked embeddings E', when disseminated to customers, act as an effective deterrent against intellectual property infringement. Simultaneously, a verifier f_V is co-trained to identify these watermarked embeddings, crucial for detecting watermarks in response embeddings E_v obtained by querying the verification dataset D_v. The loss from both the watermark model and verifier is optimized concurrently to ensure the creation of stealthy yet detectable watermarks.

In the Watermark Verification phase, if an attacker attempts a model stealing attack to replicate our embedding service, the pirated model inherits the watermarks. The detection of the model stealer, as depicted in Figure 2(c), involves querying the suspicious model with the verification dataset and utilizing the verifier to scrutinize the watermarks from the responses, thereby tracing the model stealer.

Within this protective framework, we operate under two key assumptions concerning the watermark and verification method: 1) the watermark model has access to general datasets, ensuring its effectiveness and robustness across various scenarios; 2) the verifier operates blindly, lacking access to the structure and stealing dataset of the model employed by the attacker. Despite this limitation, the verifier can interact with the suspicious EaaS until a sufficient number of responses are accumulated, allowing for effective watermark detection.

4.2 Watermark Injection

Similarly with Peng et al. (2023), we create the special token set by sampling from tokens of moderate token frequency from a general corpus. This ensures special tokens widely appear in different datasets. E_t is defined as the subset of clean embeddings that correspond to texts containing any special tokens.

The formulation of an effective watermark is a delicate balance between maintaining stealth, preserving functionality, and ensuring detectability by the verifier (Liu et al., 2023). We translate this multi-faceted objective into an optimization problem, comprising two distinct processes within our injection scheme. The first process involves training a watermark model f_W to subtly introduce signals into a subset of clean embeddings. We mandate this model to generate watermarks that adapt to the embedding distribution, thereby preserving embedding functionality. Specifically, we extract E_t based on the special token set for watermarking, leaving the remainder of the embeddings unchanged. This process can be formally expressed as:

$$\mathbf{e}' = \begin{cases} f_W(\mathbf{e};\theta_w) & \text{if } \mathbf{e} \in E_t, \\ \mathbf{e} & \text{if } \mathbf{e} \notin E_t. \end{cases}$$

The embedding $\mathbf{e}'$ will be released to customers via API. We ensure perturbations are minimal to maintain the original functionality of watermarked embeddings, by using similarity loss penalization as a guide. we measure the similarity loss $\mathcal{L}_{\text{sim}}$ between E_t and $f_W(E_t;\theta_w)$ as:

$$\mathcal{L}_{\text{sim}} = \frac{1}{|E_t|} \sum_{\mathbf{e} \in E_t} d_1(\mathbf{e}, f_W(\mathbf{e};\theta_w)), \qquad (1)$$

where d_1 is a distance matrix (like ℓ_2 distance), θ_w represents watermark model weights.

For a robust and effective watermark detection, we jointly train a verifier f_V to identify watermarks within the embeddings. This enhances watermark detectability and offers a straightforward mechanism for detecting potential model stealers. During training, the verifier is exposed to both watermarked embeddings and complete clean embeddings, enabling it to accurately discern previously injected watermarks. Formally, we cast the detection task as a binary classification problem, labeling watermarked embeddings as 'true' and clean embeddings as 'false'. We formulate the loss function for the verifier, denoted as $\mathcal{L}_{\text{ver}}$, as follows:

$$\mathcal{L}_{\text{ver}} = \frac{1}{|E_{\text{ver}}|} \sum_{\mathbf{e} \in E_{\text{ver}}, y \in L} \ell(f_V(\mathbf{e};\theta_v), y), \qquad (2)$$

where $E_{\text{ver}} = E \cup f_W(E_t; \theta_w)$, represents the combined set of both watermarked and clean embeddings, and L is the set of corresponding labels. The term ℓ serves as loss to measure the difference in classification, and θ_v are the parameters of the verifier. During the watermark injection, we simultaneously optimize the similarity loss and classification loss:

$$\min_{\theta_w, \theta_v} \mathcal{L}_{sim} + \lambda \cdot \mathcal{L}_{ver}, \tag{3}$$

which λ is a hyper-parameter that balances the two loss terms. We can solve this optimized problem using standard Stochastic Gradient Descent (SGD) (Robbins and Monro, 1951) to obtain high-quality watermarked embeddings.

To increase the robustness of GuardEmb when facing the embedding out of training distribution, we apply data augmentations into the training procedure. We randomly choose embeddings to employ three augmentation techniques, including adding Gaussian Noise, rounding the embedding values, and zeroing out values below a designated threshold. This strategy further boosts GuardEmb's generality under the more practical and challenging problem setting.

4.3 Watermark Verification

GuardEmb employs a reliable verification method for detecting potential EaaS model stealers. Using the verification dataset D_v, which consists of the test portion from the watermark training datasets, we query the suspicious EaaS and collect the embedding responses E_v. During the model stealing process, the attacker not only replicates the LLM embeddings but also inherits watermarks. Consequently, we leverage the pretrained verifier to detect watermarks within the response embeddings. Following a similar protocol as the verifier training process, the 'true' outcome indicates the successful detection of watermarks within the input embedding. We evaluate the verifier's performance by computing accuracy, recall, and F1 score. A high recall and F1 score for the verifier outcomes suggest a high confidence that the suspicious EaaS is infringing our copyright, given that it has incorporated watermarks through a model-stealing attack. Simultaneously, a high accuracy indicates the effective performance of the verifier.

5 Experiments

5.1 Experimental Setup

Datasets. We conduct comprehensive experiments across multiple widely-used NLP datasets, including SST2 (Socher et al., 2013), MIND (Wu et al., 2020), AGNews (Zhang et al., 2015) and Enron Spam (Metsis et al., 2006). To avoid the dataset bias, the token frequency on the WikiText (Merity et al., 2016) dataset is utilized to sample the special token set as EmbMarker. To have a fair competition with EmbMarker, we employ the embeddings they queried from GPT-3 text-embedding-002 API (Neelakantan et al., 2022) as the clean embeddings.

Implementation Details. We adopt the most practical setting where our watermark training datasets have no overlap with the stealing dataset. To enhance watermark generality, we concatenate multiple datasets during the training. Specifically, we collect four NLP datasets and their corresponding embeddings. For each experiment, we leverage three of them to construct the watermark training dataset and use the remaining one to measure the downstream performance and launch the model stealing attack. We set the size of special token set to be 20, with a token frequency from 0.5% to 2%. Following settings in Peng et al. (2023), we use two layers of MLP for downstream classification and *bert-base-cased* model for stolen model backbone. We utilize a single layer of Transformer encoder (Vaswani et al., 2023) as our watermark model whose hidden layer dimension is equal to the dimension of the GPT embeddings. The verifier is an LSTM model (Hochreiter and Schmidhuber, 1997) with two latent layers, followed by a linear layer. We adopt the Mean Square Error loss function to compute similarity loss and choose the binary Cross-Entropy for the classification loss. AdamW (Loshchilov and Hutter, 2019) is used to co-train the watermark model and verifier in 5 epochs. Our data augmentations randomly select embeddings to apply three techniques. 1) Adding Gaussian Noise $\mathcal{N}(0, \sigma^2)$ to embeddings, where σ^2 is a hyper-parameter listed in Table 10. 2) Rounding: round embedding values to their first decimal place. 3) Thresholding: retain only the embedding values above the threshold of 0.005, with the rest being zeroed out. All embeddings are normalized after augmentation.

