TENSORGUARD: Gradient-Based Model Fingerprinting for LLM Similarity Detection and Family Classification

Zehao Wu*†, Yanjie Zhao*†, and Haoyu Wang‡†
† Huazhong University of Science and Technology, Wuhan, China
wuzehao195@hust.edu.cn, yanjie_zhao@hust.edu.cn, haoyuwang@hust.edu.cn

***Abstract*—As Large Language Models (LLMs) become integral software components in modern applications, unauthorized model derivations through fine-tuning, merging, and redistribution have emerged as critical software engineering challenges. Unlike traditional software where clone detection and license compliance are well-established, the LLM ecosystem lacks effective mechanisms to detect model lineage and enforce licensing agreements. This gap is particularly problematic when open-source model creators, such as Meta's LLaMA, require derivative works to maintain naming conventions for attribution, yet no technical means exist to verify compliance.**

To fill this gap, treating LLMs as software artifacts requiring provenance tracking, we present TENSORGUARD, a gradient-based fingerprinting framework for LLM similarity detection and family classification. Our approach extracts model-intrinsic behavioral signatures by analyzing gradient responses to random input perturbations across tensor layers, operating independently of training data, watermarks, or specific model formats. TENSORGUARD supports the widely-adopted `safetensors` format and constructs high-dimensional fingerprints through statistical analysis of gradient features. These fingerprints enable two complementary capabilities: direct pairwise similarity assessment between arbitrary models through distance computation, and systematic family classification of unknown models via the K-Means clustering algorithm with domain-informed centroid initialization using known base models. Experimental evaluation on 58 models comprising 8 base models and 50 derivatives across five model families (Llama, Qwen, Gemma, Phi, Mistral) demonstrates 94% classification accuracy under our centroid-initialized K-Means clustering. Our work establishes a new paradigm for model similarity detection, bridging traditional software engineering practices with modern LLM distribution and compliance challenges.

I. INTRODUCTION

The proliferation of Large Language Models (LLMs) has fundamentally transformed how we conceptualize and deploy AI-powered software systems. With over one million model repositories on platforms like Hugging Face [1], LLMs have evolved from research artifacts into critical software components powering applications from code generation to intelligent assistants. However, this transformation has exposed a significant gap in software engineering practices: while traditional software enjoys mature ecosystems for clone detection [2], license compliance [3], and intellectual property protection, **the LLM domain lacks equivalent mechanisms for model lineage tracking, derivative work detection, and architectural family classification**.

This challenge is exemplified by licensing requirements from major open-source LLM providers. Meta's LLaMA 3 license explicitly requires derivative models to retain "Llama 3" in their naming [4], while other providers impose similar attribution requirements [5]. Yet unlike traditional software where static analysis tools can detect code reuse and licensing violations [6], **the model ecosystem operates without technical means to verify compliance or classify models into their originating families**. This regulatory vacuum is concerning given that as of July 2024, only 37% of publicly released models provided license information [7], suggesting widespread non-compliance with attribution requirements.

The technical challenges of LLM similarity detection and family classification differ fundamentally from traditional software analysis. While conventional approaches analyze syntactic patterns, control flow graphs, or semantic representations of source code [6], LLM analysis must capture behavioral and architectural characteristics encoded in high-dimensional parameter spaces through model fingerprinting techniques. Models undergo complex transformations through fine-tuning, parameter merging, and quantization that alter their internal representations while potentially preserving core architectural DNA from their base models, making **both similarity detection and family classification inherently dependent on robust fingerprinting methodologies**.

Existing approaches to model fingerprinting fall into three categories, each with significant limitations for practical applications. **(1) Watermarking-based methods** [8], [9] require pre-deployment modifications that may degrade model performance and can only be utilized by original model owners, rendering them unusable for third-party auditors. **(2) Output-based fingerprinting** [10], [11] relies on behavioral analysis through prompting but struggles with fine-tuned models and suffers from generation randomness that makes it difficult to establish stable fingerprints. **(3) Internal feature analysis**

* Zehao Wu and Yanjie Zhao contributed equally to this work.
† The full name of the author's affiliation is Hubei Key Laboratory of Distributed System Security, Hubei Engineering Research Center on Big Data Security, School of Cyber Science and Engineering, Huazhong University of Science and Technology.
‡ Haoyu Wang (haoyuwang@hust.edu.cn) is the corresponding author.

2643-1572/25/$31.00 ©2025 IEEE
DOI 10.1109/ASE63991.2025.00044

methods [12], [13] show promise for similarity detection, with REEF leveraging Centered Kernel Alignment (CKA) [14] to quantify representation similarity, while approaches like MoTHer [15] attempt family classification through heritage recovery. However, these methods exhibit critical limitations: poor compatibility with the dominant `safetensors` format [16], restricted support for specific model families (MoTHer supports only LLaMA2 and Stable Diffusion), lack of open-source availability, and assumptions about tensor layouts that may not generalize across diverse model architectures encountered in practice.

To bridge this gap, we propose TENSORGUARD, a gradient-based fingerprinting framework specifically designed for LLM similarity detection and family classification. Our approach treats LLMs as software artifacts requiring provenance tracking, drawing inspiration from traditional clone detection while addressing the unique challenges of neural model analysis. TENSORGUARD extracts model fingerprints by analyzing gradient responses to controlled input perturbations across tensor layers, capturing intrinsic behavioral characteristics that persist through common model modification techniques. Unlike existing approaches, our method operates independently of training data, embedded watermarks, and provides native support for the `safetensors` format, making it suitable for third-party auditing and compliance verification scenarios common in software engineering workflows.

Our key contributions are shown as follows:

- **Novel gradient-based model fingerprinting**: We introduce a perturbation-driven approach that captures model-intrinsic behavioral signatures through structured gradient analysis, creating unique fingerprints that enable both direct pairwise similarity assessment between arbitrary models and systematic family classification, contributing to the challenge of model provenance tracking in modern LLM systems.
- **Dual-capability detection framework**: We implement TENSORGUARD with native support for `safetensors` format and an extensible architecture that supports two complementary applications: distance-based similarity measurement for any model pair, and centroid-initialized clustering for classifying unknown models into established architectural families using known base models as reference points.
- **Comprehensive empirical validation**: We evaluate our approach on 58 models comprising 8 base models and 50 derivatives spanning five major model families (LLaMA, Qwen, Gemma, Phi, and Mistral). Our centroid-initialized K-Means clustering achieves 94% accuracy in family classification, while our distance-based similarity measurement provides effective benchmarks for gradient-based model comparison across arbitrary model pairs.

This work establishes a solid foundation for treating model similarity detection as a core software engineering practice, enabling the systematic development of automated compliance tools, robust license verification systems, and comprehensive intellectual property protection mechanisms that **are essential for the secure and mature deployment of LLM-based software systems**. The code and data for this work are available at https://github.com/security-pride/TensorGuard.

II. BACKGROUND

A. `Safetensors` Format

As the deployment of LLMs becomes increasingly prevalent, ensuring the security and efficiency of model serialization formats has emerged as a paramount concern. Conventional model formats, including `.bin`, `.pt`, and `.pth` files, typically employ Python's `pickle` module for serialization, thereby introducing severe security vulnerabilities due to the potential for arbitrary code execution during the deserialization process [17]. To address these critical security concerns, the Hugging Face community developed the `safetensors` format, which enhances security while preserving zero-copy and lazy-loading capabilities [18].

