https://doi.org/10.2352/EI.2026.38.4.MWSF-309

Unveiling Hidden Model Fingerprints in API-Protected LLMs

Zhiguang Yang, Shanghai University, Shanghai 200444, China; Email: yangzg@shu.edu.cn
Hanzhou Wu, Shanghai University, Shanghai 200444, China; Email: h.wu.phd@ieee.org (Corresponding author)

Abstract

Recent advances confirm that large language models (LLMs) can achieve state-of-the-art performance across various tasks. However, due to the resource-intensive nature of training LLMs from scratch, it is urgent and crucial to protect the intellectual property of LLMs against infringement. This has motivated the authors in this paper to propose a novel black-box fingerprinting technique for LLMs. We firstly demonstrate that the outputs of LLMs span a unique vector space associated with each model. We model the problem of fingerprint authentication as the task of evaluating the similarity between the space of the victim model and the space of the suspect model. To tackle with this problem, we introduce two solutions: the first determines whether suspect outputs lie within the victim's subspace, enabling fast infringement detection; the second reconstructs a joint subspace to detect models modified via parameter-efficient fine-tuning (PEFT). Experiments indicate that the proposed method achieves superior performance in fingerprint verification and robustness against the PEFT attacks. This work reveals inherent characteristics of LLMs and provides a promising solution for protecting LLMs, ensuring efficiency, generality and practicality.

Introduction

Large language models (LLMs) have become a foundational component of contemporary artificial intelligence, exhibiting exceptional performance across a broad spectrum of natural language processing tasks owing to their capacity to generate human-like text and comprehend complex semantics. Despite the considerable data volume and computational power required for their training, numerous developers continue to advance the field by releasing their models as open source. Research groups and organizations responsible for prominent models, such as LLaMA, Gemma, and Mistral [1, 2, 3, 4], have made their well-trained LLMs publicly accessible, thereby fostering a dynamic and collaborative research ecosystem. This rapidly evolving landscape is largely sustained by its open-source philosophy. Nevertheless, the same openness that accelerates innovation also renders these models vulnerable to misuse, e.g., unauthorized fine-tuning with other pre-trained models without proper attribution, or misappropriation through false claims of ownership. Consequently, safeguarding the intellectual property rights of LLMs is imperative not only for preserving their commercial value but also for promoting the sustainable and ethical development of the community.

Numerous studies have employed digital watermarking and fingerprinting methods to protect deep neural networks (DNNs). For example, Uchida *et al*. [5] introduce a regularization term to constrain the network weights for embedding watermarks. Subsequently, researchers have proposed various methodologies such as [6, 7, 8] to embed watermarks into the model parameters in white-box scenarios, where the extractor can access the entirety of model parameters. However, the challenges in accessing all the model parameters in various scenarios has led to a focus on black-box techniques for model watermarking. To this purpose, many watermarking methods utilize backdoor techniques, constructing specific input-output mappings and observing the output of the DNN model for verification such as [9, 10, 11]. Generative models, particularly image processing models, often produce contents with high entropy and sufficient information capacity to accommodate additional watermark information, which remains highly imperceptible. Wu *et al*. [12] introduce a new framework to make the output of a model contain a certain watermark. By extracting the watermark from any watermarked output, one can identify the ownership of the corresponding DNN model. Lukas *et al*. [13] propose embedding watermarks by fine-tuning the image generator, ensuring that all images produced are watermarked. Fernandez *et al*. [14] extend it to the diffusion model. Embedding watermarks through backdoor and fine-tuning techniques compromises the primary functionality of the model to a certain extent and requires significant computational resources. Song *et al*. [15] distinguish different generative models based on their artifacts and fingerprints, which can help alleviate this problem.

The emergence of superior reasoning capabilities in LLMs which require significant computational overhead, has posed new challenges for their intellectual property protection (IPP). Xu *et al*. [16] specify a confidential private key and embed it as an instructional backdoor, serving as a fingerprint. In Ref. [17], Zeng *et al*. utilize the internal parameters in Transformer [18] as a fingerprint to identify the LLMs. Existing methods typically require white-box access or fine-tuning to verify the copyright information. In contrast, the proposed fingerprinting method can be implemented in a black-box scenario without any fine-tuning.

We propose a novel fingerprinting approach for LLMs that enables LLM authentication through analysis of their output characteristics. LLMs generate coherent and contextually appropriate text by sampling from logits, which inherently encode rich model-specific information. In black-box scenarios, LLM providers often expose full or partial logit vectors via APIs, allowing users to employ different sampling strategies to produce realistic content. Carlini et al. [19] have demonstrated that it is possible to recover parts of a model solely through such API access. Building on this insight, we introduce an LLM fingerprinting framework that identifies distinctive properties of each model by analyzing its logit outputs from a new perspective.

Prior studies have demonstrated that the outputs of LLMs lie within a linear subspace determined by their parameters (see [20, 21, 22]). In our approach, we preserve the parameters of the victim model - specifically those of its final linear layer. By querying the suspect model and collecting its outputs, we leverage the retained parameters to characterize the model's unique attribution. We formalize ownership authentication as the process of measur-

ing the similarity between the vector space of the victim model and the output space of the suspect model.

We first introduce a method for rapidly determining whether a given output originates from the victim model by evaluating its compatibility with the vector space defined by the retained parameters. To handle scenarios involving Parameter-Efficient Fine-Tuning (PEFT) attacks, we further design an alignment verification approach that assesses whether the suspect model was derived from the victim model through PEFT by comparing their representational similarities. Moreover, in cases where only partial logits are accessible via an API, we demonstrate that complete logits can be reconstructed to facilitate ownership verification. Our framework enables model ownership authentication in black-box settings solely through API access, without relying on any specific architectural assumptions about the underlying LLM. This ensures broad applicability and offers a practical and effective solution for the copyright protection of LLMs. Experimental results show that our method achieves superior verification accuracy and robustness against PEFT attacks, without impairing the functional performance of the models.

