The Challenge of Identifying the Origin of Black-Box Large Language Models

Ziqing Yang[1] **Yixin Wu**[1] **Yun Shen**[2] **Wei Dai**[3] **Michael Backes**[1] **Yang Zhang**[1*]

[1]*CISPA Helmholtz Center for Information Security* [2]*HPE* [3]*TikTok Inc.*

ziqing.yang@cispa.de, ohlecode@gmail.com, yun.shen@hpe.com, weidai3141@gmail.com, {director, zhang}@cispa.de

Abstract

The tremendous commercial potential of large language models (LLMs) has heightened concerns over their unauthorized use. To address this, we focus on the task of identifying the origin of black-box LLMs. We further propose PlugAE, an effective and efficient identification method that proactively leverages LLM-specific adversarial embeddings and allows users to customize copyright tokens on a targeted query set. Extensive experiments demonstrate that PlugAE outperforms both state-of-the-art model watermarking and fingerprinting methods in accuracy and robustness. We further analyze its stealthiness and reliability from three complementary perspectives and conduct ablation studies under various configurations, confirming its practicality for real-world misuse detection.[1]

1 Introduction

Large language models (LLMs) have become integral parts in many domains (Touvron et al., 2023a; Jiang et al., 2023; Team, 2024; Groeneveld et al., 2024). Their commercial potential has led to increasing concerns over their unauthorized use, raising broader issues of fairness, accountability, and governance. Identifying the origin of a black-box LLM emerges as an intrinsic solution to the problem. Specifically, we refer to the LLM under the defender's control as the *candidate model*, and the black-box LLM under investigation as the *suspect model*. Our goal is to identify whether the suspect LLM is derived from the candidate LLM, as shown in Figure 1. Compared with *only* identifying the candidate LLM without considering its derivatives, our task is more practical and challenging, as the malicious third party can further fine-tune the candidate LLM for specific usage.

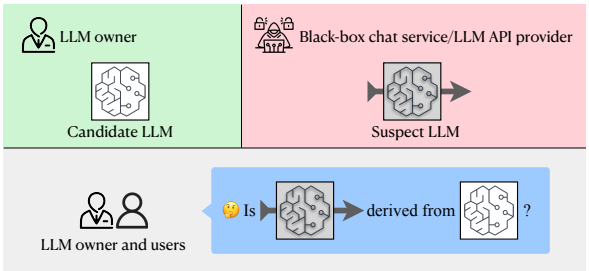


Figure 1: Scenario of black-box LLM identification, where the owner aims to verify unauthorized use of their LLM in a black-box chat service.

There have been several related works on LLM identification, which can be roughly divided into model watermarking and fingerprinting.[2] Following Cao et al. (2021); Xu et al. (2025); Ye et al. (2026), we consider the work as model watermarking when they require the model owner to embed signals into the model proactively, e.g., via backdooring (Gu et al., 2022; Li et al., 2023; Xu et al., 2024); otherwise, we consider them as fingerprinting (Diwan et al., 2021; Zeng et al., 2023; Gubri et al., 2024; Jin et al., 2024), e.g., optimizing adversarial prompts for verification. However, backdoor-based model watermarking methods inevitably alter the model's parameters and are computationally expensive. Nevertheless, existing fingerprinting methods fall short in circumstances where they only have black-box access to the suspect model.

To address these challenges, we propose PlugAE, which proactively **Plug**s the **A**dversarial token **E**mbeddings into the LLM's token-embedding layer for identification, leaving the main transformer layers unchanged. As shown in Figure 2, PlugAE first optimizes a sequence of adversarial token embeddings on a pre-defined query set Q based on the candidate LLM. Then, PlugAE plugs these token embeddings paired with customized copyright tokens into the model's token-embedding

[*]Corresponding author.

[1]Our code is available at https://github.com/TrustAIRLab/PlugAE.

[2]The terminology in this area is not always consistent.

Findings of the Association for Computational Linguistics: EMNLP 2026, pages 259–277
October 24-29, 2026 ©2026 Association for Computational Linguistics

Category	Method	Candidate Model Access	Proactive	Weights	Adaptive	Suspect Model Access
Fingerprinting	Parameter-Based	□	✗	✓	✓	□
	Query-Based	□	✗	✓	✓	■
Model Watermarking	Backdoor-Based	□	✓	✗	✗	■
	PlugAE (Ours)	□	✓	✓	✓	■

Table 1: Different types of identification methods. "Access" represents that the method requires either black-box access (■) or white-box access (□) to the corresponding LLM. "Proactive" denotes whether the LLM will be proactively modified, "Weights" represents whether the model's main transformer layers are unaltered, while "Adaptive" assesses whether the method, particularly proactive ones, remains applicable to existing derivative models

layer. This process only deals with the related token embeddings and does not alter the main transformer layers, thus preserving the model's utility. Given a suspect LLM, we can identify whether it is a derivative of the candidate LLM by querying it with the query set and predefined copyright tokens. We theoretically prove that our PlugAE is more likely to achieve the optimal than current fingerprinting methods, as they utilize a discrete optimization (Gubri et al., 2024; Jin et al., 2024). Our method is also highly efficient, e.g., we reduce trainable parameters from 7B to just 4K compared to backdoor-based model watermarking.

To demonstrate the effectiveness of PlugAE, we experiment on three candidate LLMs and 32 suspect models (30 LLMs and two real-world LLM APIs). We compare our method with both state-of-the-art model watermarking (Instructional Fingerprint (IF) (Xu et al., 2024)) and fingerprinting methods (TRAP (Gubri et al., 2024) and ProFlingo (Jin et al., 2024)). Results show that our PlugAE outperforms them, especially when identifying the further fine-tuned derivatives of the candidate model. For example, Llama-specific (Touvron et al., 2023a) PlugAE achieves a target response rate (TRR) of 0.98 on the further fine-tuned Llama model, while that of IF is 0.00. Additionally, our PlugAE achieves an AUC of 1.00 for accurate detection of Llama, whereas TRAP achieves only 0.67. These findings showcase the effectiveness of our method. We further investigate the adversarial robustness and stealthiness of PlugAE through a more detailed analysis. We also explore the influence of different token numbers and different copyright tokens, exhibiting the flexibility of our PlugAE.

Contributions. Overall, this study aims to ignite a crucial discussion on enhancing model transparency and accountability in the era of LLMs. Our contributions are as follows:

- We propose PlugAE, a lightweight LLM identification method that proactively plugs the adversarial token embeddings into the model's token-embedding layer for further identification, without modifying transformer layers or degrading the model's utility.
- Further, we theoretically prove that our PlugAE overcomes some limitations in existing LLM identification methods.
- Extensive experiments on three candidate models and 32 suspect models show that our PlugAE outperforms both state-of-the-art watermarking and fingerprinting methods.
- Our ablation study illustrates the robustness, stealthiness, and flexibility of our method, offering opportunities for user customization.

2 Related Work

Based on whether the method requires embedding signals into the model proactively (Cao et al., 2021; Xu et al., 2025; Ye et al., 2026), we categorize existing LLM identification methods into model watermarking and fingerprinting. We summarize the differences between the two types in Table 1.

Model Watermarking. Different from text watermarking that aims to trace the LLM-generated content (Kirchenbauer et al., 2023; Yoo et al., 2023; Liu et al., 2024; Zhao et al., 2024), model watermarking is a representative LLM identification method (Gu et al., 2022; Li et al., 2023; Yang et al., 2024; Xu et al., 2024; Christ et al., 2024), most of which proactively fine-tune the LLM in a backdooring way to enable traceability by the model owner. However, either full fine-tuning or LoRA fine-tuning (Hu et al., 2022) will inevitably adjust the model's behavior, raising concerns about utility. Moreover, when the model owner seeks to further

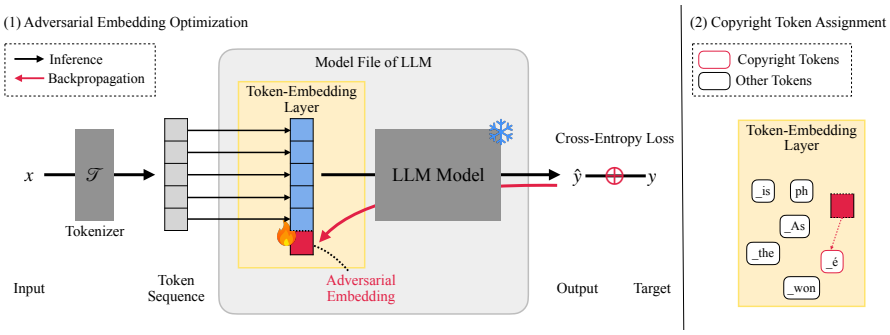


Figure 2: Framework of PlugAE. We first optimize adversarial embeddings on the tailored loss with the LLM frozen, then assign copyright tokens to adversarial embeddings by plugging them into the model's token-embedding layer.

customize the model, the watermark is prone to being eliminated after fine-tuning.

Fingerprinting. Fingerprinting refers to the task that aims to identify the origin of an LLM (Diwan et al., 2021; Zeng et al., 2023; McGovern et al., 2024; Gubri et al., 2024; Jin et al., 2024) without requiring the model to be modified before release. Existing fingerprinting methods have two categories, i.e., parameter-based and query-based. Parameter-based fingerprinting requires white-box access to the suspect model. For example, Zeng et al. (2023) propose a human-readable fingerprint that encodes invariant terms in the model's weight to a natural image. Query-based fingerprinting, instead, focuses on querying the model with well-crafted inputs and identifying based on the responses. Some works (Diwan et al., 2021; McGovern et al., 2024) extract the characteristics and distribution from the LLM's responses to some natural text inputs. Adversarial example (AE)-based methods, such as TRAP (Gubri et al., 2024) and ProFlingo (Jin et al., 2024), optimize adversarial text inputs to query the LLM for target responses.

3 Threat Model

In the era of LLMs, *adversarial third parties* may not possess the technology, computational resources, or sufficient time and funding. Thus, they intend to deliberately deploy LLMs without proper authorization, such as deploying unauthorized releases of open-source models or leaks of proprietary black-box services (Gubri et al., 2024; Jin et al., 2024). The adversary can intentionally hide the copyright or trademark of the LLM and dishonestly claim that it is their own proprietary LLM, violating most LLMs' licenses and agreements, which we have summarized in Appendix A. This inevitably accelerates unfair competition and blocks potential innovation. On the contrary, the *defender* can be the LLM owner or a third-party inspector ensuring the proper enforcement of intellectual property laws and regulations in the market. They aim to identify whether the black-box suspect model is derived from the candidate model under their control.