The `safetensors` format employs a well-defined binary structure: an initial 8-byte header specifies the length of a UTF-8 encoded JSON metadata section, which contains essential tensor information including names, shapes, data types, and byte offsets. The subsequent portion of the file stores the actual tensor data in binary format [18]. This architectural design achieves a clear separation between data and executable code, thereby eliminating the code injection vulnerabilities inherent in `pickle`-based serialization schemes. Relative to alternative formats, `safetensors` demonstrates substantial advantages across multiple dimensions: security, memory efficiency, and compatibility with large-scale model deployment pipelines. Notable features include support for lazy loading mechanisms, fine-grained layout control for grouped tensors, and hardware-efficient formats such as BF16 and FP8. Comparative analyses of existing model formats indicate that `safetensors` uniquely achieves the dual objectives of security and performance without compromising flexibility [18].

Nevertheless, despite its enhanced security architecture, the `safetensors` format remains vulnerable to certain format-level attacks, including denial-of-service exploits through overlapping tensor offsets or malformed tensor specifications. Recommended mitigation strategies encompass rigorous offset validation and comprehensive range checking during model loading procedures. As of 2025, the format has achieved widespread adoption, with over 700,000 models on the Hugging Face platform utilizing `safetensors` for distribution [16], establishing it as the de facto standard for secure LLM storage and distribution.

B. Fine-tuning Techniques for LLMs

Pre-trained LLMs are typically developed through training on extensive corpora using general-purpose objectives. While these models exhibit robust generalization capabilities, their outputs frequently fail to satisfy task-specific requirements when deployed directly in specialized domains. For example, when queried for legal counsel, an LLM may generate hallucinated or fabricated regulations due to insufficient exposure to domain-specific legal datasets during pre-training. To mitigate

such limitations, fine-tuning methodologies have been developed to adapt pre-trained models for downstream applications, thereby enhancing both task performance and alignment with user expectations [19]. Fine-tuning techniques can be systematically categorized into four principal paradigms based on their underlying mechanisms: full-parameter fine-tuning, parameter-efficient fine-tuning [20], prompt-based tuning [21], and reinforcement learning-based fine-tuning [22].

Full-Parameter Fine-Tuning (FFT) involves updating all parameters of a pre-trained model and represents the foundational fine-tuning methodology. The transfer learning framework introduced with BERT [23] exemplifies this technique. While this method achieves superior accuracy, it incurs substantial computational and storage costs, particularly when applied to large-scale models containing billions of parameters. **Parameter-Efficient Fine-Tuning (PEFT)** seeks to minimize resource requirements by adjusting only a limited subset of parameters or incorporating lightweight architectural components. Representative techniques include Adapter Tuning [24], Prefix-Tuning [25], and Low-Rank Adaptation (LoRA) [26]. Among these approaches, LoRA has garnered considerable attention due to its theoretical foundation and practical effectiveness. LoRA exploits the observation that pre-trained models exhibit low intrinsic dimensionality [27] and adapts the model by injecting low-rank decomposition matrices into frozen pre-trained weights. Analogous to LoRA, other parameter-efficient methods employ the strategy of freezing the majority of model parameters while fine-tuning a minimal set of trainable components, thereby substantially reducing training overhead [20]. **Prompt-Based Tuning** involves designing task-specific prompts to guide model behavior without modifying the underlying parameters. A closely related approach, instruction tuning, utilizes comprehensive datasets of human-authored or synthetically generated instructions to enhance the model's generalization capacity and zero-shot performance across diverse tasks [28]. **Reinforcement Learning-Based Fine-Tuning** employs human or AI-generated feedback as reward signals to align model outputs with user preferences and values. Prominent methodologies include Reinforcement Learning from Human Feedback (RLHF), pioneered by OpenAI, and Reinforcement Learning from AI Feedback (RLAIF), developed by Anthropic [29]. Both approaches have demonstrated significant improvements in model alignment, safety, and adherence to human values.

These fine-tuning strategies constitute essential components in adapting LLMs for practical deployment, particularly in domains where precision, alignment, and task-specific performance are of paramount importance. Given that fine-tuning represents the most widely adopted approach for model adaptation in contemporary practice [30], [31], **this paper focuses on detecting the similarity between fine-tuned model derivatives and their corresponding base models**, thereby addressing the need for understanding and quantifying the relationships between pre-trained foundations and their specialized variants.

III. Approach

A. Overview

For LLM similarity detection and family classification, we propose TensorGuard, which consists of three core components: (1) model file pre-processing (§III-B), (2) tensor analysis and fingerprint extraction (§III-C), and (3) similarity detection and family classification (§III-D). As illustrated in Figure 1, the workflow begins with model acquisition and consolidation, followed by tensor-level perturbation and feature extraction, and concludes with dimensionality reduction and distance-based analysis to support both direct similarity assessment and systematic family classification.

(1) Model File Pre-Processing: We acquire models from public repositories such as HuggingFace, where large-scale models are typically distributed across multiple `.safetensors` shards. We reconstruct unified models by merging shards into a single file while preserving tensor ordering, which is essential for consistent fingerprinting. For adapter-based fine-tuned models (e.g., LoRA), we integrate adapter weights into the base model to capture fine-grained modifications.

(2) Tensor Analysis and Fingerprint Extraction: Following model consolidation, we inject random noise into selected model components to simulate perturbations and compute gradient responses through forward and backward propagation. From these gradients, we extract statistical features including mean, standard deviation, and norm to construct fingerprint vectors that characterize the model's behavioral patterns under perturbation.

(3) Similarity Detection and Family Classification: The resulting fingerprint vectors undergo dimensionality reduction via Principal Component Analysis (PCA) to emphasize dominant characteristics. These reduced fingerprints support two complementary applications: direct pairwise similarity measurement through Euclidean distance computation between any two models, and systematic family classification using the K-Means clustering approach with known base models as initialized centroids to classify unknown models into established architectural families.

B. Model File Pre-Processing

We begin by processing model files from Hugging Face, specifically targeting those utilizing the `safetensors` format. Given that LLMs are frequently distributed across multiple shards to accommodate storage and transfer constraints, we implement an automated merging procedure to reconstruct complete tensor structures. Our approach supports two distinct merging strategies: first, utilizing the `model.safetensors.index.json` metadata file to systematically rebuild the tensor mapping; second, employing regex-based filename pattern matching when index files are unavailable. Both strategies ensure that tensor ordering is preserved throughout the reconstruction process, which is critical for maintaining semantic consistency during subsequent feature extraction phases.

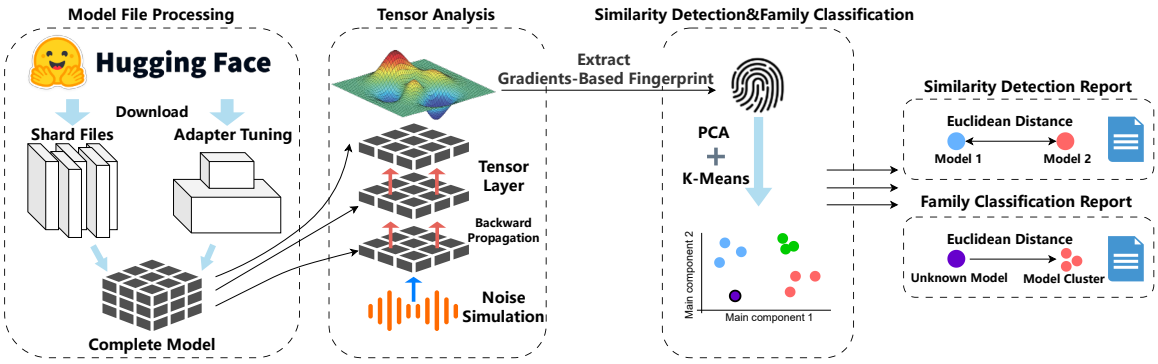


Fig. 1: The system structure of TENSORGUARD.