In summary, the main contributions of this work include:

- We analyze the characteristics of LLMs from a novel perspective and demonstrate that their outputs can serve as unique fingerprints for ownership verification.
- We propose two complementary methods for LLM ownership verification: one enables rapid identification of whether an output originates from the victim model, while the other determines whether a suspect model has been derived from the victim model through a PEFT attack.
- We introduce a technique to reconstruct complete fingerprint information from partial logits obtained via API access, enabling ownership verification in black-box scenarios.

The rest structure of this paper is organized as follows. First of all, we provide an overview of model fingerprinting, PEFT, and the assumed threat model. Then, we introduce the proposed LLM fingerprinting technique, followed by two fingerprint verification approaches. Thereafter, experimental results and analysis are provided. Finally, we conclude this paper and provide discussion.

Preliminaries

In this section, we provide a brief overview of model fingerprinting, PEFT, and the assumed threat model so that the proposed work can be better described.

Model Fingerprinting

Model fingerprinting is a technique used to uniquely identify and authenticate DNN models by analyzing their inherent characteristics. Every DNN model, even those with the identical architecture, develops distinctive patterns in its parameters and outputs due to differences in training data, initialization, and optimization. Through analyzing these patterns such as logits, embeddings, or behavioral responses to specific inputs, model fingerprinting can generate a "signature" that distinguishes one model from another. This technique enables applications such as ownership verification, detection of unauthorized copies or fine-tuned derivatives, and security auditing, and it can often be applied even in black-box scenarios where only model outputs are accessible. Through these methods, model fingerprinting provides a robust and practical way to trace, verify, and protect DNN models. In this research, we utilize the outputs of LLMs as fingerprints for verification.

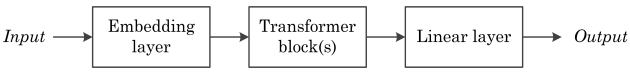


Figure 1. *The pipeline of LLMs based on the Transformer architecture.*

Parameter-Efficient Fine-Tuning (PEFT)

Training or fine-tuning DNNs from pre-trained models requires substantial computational resources, especially for LLMs that demand extensive GPU memory. Recent research has therefore focused on parameter-efficient fine-tuning (PEFT) methods, which enable effective optimization while updating only a small subset of model parameters [23, 24, 25]. In other words, PEFT is a set of techniques designed to adapt large pre-trained models to specific tasks without updating all of the model's parameters.

Instead of fine-tuning the entire model, which can be computationally expensive and memory-intensive, PEFT typically modifies only a small subset of parameters or adds lightweight modules, while keeping the majority of the model frozen. This makes fine-tuning faster, cheaper, and more storage-efficient.

Low-Rank Adaptation (LoRA) [24] has become the de facto method for PEFT, serving as the foundation for many other methods such as [25, 26]. In LoRA, the weight matrix $\mathbf{W}_{\mathrm{O}}$ is updated by the formula $\mathbf{W}_{\mathrm{N}} = \mathbf{W}_{\mathrm{O}} + \Delta\mathbf{W} = \mathbf{W}_{\mathrm{O}} + \mathbf{AB}$. During training, $\mathbf{W}_{\mathrm{O}}$ remains fixed, while the two matrices $\mathbf{A}$ and $\mathbf{B}$ encompass the least trainable parameters. In this study, we will use LoRA to mimic the PEFT attack.

Threat Model

Our threat model involves a defender, known as the model provider, and an adversary who controls a malicious user. The adversary's goal is to steal the model and falsely claim ownership. We assume that the adversary may employ PEFT attacks such as LoRA [24], to evade detection. The defender, who retains access to their own model's parameters, including the final linear layer as a fingerprint, aims to verify ownership by querying the suspect model through API access.

LLM Fingerprinting

In this section, we show that the logits outputs from LLMs span a vector space, which can be utilized for fingerprinting LLM.

LLM Outputs Span A Vector Space

The Transformer architecture has already become the base of numerous models due to its exceptional performance across a variety of tasks. LLMs are typically Transformer-based models, with their pipeline presented in Figure 1. Specifically, the input text is tokenized and converted into word embeddings, with the embedding layer represented as a matrix of size $|V| \times h$, where $|V|$ is the vocabulary size and h is the hidden size. The vocabulary size is significantly larger than the hidden size, i.e., $|V| >> h$. The embeddings are processed by the Transformer block with multiple layers, each containing a multi-head self-attention mechanism and a feed-forward layer, to calculate the intrinsic representation of the input and produce the intermediate representation $\mathbf{z}$. The last linear layer maps $\mathbf{z}$ to logits $\mathbf{s} \in \mathbf{R}^{|V|}$. The output of LLM is

generated by sampling from the logits $\mathbf{s}$. For clarity, this analysis excludes Layer Normalization [27] and RMS Normalization [28], as they primarily introduce additional multiplication terms, which do not affect the conclusions of this paper.

It is worth noting that the logits $\mathbf{s} = \mathbf{W}\mathbf{z}$, where $\mathbf{W} \in \mathbf{R}^{|V| \times h}$ is the weight matrix of the last linear layer and the rank of $\mathbf{W}$ is at most h. Every output produced by $\mathbf{W}$ is corresponding to a vector $\mathbf{s}$ that will always lie within a subspace of $\mathbf{R}^{|V|}$, spanned by the columns of $\mathbf{W}$, which is at most h-dimensional. All possible outputs of the LLM logits will span a vector space L isomorphic to the space spanned by the columns of $\mathbf{W}$, since they have the same dimensionality. Consequently, each LLM is associated with a distinctive vector space L. This uniqueness arises because the vector space $\mathbf{R}^{|V|}$ encompasses an exceedingly large number of potential subspaces for any subspace of dimension h. For example, in the case of Gemma [3], with $h = 2,048$ and $|V| = 256,000$, the number of subspaces is extremely large. Due to variations in training initialization, datasets, configurations and hardware, it is impossible for two different LLMs to cover the same vector space. This property allows us to use it as a fingerprint for LLMs and identify their ownership. The model providers only need to retain the parameters of the last linear layer in the model. Accessing the API of the suspect model enables them to retrieve the logits, thereby verifying the ownership.