As for the watermark verification, we do not deliberately craft the verification dataset. Instead, we

Table 1: Performance comparison of different methods on four datasets.

Dataset	Methods	DPA (%)	Detection Performance		
			Acc (%)	Recall	F1 score
SST2	Original	93.92	94.30	0.0132	0.0259
	EmbMarker	93.58	94.39	0.2553	0.3426
	RedAlarm	93.92	99.58	0.2832	0.4211
	GuardEmb	93.69	96.80	**0.7054**	**0.7161**
MIND	Original	77.58	89.58	0.0009	0.0018
	EmbMarker	77.47	96.28	0.6458	0.7821
	RedAlarm	77.54	65.43	0.9196	0.0348
	GuardEmb	77.51	**98.82**	0.9077	**0.9407**
AGNews	Original	94.03	95.17	0.1138	0.1649
	EmbMarker	94.00	99.88	0.9811	0.9858
	RedAlarm	92.97	99.90	0.7557	0.8426
	GuardEmb	**94.09**	99.83	**0.9832**	0.9799
Enron Spam	Original	95.40	94.10	0.0000	NaN
	EmbMarker	95.50	93.62	0.0957	0.1505
	RedAlarm	94.75	99.52	0.2731	0.3851
	GuardEmb	95.00	96.40	**0.7562**	**0.7127**

concatenate the test part of three training datasets to query the suspicious service and apply the verifier to detect our watermarks from the responses. Further experiment setting and hyper-parameter choices are defined in Appendix A.

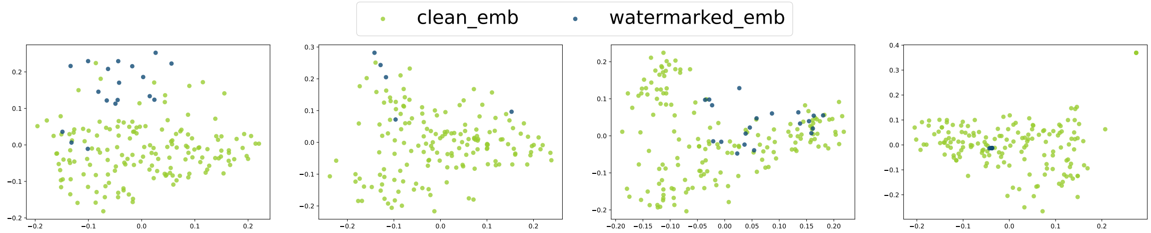
Baselines. We compare GuardEmb with three baselines: 1) Original, where we do not inject any watermarks. The verifier is trained to identify the embedding of text containing special tokens. 2) EmbMarker (Peng et al., 2023), a method to backdoor text embeddings by injecting a designated embedding as the watermark. The weight of insertion increases linearly with the number of trigger words in the text. 3) RedAlarm (Zhang et al., 2023), a method to backdoor pre-trained language models, which returns a pre-defined target embedding when a sentence contains the rare special tokens. We sample special tokens from frequency interval 0.1%-0.2% and view the target embedding as the watermark. We train the verifier to recognize the watermark for EmbMarker and RedAlarm. All baselines follow the same training setting and watermark verification procedure as GuardEmb.

Evaluation Metrics. To evaluate the efficacy of GuardEmb, we measure performance on downstream tasks and watermark detection, reported through the following metrics: 1) **Downstream Performance Accuracy (DPA)**: The functionality of the embeddings when applying to the downstream classification task and reporting the accuracy. 2) **Detection Performance**: The accuracy, recall, and F1 score of the watermark classification results from our verifier.

5.2 Main Results

Table 1 demonstrates the performance comparison of our proposed watermark scheme with other baselines. GuardEmb demonstrates a substantial improvement in detection performance across all datasets compared to EmbMarker and RedAlarm. This is attributed to the joint training of watermarks and the verifier, as well as the careful balance between the two loss components. Our watermarks are deeply embedded into the embeddings and cannot be mitigated during the fine-tuning process by the attacker. At the same time, our downstream performances remain nearly unchanged compared to the original baseline. This is because our watermarks learn and adapt to the distribution of embeddings. The similarity loss term ensures the watermarked embeddings closely align with the original ones.

Embedding Visualization. We employ PCA and t-SNE techniques to visualize high-dimensional embeddings produced by our method. This allows



(a) sst2 (b) mind (c) ag_news (d) enron

Figure 3: The PCA visualization of embeddings when frequency sets to 0.5%-2%

for a better examination of the underlying mechanism and confidential attributes of our watermarks. We plot the clean and watermarked embeddings in Figure 3 where we visualize the embeddings at the frequency interval 0.5%-2% among all datasets. The results illustrates that most of the watermarked embeddings wrap in or surround closely around the benign counterparts, which indicates a shared distribution between the two distrbutions. To be noted, as shown in Figure 3(d), even under high special token frequency, our watermarking scheme still maintains stealthiness, which makes the proposed mechanism hard to be detected by the attacker and maintains good downstream performance. Further visualizations can be found in Appendix B.

5.3 Resistant to Watermark Removal Attack

The adversary may employ various tactics to evade watermark verification after obtaining embeddings from our API. We consider several removal attacks targeting at watermarked embeddings and stolen model itself. Furthermore, we evaluate the impact on embedding utility through DPA for these attacks applied to watermarked embeddings. More results are listed in Appendix E.

Feature Rounding Attack (FRA). Inspired by confidence score rounding in defending against the membership inference attack (Jia et al., 2019), and model inversion attack (Fredrikson et al., 2015), the model stealer could round embeddings into m decimals to erase the watermark. The intuition is that watermark information is embedded in specific features. Rounding features may disrupt the watermark signal, making it hard to detect in the stealing verification process. We test the verification performance on the stolen model after the attacker applies this defense, setting m to $3, 2, 1$. Our results in Table 2 show that our watermarks are not sensitive to the feature granularity.When m is set to 1, the functionality of the embeddings is compromised because the clean embeddings, when rounded to 1 decimal place, suffer a significant DPA drop.

Table 2: The downstream and detection performance after feature rounding to m decimals on SST2.

m	DPA(%)	Detection Performance		
		Acc (%)	Recall	F1 score
4	93.69	96.80	0.7054	0.7161
3	93.81	96.76	0.7009	0.7124
2	93.46	96.85	0.6955	0.7167
1	78.33	94.27	0.0000	NaN

Feature Poisoning Attack (FPA). In this attack, the stealer introduces perturbations to the watermarked embeddings. Given the adversary's lack of access to the specifics of our watermarking method, we suppose that he crafts Gaussian noise with different variances to all embeddings. The experiments in Table 3 show that GuardEmb is not sensitive to random perturbation. A variance that is less than $2e$-2 will not cause detection performance degradation. When an excess of perturbations is added, there is a great risk of damaging embedding functionality. This method is widely used in adversarial attacks to gain sensitive information in membership inference attack (Tramèr et al., 2022) and evade detection in model evasion attack (Biggio et al., 2013).