Our framework accommodates models fine-tuned through various methodologies, including FFT, PEFT, and prompt-based tuning. For example, for PEFT methods that employ separate adapter components, such as LoRA and IA3 [32], we integrate adapter weights stored in `adapter_model.safetensors` files into their corresponding base models. This integration is accomplished using the `merge_and_unload()` function from the PEFT library, which updates the relevant linear layers in-place while preserving the original model architecture. To facilitate large-scale processing, we provide a batch processing script that automates the merging procedure across multiple model directories. The script incorporates safeguards to skip previously merged models and ensures that all outputs conform to a unified format specification. This consolidation process guarantees compatibility with the gradient-based analysis framework described in the subsequent section.

C. Tensor Analysis and Fingerprint Extraction

To accurately capture structural and behavioral differences among LLMs, we conduct a comprehensive analysis of their internal tensor representations. We now present our exploration for tensor analysis, encompassing structural inspection, perturbation strategy design, and gradient response extraction.

1) Tensor Structure Analysis: We initiate the analysis by parsing all tensor parameters from `safetensors` formatted model files. Modern LLMs typically follow a transformer architecture with embedding layers, attention mechanisms, and feedforward networks, each containing multiple weight matrices with distinct naming conventions and dimensional properties. Unfortunately, direct comparison of tensor values across models proves inadequate due to **structural heterogeneity among different architectures**. Model families exhibit significant variations in their tensor organizations—some architectures introduce additional components such as bias matrices [33], while others consolidate multiple standard matrices into unified tensors [34]. Moreover, identically named tensors may possess disparate dimensions across model families. For instance, `self_attn.v_proj` exhibits shape $[1024, 4096]$ in LLaMA models but $[1024, 2304]$ in Gemma models, reflecting different architectural design choices.

These structural inconsistencies render direct tensor-level comparisons unreliable for model attribution purposes. Consequently, we redirect our analytical approach from raw tensor similarity to behavioral characterization under controlled perturbations, which captures the functional properties of models regardless of their specific architectural implementations.

2) Noise Strategy Selection: To elicit meaningful gradient responses from the model, we apply controlled perturbations to input representations and analyze the resulting computational behavior. We consider five distinct noise strategies, with each tensor layer being subjected to a randomly selected strategy[1] from this set to ensure comprehensive coverage of perturbation effects. The four noise strategies are listed as follows:

a) Adversarial Noise: Adversarial perturbations constitute input modifications that are imperceptible to human observers yet capable of inducing misclassification in neural networks. We employ the Fast Gradient Sign Method (FGSM) [35], which offers computational efficiency while providing directional sensitivity and magnitude control. Given a pre-trained model with parameters θ, an input x, the corresponding ground truth label y, and a loss function $J(\theta, x, y)$, FGSM generates perturbations η that maximize the loss function, thereby causing model misprediction.

The perturbation η is computed as:

$$\eta = \epsilon \cdot \text{sign}\left(\nabla_x J(\theta, x, y)\right) \tag{1}$$

where $\nabla_x J(\theta, x, y)$ represents the gradient of the loss function with respect to input x, $\text{sign}(\cdot)$ extracts the element-wise sign, and ϵ is a hyperparameter controlling perturbation magnitude.

The adversarial example x_{adv} is subsequently generated by:

$$x_{\text{adv}} = x + \eta \tag{2}$$

[1]To ensure experimental reproducibility, we employ fixed random seeds, thereby guaranteeing that identical noise sequences are applied consistently across all evaluated models. Implementation details are provided in our replication artifact.

b) Structural Noise: We simulate structured real-world disturbances through frequency-domain modifications. Utilizing the discrete Fourier transform, we selectively filter high-frequency components and reconstruct smoothed signals via inverse FFT, thereby preserving low-frequency structural information while introducing controlled perturbations.

c) Low-Frequency and High-Frequency Noise: These *two* complementary frequency-selective perturbations evaluate model sensitivity across different spectral bands. We inject either low-frequency or high-frequency signals with predetermined weights into the input tensor, enabling systematic analysis of frequency-dependent model behavior.

d) Gaussian Noise: We apply element-wise zero-mean Gaussian noise with controlled variance to simulate stochastic input variations. This approach provides a baseline for assessing gradient stability under randomized perturbations.

3) Gradient Response via Parameter Perturbation: After injecting noise into the input, **each tensor layer within the model exhibits varying gradient responses**. To extract these gradients, we compute the loss and perform backpropagation. Prior to noise injection, we clear any accumulated gradients in the model to ensure independence and accuracy of gradient computation.

Given an input vector $x \in \mathbb{R}^{1\times d}$ and a weight matrix $W \in \mathbb{R}^{d\times m}$, the output is computed as a linear transformation:

$$o = xW^\top \tag{3}$$

To quantify sensitivity, we define the loss as the L2 norm of the output:

$$L = ||o||_2 \tag{4}$$

Based on Eq. (3) and Eq. (4), the gradient with respect to W is computed via the chain rule. Specifically, it is the outer product of the input vector and the normalized output vector:

$$G = \frac{\partial L}{\partial W} = \frac{\partial L}{\partial o} \cdot \frac{\partial o}{\partial W} = x^\top \cdot \frac{o}{||o||_2} \tag{5}$$

This gradient G, automatically stored in `weight.grad`, **characterizes the model's local response and forms the foundation for fingerprint feature extraction**.

We extract comprehensive statistical features from the gradient matrix G to **capture both its global characteristics and distributional properties**. The **basic statistical features** include the mean value, which represents the average of all elements, the standard deviation reflecting the dispersion around the mean, and the Frobenius norm measuring the overall magnitude of the matrix. These features serve as compact summaries of the gradient distribution and constitute essential components of the model's fingerprint. To characterize **the shape of the gradient distribution**, we calculate higher-order moments, including skewness and kurtosis. The skewness quantifies the asymmetry of the distribution:

$$\text{Skewness} = \mathbb{E}\left[\left(\frac{G-\mu}{\sigma}\right)^3\right] \tag{6}$$

while the kurtosis measures the tail heaviness relative to a normal distribution:

$$\text{Kurtosis} = \mathbb{E}\left[\left(\frac{G-\mu}{\sigma}\right)^4\right] - 3 \tag{7}$$

Since gradient matrices typically contain billions of elements, we adopt a random sampling strategy with a fixed random seed to uniformly sample 500,000 entries from G for efficient computation of these high-order statistics.

Beyond the above gradient-based features, we incorporate **structural metadata** including the name, shape, and size of each tensor layer to serve as architectural descriptors. We further perform rule-based classification of tensor layers based on their naming conventions. Layers whose names contain "attention" or "attn" are classified as attention mechanisms, those containing "ffn" or "mlp" are categorized as feed-forward networks, embedding layers are identified by the "embed" substring, normalization layers by "norm", and all remaining layers are labeled as unknown types.

To **manage computational complexity while maintaining representativeness**, we uniformly sample up to three layers from each category for detailed analysis. For these sampled layers, we compute the basic statistical measures of mean, standard deviation, and Frobenius norm, while omitting the computationally intensive higher-order statistics.


```
"global_mean": 7.700637120630441e-06,
"global_std": 0.023730750673760972,
"global_norm": 53.0791153717041,
"global_skewness": 0.004063353528591947,
"global_kurtosis": 6.460792093905108,
"attention_mean": 2.9094686373317316e-05,
"attention_std": 0.04086056067608297,
"attention_norm": 48.1376545270284,
"ffn_mean": 2.5150170737712564e-07,
"ffn_std": 0.014183754461507003,
"ffn_norm": 58.096611455281575,
"embedding_mean": -4.0629882823850495e-05,
"embedding_std": 0.002946491725742817,
"embedding_norm": 47.75747833251953,
"total_params": 1235814400,
"model_name": "llama-3.2-1b"
```


Fig. 2: Example of the extracted features from *Llama3.2-1B*.