Vector Space Reconstruction via API

We already demonstrate that the logits outputs from LLMs span a vector space denoted by L, which will be utilized as the LLM fingerprint. Obtaining the full logits of a model is not always feasible, as attackers aim to disclose minimal information to evade detection by the victim. In this subsection, we consider practical scenarios in which our approach reconstructs the vector space for fingerprint verification and only relies on API access, enabling the retrieval of complete vocabulary probabilities, top-k probabilities, or the top-1 probability.

Complete Probabilities of the Vocabulary

The API provides the complete probabilities $\mathbf{p} = \text{softmax}(\mathbf{s})$ over the vocabulary, all components of which are non-negative and have a sum equal to 1. This indicates that $\mathbf{p}$ is a point in the simplex $\Delta^{|V|-1}$, which lies within a $|V|-1$ dimensional subspace of $\mathbf{R}^{|V|}$. Additionally, $\mathbf{p}$ is also constrained by $\mathbf{s}$.

Due to normalization, the softmax function does not have a well-defined inverse transformation. However, if we omit it temporarily, and use the `CLR` (centered log-ratio) transformation, we can obtain $\mathbf{s}^\star$ that differs from $\mathbf{s}$ by a constant bias. This will introduce a one-dimensional deviation from the perspective of spatial dimensions. We can manually do one-dimensional correction and will not affect the results, as h and $|V|$ are both significantly greater than one. Therefore, we can directly reconstruct the vector space $L^\star$ corresponding to $\mathbf{s}^\star$, which is determined by

$$\mathbf{s}^\star = \texttt{CLR}(\mathbf{p}) = \log\left(\frac{\mathbf{p}}{g(\mathbf{p})}\right), \tag{1}$$

where $g(\cdot)$ denotes the geometric mean of the input. We demonstrate that $\mathbf{s}^\star$ differs from $\mathbf{s}$ by a constant bias. First, we have

$$\mathbf{p} = \text{softmax}(\mathbf{s}) = \frac{e^{\mathbf{s}}}{\sum_{i=1}^{|V|} e^{s_i}} \tag{2}$$

and

$$g(\mathbf{p}) = (\prod_{i=1}^{|V|} p_i)^{1/|V|}. \tag{3}$$

Then, we can find that

$$\begin{aligned} s_i^\star &= \log\left(\frac{p_i}{g(\mathbf{p})}\right) = \log(p_i) - \log(g(\mathbf{p})) \\ &= \log(p_i) - \frac{1}{|V|}\sum_{j=1}^{|V|}\log(p_j) \\ &= \log\left(\frac{e^{s_i}}{\sum_{j=1}^{|V|} e^{s_j}}\right) - \frac{1}{|V|}\sum_{j=1}^{|V|}\log\left(\frac{e^{s_j}}{\sum_{k=1}^{|V|} e^{s_k}}\right) \\ &= \left[s_i - \log\sum_{j=1}^{|V|} e^{s_j}\right] - \frac{1}{|V|}\sum_{j=1}^{|V|}\left[s_j - \log\sum_{k=1}^{|V|} e^{s_k}\right] \\ &= s_i - \frac{1}{|V|}\sum_{j=1}^{|V|} s_j, \end{aligned} \tag{4}$$

which means that each component of $\mathbf{s}^\star$ only differs from that of $\mathbf{s}$ by the mean of $\mathbf{s}$. Therefore, $\mathbf{s}^\star - \mathbf{s}$ forms a constant vector (bias).

Top-k Probabilities

In case that the API provides the top-k probabilities, which are the k largest elements of $\mathbf{p}$, we assume that the provider allows the user to alter token probabilities using a bias through the API. This is reasonable and common in applications, e.g., OpenAI includes this function in their API[1]. The bias is added to the specific logits of the tokens before the softmax operation, the API returns the top-k probabilities. Without the loss of generalization, assuming that $i_1, i_2, \ldots, i_m$ are the indices of selected tokens and b is the bias, the biased probabilities distribution $\mathbf{p}^\star$ can be calculated by

$$\mathbf{p}^\star(i_1, i_2, \ldots, i_m, b) = \text{softmax}\left(\mathbf{s}^\star(i_1, i_2, \ldots, i_m, b)\right), \tag{5}$$

where for $1 \le i \le |V|$, we have

$$s_i^\star(i_1, i_2, \ldots, i_m, b) = \begin{cases} s_i + b & i \in \{i_1, i_2, \ldots, i_m\}, \\ s_i & \text{otherwise.} \end{cases} \tag{6}$$

We propose a method to recover the complete probabilities $\mathbf{p} = \text{softmax}(\mathbf{s})$, i.e., $b \equiv 0$ for Eq. (5). In detail, by feeding a prompt to the model, we can obtain the top-k probabilities, which can be expressed as $\{p_{\text{idx}_1}, p_{\text{idx}_2}, \ldots, p_{\text{idx}_k}\}$, where $p_{\text{idx}_1} \ge p_{\text{idx}_2} \ge \ldots \ge p_{\text{idx}_k}$ and $\{\text{idx}_1, \text{idx}_2, \ldots, \text{idx}_k\}$ are the indices of the corresponding tokens. Obviously, we have

$$p_{\text{idx}_i} = \frac{e^{s_{\text{idx}_i}}}{\sum_{j=1}^{|V|} e^{s_j}}, \ \forall\ 1 \le i \le k. \tag{7}$$

We have collected k original probabilities $\{p_{\text{idx}_1}, p_{\text{idx}_2}, \ldots, p_{\text{idx}_k}\}$. We are to determine the remaining $|V| - k$ original probabilities. This problem can be solved by processing these $|V| - k$ tokens in batches each containing $k-1$ tokens. Suppose that we are now to