We argue that this identification task is non-trivial in reality. The defender often has black-box only access to the suspect model, as unauthorized services normally expose a query-only API to users. The suspect LLM may be further customized, for example, fine-tuning on specific datasets (Rozière et al., 2023; Tunstall et al., 2023; Ivison et al., 2023; Mitra et al., 2023) or being instructed by certain system prompts (Pro; OpenAI, 2024a). Such customization, while not intended, could help the adversary avoid identification by the defender. To further circumvent identification as the candidate model, the adversary may use specific prompts to respond to any queries about their true origin. They may also rebrand the LLM and rename the parameters. Additionally, they could rearrange the model's weights, significantly changing the direction of parameter vectors without affecting the model's performance (Zeng et al., 2023).

4 PlugAE

4.1 Preliminary

Large language models (LLMs) typically consist of a tokenizer and a model. The tokenizer maps between text x and tokens $\{t_1, \ldots, t_n\}$ (denoted as $t_{1:n}$ for simplicity). With the input tokens $t_{1:n}$, the token-embedding layer of the model converts them into *token embeddings*: $g(t_{1:n}) = \mathbf{e}_{1:n}, \mathbf{e}_i \in \mathbb{R}^d$. The remained transformer layers, parameterized by Φ, then calculates the probability of producing the next token $p_\Phi(t_{n+1}|\mathbf{e}_{1:n})$.

Overall, the LLM determines the joint probability of generating a series of tokens $t_{n+1:n+m}$ autoregressively: $p_\Phi(t_{n+1:n+m}|\mathbf{e}_{1:n}) = \prod_{i=1}^{m} p_\Phi(t_{n+i}|\mathbf{e}_{1:n+i-1})$. The typical training process of LLM leverages the negative log-likelihood (NLL) loss:

$$\mathcal{L}(y, x) = \mathcal{L}^{\mathrm{e}}(y, \mathbf{e}_{1:n}) = -\sum_{i=1}^{m} \log p(t_{n+i}|\mathbf{e}_{1:n+i-1}), \quad (1)$$

where $\mathcal{L}^{\mathrm{e}}(\cdot, \cdot)$ denotes the loss function that takes y and the token embeddings of x as input.

4.2 PlugAE Approach

Method Overview. In this work, we propose PlugAE, an efficient and effective LLM identification method that proactively Plugs the Adversarial token Embeddings into the LLM's token-embedding layer for identification, without modifying its main transformer layers Φ. Specifically, our method has two stages, i.e., *adversarial embedding optimization* and *copyright token assignment*, as shown in Figure 2. We detail the two stages below and address potential concerns.

Stage I. Adversarial Embedding Optimization. Instead of optimizing the discrete adversarial tokens, we optimize a sequence of k adversarial token embeddings $\mathbf{e}^a$. Such adaptation makes the optimization process differentiable, largely increasing the efficiency. The loss function becomes:

$$\mathcal{L}_{\texttt{PlugAE}}(y, x, \mathbf{e}^a) = \sum_{h \in H} \mathcal{L}^{\mathrm{e}}(y, \mathbf{insert}_{z_h}(\mathbf{e}^{h(x)}, \mathbf{e}^a)), \quad (2)$$

where $h(x)$ represents encoding user input x with the chat template $h \in H$, a set containing different chat templates with various system prompts. $\mathbf{insert}_{z_h}(s_1, s_2)$ denotes inserting sequence s_2 into s_1 at the position z_h. The position z_h is adaptable for different chat templates. This makes PlugAE more generalizable to complex settings, such as system prompts or instructional templates. Based on the frozen LLM, we optimize the adversarial embeddings $\mathbf{e}^a$ to minimize the loss $\mathcal{L}_{\texttt{PlugAE}}$:

$$\arg\min_{\mathbf{e}^a} \mathcal{L}_{\texttt{PlugAE}}(y, x, \mathbf{e}^a), (x, y) \in Q, \quad (3)$$

where we optimize a *universal* sequence of adversarial embeddings $\mathbf{e}^a$ for all queries in set Q.

Stage II. Copyright Token Assignment. PlugAE then assigns the obtained k adversarial embeddings $\mathbf{e}^a$ to customized copyright tokens in the LLM's token-embedding layer g. The copyright tokens refer to the tokens being used to distinguish the LLM's identity afterwards. Specifically, PlugAE allows users to design their own copyright tokens, which can be updated and modified based on users' needs. Generally, the tokens should be rarely used or under-trained (Rumbelow and Watkins, 2023; Land and Bartolo, 2024), such as "SolidGoldMagikarp" and "cf," so that the model's utility is preserved for most cases. We also show that the user can design more unique tokens, such as "mkahg," or more complex "h?!4t2rvAEVhC>K," that are not contained in the vocabulary. In such circumstances, we need to add these newly created tokens into the tokenizer, which leads to a trade-off between the rareness of tokens and the stealthiness of PlugAE. With the selected copyright tokens, PlugAE sets their mapped token embeddings in the token-embedding layer g as the optimized adversarial embeddings, accordingly.

4.3 PlugAE v.s. Model Watermarking

Compared with most model watermarking methods that require fine-tuning the LLM (including both token embeddings in g and the transformer layers Φ) via backdooring (Xu et al., 2024), PlugAE only modifies the token-embedding layer g. Thus, the model's utility is well-preserved: in practice, only 1-3 copyright tokens have specific behaviors, leaving the remaining 32K-216K tokens to act as originals. Additionally, this plug-in process is more adaptable and convenient. Imagine an LLM already has several fine-tuned derivatives before release. The model owner wants to watermark this LLM for copyright protection via backdooring. Then, every existing derivative of this LLM is required to be watermarked as well, which consumes both time and computational resources. Instead, our PlugAE only requires plugging the optimized adversarial embeddings into the existing derivatives. That is, they do not need to optimize again. Experiments in Section 5.3 show that our method remains effective under such scenarios. We further discuss remaining challenges and our vision in Appendix C.

4.4 PlugAE v.s. Fingerprinting

Query-based fingerprinting, such as TRAP (Gubri et al., 2024) and ProFlingo (Jin et al., 2024), is the mainstream fingerprinting approach to identifying the origin of a black-box LLM. They share a similar goal with our PlugAE, i.e., leveraging adversarial prompts to guide the candidate LLM to generate target outputs. As detailed in Appendix B.1, we summarize these query-based methods with a uni-

fied loss:

$$\mathcal{L}_{\text{AE}}(y, x, a) = \sum_{h \in H} \mathcal{L}(y, \textbf{insert}_{z_h}(h(x), a)). \quad (4)$$

Specifically, the unified loss $\mathcal{L}_{\text{AE}}$ can be adapted to ProFlingo (Jin et al., 2024) when we insert the adversarial example a as input x's prefix, i.e., let $\textbf{insert}_{z_h}(h(x), a) := h(a \| x)$, where $\|$ denotes sequence concatenation. Likewise, this unified loss can be translated to TRAP if we limit to *one* default chat template $H = \{h\}$ and append a after x, i.e., $\textbf{insert}_{z_h}(h(x), a) := h(x \| a)$.

However, they can only optimize on discrete tokens within the vocabulary, the size of which is much smaller than the embedding space in $\mathbb{R}^d$ where d denotes the token embedding dimensions. This inevitably limits the search space of query-based fingerprinting and bounds their capability. By comparing our $\mathcal{L}_{\texttt{PlugAE}}$ in Equation (2) and the unified loss $\mathcal{L}_{\text{AE}}$ in Equation (4), we hereby theoretically prove that PlugAE is more likely to reach the optimal compared to query-based fingerprinting.

Hypothesis 4.1. For most LLMs, we assume their embedding layer $g : V^n \to \mathbb{R}^{d \times n}$ is injective, where d is the dimension of the token embedding and n is the length of the token sequence.

We believe this hypothesis holds as the size of the vocabulary V is much smaller than the embedding space $\mathbb{R}^d$, where d denotes the token embedding dimensions. To empirically verify this hypothesis, we checked all 30 LLMs used in our experiments (see Section 5.1). Specifically, for each LLM, we calculate the cosine similarity between every pair of token embeddings. We considered two embeddings to be similar if their cosine similarity exceeded 0.99. The results of our experiments demonstrate that, across all the LLMs tested, the token embeddings were pairwise distinct. The empirical evidence supports that our hypothesis holds for most LLMs in practice.

Based on the hypothesis, we further prove that the inverse function q of the embedding layer g is non-differentiable, so that:

Lemma 4.1. *Given q is non-differentiable, $\mathcal{L}_{\texttt{PlugAE}}$ is prone to reach closer or equal to the optimal solution compared to the unified loss $\mathcal{L}_{AE}$.*

Based on Lemma 4.1, we claim that the optimization of our PlugAE is more likely to reach the optimal compared with current query-based methods that adopt the unified loss $\mathcal{L}_{\text{AE}}$. This indicates that the adversarial examples optimized by our PlugAE can extract more information from the LLM, making it more reliable for identification, which is proven experimentally in Section 5.3. Due to the page limit, the full proof can be found in Appendix B.2.

5 Experiments

5.1 Experimental Settings

Models. We utilize three LLMs as *candidate models*, including Llama (Touvron et al., 2023a), Llama2 (Touvron et al., 2023b), and Mistral (Jiang et al., 2023). We have white-box access to them and aim to identify whether they are the origin of the suspect models. Then, we collect 30 open-source LLMs and two real-world black-box APIs as *suspect models*. Specifically, models derived from the candidate models are released both officially (e.g., Llama2-Chat (Touvron et al., 2023b) developed by Meta) and by other institutes (e.g., Vicuna (Team) (fine-tuned on Llama) from LMSYS Org). We also collect other model families, such as OLMo (Groeneveld et al., 2024), and two black-box APIs, i.e., GPT4-Turbo (OpenAI, 2023a) and Claude3.5-Sonnet (Anthropic, 2024). All LLMs used are publicly available. We do not use any unauthorized LLMs in our study, thereby adhering to ethical standards and legal compliance. Model details can be found in Appendix D.

Baselines. We compare with both state-of-the-art model watermarking, Instructional Fingerprint (IF) (Xu et al., 2024), and fingerprinting methods, TRAP (Gubri et al., 2024) and ProFlingo (Jin et al., 2024). We follow their default settings. To compare with fingerprinting, we conduct candidate model-specific identifications on 32 suspect models. For model watermarking, we mock the customization process on the watermarked LLMs to evaluate the methods' robustness. Specifically, we fine-tune the watermarked/plugged model on CodeSearchNet (CSN) (Husain et al., 2019), a widely used code dataset that contains about six million functions spanning six programming languages.