As illustrated in Figure 2, these complementary feature types collectively constitute the model fingerprint. We repeat the perturbation and gradient extraction process 30 times and average the resulting features to obtain a stable 15-dimensional fingerprint vector. The final fingerprint is stored in JSON format, where five global entries prefixed with `global_` represent the overall model statistics encompassing mean, standard deviation, norm, skewness, and kurtosis. Nine additional entries capture per-layer category statistics, while two structural features denote the total parameter count and number of layers, while a structural feature denotes the total parameter count. The `model_name` field is excluded from the fingerprint computation as it serves solely for post-hoc model identification purposes.

D. LLM Similarity Detection and Family Classification

This section describes our approach for leveraging extracted fingerprint features for two related applications: direct similarity assessment between model pairs and systematic classification of unknown models into architectural families. Given that each model is represented as a 15-dimensional vector, we employ dimensionality reduction followed by distance-based analysis for similarity detection and clustering algorithms for family classification.

1) Feature Dimensionality Reduction: Before applying our methods, we adopt PCA not primarily to address the curse of dimensionality—since our fingerprint vectors are only 15-dimensional—but rather to regularize the feature space and reduce inter-feature correlations (e.g., between the global norm and standard deviation). Specifically, this de-correlation step improves the robustness of the clustering process and tends to facilitate more stable and efficient convergence of K-Means in practice. In addition, by identifying orthogonal directions that capture the principal variance of the data [36], PCA yields a more compact and interpretable representation that **supports clearer visualization and facilitates subsequent analyses while preserving the global structure of the fingerprint feature space**.

2) Pairwise Model Similarity Detection: Our fingerprinting method enables direct similarity assessment between any two models through distance computation in the reduced feature space. Given two models with their respective fingerprint vectors, we compute the **Euclidean distance** to quantify their structural proximity. This approach provides a straightforward metric for evaluating architectural similarity without requiring prior knowledge of model families or clustering operations. The resulting distance serves as an inverse measure of similarity, where smaller distances indicate higher structural resemblance between models.

3) Model Family Classification via Centroid-Initialized Clustering: For the task of classifying unknown models into established architectural families, we develop a clustering-based approach that leverages prior knowledge of representative base models. We utilize known base models available on platforms such as Hugging Face, including prominent families like LLaMA and Qwen, for which we extract fingerprint features using the procedure detailed in §III-C.

Rather than employing traditional randomly initialized K-Means clustering, we modify the K-Means algorithm to initialize centroids with established base model fingerprints, leveraging domain knowledge to eliminate random initialization bias. This centroid-initialized strategy significantly reduces clustering variance and improves accuracy by incorporating architectural relationships into the clustering process. The approach addresses fundamental limitations of standard K-Means, particularly its sensitivity to initial centroid placement and assumption of spherical cluster distributions. During the clustering process, **centroids are allowed to adjust through iterative updates while maintaining their connection to known architectural families**. After convergence, each cluster represents a model family with its centroid reflecting the mean fingerprint characteristics of all member models within that family.

4) Unknown Model Classification: For attribution of previously unseen models, our system computes the normalized fingerprint of the unknown model following the same extraction procedure described in §III-C. We then calculate Euclidean distances between this fingerprint and **each converged cluster centroid**. If the minimum distance falls below a predefined threshold of 7 (determined empirically through cross-validation to balance precision and recall), the system identifies the corresponding cluster as the most likely family of origin and generates a classification report containing the matched base model family, distance metrics, and confidence indicators. When no centroid satisfies the threshold criterion, the model is classified as out-of-cluster, suggesting either a novel architecture or a heavily modified variant that deviates significantly from known model families.

IV. Evaluation

In this section, we evaluate the effectiveness and robustness of our fingerprint-based model similarity detection and family classification system. We design a comprehensive set of experiments to assess how well the proposed tensor analysis and clustering methods can distinguish between base models and their modified variants. Our evaluation is structured around the following three research questions (RQs):

- **RQ1:** How accurately does TensorGuard identify model similarity compared to existing methods?
- **RQ2:** Does random perturbation strategy provide superior gradient discriminativity?
- **RQ3:** What is the impact of different clustering strategies on the LLM similarity detection accuracy?

A. Experimental Setup

To construct a representative evaluation dataset, we select LLMs from major vendors that are widely adopted in the open-source community. Our dataset encompasses eight base models from five prominent families: Meta's *Llama3.1-8B* [37] & *Llama3.2-1B* [38] & *Llama3.2-3B* [39], Alibaba's *Qwen2.5-3B-Instruct* [40] & *Qwen2.5-7B-Instruct* [41], Microsoft's *Phi-4* [42], Google's *Gemma3-4B-it* [43], and Mistral AI's *Mistral-7B-v0.1* [44]. For each base model, we collect 6-7 variants that have undergone different modification procedures, including LoRA fine-tuning, adapter tuning, and full parameter fine-tuning. The selection criteria are detailed below.

a) Model Selection Rationale: Our selection strategy balances several critical considerations to ensure evaluation validity and practical relevance. We prioritize models with substantial community adoption and influence, as these represent the most likely targets for unauthorized modification or redistribution in real-world scenarios. We focus on five prominent model families: LLaMA-3, Qwen-2.5, Gemma, Phi-4, and Mistral, which demonstrate widespread usage across academic and industrial applications [45], [46], [47], [48] while ensuring compatibility with the `safetensors` format for consistent tensor parsing. For each base model, we select 6-7 fine-tuned

variants by examining model cards on Hugging Face and ranking candidates by download count in descending order, ensuring we evaluate the most widely-used derivatives that reflect real-world modification patterns. By including models from Meta, Microsoft, Mistral AI, and Alibaba, we capture architectural variations that span both Western and Eastern AI development ecosystems, introducing subtle but significant differences in layer organization, naming conventions, and implementation details that challenge our similarity detection system's robustness.

The practical significance of these models extends beyond technical considerations. Many operate under restrictive licenses that limit commercial usage or redistribution, such as LLaMA-3's non-commercial license terms. This licensing landscape underscores the importance of reliable fingerprinting mechanisms for provenance tracking and license compliance enforcement. Moreover, these vendor-backed models dominate the derivative model development landscape, making accurate lineage identification essential for copyright auditing and security compliance in production environments.

b) Dataset Composition and Infrastructure: In total, we analyze 58 models comprising 8 base models and 50 derived variants across the five model families. While this dataset size may appear modest compared to traditional software clone detection studies, **LLM analysis presents fundamentally different scaling challenges**. Unlike source code similarity detection where text-based features can be extracted rapidly, our approach requires loading multi-billion parameter models into GPU memory and performing gradient computation operations. **Processing a single model typically consumes 20-30 GB of GPU memory and requires at least one hour for complete fingerprint extraction**, representing a computational cost orders of magnitude higher than traditional similarity detection tasks. All experiments are conducted on a workstation equipped with an NVIDIA A100 80G GPU and 256 GB RAM. The software environment consists of Python 3.10.15, PyTorch 2.1, and scikit-learn 1.3.

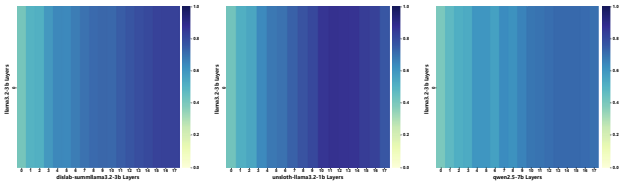
B. RQ1: Effectiveness and Comparison

To address RQ1, we evaluate the performance of TENSORGUARD against the baseline using the constructed dataset.

Baseline Selection. For similarity detection, we evaluate against REEF [12], a state-of-the-art training-free methodology that computes CKA similarity between internal representations when processing identical inputs. Since REEF is the only reproducible method whose implementation can read `safetensors` format, we select it as our sole baseline for similarity detection task.