[1] https://help.openai.com/en/articles/5247780-using-logit-bias-to-alter-token-probability-with-the-openai-api

process $k-1$ tokens, whose indices are $\{i_1, i_2, ..., i_{k-1}\}$, we add a bias b to the logits of these $k-1$ tokens and the idx_1-th token to push them into the 'top-k'. The biased probabilities distribution can therefore be calculated by

$$\mathbf{p}^\star(\text{idx}_1, i_1, \ldots, i_{k-1}, b) = \text{softmax}\left(\mathbf{s}^\star(\text{idx}_1, i_1, \ldots, i_{k-1}, b)\right), \quad (8)$$

where for $1 \le i \le |V|$, we have

$$s_i^\star(\text{idx}_1, i_1, \ldots, i_{k-1}, b) = \begin{cases} s_i + b & i \in \{\text{idx}_1, i_1, \ldots, i_{k-1}\}, \\ s_i & \text{otherwise.} \end{cases} \quad (9)$$

We can write

$$p_{i_r} = \text{softmax}(s_{i_r}) = \frac{e^{s_{i_r}}}{\sum_{j=1}^{|V|} e^{s_j}}, \ \forall\, i_r \in \{\text{idx}_1, i_1, \ldots, i_{k-1}\}, \quad (10)$$

and

$$\begin{aligned} p_{i_r}^\star &= \text{softmax}(s_{i_r}^\star) = \frac{e^{s_{i_r}^\star}}{\sum_{j=1}^{|V|} e^{s_j^\star}} \\ &= \frac{e^{s_{i_r}^\star}}{\sum_{j \notin \{\text{idx}_1, i_1, \ldots, i_{k-1}\}} e^{s_j^\star} + \sum_{j \in \{\text{idx}_1, i_1, \ldots, i_{k-1}\}} e^{s_j^\star}} \\ &= \frac{e^{s_{i_r}+b}}{\sum_{j \notin \{\text{idx}_1, i_1, \ldots, i_{k-1}\}} e^{s_j} + \sum_{j \in \{\text{idx}_1, i_1, \ldots, i_{k-1}\}} e^{s_j+b}}, \end{aligned} \quad (11)$$

for any $i_r \in \{\text{idx}_1, i_1, \ldots, i_{k-1}\}$. Obviously,

$$\frac{p_{\text{idx}_1}}{p_{i_r}} = \frac{e^{s_{\text{idx}_1}}}{e^{s_{i_r}}} \quad (12)$$

and

$$\frac{p_{\text{idx}_1}^\star}{p_{i_r}^\star} = \frac{e^{s_{\text{idx}_1}^\star}}{e^{s_{i_r}^\star}} = \frac{e^{s_{\text{idx}_1}+b}}{e^{s_{i_r}+b}} = \frac{e^{s_{\text{idx}_1}}}{e^{s_{i_r}}}. \quad (13)$$

Therefore, for any $i_r \in \{i_1, \ldots, i_{k-1}\}$,

$$p_{i_r} = p_{\text{idx}_1} \cdot \frac{p_{i_r}^\star}{p_{\text{idx}_1}^\star}, \quad (14)$$

indicating that the original probability for the i_r-th token p_{i_r} can be recovered since p_{idx_1}, $p_{i_r}^\star$ and $p_{\text{idx}_1}^\star$ are all known. By querying the model with the same prompt, the complete probabilities $\mathbf{p}$ can be perfectly reconstructed, according to the above operation.

It is remarked that the above operation processes $k-1$ tokens at each time. In fact, it is free for us to process less then k tokens at each time. Therefore, we should perform at least $(|V|-k)/(k-1)$ times. In case that $k-1$ cannot divide $|V|-k$, we need to perform once more. In addition, at each time, we have to choose a suitable value for b, which can be done by a heuristic fashion.

*Top-*1 *Probability*

The API returns the highest probability, which is equivalent to obtaining the top-k probabilities with $k=1$. For this scenario, we present a method to recover the complete probabilities $\mathbf{p}$. Let us introduce a substantial bias b to the i-th token, thereby elevating it to the top position, resulting in the biased probability $p_i^\star$. The unbiased probability p_i for the i-th token can be determined by

$$p_i = \frac{1}{e^{b - \log p_i^\star} - e^b + 1}. \quad (15)$$

This can be easily verified. First of all, we have

$$p_i = \frac{e^{s_i}}{\sum_{j=1}^{|V|} e^{s_j}} \quad (16)$$

and

$$p_i^\star = \frac{e^{s_i+b}}{\sum_{j=1, j\neq i}^{|V|} e^{s_j} + e^{s_i+b}}, \quad (17)$$

which can be rewritten as

$$\begin{aligned} p_i^\star &= \frac{e^{s_i+b}}{\sum_{j=1}^{|V|} e^{s_j} - e^{s_i} + e^{s_i+b}} \\ &= \frac{e^{s_i} \cdot e^b}{\sum_{j=1}^{|V|} e^{s_j} - e^{s_i} + e^{s_i} \cdot e^b} \\ &= \frac{\left(p_i \cdot \sum_{j=1}^{|V|} e^{s_j}\right) \cdot e^b}{\sum_{j=1}^{|V|} e^{s_j} - \left(p_i \cdot \sum_{j=1}^{|V|} e^{s_j}\right) + \left(p_i \cdot \sum_{j=1}^{|V|} e^{s_j}\right) \cdot e^b} \\ &= \frac{p_i \cdot e^b}{1 - p_i + p_i \cdot e^b} \\ &= \frac{e^b}{p_i^{-1} - 1 + e^b}. \end{aligned} \quad (18)$$

Therefore, we have

$$p_i^{-1} = \frac{e^b}{p_i^\star} - e^b + 1. \quad (19)$$

Theoretically, this method could recover the unbiased probability in the top-k scenario, though it would require a substantially larger number of queries. However, its practical applicability is limited due to exponential operations, which introduce numerical instability and can adversely affect subsequent results. In contrast, the method described previously relies solely on proportional operations, thereby largely avoiding numerical instability issues.