Configurations. In experiments, we place the adversarial embeddings as the prefix of the input query and use the query set of ProFlingo. We set the number of adversarial token embeddings k to 1 and set the copyright token to a randomly generated 5-digit string "mkahg." The learning rate is set to 0.1 with the Adam optimizer for running 30 epochs. We use the top-p of 1.0 and temperature of 1.0 as default settings when generating responses. In eval-

Method	Llama			Llama2			Mistral		
	○	●	◐	○	●	◐	○	●	◐
IF	0.00	1.00	0.00	0.00	1.00	0.88	0.00	1.00	0.25
PlugAE	0.00	1.00	0.98	0.00	1.00	0.98	0.00	1.00	0.78

Table 2: TRR of model watermarking and our PlugAE on three different candidate LLMs. ○ denotes the original LLMs without being modified; ● represents the modified LLMs either being watermarked or modified by PlugAE; ◐ depicts modified LLMs that are further fine-tuned on the CSN dataset.

uation, we use the prompt template suggested in its repository for each model if available. Otherwise, we default to the zero-shot prompt template from FastChat (Zheng et al., 2023). For fine-tuning the watermarked LLMs, we adhere to the training parameters of Alpaca (Taori et al., 2023) and run for 15,000 steps on four NVIDIA A100 GPUs with 40 GB of memory. Then, we evaluate the utility of the fine-tuned LLMs on HellaSwag (Zellers et al., 2019) and MMLU (Hendrycks et al., 2021) under the zero-shot setting.

Evaluation Metric. We use the target response rate (TRR) for evaluation, defined as the ratio of the target outputs among all responses, i.e., $\text{TRR} = \frac{\#\text{Correct target outputs}}{\#\text{Total outputs}}$. A higher TRR suggests stronger evidence that the suspect model is derived from the candidate model. Considering this as a binary classification task, we also report the classification accuracy and calculate the ROC/AUC curve when evaluating on 32 suspect models.

5.2 Experiment: PlugAE v.s. Model Watermarking

Experimental results of our PlugAE depicted in Table 2 show its effectiveness in identifying and robustness against fine-tuning. We observe that both PlugAE and IF achieve a TRR of 1.00 when we identify the suspect model without further fine-tuning. Then we mock the customization process and fine-tune the modified LLM on CSN. We observe that PlugAE maintains a high TRR of 0.98 on both fine-tuned Llama and Llama2 models, which outperforms IF by 0.98 and 0.10, respectively. This indicates that the performance of backdoor-based model watermarking relies on the customization dataset and shows that our method is more robust against fine-tuning compared with model watermarking.

The utility of LLMs is examined in Appendix E.1. The results show that fine-tuning and model watermarking do not affect the model's utility. We also find that our PlugAE achieves exactly the same performance as the original model under the same random seed. This makes sense as we only insert several adversarial token embeddings and do not need to modify the model Φ.

Besides, we investigate the influence of fine-tuning on the plugged adversarial embeddings, i.e., the token embedding of the copyright token. Specifically, we calculate the cosine similarity between the original adversarial embeddings and them in the fine-tuned LLM. Results show that the adversarial embeddings remain the same after fine-tuning in all three candidate LLMs, i.e., all cosine similarities are 1.00. This is intuitive as we assume the copyright token is defined by the model owner, which should be rare and hard to guess. This finding provides additional support for our earlier observation, i.e., PlugAE is more robust against fine-tuning than backdoor-based model watermarking.

5.3 Experiment: PlugAE v.s. Fingerprinting

We simulate the fine-tuning scenario in the real world by plugging the copyright token of the candidate model into their existing derivatives. For example, we add the copyright token of Llama2 into Llama2-Chat and the other nine derivatives. This is practical as the token embedding of copyright token is robust against fine-tuning, as shown in Section 5.2. Based on this, we compare our PlugAE with two state-of-the-art fingerprinting methods, TRAP (Gubri et al., 2024) and ProFlingo (Jin et al., 2024) on 32 suspect models.

Figure 3 shows the TRRs of each method for identifying three candidate LLMs on 32 suspect models. We observe that our PlugAE outperforms fingerprinting in detecting both non-derivatives and derivatives. For example, given a candidate model with PlugAE, the TRRs of suspect models that are non-derivatives are 0.00. This is because those models do not have the corresponding copyright token or the adversarial embeddings. In this sense, when we query the model with copyright tokens, non-derivative models would not respond with the target answer. Instead, ProFlingo tends to overestimate some non-derivatives as the candidate LLM's derivative. For example, Llama-based ProFlingo achieves a TRR of 0.20 on Mistral-v0.3-Instruct, while it only achieves 0.14 on Guanaco, the fine-tuned version of Llama. For derivatives, our PlugAE achieves generally higher TRRs than ProFlingo, while TRAP fails to distinguish fine-

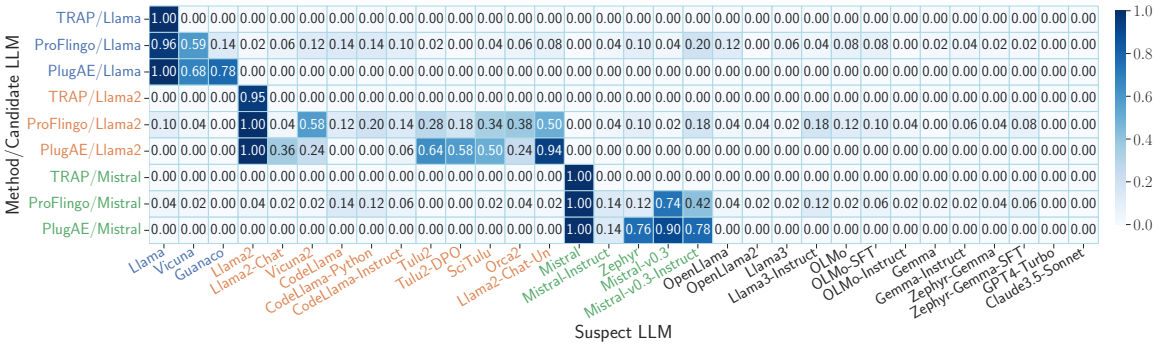


Figure 3: Performance (TRR) of different methods for three candidate LLMs on 32 suspect LLMs. A higher TRR indicates that the suspect LLM is more probable to be derived from the candidate LLM. Note that the same color indicates that the suspect LLM is derived from the candidate LLM.

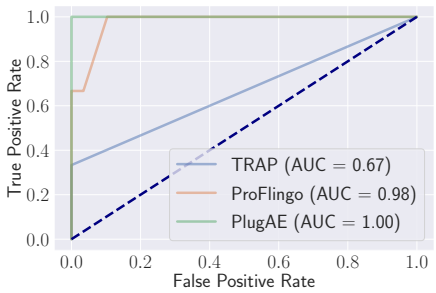


Figure 4: ROC curve of different methods on Llama.

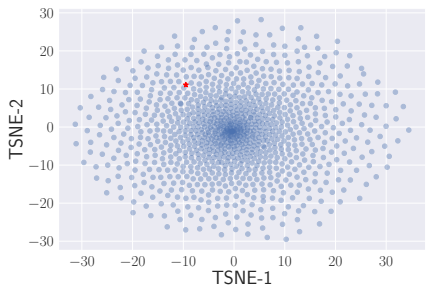


Figure 5: T-SNE visualization of adversarial embedding (red) and 1,000 random token embeddings in Llama.

tuned versions. For example, based on Llama, PlugAE achieves a TRR of 0.78 on the derivative Guanaco, while TRAP and ProFlingo achieve 0.00 and 0.14, respectively. Similarly, Mistral-based PlugAE achieves a TRR of 0.76 on Zephyr, while TRAP and ProFlingo only have a TRR of 0.00 and 0.12. This indicates that our PlugAE illustrates more confidence in distinguishing between the derivative and non-derivative LLMs.

Considering this as a binary classification task, we calculate the ROC/AUC curve for each method on three LLMs. As shown in Figure 4, our PlugAE achieves an AUC of 1.00 on Llama, outperforming the other two methods. This further shows the effectiveness of our method. The results of the other two models can be found in Appendix E.2

5.4 Stealthiness

We investigate the stealthiness of our PlugAE considering three aspects, i.e., copyright tokens, the adversarial token embeddings, and the query set Q.

Copyright Tokens. We give users a high degree of flexibility in customizing their own copyright tokens and provide some suggestions. Generally, the copyright tokens should be rarely used so that they will not be triggered for normal inputs. We also mentioned that the users can add new tokens to the tokenizer, as it is natural for LLMs to have their own tokenizer (Touvron et al., 2023b; Jiang et al., 2023; Team, 2024), i.e., there is no unified tokenizer that works for all LLMs. The vocabulary size also makes it hard to check all tokens, e.g., there are 32,000 tokens in Llama2 and 256,000 tokens in Gemma (Team, 2024). We can even remove the same number of rare or under-trained tokens to keep the vocabulary size. Further, when $k > 1$, we can use a cryptographically secure pseudorandom number generator (CSPRNG) (Suchanek et al., 2011) to select the copyright token sequence from a set of rare tokens. All these strategies make it difficult to perceive copyright tokens.

Adversarial Token Embeddings. We further explore whether we can detect copyright tokens by analyzing token embeddings. For each candidate LLM, we randomly sample 1,000 token embeddings and use t-SNE to visualize them and the corresponding adversarial embedding in this model. Due to the page limit, we exhibit the results of Llama (see Figure 5). The visualization results on all three LLMs can be found in Figure 9 in Appendix E.3. We observe that the adversarial em-

Suspect Model	PlugAE	PlugAE (New Query Set)
Llama2	1.00	1.00
Llama	0.00	0.02
Mistral	0.00	0.02
OpenLlama	0.00	0.02
OLMo	0.00	0.02

Table 3: TRR of Llama2-specific PlugAE with a new query set on different suspect LLMs.