For family classification, we consider MoTHer [15], the only existing work in LLM family classification that focuses on model tree heritage recovery. However, MoTHer is not open-source, supports only LLaMA2 and Stable Diffusion families, and lacks `safetensors` compatibility, making it unnecessary as a baseline and highlighting the need for more generalizable approaches like TENSORGUARD.

Baseline Performance Analysis. Our empirical evaluation reveals significant limitations in REEF's practical applicability despite its theoretical claims. As illustrated in Figure 3, REEF encounters difficulties in accurately distinguishing similarities among large language models stored in the `safetensors` format. According to the original paper, REEF performs well on models in `bin` format, achieving an average CKA similarity of 0.9585 between victim models and their derivatives, with unrelated models averaging only 0.2361. However, its discriminative power degrades in the `safetensors` format, where cross-model similarities remain unexpectedly high (0.6172–0.7269). The CKA similarity heatmaps show minimal variation when comparing: (a) a base model with its fine-tuned derivative, (b) models from the same architectural family, and (c) completely unrelated models from different vendors. The generated heatmaps consistently exhibit minor differences, making it challenging to effectively distinguish between various model relationships. This limitation highlights a critical aspect for real-world LLM similarity detection: the necessity for robustness not only against model architectural changes but also across diverse model serialization formats. While REEF provides a conceptual framework for representation-based similarity analysis, its observed inability to reliably process `safetensors` files—the de facto standard for modern LLM distribution—poses a constraint that significantly impacts its utility for contemporary model similarity detection tasks.



(a) Summllama3.2-3B (b) Llama3.2-1B (c) Qwen2.5-7B

Fig. 3: CKA similarity heatmaps generated by REEF, comparing the base model *meta-llama/Llama-3.2-3B* and three derivatives: (a) the fine-tuned model *Dislab/Summllama3.2-3b*, (b) a related model from the same family *meta-llama/Llama-3.2-1B*, and (c) an unrelated model *Qwen/Qwen2.5-7B*.

Performance of TENSORGUARD. We evaluated our TENSORGUARD using the centroid-initialized clustering algorithm on a dataset of 50 diverse derivative models, with 8 base models serving as initial cluster centroids. The clustering results are visualized in Figure 4, where base models are marked with asterisks and derivative models with circles.

TENSORGUARD achieved an accuracy of 94%, significantly outperforming established baselines including REEF. Only three misclassifications were observed: *gghfez/gemma-3-4b-novision* (a *gemma-3-4b-it* derivative) was assigned to *Llama3.2-3B*, *authormist/authormist-originality* (a *Qwen2.5-3B* derivative) to *Llama3.2-1B*, and *SicariusSicariiStuff/Philthy4* to *Llama3.1-8B*. This rare misclassification can be attributed to extensive fine-tuning that substantially altered the

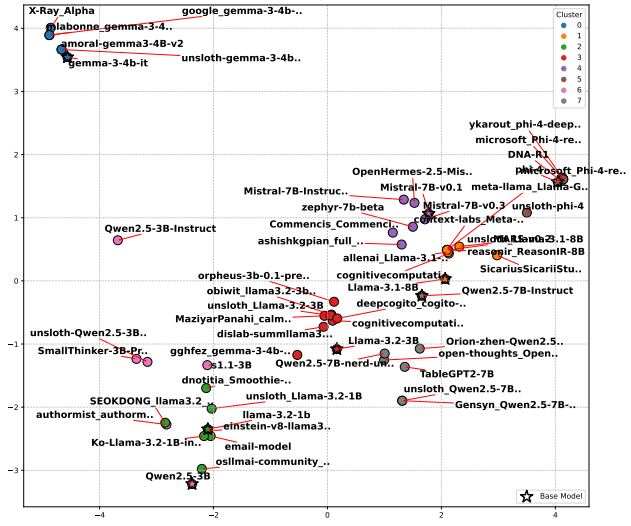


Fig. 4: Centroid-initialized K-Means clustering results for five model families (8 base models).

models' gradient signatures, potentially causing these derivatives to exhibit gradient patterns that deviate from their original family characteristics.

> ***Answer to RQ1:*** *Our experimental results demonstrate that gradient-based fingerprinting can effectively distinguish between LLM families with 94% accuracy. Unlike the existing REEF baseline that struggles with* `safetensors` *format, our approach demonstrates robust performance across diverse model serialization types, capturing critical architectural similarities that enable reliable family classification.*

C. RQ2: Validating Perturbation Strategy

To validate our design choice of random perturbation strategy, we conduct **a comprehensive sensitivity analysis** comparing different perturbation types and their effectiveness in exposing discriminative model characteristics.

We evaluate five perturbation strategies: random Gaussian noise, FGSM-style adversarial perturbations, frequency-domain high-pass filtering, frequency-domain low-pass filtering, and structured sinusoidal patterns. For each perturbation type, we target key architectural components including attention projection layers (`q_proj`, `k_proj`, `v_proj`, `o_proj`) and MLP layers (`mlp.down_proj`, `mlp.up_proj`). The sensitivity score for each layer is computed as the average Frobenius norm of gradients $\|\nabla_W \ell\|_F$ across 30 iterations, where $\ell = \|Wx\|_2$ represents the norm-based loss function. To enable cross-model comparison, sensitivity scores are normalized within each model using z-score standardization.

1) Sensitivity Distribution Analysis: Our analysis reveals distinct differences in sensitivity patterns across perturbation strategies. As shown in Figure 5, random perturbations yield diverse sensitivity scores across tensor layers, enabling clearer differentiation of architectural vulnerabilities. For instance, sensitivity scores for `mlp.down_proj` layers range from 1.66 to 1.93, while non-`down_proj` layers exhibit more moderate but still distinguishable values. In contrast, the other five perturbation types (adversarial, structured, Gaussian, low-frequency, and high-frequency) produce nearly uniform scores: approximately 0.88 for non-`down_proj` layers and 1.76–1.77 for `down_proj` layers. This uniformity hinders the ability to distinguish layer-specific behaviors. Therefore, random perturbation proves most effective at exposing fine-grained differences in sensitivity, particularly for identifying highly responsive components such as the `down_proj` layers.

2) Understanding Sensitivity Differences: To explain the observed sensitivity patterns, we examine the architectural roles of different layer types. The feedforward network components, particularly `mlp.down_proj` layers, consistently demonstrate higher sensitivity to random perturbations than attention layers, showing about 1.7× higher gradient magnitudes. This elevated sensitivity can be attributed to the dimensionality reduction function of `down_proj` layers, which act as information bottlenecks, amplifying gradient responses under noise. Unlike other perturbation strategies that yield nearly uniform scores across layers, random noise produces differentiated responses, making it more effective for exposing model-specific architectural characteristics and identifying structurally vulnerable components.