Fingerprint Verification

We introduce two methods for LLM fingerprint verification. The first method involves verifying whether the outputs of the suspect LLM occupy the same space as those of the victim LLM. It means that the suspect model and the victim model share the same last linear layer, facilitating the rapid identification of model infringement. If the model was fine-tuned, indicating that the parameters of the last linear layer have been modified, i.e., changes occur in the vector space, which makes verification challenging. To deal with this problem, we propose an alignment verification method to resolve this challenge. This method calculates the joint space dimension formed by the output vector space of the suspect model and the parameter space of the victim model. Model infringement is identified by comparing the similarity of these two spaces. Figure 2 shows the sketch of LLM fingerprint verification.

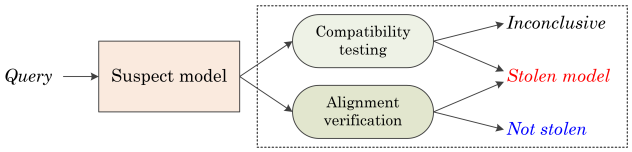


Figure 2. *The sketch of LLM fingerprint verification.*

Compatibility Testing

As demonstrated above, the LLM outputs span a vector space L isomorphic to the space spanned by the columns of the last linear layer $\mathbf{W}$. We slightly abuse the notation by using L to represent the space by the columns of the last linear layer $\mathbf{W}$ in the following. We retain the last linear layer of LLM as private, serving as the fingerprint of the victim model. The suspect model is queried through the API to obtain its logits output $\mathbf{s}$.

For the suspect model derived from the victim model without modification of the last linear layer, the logits $\mathbf{s}$ is expected to lie within L. This can be formally verified by attempting to solve the linear system $\mathbf{W}\mathbf{x} = \mathbf{s}$, to determine whether $\mathbf{s}$ belongs to L. If a solution exists, the suspect model is considered to be derived from the victim model. In practice, due to numerical errors introduced by floating-point computations, we instead compute the Euclidean distance d between $\mathbf{s}$ and the subspace L by

$$d = \|\mathbf{s} - \mathbf{W}\hat{\mathbf{x}}\|, \tag{20}$$

where $\hat{\mathbf{x}}$ denotes the solution of $\mathbf{W}\mathbf{x} = \mathbf{s}$. Let e represent the error tolerance induced by numerical instability. If $d < e$, we conclude that $\mathbf{s}$ is compatible with L, indicating that the suspect model is likely derived from the victim model.

When only the output probabilities $\mathbf{p}$ are accessible, and logits $\mathbf{s}^\star$ are reconstructed using the method described previously, we account for the inherent one-dimensional deviation by appending a constant column vector to $\mathbf{W}$, yielding $\mathbf{W}^\star = [\mathbf{W}, \mathbf{1}]$. The verification procedure is thereafter carried out using $\mathbf{W}^\star$ in place of $\mathbf{W}$, following the same methodology outlined above.

Alignment Verification

Since PEFT alters parts of the model parameters, the verification method described above can no longer be applied reliably. In line with the manifold hypothesis, high-dimensional data can be mapped onto a low-dimensional latent manifold, within which different tasks are controlled by distinct feature directions. Therefore, when fine-tuning for a specific task, it is assumed to impact only a portion of the vector space L, rather than the entire space. Especially, in PEFT attacks, the model is fine-tuned with a low-rank matrix. The original matrix $\mathbf{W}_\mathrm{O}$ is updated by

$$\mathbf{W}_\mathrm{N} = \mathbf{W}_\mathrm{O} + \Delta\mathbf{W}, \tag{21}$$

where $\Delta\mathbf{W}$ is a matrix of rank no more than k. If there is substantial overlap between between of the output space of the suspect model and the space spanned by the columns of the victim model $\mathbf{W}$, the suspect model closely resembles the victim model and may have been derived through unauthorized replication.

We calculate the dimension formed by the union of the vector space of the suspect model and the parameter space of the victim model, denoted by L_sum. If the dimension of L_sum is quite close to that of the space of the victim model, it indicates that the suspect model is derived from the victim model. Otherwise, it is not. It can be controlled by using a threshold. For those matrices containing numerous floating-point numbers, directly calculating their rank is not advisable, as it leads to significant errors due to numerical inaccuracies. Instead, we here propose to calculate the dimension difference Δr between L_sum and L (corresponding to $\mathbf{W}$) to determine whether the suspect model is derived from the victim model. If Δr is smaller than a small threshold (or say Δr is significantly smaller than the hidden size h), it means strong alignment and supports the inference that the suspect model originates from the victim model. In scenarios where only output probabilities are accessible, Δr will only experience a numerical disturbance of one, which will not affect our results. Algorithm 1 gives the pseudocode for dimension difference calculation.

Algorithm 1 Pseudocode for dimension difference calculation

Input: The parameter matrix $\mathbf{W}$, the logits set $S = \{\mathbf{s}_1, \mathbf{s}_2, ..., \mathbf{s}_q\}$, the error term e_r.
Output: The dimension difference Δr.

```
Initiate Δr = 0 and W_sum = W
for i = 1, 2, ..., q do
    Solve W_sum · x_i = s_i to obtain x̂_i
    Calculate d_i = ||s_i − W_sum · x̂_i||
    if d_i > e_r then
        Δr = Δr + 1
        W_sum = [W_sum, s_i]
    end if
end for
return Δr
```

Experimental Results and Analysis

In this section, we are to report experimental results and provide analysis to demonstrate the applicability of our work.