Table 3: The downstream and detection performance after FPA on SST2.

Variance	DPA(%)	Detection Performance		
		Acc (%)	Recall	F1 score
0	93.69	96.80	0.7054	0.7161
5e-3	94.46	96.68	0.6785	0.7009
1e-2	93.35	96.65	0.6851	0.7007
2e-2	91.74	96.69	0.6760	0.7004
5e-2	85.67	96.30	0.6810	0.6783

Distillation Attack(DA). To evaluate the robustness and resilience of our embedded watermarks

Table 4: The detection performance after DA across four datasets.

Dataset	Detection Performance		
	Acc (%)	Recall	F1 score
SST2	98.17	0.6667	0.8000
MIND	98.68	0.9969	0.8171
AGNews	99.68	0.9741	0.9835
Enron Spam	99.25	0.6216	0.7541

against distillation attacks, we simulate an adversary distilling the stolen model into a smaller "student" model. Specifically, we use the stolen *bert-base-cased* model with 12 encoder layers as the "teacher" and distill it into a 6-layer student model capable of performing the same downstream task. Following the knowledge distillation benchmark (Shah et al., 2020), we train the student model with teacher's output logits and embedding similarity loss. we train the student model using the teacher's output logits and an embedding similarity loss. Additionally, we incorporate a classification head into the student model to maintain its performance on the downstream classification task.

The detection performance of our watermark verification on this distilled model is presented in Table 4. As shown, GuardEmb remains robust against model distillation attacks, achieving detection results comparable to those obtained with the full-sized stolen model on both the Mind and AGNews datasets. These results demonstrate the effectiveness of our watermarks even when the adversary alters the model structure and size through distillation.

5.4 Ablation Study

Special Token Frequency. We evaluate the impact of the special token frequency by sampling an equivalent number of special tokens from different word frequency intervals and adjusting the hyper-parameter λ for optimal watermark performance. The results for the SST2 dataset are detailed in Table 5 and additional results can be found in Appendix D.2. The results show that GuardEmb demonstrates robust performance across a wide range of special token frequencies. It maintains a high downstream accuracy on the middle and high-frequency interval, which is attributed to the increase of similarity loss term penalization. Besides, special tokens from the high-frequency interval contribute to a large number of watermarked embeddings, resulting in improved detection performance. Conversely, we find poor DPA and detection performance on SST2 when the frequency interval is 0.1%-0.2% where the inserted watermark sacrifices DPA to increase detectability. This is attributed to the low watermarking rate leading to an imbalanced label distribution, making it challenging to train a strong verifier. The embedding visualizations also verify our analysis: low-frequency special tokens are more likely to lead to outliers watermarks. Our watermarking scheme is better suited for scenarios with a high special token frequency.

Table 5: The downstream and detection performance under different special token frequencies on SST2.

Frequency	DPA(%)	Detection Performance		
		Acc (%)	Recall	F1 score
0.1%-0.2%	93.12	96.18	0.5426	0.1124
0.5%-2%	93.69	96.80	0.7054	0.7161
2%-5%	93.58	94.65	0.4126	0.5439
10%-20%	93.58	88.94	0.7280	0.7914
20%-50%	93.69	91.71	0.9711	0.9466

Table 6: The results of our methods w/wo data augmentations.

Dataset	DPA (%)	Detection Performance		
		Acc (%)	Recall	F1 score
SST2	93.69	96.80	0.7054	0.7161
No Aug	91.28	97.26	0.7282	0.7529
Mind	77.51	98.82	0.9077	0.9407
No Aug	77.21	98.74	0.8828	0.9355
AGNews	94.09	99.83	0.9832	0.9799
No Aug	86.00	99.86	0.9791	0.9834
Enron Spam	95.00	96.40	0.7562	0.7127
No Aug	94.70	94.64	0.7785	0.6316

Data Augmentation. In this subsection, we investigate the effect of augmentation (aug) techniques for embedding data. Given that traditional text aug techniques are largely incompatible with embedding data, we come up with three methods to modify our embeddings. The aug techniques include thresholding, rounding, and adding Gaussian Noise. The experiment results are presented in Table 6, where we observe that random augmentations increase DPA while having mixed effects on Detection Performance. The watermarks trained without aug drop DPA from 0.3% to 8%. Co-training watermarking and verifying models can lead to overfitting on the training datasets. The aug-

Table 7: The results of different λ on SST2.

λ	**DPA** (%)	**Detection Performance**		
		Acc (%)	Recall	F1 score
5	94.15	94.34	0.0356	0.0672
10	93.69	94.65	0.2065	0.3065
12	93.69	96.80	0.7054	0.7161
15	91.74	97.26	0.7381	0.7550

Table 8: The results of different stealing learning rate on SST2.

Stolen LR	**Detection Performance**		
	Acc (%)	Recall	F1 score
3e-5	95.89	0.5846	0.6196
5e-5	96.40	0.7009	0.6906
1e-4	97.13	0.7348	0.7461
2e-4	97.01	0.7720	0.7471
5e-4	96.96	0.7927	0.7495

mentation we add improves the generalization of the watermark model. Remind that the test dataset has no overlap with watermark training datasets. Consequently, our watermarks can adapt to new embeddings and minimize the impact on downstream classification tasks.

Watermark Loss Weight λ. We demonstrate the impact of λ associated with the watermark binary loss term on downstream and detection performance. We observe that an increase in λ leads to an improvement in detection performance but a reduction in downstream performance. From the visualization on SST2, it is obvious a larger λ compels the watermarked embeddings to evenly distribute along with the clean embeddings.

Fine-tuning Learning Rate. Chen et al. (2021) reveals the risk of watermark removal by using a large learning rate during fine-tuning. Fine-tuning out-of-distribution data with large strides may cause the model to remove the watermark. Therefore, we test the impact of the stealing learning rates and report the results in Table 8 and Appendix D.4. We find that the stealer's model will not forget about the watermarks with the increase in fine-tuning learning rate.

6 Conclusion

In this paper, we develop a learnable embedding watermark method, called GuardEmb, which aims to detect the copyright infringement of LLM embeddings. GuardEmb maintains the embedding utility and improves the accuracy of watermark detection compared to the previous method. We first sample special tokens from the suitable token frequency interval. Then we watermark the embedding corresponding to text that contains special tokens by slightly tuning. Meanwhile, we train a verifier to distinguish the watermarked and clean embeddings. This verifier will be applied to detect the watermarks from suspicious embeddings we query from the API, returning the confidence score about the watermark's existence. Various experiments demonstrate the effect and robustness of GuardEmb.