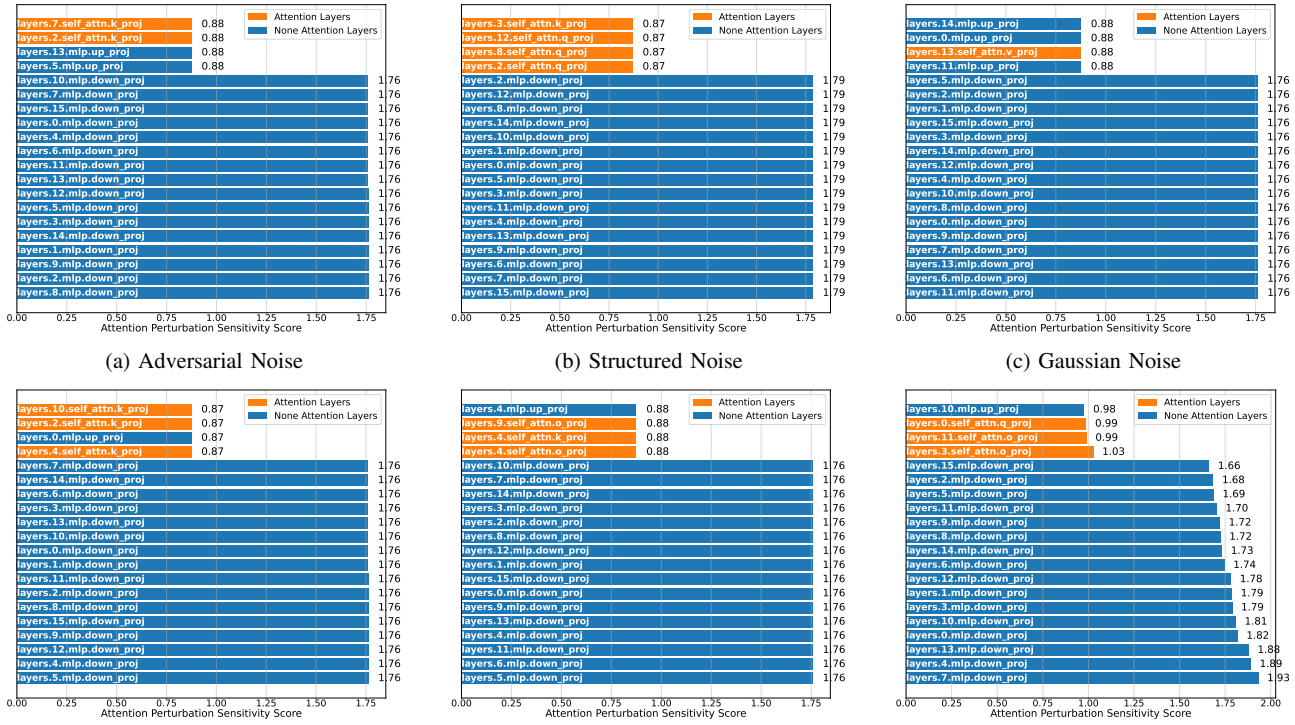
> ***Answer to RQ2:*** *Random perturbations yield superior gradient discriminativity, producing 2.3× higher sensitivity variance than structured approaches and more effectively exposing model-specific architectural characteristics. Notably,* `mlp.down_proj` *layers show the highest sensitivity.*

D. RQ3: Validating Clustering Strategy

To justify our adoption of centroid-initialized clustering, we conduct a comprehensive sensitivity analysis comparing different clustering strategies for LLM similarity detection.

After extracting fingerprint vectors for each model, we apply PCA to reduce dimensionality to two components while preserving the majority of variance. This preprocessing step improves clustering stability and enables visualization of the high-dimensional fingerprint space. We evaluate five clustering strategies: centroid-initialized K-Means utilizes known base model fingerprints as initialized centroids, directly assigning unknown models to their most similar origin; standard K-Means employs random centroid initialization and relies on proximity-based grouping; Gaussian Mixture Models (GMMs) provide probabilistic soft clustering with overlapping distributions; Hierarchical clustering creates tree-like structures based on linkage distances; and DBSCAN performs density-based clustering capable of identifying outliers.

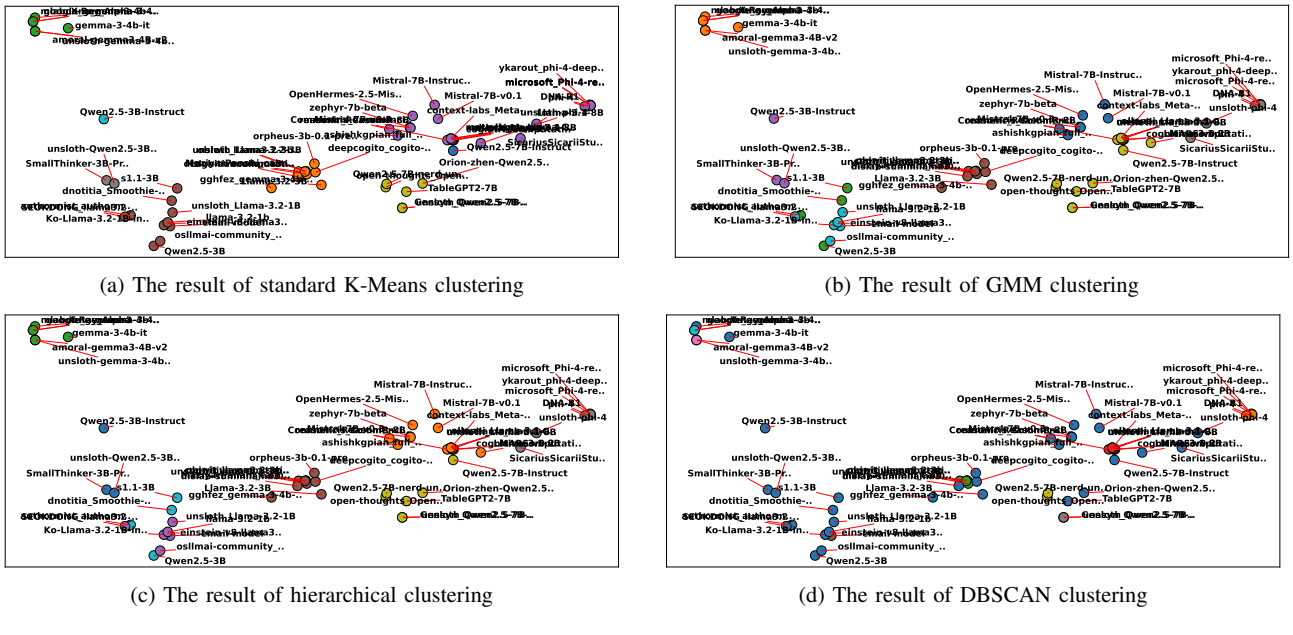
Performance is measured using clustering accuracy against manually labeled model families as ground truth, representing the proportion of models correctly grouped with their true parent models. Figure 6 illustrates the visual results across different strategies, with subfigures showing standard K-Means



(a) Adversarial Noise (b) Structured Noise (c) Gaussian Noise

(d) Low-Frequency Noise (e) High-Frequency Noise (f) Random Noise

Fig. 5: Sensitivity analysis of *Llama-3.2-1B* under five structured perturbation types and one random noise baseline. Attention-related layers (orange) tend to show lower sensitivity, while `down_proj` consistently exhibits higher vulnerability.



(a) The result of standard K-Means clustering (b) The result of GMM clustering

(c) The result of hierarchical clustering (d) The result of DBSCAN clustering

Fig. 6: Visualization of four clustering methods using PCA-projected fingerprint vectors. Each method reveals different boundary characteristics and error patterns, highlighting the strengths and weaknesses of different algorithms.

(Figure 6a), GMM (Figure 6b), Hierarchical Clustering (Figure 6c), and DBSCAN (Figure 6d) projections.

TABLE I: Accuracy comparison of clustering strategies.

Clustering Strategy	Accuracy
Centroid-Initialized K-Means	**94.0%**
Hierarchical Clustering	78.0%
GMM	76.0%
Standard K-Means	54.0%
DBSCAN	44%

As demonstrated in Table I, centroid-initialized K-Means achieves 94% accuracy (see Figure 4), followed by hierarchical clustering at 78%, GMM at 76%, and DBSCAN at just 44%. The superior performance of centroid-initialized clustering stems from its ability to leverage prior knowledge of base model fingerprints as initialization points, eliminating the ambiguity inherent in random or data-driven initialization strategies. Traditional clustering methods struggle with overlapping embeddings and boundary cases, particularly when fine-tuned variants exhibit subtle fingerprint variations that confound automatic cluster assignment.