Setup

We use Gemma [3] with a hidden size of 2048 as the victim model and fine-tune it by either LoRA [24] or QLoRA [25] as the adversarial setting. We set the rank to 16, 32 and 64. The models are fine-tuned on the Alpaca dataset [29] and SAMSum dataset [30] to simulate different scenarios. We also compare with a new version termed as Gemma-2, which shares the same structure but under a novel training method, leading to substantial income and completely different outputs for the identical inputs. For each experiment, we generate (or recover) 300 complete logits (or probabilities) for simulation. The error parameter e_r is fixed to 1 across all experiments. The following subsections report results for compatibility testing and alignment verification, followed by a case study that illustrates the practical applicability of our method[2].

Compatibility Testing Results

In the compatibility testing, we assume that malicious users might release their stolen model either in its original form or after fine-tuning. Fine-tuning is applied to the attention mechanism in the intermediate layers or the last linear layer. Here, we retain the parameters of the model's last linear layer and conduct verification experiments across various scenarios. The average Euclidean

[2]Code available: https://github.com/solitude-alive/llm-fingerprint

Table 1. Compatibility testing results on different model versions and models fine-tuned with different modules and ranks.

Model	Dataset	Scenario			
		Full logits	Full probabilities	Top-5 probabilities	Top-1 probabilities
Gemma	/	8.0×10^{-5}	3.4×10^{-3}	3.4×10^{-3}	8.3×10^{-5}
Gemma-2		5.5×10^{5}	5.5×10^{5}	5.5×10^{5}	5.5×10^{5}
qkv-16	SAMSum	9.7×10^{-5}	1.4×10^{-4}	1.4×10^{-4}	1.0×10^{-4}
qkv-32		8.5×10^{-5}	3.6×10^{-3}	3.6×10^{-3}	9.6×10^{-5}
qkv-64		1.1×10^{-4}	9.5×10^{-4}	9.5×10^{-4}	9.9×10^{-5}
linear-16		7.3×10^{5}	7.2×10^{5}	7.2×10^{5}	7.2×10^{5}
linear-32		6.9×10^{5}	6.8×10^{5}	6.8×10^{5}	6.9×10^{5}
linear-64		7.6×10^{5}	7.5×10^{5}	7.5×10^{5}	7.5×10^{5}
qkv-16	Alpaca	8.3×10^{-5}	5.7×10^{-4}	5.7×10^{-4}	8.4×10^{-4}
qkv-32		7.0×10^{-5}	2.8×10^{-3}	2.8×10^{-3}	8.5×10^{-5}
qkv-64		6.8×10^{-5}	1.6×10^{-4}	1.6×10^{-4}	8.1×10^{-5}
linear-16		6.8×10^{5}	6.7×10^{5}	6.7×10^{5}	6.7×10^{5}
linear-32		6.7×10^{5}	6.6×10^{5}	6.6×10^{5}	6.6×10^{5}
linear-64		6.9×10^{5}	6.8×10^{5}	6.8×10^{5}	6.9×10^{5}

Table 2. Dimension difference results on different model versions and models fine-tuned with different modules and ranks.

Model	Dataset	Scenario			
		Full logits	Full probabilities	Top-5 probabilities	Top-1 probabilities
Gemma	/	0	1	1	1
Gemma-2		300	300	300	300
qkv-16	SAMSum	0	1	1	1
qkv-32		0	1	1	1
qkv-64		0	1	1	1
linear-16		10	11	11	11
linear-32		11	12	12	11
linear-64		10	11	11	11
qkv-16	Alpaca	0	1	1	1
qkv-32		0	1	1	1
qkv-64		0	1	1	1
linear-16		15	16	16	16
linear-32		20	21	21	20
linear-64		21	22	22	21

distance d is used to quantify the performance. As shown in Table 1, each column corresponds to the results of d under different API scenarios, while each row represents the outcomes for different models. 'qkv' and 'linear' represent LoRA module was applied to the query, key and value module in the attention mechanism or the linear module in the last layer, with their suffixes indicating the rank of LoRA. The experimental results clearly reveal that for models fine-tuned on the last layer or not, a significant difference exits between them and other models. This distinction is adequate for ascertaining ownership, verifying the effectiveness of the proposed method. It is worth to note that although this is the average results, this consistency will hold even with only a few samples in practice, thus also enabling rapid verification.

Alignment Verification Results

In case of alignment verification, Table 2 presents the results of the dimension difference calculation, i.e., Δr. The configuration of PEFT attacks is the same as that in the previous subsection. As shown in Table 2, the proposed method consistently distinguishes between different types of fine-tuning across all four API scenarios. For models fine-tuned on attention modules, the metric returns uniformly small Δr, indicating strong consistency with the original model. In contrast, models fine-tuned on the final linear layer exhibit substantially larger Δr that grows systematically with LoRA rank. All these observations are not merely highlighting the difference between qkv and linear fine-tuning; rather, they directly demonstrate the discriminative power of our method. Regardless of the fine-tuning location, fine-tuning intensity, or the granularity of API outputs, our method reliably captures the structural deviations introduced by PEFT, enabling effective identification of stolen or fine-tuned variants. The consistent trends observed across all API scenarios further validate the robustness and sample efficiency of the proposed verification method. Table 3 reports the dimension difference results for models fine-tuned with

Table 3. Dimension difference results for models fine-tuned with QLoRA at different ranks across different datasets.

Model	Dataset	Scenario			
		Full logits	Full probabilities	Top-5 probabilities	Top-1 probabilities
Gemma	/	0	1	1	1
Gemma-2		300	300	300	300
qkv-16	SAMSum	0	1	1	1
qkv-32		0	1	1	1
qkv-64		0	1	1	1
linear-16		9	10	10	10
linear-32		10	11	11	11
linear-64		8	9	9	9
qkv-16	Alpaca	0	1	1	1
qkv-32		0	1	1	1
qkv-64		0	1	1	1
linear-16		16	17	17	17
linear-32		20	21	21	21
linear-64		23	24	24	24

QLoRA at different ranks across different datasets. From this table, we can draw out the same conclusions.