Limitations

In this paper, we introduce GuardEmb, a dynamic embedding watermarking method designed to safeguard LLM Embedding as a Service (EaaS). Our experiments in Section 5.3 underscore GuardEmb's resilience against two watermark removal attacks. However, we acknowledge a potential limitation: if the existence or methodology of the watermarks becomes widely known, adversaries may refine their strategies, posing a challenge to GuardEmb's efficacy. For example, if the attacker applies some similarity invariant attacks (e.g. dimension-shift attacks, moving the last dimension of embeddings to the front), our previous verification techniques become ineffective. To proactively address this, we advocate equipping adversaries with enhanced capabilities and knowledge of the watermarking process, enabling a thorough assessment of GuardEmb's resilience against more sophisticated attacks.

Furthermore, the current watermark verification process involves querying the potential EaaS with a thousand-level dataset to retrieve responses. Ideally, the detection of unauthorized use should necessitate fewer queries, preferably fewer than 100, to ensure efficiency and reduce verification costs. Another intrinsic limitation of GuardEmb is its ability to detect past or ongoing attacks, akin to ownership verification, rather than actively prohibiting attacks. Recognizing this constraint, we plan to explore these areas further in future investigations.

References

Yossi Adi, Carsten Baum, Moustapha Cisse, Benny Pinkas, and Joseph Keshet. 2018. Turning your weakness into a strength: Watermarking deep neural networks by backdooring. In *27th USENIX Security Symposium (USENIX Security 18)*, pages 1615–1631, Baltimore, MD. USENIX Association.

Battista Biggio, Igino Corona, Davide Maiorca, Blaine Nelson, Nedim Šrndić, Pavel Laskov, Giorgio Giacinto, and Fabio Roli. 2013. *Evasion Attacks against Machine Learning at Test Time*, page 387–402. Springer Berlin Heidelberg.

Xinyun Chen, Wenxiao Wang, Chris Bender, Yiming Ding, Ruoxi Jia, Bo Li, and Dawn Song. 2021. Refit: A unified watermark removal framework for deep learning systems with limited data. In *Proceedings of the 2021 ACM Asia Conference on Computer and Communications Security*, ASIA CCS '21. ACM.

Jacson Rodrigues Correia-Silva, Rodrigo F. Berriel, Claudine Badue, Alberto F. de Souza, and Thiago Oliveira-Santos. 2018. Copycat cnn: Stealing knowledge by persuading confession with random non-labeled data. In *2018 International Joint Conference on Neural Networks (IJCNN)*. IEEE.

Ingemar J. Cox, Matthew L. Miller, Jeffrey A. Bloom, Jessica Fridrich, and Ton Kalker. 2008. Digital watermarking and steganography (second edition). In Ingemar J. Cox, Matthew L. Miller, Jeffrey A. Bloom, Jessica Fridrich, and Ton Kalker, editors, *Digital Watermarking and Steganography (Second Edition)*, second edition edition, The Morgan Kaufmann Series in Multimedia Information and Systems, pages 105–135. Morgan Kaufmann, Burlington.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. Bert: Pre-training of deep bidirectional transformers for language understanding.

Adam Dziedzic, Franziska Boenisch, Mingjian Jiang, Haonan Duan, and Nicolas Papernot. 2023. Sentence embedding encoders are easy to steal but hard to defend. In *ICLR 2023 Workshop on Pitfalls of limited data and computation for Trustworthy ML*.

Adam Dziedzic, Nikita Dhawan, Muhammad Ahmad Kaleem, Jonas Guan, and Nicolas Papernot. 2022. On the difficulty of defending self-supervised learning against model extraction.

Pierre Fernandez, Guillaume Couairon, Hervé Jégou, Matthijs Douze, and Teddy Furon. 2023. The stable signature: Rooting watermarks in latent diffusion models.

Matt Fredrikson, Somesh Jha, and Thomas Ristenpart. 2015. Model inversion attacks that exploit confidence information and basic countermeasures. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, CCS '15, page 1322–1333, New York, NY, USA. Association for Computing Machinery.

Xueluan Gong, Qian Wang, Yanjiao Chen, Wang Yang, and Xinchang Jiang. 2020. Model extraction attacks and defenses on cloud-based machine learning models. *IEEE Communications Magazine*, 58(12):83–89.

Chenxi Gu, Chengsong Huang, Xiaoqing Zheng, Kai-Wei Chang, and Cho-Jui Hsieh. 2023. Watermarking pre-trained language models with backdooring.

Tianyu Gu, Brendan Dolan-Gavitt, and Siddharth Garg. 2019. Badnets: Identifying vulnerabilities in the machine learning model supply chain.

Dorjan Hitaj and Luigi V. Mancini. 2018. Have you stolen my model? evasion attacks against deep neural network watermarking techniques.

Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural computation*, 9:1735–80.

Hengrui Jia, Christopher A. Choquette-Choo, Varun Chandrasekaran, and Nicolas Papernot. 2021. Entangled watermarks as a defense against model extraction.

Jinyuan Jia, Ahmed Salem, Michael Backes, Yang Zhang, and Neil Zhenqiang Gong. 2019. Memguard: Defending against black-box membership inference attacks via adversarial examples.

Abhinav Ramesh Kashyap, Thanh-Tung Nguyen, Viktor Schlegel, Stefan Winkler, See-Kiong Ng, and Soujanya Poria. 2023. Beyond words: A comprehensive survey of sentence representations.

Peixuan Li, Pengzhou Cheng, Fangqi Li, Wei Du, Haodong Zhao, and Gongshen Liu. 2023. Plmmark: A secure and robust black-box watermarking framework for pre-trained language models. *Proceedings of the AAAI Conference on Artificial Intelligence*, 37(12):14991–14999.

Yiming Li, Yang Bai, Yong Jiang, Yong Yang, Shu-Tao Xia, and Bo Li. 2022. Untargeted backdoor watermark: Towards harmless and stealthy dataset copyright protection. In *Advances in Neural Information Processing Systems*, volume 35, pages 13238–13250. Curran Associates, Inc.

Aiwei Liu, Leyi Pan, Yijian Lu, Jingjing Li, Xuming Hu, Lijie Wen, Irwin King, and Philip S. Yu. 2023. A survey of text watermarking in the era of large language models.

Yupei Liu, Jinyuan Jia, Hongbin Liu, and Neil Zhenqiang Gong. 2022. Stolenencoder: Stealing pre-trained encoders in self-supervised learning. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, CCS '22, page 2115–2128, New York, NY, USA. Association for Computing Machinery.

Ilya Loshchilov and Frank Hutter. 2019. Decoupled weight decay regularization.

Xiyang Luo, Ruohan Zhan, Huiwen Chang, Feng Yang, and Peyman Milanfar. 2020. Distortion agnostic deep watermarking. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*.

Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. 2016. Pointer sentinel mixture models.

Vangelis Metsis, Ion Androutsopoulos, and Georgios Paliouras. 2006. Spam filtering with naive bayes - which naive bayes? In *International Conference on Email and Anti-Spam*.