Suspect Model	Empty	Basic	Helpful	Alpaca	ChatGPT
Llama2	0.88	1.00	0.98	1.00	1.00
Llama	0.00	0.00	0.00	0.00	0.00
Mistral	0.02	0.00	0.04	0.00	0.00
OpenLlama	0.02	0.00	0.00	0.00	0.00
OLMo	0.00	0.00	0.00	0.02	0.00

Table 4: TRR of Llama2-specific PlugAE with various system prompts on different suspect LLMs.

bedding for Llama is within the circle of its token embeddings, i.e., it is not an outlier. In this sense, it is challenging to detect the adversarial embedding using outlier detection methods, exhibiting the stealthiness of our PlugAE.

Query Set. We allow users to customize their query set Q, which further complicates the detection of PlugAE. Even if the attacker knows the copyright tokens, it is challenging for them to deduce our exact approach through reverse engineering. We further show that our PlugAE does not rely on any specific query set. Specifically, we use ChatGPT (cha) to generate a similar query set with 50 questions. Then we conduct PlugAE on Llama2 and evaluate on different LLMs. The results shown in Table 3 demonstrate the robustness of our method using different query sets.

5.5 Ablation Study

Copyright Tokens. In the main experiments, PlugAE takes a random 5-digit string as the copyright token. To investigate the influence of different copyright token choices, we experiment with the other two different copyright tokens. The first is that we randomly generate a 20-digit-long string as the copyright token, i.e., "c*zNFj.7}4E5~hHAq9Nh." The other is "cf," which exists in Mistral-7b's default vocabulary but not in Llama or Llama2's vocabulary. In the experiments, we got the same results using different copyright tokens. This is intuitive, as different copyright tokens correspond to the same token embedding.

Chat Templates. We follow the official chat template for different LLMs, so that the derivatives use different chat templates and system prompts from the original LLM. For example, the chat template of CodeLlama is different from that of the base Llama2 model. Results shown in Figure 3 demonstrate the robustness of our PlugAE regarding different prompt wrappers.

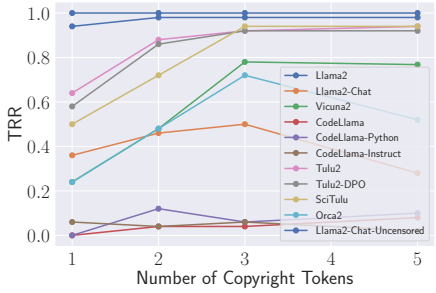


Figure 6: Influence of the number of tokens on Llama2. TRRs for non-derivatives are consistently 0.0.

System Prompts. We also evaluate PlugAE on LLMs with various system prompts. We use five different system prompts: empty, basic, helpful, Alpaca, and ChatGPT style, as detailed in Table 10 in the appendix. We apply these system prompts to the suspect LLMs and evaluate the Llama2-specific PlugAE on these suspect LLMs. Results are shown in Table 4. We observe that although there are some variants with different system prompts, we can still identify the derivatives and the non-derivatives of Llama2 model. This indicates that our PlugAE does not rely on fragile system prompts.

Numbers of Tokens. To investigate the influence of the number of copyright tokens, we further experiment with the number of tokens of 2, 3, and 5 on the Llama2 model. For copyright token selection, we randomly generate a 5-digit string for each token embedding. Results in Figure 6 show that a larger token number could achieve a better TRR on the derivatives. For instance, the TRR on SciTulu increases from 0.50 to 0.94 when the token number increases to 5. However, when the token number is too large, i.e., 5, the TRRs on the derivatives would even drop. For example, the TRR on Orca2 drops to 0.52 when the token number is 5, while it is 0.72 when the token number is 3. Note that the TRRs for non-derivatives remain 0.

More Advanced Models. To demonstrate the generalizability of PlugAE, we extended experiments to larger and more frontier LLMs. We use Llama3.1 (Meta, 2024) (meta-llama/Llama-3.1-8B, released in 2024) and Mistral-24B (Mistral AI,

Candidate	Llama3.1	Llama3.1-Chat	Mistral-24B	Mistral-24B-Chat	Mistral-24B-Chatml	GPT5	Gemini2.5	Claude4.5
Llama3.1	0.94	0.82	0.00	0.00	0.00	0.00	0.00	0.00
Mistral-24B	0.00	0.00	0.92	0.80	0.86	0.00	0.00	0.00

Table 5: TRR of Llama3.1- and Mistral-24B-specific PlugAE on eight suspect models.

2025) (mistralai/Mistral-Small-24B-Instruct-2501, released in 2025) as candidate models. And we take Llama3.1, Llama3.1-Chat (meta-llama/Llama-3.1-8B-Instruct), Mistral-24B, Mistral-24B-Chat (mistralai/Mistral-Small-3.1-24B-Instruct-2503), and Mistral-24B-Chatml (IntervitensInc/Mistral-Small-24B-Instruct-2501-chatml), and three frontier APIs as suspect models. The APIs include GPT5 (OpenAI, 2026) (gpt-5-mini-2025-08-07), Gemini2.5 (Platform) (gemini-2.5-flash), and Claude4.5 (Anthropic, 2025b) (claude-sonnet-4-5-20250929). As shown in Table 5, PlugAE consistently achieves high TRR across derivatives, while unrelated models and APIs yield 0 TRR. This confirms strong detectability of our PlugAE.

6 Conclusion

In this paper, we address the challenge of identifying the origin of black-box LLMs, motivated by the increasing risks of copyright infringement and unfair competition. To this end, we introduce PlugAE, a simple yet effective identification framework that integrates adversarial token embeddings into the token-embedding layer of LLMs without modifying the model's main transformer layers. Our theoretical analysis reveals the inner limitations of both fingerprinting and model watermarking methods, and positions PlugAE as a robust alternative. Extensive experiments on 30 LLMs and two real-world LLM API services empirically validate the effectiveness of PlugAE, consistently outperforming state-of-the-art LLM identification techniques. We further exploit the stealthiness of our PlugAE from three distinct perspectives. Finally, the ablation study under various configurations confirms the robustness and flexibility of our method.

Limitations

While our PlugAE demonstrates strong effectiveness in identifying the origin of LLMs, it is a proactive method and still requires modifying the corresponding LLM before release. This introduces the risk of the modification being detected or removed. We leave this as a future direction. Further, due to the project timeline, we opted for some older LLM versions rather than the most current one, while they are compatible with the LLMs used in the baselines. To keep updated, we have experimented with more advanced and larger models and validated the effectiveness of our method in Section 5.5.

Ethical Consideration

By raising awareness of the risks of the unauthorized use of LLMs, our work aims to contribute to developing a more robust and secure open-source environment and calls for a more reliable way to identify LLMs. With theoretical analysis, we explore the inner limitations of current query-based fingerprinting methods and provide an alternative way to overcome such limitations. We believe our work can motivate more advanced identification methods in this field. All LLMs are specified in Appendix D, and the datasets used in this research are in English. They are either publicly available or generated by LLMs, ensuring no inclusion of personally identifiable information and eliminating user de-anonymization risks. We do not use any unauthorized LLMs in our study, thereby adhering to ethical standards and legal compliance.

Acknowledgment

We thank all anonymous reviewers, ACs, and SACs for their constructive comments.

References

ChatGPT. https://chat.openai.com/chat.

PromptDB. https://promptdb.param.codes/.

2004. Apache License, Version 2.0. https://www.apache.org/licenses/LICENSE-2.0.

2023. https://ai.meta.com/llama/license/.

2024. https://eur-lex.europa.eu/eli/reg/2024/1689.

2025. https://mistral.ai/terms/.

Ai2. 2023. AI2 ImpACT License – Low Risk Artifacts. https://huggingface.co/allenai/tulu-2-7b/blob/main/LICENSE.md.

Anthropic. 2024. Claude 3.5 Sonnet. https://www.anthropic.com/news/claude-3-5-sonnet.

Anthropic. 2025a. Consumer Terms of Service. https://www.anthropic.com/legal/consumer-terms.

Anthropic. 2025b. Introducing Claude 4. https://www.anthropic.com/news/claude-4.

Xiaoyu Cao, Jinyuan Jia, and Neil Zhenqiang Gong. 2021. IPGuard: Protecting Intellectual Property of Deep Neural Networks via Fingerprinting the Classification Boundary. In *ACM Asia Conference on Computer and Communications Security (ASIACCS)*, pages 14–25. ACM.

Lin Chen, Jinsong Li, Xiaoyi Dong, Pan Zhang, Conghui He, Jiaqi Wang, Feng Zhao, and Dahua Lin. 2023. ShareGPT4V: Improving Large Multi-Modal Models with Better Captions. *CoRR abs/2311.12793*.

Miranda Christ, Sam Gunn, Tal Malkin, and Mariana Raykova. 2024. Provably Robust Watermarks for Open-Source Language Models. *IACR Cryptology ePrint Archive*.

Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. 2023. QLoRA: Efficient Finetuning of Quantized LLMs. *CoRR abs/2305.14314*.

Nirav Diwan, Tanmoy Chakraborty, and Zubair Shafiq. 2021. Fingerprinting Fine-tuned Language Models in the Wild. In *Annual Meeting of the Association for Computational Linguistics and International Joint Conference on Natural Language Processing (ACL/IJCNLP)*, pages 4652–4664. ACL.

Xinyang Geng and Hao Liu. 2023. OpenLLaMA: An Open Reproduction of LLaMA. https://github.com/openlm-research/open_llama.

Dirk Groeneveld, Iz Beltagy, Pete Walsh, Akshita Bhagia, Rodney Kinney, Oyvind Tafjord, Ananya Harsh Jha, Hamish Ivison, Ian Magnusson, Yizhong Wang, Shane Arora, David Atkinson, Russell Authur, Khyathi Raghavi Chandu, Arman Cohan, Jennifer Dumas, Yanai Elazar, Yuling Gu, Jack Hessel, and 24 others. 2024. OLMo: Accelerating the Science of Language Models. *CoRR abs/2402.00838*.

Chenxi Gu, Chengsong Huang, Xiaoqing Zheng, Kai-Wei Chang, and Cho-Jui Hsieh. 2022. Watermarking Pre-trained Language Models with Backdooring. *CoRR abs/2210.07543*.

Martin Gubri, Dennis Ulmer, Hwaran Lee, Sangdoo Yun, and Seong Joon Oh. 2024. TRAP: Targeted Random Adversarial Prompt Honeypot for Black-Box Identification. In *Annual Meeting of the Association for Computational Linguistics (ACL)*. ACL.

Eric Hartford. 2023. Uncensored Models. https://erichartford.com/uncensored-models.

Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. 2021. Measuring Massive Multitask Language Understanding. In *International Conference on Learning Representations (ICLR)*.

Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2022. LoRA: Low-Rank Adaptation of Large Language Models. In *International Conference on Learning Representations (ICLR)*.