It is noted that $\Delta r = 300$ reported for Gemma-2 in Table 2 and Table 3 is a result of the fact that we utilize only 300 valid complete logits (or probabilities) for simulation. If this restriction is relaxed and more outputs are included, the observed Δr would likely increase proportionally. Indeed, analysis of 3000 outputs from Gemma-2 yields $\Delta r = 2052$, which suggests extensive fine-tuning or even training from scratch. This further emphasizes the uniqueness and reliability of the proposed fingerprint.

Case Study

We conduct a case study on five Llama-family models sharing same architectures to further evaluate the effectiveness of our fingerprinting method under realistic conditions of high model similarity. In Figure 3, the contribution logits and weights, computed via the Euclidean distance metric and visualized after logarithmic transformation, exhibit a clear and stable pattern, that is, only the diagonal entries show markedly elevated values, while all off-diagonal entries remain uniformly low. This indicates that our metric consistently identifies each model as most compatible with itself and effectively rejects all other closely related variants. Crucially, this pronounced diagonal dominance does not arise from subtle architectural differences but directly reflects the discriminative strength and reliability of our method. Even among models with identical architectures and highly similar training objectives, the resulting fingerprints remain distinctive and stable, enabling robust and trustworthy model attribution in practical applications.

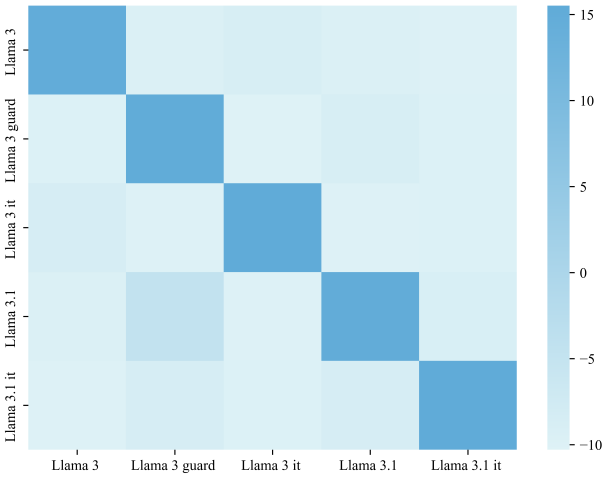


Figure 3. *A case study on Llama-family models.*

Conclusion and Discussion

In this paper, we introduce a novel and general fingerprinting framework for large language models that enables reliable ownership verification in black-box settings without retraining or modifying the target model. Across multiple model families, API granularities, and PEFT attack configurations, our experiments show that the proposed method produces distinctive and stable fingerprints. The approach consistently achieves high verification accuracy and remains robust to LoRA fine-tuning on various modules. These findings highlight persistent structural properties of LLMs that enable dependable model attribution even under adversarial conditions. Overall, this research offers a practical and effective solution for LLM provenance tracking and ownership protection, and outlines a promising direction for future research on secure deployment of generative models.

Acknowledgment

This study was financially supported by the Nanning "Yong Jiang" Program under Grant Number RC20250102, Science and Technology Commission of Shanghai Municipality under Grant Number 24ZR1424000, and Xizang Autonomous Region Central Guided Local Science and Technology Development Fund Project under Grant Number XZ202401YD0015. This research was also partly supported by the National Natural Science Foundation of China under Grant Number U23B2023.

References

[1] H. Touvron, T. Lavril, G. Izacard *et al.* LLaMA: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*, 2023.

[2] H. Touvron, L. Martin, K. Stone *et al.* Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*, 2023.

[3] T. Mesnard, C. Hardin, R. Dadashi *et al.* Gemma: Open models based on gemini research and technology. *arXiv preprint arXiv:2403.08295*, 2024.

[4] A. Jiang, A. Sablayrolles, A. Mensch *et al.* Mistral 7B. *arXiv preprint arXiv:2310.06825*, 2023.

[5] Y. Uchida, Y. Nagai, S. Sakazawa, S. Satoh. Embedding watermarks into deep neural networks. In: *Proc. ACM International Conference on Multimedia Retrieval*, pp. 269-277, 2017.

[6] J. Wang, H. Wu, X. Zhang, Y. Yao. Watermarking in deep neural networks via error back-propagation. In: *Proc. IS&T Electronic Imaging, Media Watermarking, Security and Forensics*, pp. 22-1-22-9(9), 2020.

[7] B. Rouhani, H. Chen, F. Koushanfar. DeepSigns: An end-to-end watermarking framework for ownership protection of deep neural networks. In: *Proc. International Conference on Architectural Support for Programming Languages and Operating Systems*, pp. 485-497, 2019.

[8] P. Fernandez, G. Couairon, T. Furon, M. Douze. Functional invariants to watermark large Transformers. In: *Proc. IEEE International Conference on Acoustics, Speech and Signal Processing*, pp. 4815-4819, 2024.

[9] Y. Adi, C. Baum, M. Cisse *et al.* Turning your weakness into a strength: watermarking deep neural networks by backdooring. In: *Proc. USENIX Conference on Security Symposium*, pp. 1615-1631, 2018.

[10] X. Zhao, H. Wu, X. Zhang. Watermarking graph neural networks by random graphs. In: *Proc. IEEE International Symposium on Digital Forensics and Security*, pp. 1-6, 2021.

[11] Y. Liu, H. Wu, X. Zhang. Robust and imperceptible black-box dnn watermarking based on fourier perturbation analysis and frequency sensitivity clustering. *IEEE Transactions on Dependable and Secure Computing*, pp. 21, no. 6, pp. 5766-5780, 2024.

[12] H. Wu, G. Liu, Y. Yao, X. Zhang. Watermarking neural networks with watermarked images. *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 31, no. 7, pp. 2591-2601, 2021.

[13] N. Lukas, F. Kerschbaum. PTW: Pivotal tuning watermarking for pre-trained image generators. In: *Proc. USENIX Conference on Security Symposium*, pp. 2241-2258, 2023.

[14] P. Fernandez, G. Couairon, H. Jegou *et al.* The stable signature: Rooting watermarks in latent diffusion models. In: *IEEE International Conference on Computer Vision*, pp. 22466-22477, 2023.