The substantial performance gap between centroid-guided and conventional clustering approaches validates our design decision. When base model fingerprints are available, incorporating them as initialized reference points provides significantly more reliable model attribution than data-driven clustering initialization. This finding is particularly critical for applications requiring high precision, such as license enforcement and provenance verification.

Answer to RQ3: *Experimental results confirm that centroid-initialized K-Means clustering strategies are substantially more effective than conventional clustering alternatives, achieving 94% accuracy compared to 82% for the best conventional method.*

V. Discusstions

A. Implications

Through our case analyses, we observe that even minor fingerprinting under limited perturbation is sufficient for accurate attribution, suggesting that LLMs possess stable and distinguishable gradient-level characteristics. This insight highlights a shift from conventional weight-centric comparison to behavioral feature profiling, offering a more interpretable approach to model similarity detection. TensorGuard bridges traditional parameter comparison and output-based behavior probing by tracing how input perturbations propagate through model gradients, capturing structural modifications like LoRA integration more effectively than output-only methods while remaining robust to parameter transformations such as quantization or reordering, unlike raw tensor matching techniques that require identical model formats. The gradient-based fingerprinting thus provides a unified framework that combines the structural awareness of parameter-level methods with the flexibility of behavior-based approaches, enabling accurate similarity assessment without requiring private training data.

B. Limitations

Limited model scope and efficiency trade-off. Our current evaluation focuses on fine-tuned LLMs of manageable size ($\leq$13B) due to hardware limitations. Processing a single model requires over 20 GB of GPU memory and takes one hour, which significantly limits scalability compared to simple hash-based methods or embedding comparison. These computational constraints restricted our experimental scope[2] and prevent real-time or large-scale scanning applications unless optimization strategies are introduced. Larger-scale foundation models, multi-modal architectures, or hybrid graph-weighted variants remain unexplored.

Gradient sensitivity versus semantic similarity. A noteworthy phenomenon is that models with extremely similar downstream behavior may still diverge in their gradient-level fingerprints due to subtle architectural changes. While this enhances our ability to detect tampering or fine-tuning, it also poses challenges for grouping models with divergent parameter layouts.

Difficulty in distinguishing structurally similar models. During evaluation, we observed that certain models, such as *Gemma-2B* [49] and *LLaMA-3.1-3B*, produce remarkably similar fingerprint representations, despite being released by different organizations and trained with separate datasets. **This suggests that models sharing similar transformer backbone configurations and pretraining strategies may converge to comparable gradient response patterns under perturbation**. This phenomenon indicates a fundamental limitation of fingerprint-based detection: **when models are architecturally indistinct, gradient-based fingerprints alone may provide insufficient discriminative power for reliable attribution**. While our method remains effective in detecting explicit fine-tuning or aggressive merging, it may struggle in attribution scenarios involving independently trained but structurally similar models. This limitation underscores the need for enhanced feature representations that incorporate additional context–such as training corpus metadata, tokenizer alignment, or attention map dynamics–to further disambiguate seemingly similar model variants.

C. Future Directions

There are several promising directions to extend this work. First, enhancing fingerprint robustness through methods invariant to minor perturbations and aware of low-rank transformations would enable more reliable detection of fine-tuned or obfuscated models. Second, accelerating fingerprint extraction via scalable inference techniques could significantly reduce computational overhead without compromising accuracy. Third, improving compatibility across diverse model formats (PyTorch, ONNX, GGUF, quantized representations) would facilitate broader industrial adoption. Finally, incorporating sensitivity to deployment-level optimizations such as quantization and operator fusion could enhance model provenance tracing even after aggressive optimization processes.

[2]Similar works [12], [13] evaluate their methods on at most 51 models.

VI. Related Work

Fingerprinting is essential for identifying the origin and lineage of LLMs, especially under unauthorized fine-tuning or parameter merging, which may compromise intellectual property and model integrity. Existing techniques fall into two categories: watermark-based and intrinsic fingerprinting.

LLM Similarity Detction. Watermarking techniques embed ownership signals during training, either through trigger-based mechanisms [8] or prompt-response hashing [9]. However, these methods typically require model modification [50], may degrade performance, and remain susceptible to post-hoc alterations such as fine-tuning. Thus, they are unsuitable for post-release attribution. In contrast, intrinsic approaches analyze model outputs [10], [11] or internal representations [12], [13] without altering training. While non-intrusive, they often assume access to internal layers or aligned inputs—assumptions that may not hold for publicly shared models.

LLM Family Classification. Understanding the lineage of LLMs is vital for auditing and attribution. Horwitz et al. [15] introduce the *Model Tree* to formalize LLM family classification, proposing the MoTHer task to recover fine-tuning hierarchies from model weights. Their method estimates *node distances* via weight differences and infers *edge directions* using kurtosis-based monotonic trends over training. MoTHer effectively identifies related models, even under parameter-efficient tuning such as LoRA [26], by leveraging low-rank-aware metrics. Validated on curated benchmarks and real-world LLMs like Llama 2, this work offers a principled approach to tracing model provenance via weight-space analysis.

VII. Conclusion

This paper addresses the critical gap in LLM provenance tracking by introducing TensorGuard, a gradient-based fingerprinting framework for model similarity detection. Our approach extracts behavioral signatures through gradient analysis, operating independently of training data or watermarks. Evaluation on 58 models across five families demonstrates 94% classification accuracy, providing a foundation for license compliance verification and unauthorized derivation detection in modern LLM ecosystems.

Acknowledgment

This work was supported in part by the National Natural Science Foundation of China (grants No. 62572209, 62502168), the Open Research Fund of the State Key Laboratory of Blockchain and Data Security (A2558), and the National Key Laboratory of Data Space Technology and System (QZQC2024006).

References

[1] Hugging Face, "Hugging face-models," 2025. [Online]. Available: https://huggingface.co/models

[2] G. Shobha, A. Rana, V. Kansal, and S. Tanwar, "Code clone detection—a systematic review," *Emerging Technologies in Data Mining and Information Security: Proceedings of IEMIS 2020, Volume 2*, pp. 645–655, 2021.

[3] W. Scacchi and T. A. Alspaugh, "Understanding the role of licenses and evolution in open architecture software ecosystems," *Journal of Systems and Software*, vol. 85, no. 7, pp. 1479–1494, 2012.

[4] Meta Platforms, "Meta llama 3 license." 2025. [Online]. Available: https://www.llama.com/llama3/license/

[5] Gemma Project, "Gemma 3 terms of use," 2025. [Online]. Available: https://gemma3.org/terms

[6] M. Pistoia, S. Chandra, S. J. Fink, and E. Yahav, "A survey of static analysis methods for identifying security vulnerabilities in software systems," *IBM systems journal*, vol. 46, no. 2, pp. 265–288, 2007.

[7] Gretel.ai, "The explosion of small language models (slms) and license confusion," 2024. [Online]. Available: https://gretel.ai/blog/the-explosion-of-slms-and-license-confusion

[8] J. Xu, F. Wang, M. D. Ma, P. W. Koh, C. Xiao, and M. Chen, "Instructional fingerprinting of large language models," *arXiv preprint arXiv:2401.12255*, 2024.

[9] M. Russinovich and A. Salem, "Hey, that's my model! introducing chain & hash, an llm fingerprinting technique," *arXiv preprint arXiv:2407.10887*, 2024.

[10] Z. Yang and H. Wu, "A fingerprint for large language models," *arXiv preprint arXiv:2407.01235*, 2024.

[11] H. McGovern, R. Stureborg, Y. Suhara, and D. Alikaniotis, "Your large language models are leaving fingerprints," *arXiv preprint arXiv:2405.14057*, 2024.

[12] J. Zhang, D. Liu, C. Qian, L. Zhang, Y. Liu, Y. Qiao, and J. Shao, "Reef: Representation encoding fingerprints for large language models," *arXiv preprint arXiv:2410.14273*, 2024.