Hamel Husain, Ho-Hsiang Wu, Tiferet Gazit, Miltiadis Allamanis, and Marc Brockschmidt. 2019. CodeSearchNet Challenge: Evaluating the State of Semantic Code Search. *CoRR abs/1909.09436*.

Hamish Ivison, Yizhong Wang, Valentina Pyatkin, Nathan Lambert, Matthew E. Peters, Pradeep Dasigi, Joel Jang, David Wadden, Noah A. Smith, Iz Beltagy, and Hannaneh Hajishirzi. 2023. Camels in a Changing Climate: Enhancing LM Adaptation with Tulu 2. *CoRR abs/2311.10702*.

Albert Q. Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, Lélio Renard Lavaud, Marie-Anne Lachaux, Pierre Stock, Teven Le Scao, Thibaut Lavril, Thomas Wang, Timothée Lacroix, and William El Sayed. 2023. Mistral 7B. *CoRR abs/2310.06825*.

Heng Jin, Chaoyu Zhang, Shanghao Shi, Wenjing Lou, and Y. Thomas Hou. 2024. ProFLingo: A Fingerprinting-based Copyright Protection Scheme for Large Language Models. *CoRR abs/2405.02466*.

John Kirchenbauer, Jonas Geiping, Yuxin Wen, Jonathan Katz, Ian Miers, and Tom Goldstein. 2023. A Watermark for Large Language Models. In *International Conference on Machine Learning (ICML)*, pages 17061–17084. PMLR.

Sander Land and Max Bartolo. 2024. Fishing for Magikarp: Automatically Detecting Under-trained Tokens in Large Language Models. In *Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 11631–11646. ACL.

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. In *AAAI Conference on Artificial Intelligence (AAAI)*, pages 14991–14999. AAAI.

Aiwei Liu, Leyi Pan, Xuming Hu, Shiao Meng, and Lijie Wen. 2024. A Semantic Invariant Robust Watermark for Large Language Models. In *International Conference on Learning Representations (ICLR)*. ICLR.

Haotian Liu, Chunyuan Li, Qingyang Wu, and Yong Jae Lee. 2023. Visual Instruction Tuning. In *Annual Conference on Neural Information Processing Systems (NeurIPS)*. NeurIPS.

Hope McGovern, Rickard Stureborg, Yoshi Suhara, and Dimitris Alikaniotis. 2024. Your Large Language Models Are Leaving Fingerprints. *CoRR abs/2405.14057*.

Meta. 2024. Introducing Llama 3.1: Our most capable models to date. https://ai.meta.com/blog/meta-llama-3-1/.

Mistral AI. 2025. Mistral Small 3. https://mistral.ai/news/mistral-small-3/.

Arindam Mitra, Luciano Del Corro, Shweti Mahajan, Andrés Codas, Clarisse Simões, Sahaj Agrawal, Xuxi Chen, Anastasia Razdaibiedina, Erik Jones, Kriti Aggarwal, Hamid Palangi, Guoqing Zheng, Corby Rosset, Hamed Khanpour, and Ahmed Awadallah. 2023. Orca 2: Teaching Small Language Models How to Reason. *CoRR abs/2311.11045*.

OpenAI. 2023a. GPT-4 Technical Report. *CoRR abs/2303.08774*.

OpenAI. 2023b. Introducing GPTs. https://openai.com/blog/introducing-gpts.

OpenAI. 2024a. Introducing the GPT Store. https://openai.com/index/introducing-the-gpt-store/.

OpenAI. 2024b. Terms of Use. https://openai.com/policies/row-terms-of-use/revisions/2024-12-11/.

OpenAI. 2025. OpenAI Services Agreement. https://openai.com/policies/may-2025-business-terms/.

OpenAI. 2026. GPT-5.4. https://developers.openai.com/api/docs/models/gpt-5.4.

Gemini Agent Platform. Gemini 2.5 Flash. https://docs.cloud.google.com/gemini-enterprise-agent-platform/models/gemini/2-5-flash.

Baptiste Rozière, Jonas Gehring, Fabian Gloeckle, Sten Sootla, Itai Gat, Xiaoqing Ellen Tan, Yossi Adi, Jingyu Liu, Tal Remez, Jérémy Rapin, Artyom Kozhevnikov, Ivan Evtimov, Joanna Bitton, Manish Bhatt, Cristian Canton-Ferrer, Aaron Grattafiori, Wenhan Xiong, Alexandre Défossez, Jade Copet, and 6 others. 2023. Code Llama: Open Foundation Models for Code. *CoRR abs/2308.12950*.

Jessica Rumbelow and Matthew Watkins. 2023. SolidGoldMagikarp (plus, prompt generation). https://www.lesswrong.com/posts/aPeJE8bSo6rAFoLqg/solidgoldmagikarp-plus-prompt-generation.

Fabian M. Suchanek, David Gross-Amblard, and Serge Abiteboul. 2011. Watermarking for Ontologies. In *International Semantic Web Conference (ISWC)*, pages 697–713. Springer.

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

Gemma Team. 2024. Gemma: Open Models Based on Gemini Research and Technology. *CoRR abs/2403.08295*.

The Vicuna Team. Vicuna: An Open-Source Chatbot Impressing GPT-4 with 90%* ChatGPT Quality. https://lmsys.org/blog/2023-03-30-vicuna/.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurélien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. 2023a. LLaMA: Open and Efficient Foundation Language Models. *CoRR abs/2302.13971*.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, Dan Bikel, Lukas Blecher, Cristian Canton-Ferrer, Moya Chen, Guillem Cucurull, David Esiobu, Jude Fernandes, Jeremy Fu, Wenyin Fu, and 49 others. 2023b. Llama 2: Open Foundation and Fine-Tuned Chat Models. *CoRR abs/2307.09288*.

Lewis Tunstall, Edward Beeching, Nathan Lambert, Nazneen Rajani, Kashif Rasul, Younes Belkada, Shengyi Huang, Leandro von Werra, Clémentine Fourrier, Nathan Habib, Nathan Sarrazin, Omar Sanseviero, Alexander M. Rush, and Thomas Wolf. 2023. Zephyr: Direct Distillation of LM Alignment. *CoRR abs/2310.16944*.

Weihan Wang, Qingsong Lv, Wenmeng Yu, Wenyi Hong, Ji Qi, Yan Wang, Junhui Ji, Zhuoyi Yang, Lei Zhao, Xixuan Song, Jiazheng Xu, Bin Xu, Juanzi Li, Yuxiao Dong, Ming Ding, and Jie Tang. 2023. CogVLM: Visual Expert for Pretrained Language Models. *CoRR abs/2311.03079*.

Jiashu Xu, Fei Wang, Mingyu Derek Ma, Pang Wei Koh, Chaowei Xiao, and Muhao Chen. 2024. Instructional Fingerprinting of Large Language Models. *CoRR abs/2401.12255*.

Zhenhua Xu, Xubin Yue, Zhebo Wang, Qichen Liu, Xixiang Zhao, Jingxuan Zhang, Wenjun Zeng, Wengpeng Xing, Dezhang Kong, Changting Lin, and Meng Han. 2025. Copyright Protection for Large Language Models: A Survey of Methods, Challenges, and Trends. *CoRR abs/2508.11548*.

Ziqing Yang, Michael Backes, Yang Zhang, and Ahmed Salem. 2024. SOS! Soft Prompt Attack Against Open-Source Large Language Models. *CoRR abs/2407.03160*.

Yuan Yao, Tianyu Yu, Ao Zhang, Chongyi Wang, Junbo Cui, Hongji Zhu, Tianchi Cai, Haoyu Li, Weilin Zhao, Zhihui He, Qianyu Chen, Huarong Zhou, Zhensheng Zou, Haoye Zhang, Shengding Hu, Zhi Zheng, Jie Zhou, Jie Cai, Xu Han, and 4 others. 2024. MiniCPM-V: A GPT-4V Level MLLM on Your Phone. *CoRR abs/2408.01800*.

Peigen Ye, Huali Ren, Zhengdao Li, Anli Yan, Hongyang Yan, Shaowei Wang, and Jin Li. 2026. Securing Large Language Models: A Survey of Watermarking and Fingerprinting Techniques. *ACM Computing Surveys*.

KiYoon Yoo, Wonhyuk Ahn, Jiho Jang, and Nojun Kwak. 2023. Robust Multi-bit Natural Language Watermarking through Invariant Features. In *Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 2092–2115. ACL.

Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. 2019. HellaSwag: Can a Machine Really Finish Your Sentence? In *Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 4791–4800. ACL.

Boyi Zeng, Chenghu Zhou, Xinbing Wang, and Zhouhan Lin. 2023. HuRef: HUman-REadable Fingerprint for Large Language Models. *CoRR abs/2312.04828*.

Xuandong Zhao, Prabhanjan Vijendra Ananth, Lei Li, and Yu-Xiang Wang. 2024. Provable Robust Watermarking for AI-Generated Text. In *International Conference on Learning Representations (ICLR)*. ICLR.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric P. Xing, Hao Zhang, Joseph E. Gonzalez, and Ion Stoica. 2023. Judging LLM-as-a-Judge with MT-Bench and Chatbot Arena. In *Annual Conference on Neural Information Processing Systems (NeurIPS)*, pages 46595–46623. NeurIPS.

Andy Zou, Zifan Wang, Nicholas Carlini, Milad Nasr, J. Zico Kolter, and Matt Fredrikson. 2023. Universal and Transferable Adversarial Attacks on Aligned Language Models. *CoRR abs/2307.15043*.

In the appendix, we provide details and results omitted in the main text.

- Appendix A: LLM licenses and agreements of six mainstream LLM organizations on the market, including OpenAI, Anthropic, Meta, Ai2, Google, and Mistral AI.
- Appendix B: Theoretical analysis and proof details on fingerprinting and model watermakring.
- Appendix C: Based on our theoretical and empirical analysis, we highlight challenges and opportunities in the task of LLM identification and copyright protection.
- Appendix D: Details of the models and datasets we use in the experiments.
- Appendix E: Additional experiments, including the utility evaluation, AUC/ROC and TSNE results on three LLMs, and counterexample analysis.

A LLM Licenses and Agreements

We summarize the types of services provided by the most mainstream LLMs on the market, along with their usage restrictions in Table 6. Among the six organizations we collected, OpenAI and Anthropic focus on providing API services, Meta and Ai2 release open-source models, while Google and Mistral AI have both API services and open-source models. We further analyze the usage restrictions in their licenses.