Arvind Neelakantan, Tao Xu, Raul Puri, Alec Radford, Jesse Michael Han, Jerry Tworek, Qiming Yuan, Nikolas Tezak, Jong Wook Kim, Chris Hallacy, Johannes Heidecke, Pranav Shyam, Boris Power, Tyna Eloundou Nekoul, Girish Sastry, Gretchen Krueger, David Schnurr, Felipe Petroski Such, Kenny Hsu, Madeleine Thompson, Tabarak Khan, Toki Sherbakov, Joanne Jang, Peter Welinder, and Lilian Weng. 2022. Text and code embeddings by contrastive pre-training.

Daryna Oliynyk, Rudolf Mayer, and Andreas Rauber. 2023. I know what you trained last summer: A survey on stealing machine learning models and defences. *ACM Comput. Surv.*, 55(14s).

OpenAI. 2021. Openai api.

Wenjun Peng, Jingwei Yi, Fangzhao Wu, Shangxi Wu, Bin Bin Zhu, Lingjuan Lyu, Binxing Jiao, Tong Xu, Guangzhong Sun, and Xing Xie. 2023. Are you copying my model? protecting the copyright of large language models for EaaS via backdoor watermark. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 7653–7668, Toronto, Canada. Association for Computational Linguistics.

Herbert Robbins and Sutton Monro. 1951. A Stochastic Approximation Method. *The Annals of Mathematical Statistics*, 22(3):400 – 407.

Sunandini Sanyal, Sravanti Addepalli, and R. Venkatesh Babu. 2022. Towards data-free model stealing in a hard label setting.

Giorgio Severi, Jim Meyer, Scott Coull, and Alina Oprea. 2021. Explanation-Guided backdoor poisoning attacks against malware classifiers. In *30th USENIX Security Symposium (USENIX Security 21)*, pages 1487–1504. USENIX Association.

Het Shah, Avishree Khare, Neelay Shah, and Khizir Siddiqui. 2020. Kd-lib: A pytorch library for knowledge distillation, pruning and quantization.

Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D. Manning, Andrew Ng, and Christopher Potts. 2013. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 1631–1642, Seattle, Washington, USA. Association for Computational Linguistics.

Florian Tramèr, Fan Zhang, Ari Juels, Michael K. Reiter, and Thomas Ristenpart. 2016. Stealing machine learning models via prediction APIs. In *25th USENIX Security Symposium (USENIX Security 16)*, pages 601–618, Austin, TX. USENIX Association.

Florian Tramèr, Reza Shokri, Ayrton San Joaquin, Hoang Le, Matthew Jagielski, Sanghyun Hong, and Nicholas Carlini. 2022. Truth serum: Poisoning machine learning models to reveal their secrets.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2023. Attention is all you need.

Fangzhao Wu, Ying Qiao, Jiun-Hung Chen, Chuhan Wu, Tao Qi, Jianxun Lian, Danyang Liu, Xing Xie, Jianfeng Gao, Winnie Wu, and Ming Zhou. 2020. MIND: A large-scale dataset for news recommendation. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 3597–3606, Online. Association for Computational Linguistics.

Jialong Zhang, Zhongshu Gu, Jiyong Jang, Hui Wu, Marc Ph. Stoecklin, Heqing Huang, and Ian Molloy. 2018. Protecting intellectual property of deep neural networks with watermarking. In *Proceedings of the 2018 on Asia Conference on Computer and Communications Security*, ASIACCS '18, page 159–172, New York, NY, USA. Association for Computing Machinery.

Xiang Zhang, Junbo Zhao, and Yann LeCun. 2015. Character-level convolutional networks for text classification. In *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 1*, NIPS'15, page 649–657, Cambridge, MA, USA. MIT Press.

Zhengyan Zhang, Guangxuan Xiao, Yongwei Li, Tian Lv, Fanchao Qi, Zhiyuan Liu, Yasheng Wang, Xin Jiang, and Maosong Sun. 2023. Red alarm for pre-trained models: Universal vulnerability to neuron-level backdoor attacks. *Machine Intelligence Research*, 20(2):180–193.

Jiren Zhu, Russell Kaplan, Justin Johnson, and Li Fei-Fei. 2018. Hidden: Hiding data with deep networks. In *Proceedings of the European Conference on Computer Vision (ECCV)*.

A Experiment Setting

We provide further details about the experimental settings and list the complete set of hyperparameters in Table 10.

A.1 Watermark Setting

In our experiment, we selected a transformer encoder layer to inject watermarks to embeddings linked with special tokens. The verifier is an LSTM model with two latent layers followed by a linear transformation to generate the possibility of the watermark. We investigate the impact of the structure choice through ablation experiments in Table 9. The result finds that RNN and Attention structure perform better on this task compared to MLP. We set a smaller learning rate of $3e$-4 for the watermark model and a larger one of $5e$-4 for the verifier, forcing the smaller embedding perturbation but quicker detection.

A.2 Downstream Classifier Setting

We measure the functionality of the embeddings by applying them to the downstream classification task. The classifier is a two-layer MLP with a ReLU activation function and a dropout layer. Cross-entropy loss is utilized for training the classifier. We apply the early stop technique, in which the training process continues until the validation loss exceeds three times the current minimal validation loss.

A.3 Attacker Setting

We denote the attacker who queries our EaaS API with natural language data D_s to gain embeddings E_s for training a pirated embedding service f_S. Assuming the attacker employs Mean Square Error (MSE) to compute the stealing loss term, the optimization problem can be formulated as follows:

$$\min_{\theta_s} \mathcal{L}_{\text{steal}} = \frac{1}{|E_s|} \sum_{\mathbf{d},\mathbf{e}\in D_s,E_s} \|f_S(\mathbf{d};\theta_s),\mathbf{e}\|_2^2,$$

where θ_s represents the parameter in the attacker's unauthorized embedding service.

During our experiment, we utilize the pretrained language model Bert (Devlin et al., 2019) to execute the model stealing attack. We applied a 2-layer MLP to transform the output of the [CLS] token to the dimension of the target embedding. During training, we set a smaller learning rate for the pretrained module and a larger one for the linear transform module.

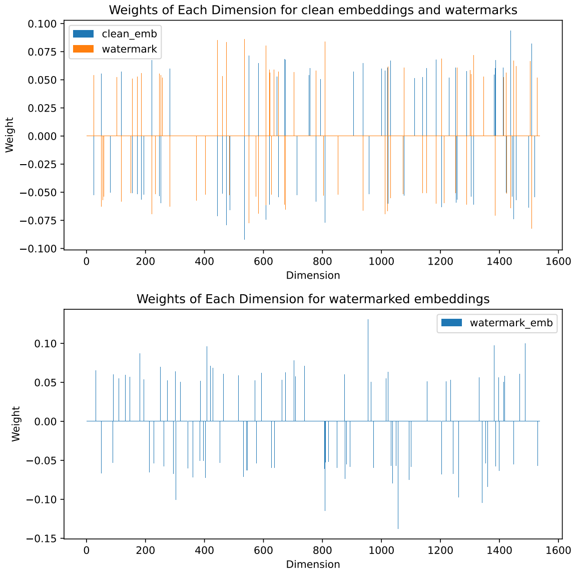


Figure 4: The PCA analysis of embeddings and watermarks

B Embedding Visualization

In this section, we present the embedding visualization by the PCA and t-SNE algorithms. We observe that watermarked embeddings are closely wrapped in clean embeddings, indicating a minimal distance between the watermarked and clean embeddings within the latent space. It is interesting to observe that as the token frequency increases, the watermarked embeddings on Mind approach the benign ones because the watermarked embeddings have to be conspicuous to keep the detectability when the special token frequency is low.