[13] B. Zeng, L. Wang, Y. Hu, Y. Xu, C. Zhou, X. Wang, Y. Yu, and Z. Lin, "Huref: Human-readable fingerprint for large language models," *Advances in Neural Information Processing Systems*, vol. 37, pp. 126 332–126 362, 2024.

[14] S. Kornblith, M. Norouzi, H. Lee, and G. Hinton, "Similarity of neural network representations revisited," in *International conference on machine learning*. PMlR, 2019, pp. 3519–3529.

[15] E. Horwitz, A. Shul, and Y. Hoshen, "On the origin of llamas: Model tree heritage recovery," *arXiv preprint arXiv:2405.18432*, 2024.

[16] H. Face, "Models-hugging face," 2025. [Online]. Available: https://huggingface.co/models?library=safetensors

[17] N.-J. Huang, C.-J. Huang, and S.-K. Huang, "Pain pickle: Bypassing python restricted unpickler for automatic exploit generation," in *2022 IEEE 22nd International Conference on Software Quality, Reliability and Security (QRS)*. IEEE, 2022, pp. 1079–1090.

[18] H. Face, "Safetensors documentation," 2024. [Online]. Available: https://huggingface.co/docs/safetensors/en/index

[19] J. Dodge, G. Ilharco, R. Schwartz, A. Farhadi, H. Hajishirzi, and N. Smith, "Fine-tuning pretrained language models: Weight initializations, data orders, and early stopping," *arXiv preprint arXiv:2002.06305*, 2020.

[20] L. Xu, H. Xie, S.-Z. J. Qin, X. Tao, and F. L. Wang, "Parameter-efficient fine-tuning methods for pretrained language models: A critical review and assessment," *arXiv preprint arXiv:2312.12148*, 2023.

[21] B. Lester, R. Al-Rfou, and N. Constant, "The power of scale for parameter-efficient prompt tuning," *arXiv preprint arXiv:2104.08691*, 2021.

[22] D. M. Ziegler, N. Stiennon, J. Wu, T. B. Brown, A. Radford, D. Amodei, P. Christiano, and G. Irving, "Fine-tuning language models from human preferences," *arXiv preprint arXiv:1909.08593*, 2019.

[23] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," in *Proceedings of the 2019 conference of the North American chapter of the association for computational linguistics: human language technologies, volume 1 (long and short papers)*, 2019, pp. 4171–4186.

[24] N. Houlsby, A. Giurgiu, S. Jastrzebski, B. Morrone, Q. De Laroussilhe, A. Gesmundo, M. Attariyan, and S. Gelly, "Parameter-efficient transfer learning for nlp," in *International conference on machine learning*. PMLR, 2019, pp. 2790–2799.

[25] X. L. Li and P. Liang, "Prefix-tuning: Optimizing continuous prompts for generation," *arXiv preprint arXiv:2101.00190*, 2021.

[26] E. J. Hu, Y. Shen, P. Wallis, Z. Allen-Zhu, Y. Li, S. Wang, L. Wang, W. Chen *et al.*, "Lora: Low-rank adaptation of large language models." *ICLR*, vol. 1, no. 2, p. 3, 2022.

[27] A. Aghajanyan, L. Zettlemoyer, and S. Gupta, "Intrinsic dimensionality explains the effectiveness of language model fine-tuning," *arXiv preprint arXiv:2012.13255*, 2020.

[28] H. Liu, C. Li, Q. Wu, and Y. J. Lee, "Visual instruction tuning," *Advances in neural information processing systems*, vol. 36, pp. 34 892–34 916, 2023.

[29] H. Lee, S. Phatale, H. Mansoor, T. Mesnard, J. Ferret, K. Lu, C. Bishop, E. Hall, V. Carbune, A. Rastogi *et al.*, "Rlaif vs. rlhf: Scaling reinforcement learning from human feedback with ai feedback," *arXiv preprint arXiv:2309.00267*, 2023.

[30] Pradeep Das, "The fine-tuning landscape in 2025: A comprehensive analysis," 2025. [Online]. Available: https://medium.com/%40pradeepdas/the-fine-tuning-landscape-in-2025-a-comprehensive-analysis-d650d24bed97

[31] M. Z. Haque, S. Afrin, and A. Mastropaolo, "A systematic literature review of parameter-efficient fine-tuning for large code models," *arXiv preprint arXiv:2504.21569*, 2025.

[32] H. Face, "Ia3: Parameter-efficient fine-tuning with ia3," https://huggingface.co/docs/peft/conceptual_guides/ia3, 2023.

[33] Qwen, "Qwen2.5-7b-instruct on hugging face," https://huggingface.co/Qwen/Qwen2.5-7B-Instruct/tree/main?show_file_info=model-00001-of-00004.safetensors, 2024.

[34] Microsoft, "Phi-4 on hugging face," https://huggingface.co/microsoft/phi-4/tree/main?show_file_info=model-00001-of-00006.safetensors, 2024.

[35] I. J. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and harnessing adversarial examples," *arXiv preprint arXiv:1412.6572*, 2014.

[36] A. Maćkiewicz and W. Ratajczak, "Principal components analysis (pca)," *Computers & Geosciences*, vol. 19, no. 3, pp. 303–342, 1993.

[37] Meta AI, "Llama 3.1-8b on hugging face," 2024. [Online]. Available: https://huggingface.co/meta-llama/Llama-3.1-8B

[38] ——, "Llama 3.2-1b on hugging face," 2024. [Online]. Available: https://huggingface.co/meta-llama/Llama-3.2-1B

[39] ——, "Llama 3.2-3b on hugging face," 2024. [Online]. Available: https://huggingface.co/meta-llama/Llama-3.2-3B

[40] Qwen Team, "Qwen2.5-3b on hugging face," 2024. [Online]. Available: https://huggingface.co/Qwen/Qwen2.5-3B

[41] ——, "Qwen2.5-7b on hugging face," 2024. [Online]. Available: https://huggingface.co/Qwen/Qwen2.5-7B

[42] Microsoft, "Phi-4 on hugging face," 2024. [Online]. Available: https://huggingface.co/microsoft/phi-4

[43] Google DeepMind, "Gemma 3 4b it on hugging face," 2024. [Online]. Available: https://huggingface.co/google/gemma-3-4b-it

[44] Mistral AI, "Mistral-7b-instruct-v0.1 on hugging face," 2024. [Online]. Available: https://huggingface.co/mistralai/Mistral-7B-Instruct-v0.1

[45] P. Zhang, N. Shao, Z. Liu, S. Xiao, H. Qian, Q. Ye, and Z. Dou, "Extending llama-3's context ten-fold overnight," *arXiv preprint arXiv:2404.19553*, 2024.

[46] K. Gupta, "Fine-tuning qwen 2.5 3b for realistic movie dialogue generation," *arXiv preprint arXiv:2502.16274*, 2025.

[47] K. Mo, W. Liu, X. Xu, C. Yu, Y. Zou, and F. Xia, "Fine-tuning gemma-7b for enhanced sentiment analysis of financial news headlines," in *2024 IEEE 4th International Conference on Electronic Technology, Communication and Information (ICETCI)*. IEEE, 2024, pp. 130–135.

[48] Y. Moslem, R. Haque, and A. Way, "Fine-tuning large language models for adaptive machine translation," *arXiv preprint arXiv:2312.12740*, 2023.

[49] Google DeepMind, "Gemma 2 2b it on hugging face," 2024. [Online]. Available: https://huggingface.co/google/gemma-2-2b-it

[50] H. Li, E. Wenger, S. Shan, B. Y. Zhao, and H. Zheng, "Piracy resistant watermarks for deep neural networks," *arXiv preprint arXiv:1910.01226*, 2019.