Our analysis reveals that many impose constraints on commercial use. Specifically, OpenAI requires businesses and developers that could benefit from their services to agree to further business terms (OpenAI, 2024b, 2025). Anthropic stipulates that their services cannot be used for commercial or business purposes (Anthropic, 2025a). Mistral AI does not allow their services to be used for the benefit of a third party unless otherwise agreed in a separate contract with them (mis, 2025). Meta stipulates that if a third-party product or service exceeds 700 million monthly active users, they must apply for a special license from Meta; otherwise, they are not authorized to use Llama 2 (lla, 2023). Ai2 requires a third party to complete a derivative impact report with all model and data derivatives (Ai2, 2023). Moreover, none of these organizations permit third parties to rebrand their services, such as by obscuring or removing copyright and trademark information.

B Theoretical Analysis on Fingerprinting and Model Watermarking

Our `PlugAE` utilizes the advantage of both query-based fingerprinting and model watermarking. In this section, we theoretically analyze and prove that `PlugAE` could be more optimal compared with current query-based fingerprinting and model watermarking methods.

B.1 Query-Based Fingerprinting

Query-based fingerprinting (Gubri et al., 2024; Jin et al., 2024) primarily optimizes an LLM-specific adversarial prompt for the candidate model, which can serve as a prefix/suffix to the input query to generate specific outputs. We first formalize existing state-of-the-art query-based fingerprinting under a unified loss function. Then we conduct a comprehensive investigation of their inherent limitations. The notations can be found in Table 7.

TRAP. TRAP (Gubri et al., 2024) is based on GCG (Zou et al., 2023). Its objective is to optimize an adversarial suffix, denoted as a, that manipulates the reference model into producing a predetermined target response y, given the input query x. Then, it uses the suffix to query the black-box LLM and compare the output with the target string. A match indicates that the target candidate model may power the black-box LLM. TRAP's loss function can be formulated as follows:

$$\mathcal{L}_{\text{TRAP}}(y, x, a) = \mathcal{L}(y, h(x \| a)), \tag{5}$$

where $h \in H$ represents the candidate model's prompt template, and $\|$ denotes the concatenation between texts. Then, TRAP optimizes the adversarial suffix a to minimize this loss function by iteratively comparing the loss value and replacing the tokens in the suffix.

ProFlingo. ProFlingo (Jin et al., 2024) improves TRAP in four aspects. First, ProFlingo uses a different query set. Second, ProFlingo's optimization objective is an adversarial prefix instead of a suffix. Third, instead of using only a single prompt template, ProFlingo optimizes several templates simultaneously to improve the algorithm's robustness. Furthermore, ProFlingo proposes a new way to update the adversarial prefix, which enhances the process of generating adversarial examples and the efficiency of the overall algorithm. ProFlings's

Usage Scenario	Organization					
	OpenAI	Anthropic	Google	Mistral AI	Meta	Ai2
Open-Source Model[1]	✗	✗	✓	✓	✓	✓
Query-Only API Service	✓	✓	✓	✓	✗	✗
Fine-Tuning API Service	✓	✓	✓	✓	✗	✗
Commercial Use	✗	✗	-	✗	✓	✗
Commercial Use (Special Permission)	✓	-	✓	✓	✓	✓
Rebranding	✗	✗	✗	✗	✗	✗[2]

[1] The open-source models include licenses not yet approved by the Open Source Initiative (https://opensource.org/license).
[2] Note that Ai2's OLMo series is under Apache 2.0 license (apa, 2004) and allows rebranding, while there are other LLMs, such as tulu-2 series, are under the AI2 ImpACT License for Low Risk Artifacts (https://huggingface.co/allenai/tulu-2-7b/blob/main/LICENSE.md), which forbids rebranding.

Table 6: Usage scenarios by each organization's model/API licenses. "-" denotes *not applicable*.

loss function is formulated as follows:

$$\mathcal{L}_{\text{ProFlingo}}(y, x, a) = \sum_{h \in H} \mathcal{L}(y, h(a \| x)), \tag{6}$$

where H is a collection of prompt templates that can be manually designed. The adversarial prefix a is optimized to minimize its loss function.

Unified Loss. Based on Equation (5) and Equation (6), we present a unified loss function summarizing the above query-based identification methods (Equation (4)):

$$\mathcal{L}_{\text{AE}}(y, x, a) = \sum_{h \in H} \mathcal{L}(y, \mathbf{insert}_{z_h}(h(x), a)), \tag{7}$$

where H is a prompt template set containing different instructional templates with various system prompts, and $h(x)$ represents encoding user input x with the template h. $\mathbf{insert}_z(s_1, s_2)$ denotes inserting sequence s_2 into s_1 at the position z_h. The position z_h is adaptable for different prompt templates h, enabling support for various prompt templates.

Limitations of TRAP and ProFlingo. We compare TRAP and ProFlingo under the unified loss $\mathcal{L}_{\text{AE}}$. As discussed above, the unified loss can be translated into the $\mathcal{L}_{\text{ProFlingo}}$ loss function when $\mathbf{insert}_{z_h}(h(x), a) := h(a \| x)$. Likewise, this unified loss can be adapted to $\mathcal{L}_{\text{TRAP}}$ by defining the prompt template collection as $H = \{h\}$, where h represents the default template of the candidate model, and appending a after x, i.e., setting $\mathbf{insert}_{z_h}(h(x), a) := h(x \| a)$. We find $\mathcal{L}_{\text{TRAP}}$ is more restricted as it optimizes to only one prompt template, while $\mathcal{L}_{\text{ProFlingo}}$ optimizes several prompt templates simultaneously (see Equation (6)), which allows a more generalizable adversarial text that is resilient to larger disturbances. This could explain why TRAP can only identify the candidate model itself, while ProFlingo can identify more derivatives of the candidate model, as shown in Section 5.3.

Recall that we focus on the task of black-box LLM identification, where interaction with the target model is restricted to text inputs. The defender optimizes an adversarial prefix/suffix to query the suspect model. However, because text is composed of discrete tokens, the optimization process inherently operates in a discrete space. For example, ProFlingo first calculates the gradient for each token t_i^a in the prefix a in each epoch as $\nabla_{t_i^a} \mathcal{L}_{\text{AE}}$. As we can only replace t_i^a instead of optimizing it directly, ProFlingo updates the adversarial prefix by replacing multiple tokens in each epoch. Based on the top-k tokens with the most negative gradients, a set of candidate tokens for replacement is sampled. Subsequently, the loss associated with each potential replacement token is computed. The token with the minimal loss is then selected. TRAP follows a similar process, optimizing over a discrete set of inputs. However, such discrete optimization approaches present significant challenges. Given the limited number of tokens and the computationally intensive nature of the optimization process, it is difficult to achieve a local optimum by design.

B.2 Reaching the Optimal of Query-Based Fingerprinting

Based on the loss function of our PlugAE and the unified loss Equation (4), we theoretically prove that PlugAE can reach the optimal of query-based methods by analyzing the optimal loss.

Hypothesis B.1. For most LLMs, we assume their

Variants		Functions
m, n, k	Integers	$\mathbf{encode}(x) = t^x_{1:n}, t^x_i \in V$
x, y, a	Texts	$\mathbf{decode}(t) = x^t$
$t_{1:n}$	Sequence of tokens	$g(t_{1:n}) = \mathbf{e}_{1:n}, \mathbf{e}_i \in \mathbb{R}^d$
$\mathbf{e}_{1:n}$	Sequence of token embeddings	$p_\Phi(t_{n+1}\|\mathbf{e}_{1:n}) = g \circ p_\theta(t_{n+1}\|t_{1:n})$
V	Set of all possible tokens	$p_\Phi(t_{n+1:n+m}\|\mathbf{e}_{1:n}) = \prod_{i=1}^m p_\Phi(t_{n+i}\|\mathbf{e}_{1:n+i-1})$
h	Prompt template	$\mathcal{L}^{\mathrm{e}}(t_{n+1:n+m}, \mathbf{e}_{1:n}) = -\log p_\Phi(t_{n+1:n+m}\|\mathbf{e}_{1:n})$
H	Set of prompt templates	$p_\theta(t_{n+1:n+m}\|t_{1:n}) = \prod_{i=1}^m p_\theta(t_{n+i}\|t_{1:n+i-1})$
$\|$	Text/Sequence concatenation	$\mathcal{L}(t_{n+1:n+m}, t_{1:n}) = -\log p_\theta(t_{n+1:n+m}\|t_{1:n})$
$\circ$	Function composition	
$p_\Phi(t_{n+1}\|\mathbf{e}_{1:n})$	Probability of the next token	
$p_\theta(t_{n+1}\|t_{1:n})$	Probability of the next token	
$\mathbf{insert}_z(s_1, s_2)$	Insert s_2 into s_1 at position z	

Table 7: Notations.

embedding layer $g : V^n \to \mathbb{R}^{d\times n}$ is injective, where d is the dimension of the token embedding and n is the length of the token sequence.

In practice, token embeddings in LLMs are often represented as d-dimensional vectors. Consequently, the size of the vocabulary V is much smaller than the embedding space $\mathbb{R}^d$. Such a high dimensionality increases the likelihood that different tokens will have distinct embeddings, thereby making g more likely to be injective. To empirically verify this hypothesis, we experiment with our three candidate LLMs and the other 18 LLMs. Specifically, for each LLM, we calculate the cosine similarity between every pair of token embeddings. We considered two embeddings to be similar if their cosine similarity exceeded 0.99. The results of our experiments demonstrate that, across all the LLMs tested, the token embeddings were pairwise distinct. The empirical evidence supports that our Hypothesis B.1 holds for most LLMs in practice.