C Quantitative Analysis of Watermarks

We present the nature of watermarks from a quantitative perspective here. We propose a hypothesis that our watermarks only modify unimportant dimensions of clean embeddings by analyzing clean embeddings, watermarked embeddings, and the differences between them (i.e., watermarks) with the PCA algorithm. We visualize the weight of each dimension to the most dominant component and zero out the weights under 0.05. In the Figure 4, we discover that many high-weighted dimensions between clean embeddings and watermarks are dissimilar, indicating that our watermarks primarily modify dimensions with lower contributions to the clean embeddings. Thus, we can embed detectable watermarks in the embeddings while preserving their functionality.

Table 9: The impact of the watermark structure choice on SST2

Module	Layer	DPA (%)	Detection Performance		
			acc (%)	recall	F1 score
Watermark	Transformer Encoder	93.69	96.80	0.7054	0.7161
	MLP	93.00	94.12	0.0000	NaN
Verifier	LSTM	93.69	96.80	0.7054	0.7161
	MLP	93.35	95.57	0.3753	0.4925

D Hyper-parameter Setting

D.1 Impact of the verification dataset size

Small queries can also be effective during the watermark verification process. In the experiment, we assess the watermarks with randomly sampled subsets from the original verification dataset. The experiment outcomes in Appendix D.1 illustrate how the size of the verification dataset impacts performance. As the dataset size decreases, we observed a slight decline in detection performance, but the detection rate still remained high. Notably, datasets like SST2 and AGNews achieved high scores with a dataset size of 100, owing to the infrequent presence of special tokens.

D.2 Impact of the Special Token Frequency

We present the impact of special token frequency on other datasets in Table 12, where we address the almost same conclusion as 5.4.

D.3 Impact of the Hyper-parameter λ

We conduct the experiments to evaluate the impact on other datasets in addition to SST2. The outcomes show that both downstream and detection performance remain high on AGNews with the increase of the λ. We also observe significant performance improvement on the Enron Spam dataset. Because the watermarking ratio is low therefore requires high λ to make watermarks detective.

D.4 Impact of the Stealing Learning Rate

In this subsection, we illustrate the impact of the stealing learning rate in Tables 16 to 18. To retrieve better model stealing performance, we set a small learning rate for the pretrained model and a larger one for the linear transform module. We find that on Mind, AGNews and Enron Spam datasets, the stealing learning rate 5e-4 adversely affects detection performance, possibly due to the learning rate being too large for effective model fine-tuning. While on Enron, the small stealing learning rate fails to steal the embeddings, leading to the low similarity between the stolen embeddings and watermarked embeddings. The low similarity makes it challenging for the verifier to detect the watermarks.

E Watermark Removal Attack

We demonstrate our watermark method can withstand watermark removal attacks on various datasets in addition to SST2. The conclusion aligns with that of Section 5.3: these attacks cannot erase the watermark without compromising the utility of embeddings. Tables 19 to 21 show the watermark verification performance after the he casts the feature rounding attack to watermarked embeddings. Tables 22 to 24, show the watermark verification performance when he launches the feature poisoning attack.

F Experimental Environments

We conduct experiments on a Linux server with CentOS Linux. The server has seven NVIDIA GeForce RTX 3090 and one NVIDIA GeForce RTX 4090 with CUDA 12.2. We use pytorch 2.0.1.

Table 10: The full hyper-parameter setting in our experiments.

Module	Hyper-parameter	SST2	Mind	AGNews	Enron
Watermark	learning rate	3e-4	3e-4	3e-4	3e-4
	λ	12	5	10	15
	batch size	32	32	32	32
	σ^2	0.01	0.01	0.05	0.01
Verifier	learning rate	5e-4	5e-4	5e-4	5e-4
	hidden layer	64	64	64	64
	dropout rate	0.5	0.5	0.5	0.5
Downstream Classifier	learning rate	2e-3	2e-3	2e-3	2e-3
	batch size	32	32	32	32
	hidden layer	256	256	256	256
	dropout rate	0.2	0.2	0.2	0.2
Stolen Model	batch size	32	32	32	32
	hidden layer	1536	1536	1536	1536

Table 11: The detection performance under different size of verification datasets. The items in the table are Accuracy, Recall, and F1 Score respectively.

Size	SST2	Mind	AGNews	Enron Spam
100	100.0% | 1.0 | 1.0	99.0% | 0.50 | 0.67	100.0% | 1.0 | 1.0	97.0% | 1.0 | 0.57
1000	98.4% | 0.70 | 0.61	98.3% | 0.66 | 0.78	99.3% | 0.81 | 0.90	96.3% | 0.83 | 0.64
5000	98.4% | 0.74 | 0.71	97.8% | 0.88 | 0.87	99.7% | 0.91 | 0.95	97.5% | 0.87 | 0.69
10000	98.6% | 0.78 | 0.75	97.9% | 0.90 | 0.88	99.8% | 0.94 | 0.97	98.0% | 0.94 | 0.97

Table 12: The downstream and detection performance under different special token frequencies

Dataset	Frequency Interval	DPA (%)	Detection Performance		
			acc (%)	recall	F1 score
Mind	0.1%-0.2%	77.66	99.49	0.5657	0.6788
	0.5%-2%	77.51	98.82	0.9077	0.9407
	2%-5%	77.58	96.63	0.8168	0.8960
	10%-20%	77.66	98.18	0.9720	0.9845
	20%-50%	77.52	84.94	0.8888	0.9147
AGNews	0.1%-0.2%	93.91	99.96	0.9459	0.9417
	0.5%-2%	94.09	99.83	0.9832	0.9799
	2%-5%	93.99	98.79	0.8351	0.8843
	10%-20%	94.07	99.43	0.9794	0.9862
	20%-50%	94.09	71.02	1.000	0.8306
Enron	0.1%-0.2%	95.85	99.61	0.1708	0.2870
	0.5%-2%	95.00	96.40	0.7562	0.7127
	2%-5%	93.10	95.03	0.7507	0.6906
	10%-20%	95.20	91.64	0.9795	0.8769
	20%-50%	95.95	86.63	0.9992	0.9202

Table 13: The impact of lambda on Mind.