[15] H. Song, M. Khayatkhoei, W. AbdAlmageed. ManiFPT: Defining and analyzing fingerprints of generative models. In: *Proc. IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 10971-10981, 2024.

[16] J. Xu, F. Wang, M. Ma *et al.* Instructional fingerprinting of large language models. In: *Proc. NAACL*, pp. 3277-3306, 2024.

[17] B. Zeng, L. Wang, Y. Hu *et al.* HuRef: Human-readable fingerprint for large language models. In: *Proc. Neural Information Processing Systems (NeurIPS)*, 2024.

[18] A. Vaswani, N. Shazeer, N. Parmar *et al.* Attention is all you need. In: *Proc. Neural Information Processing Systems (NeurIPS)*, 2017.

[19] N. Carlini, D. Paleka, K. D. Dvijotham *et al.* Stealing part of a production language model. In: *Proc. International Conference on Machine Learning*, pp. 5680-5705, 2024.

[20] Z. Yang, Z. Dai, R. Salakhutdinov *et al.* Breaking the softmax bottleneck: A high-rank RNN language model. In: *Proc. International Conference on Learning Representations*, 2018.

[21] M. Finlayson, J. Hewitt, A. Koller *et al.* Closing the curious case of neural text degeneration. In: *Proc. International Conference on Learning Representations*, 2024.

[22] M. Finlayson, X. Ren, S. Swayamdipta. Logits of API-protected LLMs leak proprietary information. In: *Proc. Conference on Language Model*, 2024.

[23] N. Houlsby, A. Giurgiu, S. Jastrzebski *et al.* Parameter-efficient transfer learning for NLP. In: *Proc. International Conference on Machine Learning*, pp. 2790-2799, 2019.

[24] E. Hu, Y. Shen, P. Wallis *et al.* LoRA: Low-rank adaptation of large language models. In: *Proc. International Conference on Learning Representations*, 2022.

[25] T. Dettmers, A. Pagnoni, A. Holtzman, L. Zettlemoyer. QLoRA: Efficient finetuning of quantized LLMs. In: *Proc. Neural Information Processing Systems (NeurIPS)*, 2023.

[26] F. Meng, Z. Wang, M. Zhang. PiSSA: Principal singular values and singular vectors adaptation of large language models. In: *Proc. Neural Information Processing Systems (NeurIPS)*, 2024.

[27] J. Ba, J. Kiros, G. E. Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.

[28] B. Zhang, R. Sennrich. Root mean square layer normalization. In: *Proc. Neural Information Processing Systems (NeurIPS)*, 2019.

[29] Y. Dubois, X. Li, R. Taori *et al.* AlpacaFarm: A simulation framework for methods that learn from human feedback. In: *Proc. Neural Information Processing Systems (NeurIPS)*, 2023.

[30] B. Gliwa, I. Mochol, M. Biesek, A. Wawer. SAMSum Corpus: A human-annotated dialogue dataset for abstractive summarization. In: *Proc. 2nd Workshop on New Frontiers in Summarization*, 2019.

Author Biography

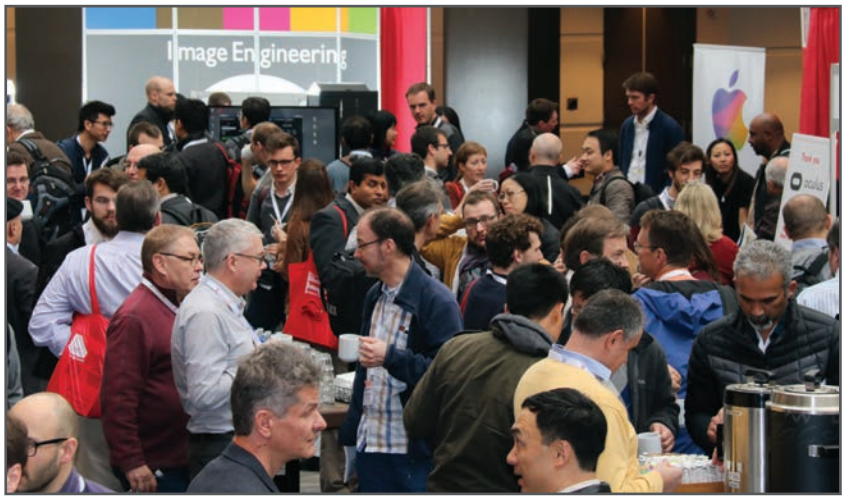
Zhiguang Yang received his BS and MS degrees from Shanghai University, Shanghai, China, in 2022 and 2025, respectively. His research interests include deep learning, large language models, digital watermarking and fingerprinting. He is now an Algorithm Engineer at a startup.

Hanzhou Wu received his BS and PhD degrees from Southwest Jiaotong University, Chengdu, China, in 2011 and 2017, respectively. He was a Visiting Scholar in New Jersey Institute of Technology, New Jersey, USA, from 2014 to 2016. He was a Research Scientist in Institute of Automation, Chinese Academy of Sciences, Beijing, China, from 2017 to 2019. He is now an Associate Professor in Shanghai University, Shanghai, China. His research interests include steganography, steganalysis, digital watermarking and digital forensics. He has published more than 100 research articles in peer journals and conferences. He has also written four book chapters. He served as the Organization Chair for 2022 IEEE International Workshop on Information Forensics and Security, and serves as an Associate Editor for IEEE Signal Processing Letters started from 2025.

JOIN US AT THE NEXT EI!

electronic IMAGING

Imaging across applications . . . Where industry and academia meet!



• SHORT COURSES • EXHIBITS • DEMONSTRATION SESSION • PLENARY TALKS •
• INTERACTIVE PAPER SESSION • SPECIAL EVENTS • TECHNICAL SESSIONS •

www.electronicimaging.org

ADVANCING IMAGING SCIENCE 75 1947 IS&T 2022 imaging.org