Lemma B.1. *Based on Hypothesis B.1, the inverse function q of the embedding layer g is non-differentiable.*

Proof. Let g be the mapping from tokens to embeddings, and let q be its inverse, defined as:

$$q : \mathbb{R}^{d\times n} \to V^n, \tag{8}$$

where d is the dimension of the token embedding, and n is the length of the embedding sequence. Note that V is a discrete space, which contains predefined tokens that are discontinuous. Thus, q is non-differentiable due to the discrete nature of its output. □

Lemma B.2. *Based on Hypothesis B.1, the unified loss $\mathcal{L}_{AE}$ is derived from* PlugAE*'s loss $\mathcal{L}^e_{\texttt{PlugAE}}$ with the application of the inverse function q.*

Proof. Firstly, we prove $q \circ \mathcal{L}^{\mathrm{e}}(t_{n+1:n+m}, \mathbf{e}_{1:n}) = \mathcal{L}(t_{n+1:n+m}, t^{\mathbf{e}}_{1:n})$. Based on Equation (8), q maps a token embedding sequence to a discrete token sequence. Thus, $q \circ \mathcal{L}^{\mathrm{e}}(\cdot)$ is a transformation of input domains from $\mathbb{R}^{d\times n}$ to V^n. As $p_\Phi(t_{n+1}|\mathbf{e}_{1:n}) = g \circ p_\theta(t_{n+1}|t_{1:n})$, we can infer $q \circ p_\Phi(t_{n+1}|\mathbf{e}_{1:n}) = q \circ g \circ p_\theta(t_{n+1}|t_{1:n}) = p_\theta(t_{n+1}|t_{1:n})$.

$$\begin{aligned}
&q \circ \mathcal{L}^{\mathrm{e}}(t_{n+1:n+m}, \mathbf{e}_{1:n}) \\
&\quad = q \circ -\log p_\Phi(t_{n+1:n+m}|\mathbf{e}_{1:n}) \\
&\quad = q \circ -\sum_{i=1}^m \log p_\Phi(t_{n+i}|\mathbf{e}_{1:n+i-1}) \\
&\quad = -\sum_{i=1}^m q \circ \log p_\Phi(t_{n+i}|\mathbf{e}_{1:n+i-1}) \\
&\quad = -\sum_{i=1}^m \log p_\theta(t_{n+i}|t^{\mathbf{e}}_{1:n+i-1}) \\
&\quad = -\log \prod_{i=1}^m p_\theta(t_{n+i}|t^{\mathbf{e}}_{1:n+i-1}) \\
&\quad = -\log p_\theta(t_{n+1:n+m}|t^{\mathbf{e}}_{1:n}) = \mathcal{L}(t_{n+1:n+m}, t^{\mathbf{e}}_{1:n}).
\end{aligned} \tag{9}$$

We conclude $q \circ \mathcal{L}^{\mathrm{e}}(t_{n+1:n+m}, \mathbf{e}_{1:n}) = \mathcal{L}(t_{n+1:n+m}, t^{\mathbf{e}}_{1:n})$.

Then, we prove $q(\mathbf{insert}_z(\mathbf{e}_{1:n}, \mathbf{e}'_{1:m})) = \mathbf{insert}_z(q(\mathbf{e}_{1:n}), q(\mathbf{e}'_{1:m}))$. Let z be the k-th position of $\mathbf{e}_{1:n}$, $k \in [0, n]$, $s_{1:0}$ and $s_{n+1:n}$ denote a null sequence.

From left:

$$\begin{aligned}
q(\mathbf{insert}_z(\mathbf{e}_{1:n}, \mathbf{e}'_{1:m})) &= q(\{\mathbf{e}_{1:k}, \mathbf{e}'_{1:m}, \mathbf{e}_{k+1:n}\}) \\
&= t^{\{\mathbf{e}_{1:k}, \mathbf{e}'_{1:m}, \mathbf{e}_{k+1:n}\}} \\
&= \{t^{\mathbf{e}}_{1:k}, t^{\mathbf{e}'}_{1:m}, t^{\mathbf{e}}_{k+1:n}\}.
\end{aligned} \tag{10}$$

For the right side:

$$\begin{aligned}
\mathbf{insert}_z(q(\mathbf{e}_{1:n}), q(\mathbf{e}'_{1:m})) &= \mathbf{insert}_z(t^{\mathbf{e}}_{1:n}, t^{\mathbf{e}'}_{1:m}) \\
&= \{t^{\mathbf{e}}_{1:k}, t^{\mathbf{e}'}_{1:m}, t^{\mathbf{e}}_{k+1:n}\}.
\end{aligned} \tag{11}$$

Thus, we conclude $q(\mathbf{insert}_z(\mathbf{e}_{1:n}, \mathbf{e}'_{1:m})) =$

$\mathbf{insert}_z(q(\mathbf{e}_{1:n}), q(\mathbf{e}'_{1:m}))$. Finally, we have:

$$
\begin{aligned}
& q \circ \mathcal{L}^{\mathrm{e}}_{\texttt{PlugAE}}(y, x, \mathbf{e}^a_{1:m}) \\
&\quad = q \circ \sum_{h \in H} \mathcal{L}^{\mathrm{e}}(t^y, \mathbf{insert}_{z_h}(\mathbf{e}^{h(x)}, \mathbf{e}^a_{1:m})) \\
&\quad = \sum_{h \in H} \mathcal{L}(t^y, q(\mathbf{insert}_{z_h}(\mathbf{e}^{h(x)}, \mathbf{e}^a_{1:m}))) \\
&\quad = \sum_{h \in H} \mathcal{L}(t^y, \mathbf{insert}_{z_h}(q(\mathbf{e}^{h(x)}), q(\mathbf{e}^a_{1:m}))) \\
&\quad = \sum_{h \in H} \mathcal{L}(t^y, \mathbf{insert}_{z_h}(t^{h(x)}, t^a_{1:m})) = \mathcal{L}_{\mathrm{AE}}(y, x, a).
\end{aligned} \tag{12}
$$

□

Lemma B.3. *Given q is non-differentiable (Lemma B.1), $\mathcal{L}^{e}_{\texttt{PlugAE}}$ can reach closer or equal to the optimal solution compared to the unified loss $\mathcal{L}_{AE}$.*

Proof. Based on Lemma B.2, $\mathcal{L}_{\mathrm{AE}} = q \circ \mathcal{L}^{\mathrm{e}}_{\texttt{PlugAE}}$. The non-differentiability of q introduces points in the loss landscape where gradients are either undefined or lead to instability. This can cause optimization algorithms, particularly those relying on gradient descent, to struggle with convergence, potentially getting stuck in flat regions or diverging near discontinuities. On the other hand, $\mathcal{L}^{\mathrm{e}}_{\texttt{PlugAE}}$, provides a smooth loss landscape. Gradient-based methods can reliably compute gradients and update parameters effectively, allowing for consistent progress toward the optimal solution. Thus, $\mathcal{L}^{\mathrm{e}}_{\texttt{PlugAE}}$ is more likely to reach closer or equal to the optimal solution compared to $\mathcal{L}_{\mathrm{AE}}$, which is hindered by the non-differentiable function q. □

Claim B.1. *Based on Lemma B.3, the optimization of our* PlugAE *can be closer to the optimal compared with current query-based methods that adopt the unified loss $\mathcal{L}_{AE}$.*

Claim B.1 shows that our PlugAE can explore the extent of existing passive identification methods.

B.3 Model Watermarking

Model watermarking allows model owners to proactively protect their intellectual property before model distribution. They mainly fine-tune the LLM to learn a watermark pattern. Our PlugAE differs from these methods in the following two aspects. Firstly, our PlugAE does not require modifying the model's main transformer layers, thus preserving the model's utility. This is different from model watermarking, which requires updating the model's parameters to learn the watermark pattern. The second aspect is that we allow the model owner to define the copyright tokens. This not only reduces the collision of copyright tokens for different models using our PlugAE, but also provides chances that the corresponding adversarial token embeddings could remain after further fine-tuning.

C Challenges and Our Vision

Our comprehensive experiments on various methods have illustrated the difficulties of identifying the origin of a black-box LLM. Drawing from theoretical and empirical analysis in this study, we highlight the challenges and opportunities.

Challenges. The first challenge is that the auditor may not have sufficient knowledge of practical customizations applied to LLMs. For example, the LLM may be further fine-tuned with specific system prompts and equipped with specific hyperparameters (Gubri et al., 2024; Jin et al., 2024). The model's structure and weights can also be modified or rearranged (Zeng et al., 2023). For example, existing methods and PlugAE are proven ineffective for models that adopt the Llama architecture yet are trained independently (e.g., OpenLlama).

The second challenge is the black-box access to the suspect model. It implies that we can only query the model with text and obtain a text response. This is inevitable, as the malicious third party tends to conceal the model's true origin. Such black-box access inevitably limits the identification methods, as they are constrained to querying the model and analyzing the text outputs. For example, existing query-based fingerprinting relies on discrete optimization, as they cannot directly query the LLM with embeddings but with discrete texts. As analyzed in Appendix B, such discrete optimization makes it hard to reach the optimum, leading to performance degradation. Additionally, the adapter variant of IF (Xu et al., 2024) is not applicable as we are unable to insert an F-Adapter into the black-box LLM. Our experimental results show that PlugAE surpasses existing methods, though its efficacy declines when substantial weight modifications are present.

Our Vision. As we delve deeper into methods for identifying black-box LLMs, there is still much to be done to safeguard the intellectual property rights of developers and owners of LLMs, as well as other foundation models (Liu et al., 2023; Chen et al., 2023; Wang et al., 2023; Yao et al., 2024). To begin with, we present a real example of two large

Model Name	Model Path
Llama (Touvron et al., 2023a)	yahma/llama-7b-hf
Vicuna (Team)	lmsys/vicuna-7b-v1.3
Guanaco (Dettmers et al., 2023)	TheBloke/guanaco-7B-HF
Llama2 (Touvron et al., 2023b)	meta-llama/Llama-2-7b-hf
Llama2-Chat (Touvron et al., 2023b)	meta-llama/Llama-2-7b-chat-hf
Vicuna2 (Team)	lmsys/vicuna-7b-v1.5
CodeLlama (Rozière et al., 2023)	codellama/CodeLlama-7b-hf
CodeLlama-Python (Rozière et al., 2023)	codellama/CodeLlama-7b-Python-hf
CodeLlama-Instruct (Rozière et al., 2023)	codellama/CodeLlama-7b-Instruct-hf
Tulu2 (Ivison et al., 2023)	allenai/tulu-2-7b
Tulu2-DPO (Ivison et al., 2023)	allenai/tulu-2-dpo-7b
SciTulu (Ivison et al., 2023)	allenai/scitulu-7b
Orca2 (Mitra et al., 2023)	microsoft/Orca-2-7b
Llama2-Chat-Un (Hartford, 2023)	georgesung/llama2_7b_chat_uncensored
Mistral (Jiang et al., 2023)	mistralai/Mistral-7B-v0.1
Mistral-Instruct (Jiang et al., 2023)	mistralai/Mistral-7B-Instruct-v0.1
Zephyr (Tunstall et al., 2023)	HuggingFaceH4/zephyr-7b-beta
Mistral-v0.3 (Jiang et al., 2023)	mistralai/Mistral-7B-v0.3
Mistral-v0.3-Instruct (Jiang et al., 2023)	mistralai/Mistral-7B-Instruct-v0.3
Gemma (Team, 2024)	google/gemma-7b
Gemma-Instruct (Team, 2024)	google/gemma-7b-it
Zephyr-Gemma (Tunstall et al., 2023)	HuggingFaceH4/zephyr-7b-gemma-v0.1
Zephyr-Gemma-SFT (Tunstall et al., 2023)	HuggingFaceH4/zephyr-7b-gemma-sft-v0.1
OpenLlama (Geng and Liu, 2023)	openlm-research/open_llama_7b
OpenLlama2 (Geng and Liu, 2023)	openlm-research/open_llama_7b_v2
Llama3 (Team, 2024)	meta-llama/Meta-Llama-3-8B
Llama3-Instruct (Team, 2024)	meta-llama/Meta-Llama-3-8B-Instruct
OLMo (Groeneveld et al., 2024)	allenai/OLMo-7B-hf
OLMo-SFT (Groeneveld et al., 2024)	allenai/OLMo-7B-SFT-hf
OLMo-Instruct (Groeneveld et al., 2024)	allenai/OLMo-7B-Instruct-hf
GPT4-Turbo (OpenAI, 2023a)	gpt-4-turbo-2024-04-09
Claude3.5-Sonnet (Anthropic, 2024)	claude-3-5-sonnet-20240620