λ	**DPA** (%)	**Detection Performance**		
		acc (%)	recall	F1 score
3	77.54	98.77	0.8902	0.9373
5	77.51	98.82	0.9077	0.9407
7	77.67	98.69	0.8819	0.9331
10	76.24	96.70	0.7325	0.8211

Table 14: The impact of lambda on AGNews.

λ	**DPA** (%)	**Detection Performance**		
		acc (%)	recall	F1 score
5	93.80	99.88	0.9811	0.9858
7	93.96	99.86	0.9832	0.9828
10	94.09	99.83	0.9832	0.9799
15	93.80	99.89	0.9811	0.9865

Table 15: The impact of lambda on Enron.

λ	**DPA** (%)	**Detection Performance**		
		acc (%)	recall	F1 score
10	95.70	94.10	0.0000	NaN
12	95.60	94.44	0.2624	0.3579
15	95.00	96.40	0.7562	0.7127
20	95.05	95.73	0.8106	0.6915

Table 16: The impact of the stealing learning rate on Mind.

Stolen LR	**Detection Performance**		
	acc (%)	recall	F1 score
3e-5	96.96	0.7122	0.8292
5e-5	98.94	0.9050	0.9465
1e-4	98.83	0.9142	0.9416
2e-4	98.79	0.9077	0.9394
5e-4	89.65	0.0000	NaN

Table 17: The impact of the stealing learning rate on AGNews.

Stolen LR	**Detection Performance**		
	acc (%)	recall	F1 score
3e-5	99.88	0.9811	0.9861
5e-5	99.88	0.9785	0.9858
1e-4	99.90	0.9811	0.9881
2e-4	99.89	0.9852	0.9865
5e-4	96.81	0.0000	NaN

Table 18: The impact of the stealing learning rate on Enron.

Stolen LR	**Detection Performance**		
	acc (%)	recall	F1 score
3e-5	94.10	0.0000	NaN
5e-5	94.10	0.0000	NaN
1e-4	95.61	0.3214	0.4636
2e-4	97.88	0.8498	0.8258
5e-4	94.10	0.0000	NaN

Table 19: The downstream and detection performance after FRA on Mind.

m	**DPA**(%)	**Detection Performance**		
		acc (%)	recall	F1 score
4	77.51	98.67	0.9004	0.9335
3	77.65	98.96	0.9253	0.9485
2	77.56	98.72	0.9041	0.9360
1	61.36	89.66	0.0009	0.0018

Table 20: The downstream and detection performance after FRA on AGNews.

m	**DPA**(%)	**Detection Performance**		
		acc (%)	recall	F1 score
4	94.09	99.88	0.9832	0.9858
3	93.54	99.86	0.9825	0.9838
2	93.28	99.88	0.9832	0.9858
1	73.62	96.81	0.0000	NaN

Table 21: The downstream and detection performance after FRA on Enron.

m	**DPA**(%)	**Detection Performance**		
		acc (%)	recall	F1 score
4	95.00	96.41	0.7467	0.7104
3	95.20	96.61	0.7145	0.7131
2	95.25	96.49	0.7335	0.7116
1	85.90	94.10	0.0000	NaN

Table 22: The downstream and detection performance after FPA on Mind.

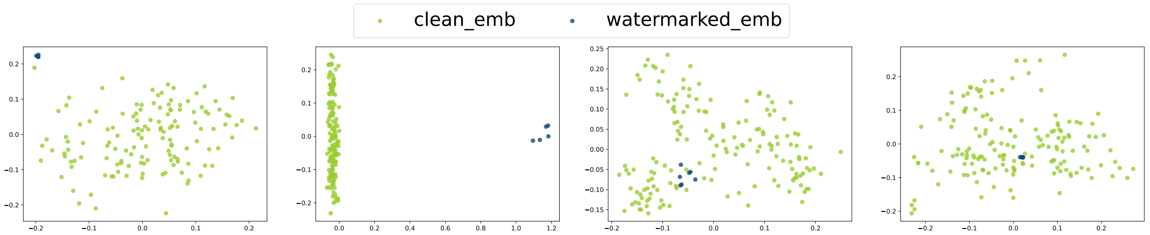
Variance	**DPA**(%)	**Detection Performance**		
		acc (%)	recall	F1 score
5e-3	77.32	98.67	0.9004	0.9335
1e-2	76.98	98.77	0.9031	0.9382
2e-2	75.63	98.74	0.8976	0.9365
5e-2	69.46	96.81	0.7011	0.8198

Table 23: The downstream and detection performance after FPA on AGNews.

Variance	**DPA**(%)	**Detection Performance**		
		acc (%)	recall	F1 score
5e-3	93.25	99.88	0.9832	0.9858
1e-2	93.01	99.90	0.9832	0.9875
2e-2	91.42	99.86	0.9832	0.9828
5e-2	83.32	99.88	0.9811	0.9855

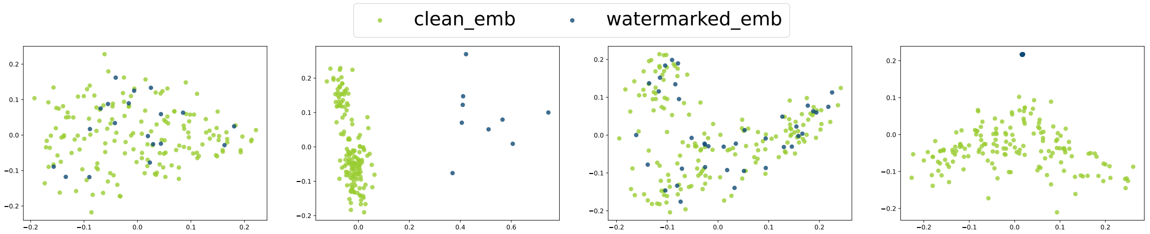
Table 24: The downstream and detection performance after FPA on Enron.

Variance	**DPA**(%)	**Detection Performance**		
		acc (%)	recall	F1 score
5e-3	95.55	96.41	0.7467	0.7104
1e-2	93.75	95.62	0.7488	0.6688
2e-2	91.80	95.77	0.6807	0.6554
5e-2	84.50	94.10	0.0000	NaN



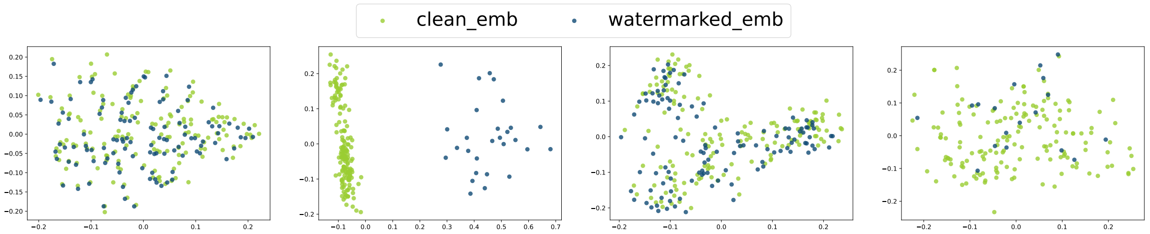
(a) sst2 (b) mind (c) ag_news (d) enron

Figure 5: The PCA visualization of embeddings when the special token frequency sets to 0.1%-0.2%