Table 8: Details of the LLMs used in the experiments. The LLMs in one group indicate that they are derivatives of the LLMs in bold. Note that the OpenLlama series uses the same framework as Llama, but they are pre-trained on different datasets.

vision-language models (VLMs) as an introduction. In June 2024, a user on GitHub raised concerns that the Llama3-V model may have improperly copied elements from the MiniCPM-Llama3-V 2.5 project (Yao et al., 2024), prompting a thorough investigation by the authors of the accused model.[3] The inquiry revealed striking similarities in code structure and minor variable name changes between Llama3-V and MiniCPM-Llama3-V 2.5. Further similarities were found in behavior, particularly in specialized tasks such as recognizing Tsinghua Bamboo Characters, which were from the in-house training data of MiniCPM-Llama3-V 2.5. This serves as a clear indicator of the infringement and exemplifies the potential benefits of proactive identification strategies.

Beyond technological solutions, we emphasize the importance of nurturing a robust open-source community. Such communities not only serve as guardians, preventing the unauthorized use of copyrighted material, but also as incubators for technological innovation. The incident with Llama3-V highlights the pivotal role that the community can play in uncovering and addressing infringement. Moreover, by participating in open-source projects, researchers and developers gain access to a collaborative environment where they can innovate and refine technologies at a pace that individual efforts cannot match.

Furthermore, while current licensing practices for foundation models are extensive (see Table 6), there is a notable deficiency in the regulation of the

[3] https://github.com/OpenBMB/MiniCPM-V/issues/196.

Dataset	Model	Original	PlugAE	IF	PlugAE-CSN	IF-CSN
HellaSwag	Llama	0.57±0.005	0.57±0.005	0.57±0.005	0.57±0.005	0.57±0.005
	Llama2	0.57±0.005	0.57±0.005	0.58±0.005	0.58±0.005	0.58±0.005
	Mistral	0.61±0.005	0.61±0.005	0.61±0.005	0.60±0.005	0.60±0.005
MMLU	Llama	0.32±0.059	0.32±0.059	0.31±0.057	0.32±0.059	0.33±0.059
	Llama2	0.41±0.088	0.41±0.088	0.41±0.089	0.40±0.085	0.42±0.085
	Mistral	0.60±0.133	0.60±0.133	0.60±0.135	0.57±0.126	0.59±0.115

Table 9: Utility on HellaSwag and MMLU. [Method]-CSN denotes that the model is first proactively modified by [Method] and then fine-tuned on CSN.

training data used to develop these models. The vast corpora required to train such models are often neither open-sourced nor clearly documented (OpenAI, 2023b; Touvron et al., 2023a; Jiang et al., 2023), raising significant concerns about copyright compliance and transparency. This lack of oversight and regulation can lead to ambiguities in ownership and ethical use, which are crucial for maintaining trust and integrity within the AI community. To address these challenges, we advocate for clearer legislation and more stringent regulatory frameworks that not only keep pace with technological advancements but also ensure that the copyrights of large training datasets are adequately protected (EUA, 2024).

D Experimental Settings

Table 8 shows the details of the models used in our experiments, including the versions we used.

As for the dataset, CodeSearchNet (CSN) (Husain et al., 2019) is a widely used dataset that contains about six million functions from open-source code spanning six programming languages (Go, Java, JavaScript, PHP, Python, and Ruby).

E Experiments

E.1 Utility on Original, Modified, and Fine-tuned LLMs

We evaluate the performance of all original, modified, and fine-tuned LLMs on HellaSwag and MMLU datasets, which are widely used as benchmarks for evaluating LLMs. Table 9 shows the results. We observe that IF does not significantly affect the model's utility, while our PlugAE keeps exactly the same results on both datasets under the same random seed. This is because our PlugAE does not modify the model's main transformer layers, so there will be no difference if there are no copyright tokens in the input. The results also support that our fine-tuning does not affect the model's utility.

E.2 AUC/ROC Results of Three LLMs

Figure 8 shows the AUC/RUC Curve of each method on three different LLMs. As shown in the results, our PlugAE achieves an AUC of 1.00 on both Llama and Mistral, outperforming the other two methods. On Llama2, PlugAE achieves an AUC of 0.91, while the AUCs of TRAP and ProFlingo are 0.55 and 0.93. This further shows the effectiveness of our method.

E.3 TSNE Visualization of Three LLMs

Figure 9 shows the visualization results of the adversarial embeddings and other randomly selected token embeddings of all three LLMs. We observe that the adversarial embedding for each LLM is within the circle of its token embeddings; there are even several original token embeddings behaving like outliers. In this sense, it is challenging to detect the adversarial embedding using outlier detection methods, exhibiting the stealthiness of our PlugAE.

E.4 Counterexample Analysis

Although our method outperforms most baselines, there are still some counterexamples. Based on Llama2, the TRRs of CodeLlama and CodeLlama-Python are 0.00, while the TRR of CodeLlama-Instruct is 0.06. Thus, we further examined the reason why PlugAE performs worse on these models. Specifically, we measure the difference between the candidate model and its derivatives. We compute the cosine distance between the weights of the two models. Figure 7 shows the relationship between the TRRs and the cosine distance between the related suspect models and the candidate model. A larger cosine distance indicates a bigger weight shift from the candidate model. We observe that the performance of PlugAE would drop to around 0.00 when the cosine distance is too large. For example, the cosine distance between CodeL-

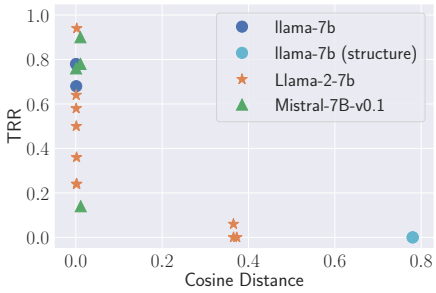
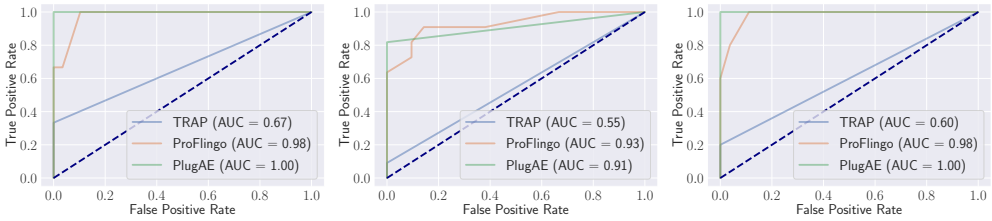


Figure 7: Relationship of TRRs and the cosine distances between the suspect models and the candidate model.

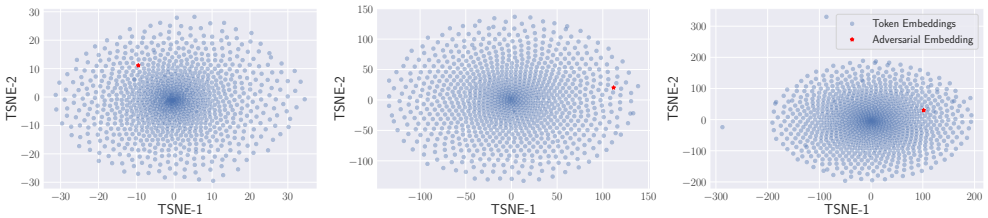
lama and Llama2 reaches 0.3653, which is much higher than other Llama2 derivatives, e.g., Tulu2 (0.0009). `PlugAE` only achieves a TRR of 0.00 on CodeLlama while reaching a TRR of 0.64 on Tulu2. This suggests that `PlugAE` could be dodged when there is a significant weight shift from the candidate model.

Additionally, OpenLlama presents a permissively licensed open-source reproduction of Meta AI's Llama model, i.e., it uses Llama's structure and is trained on another dataset by another organization. We compute the cosine distance between it and its derivatives with Llama as well, and label them as Llama (structure). Results in Figure 7 show that their weights are more distant from Llama, and `PlugAE` failed to identify them as Llama. This provides a new scenario where the adversary steals the architecture of an LLM illicitly, trains it from scratch, and distributes it for unauthorized use, e.g., commercial purposes. We do not include a discussion on this topic and leave it as future work.



(a) Llama (b) Llama2 (c) Mistral

Figure 8: ROC curve of different methods on three candidate LLMs.



(a) Llama (b) Llama2 (c) Mistral

Figure 9: T-SNE visualization of the adversarial embedding and 1,000 randomly sampled token embeddings in different candidate LLMs.

Style Name	System Prompt
Empty	-
Basic	"You are a concise AI assistant. Give short, direct answers with minimal explanation unless the user asks for more detail."
Helpful	"You are a helpful, respectful, and honest assistant. Your goal is to help the user solve problems efficiently. Be polite, practical, and focus on giving useful step-by-step guidance."
Alpaca	"Below is an instruction from a user. Provide a detailed and thoughtful response that completes the request."
ChatGPT	"You are ChatGPT, a conversational AI created by OpenAI. You give clear, structured answers, explain reasoning when helpful, and maintain a friendly but professional tone."

Table 10: Different system prompt styles.