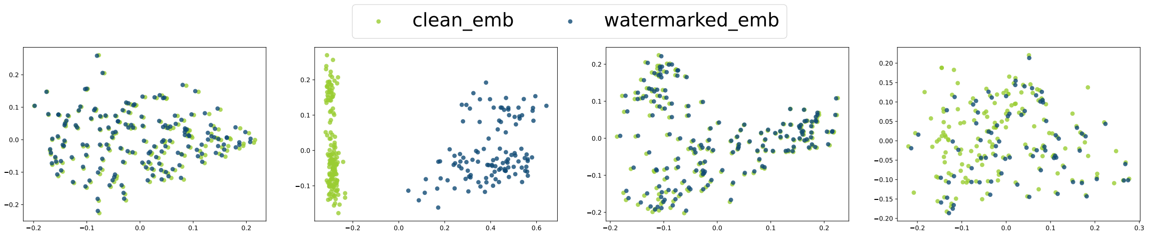
(a) sst2 (b) mind (c) ag_news (d) enron

Figure 6: The PCA visualization of embeddings when the special token frequency sets to 2%-5%



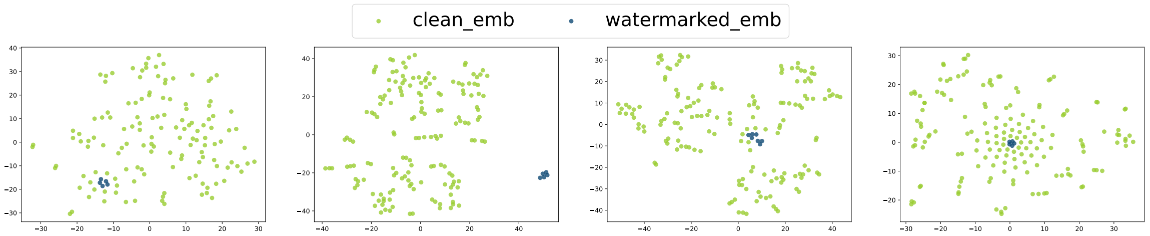
(a) sst2 (b) mind (c) ag_news (d) enron

Figure 7: The PCA visualization of embeddings when the special token frequency sets to 10%-20%



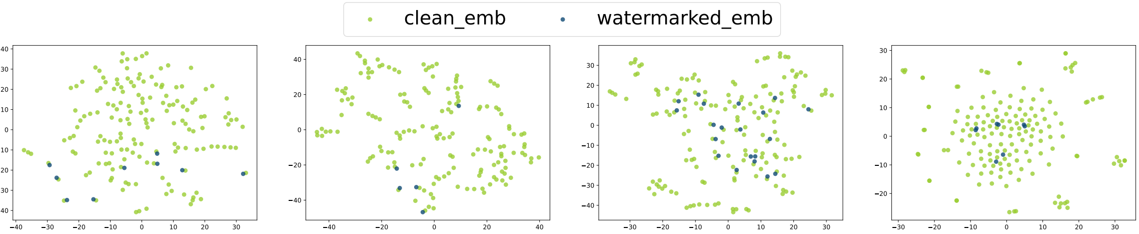
(a) sst2 (b) mind (c) ag_news (d) enron

Figure 8: The PCA visualization of embeddings when the special token frequency sets to 20%-50%



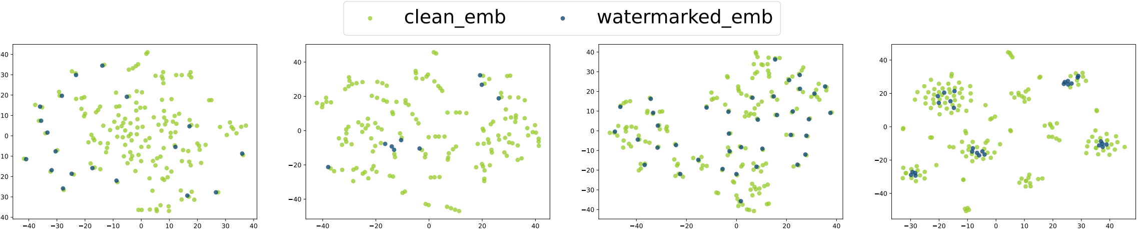
(a) sst2 (b) mind (c) ag_news (d) enron

Figure 9: The t-SNE visualization of embeddings when the special token frequency sets to 0.1%-0.2%



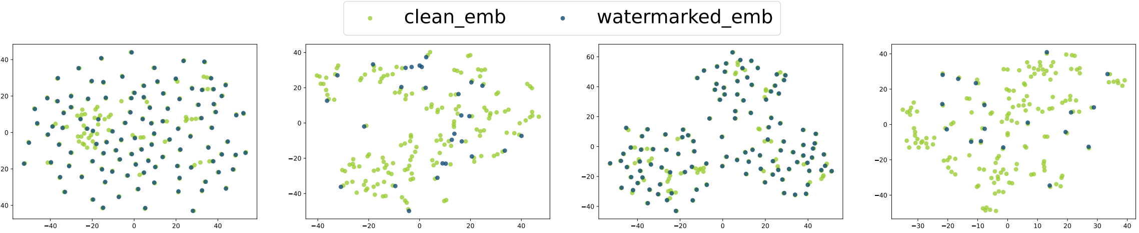
(a) sst2 (b) mind (c) ag_news (d) enron

Figure 10: The t-SNE visualization of embeddings when the special token frequency sets to 0.5%-2%



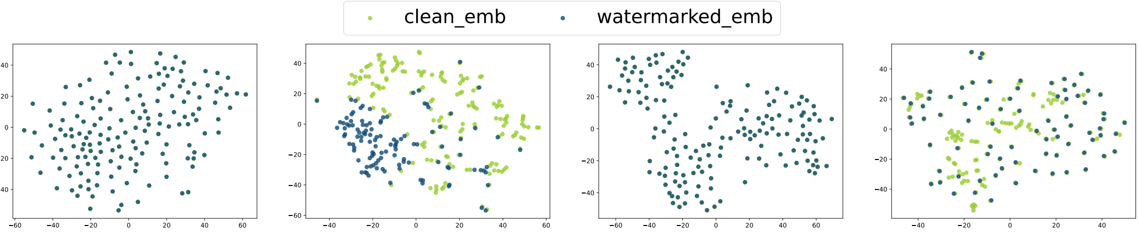
(a) sst2 (b) mind (c) ag_news (d) enron

Figure 11: The t-SNE visualization of embeddings when the special token frequency sets to 2%-5%



(a) sst2 (b) mind (c) ag_news (d) enron

Figure 12: The t-SNE visualization of embeddings when the special token frequency sets to 10%-20%



(a) sst2 (b) mind (c) ag_news (d) enron

Figure 13: The t-SNE visualization of embeddings when the special token frequency sets to 20%-50%