TensorLock: Recovering Model Dependency for Model Supply Chain

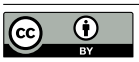
SUSHENG WU*, Fudan University, China
ZIQIAN CHEN*, Fudan University, China
CHENGYUAN LI, Fudan University, China
KAIFENG HUANG✉, Tongji University, China
ZEKAI CHEN, Fudan University, China
YIJIAN WU, Fudan University, China
BIHUAN CHEN✉, Fudan University, China
YIHENG CAO, Fudan University, China
ZHUOTONG ZHOU, Fudan University, China
YIHENG HUANG, Fudan University, China
XIN PENG, Fudan University, China

The public models on the model hosting platforms have undergone exponential growth, allowing developers to build upon existing models rather than training from scratch. These models are continuously reused, modified, and re-distributed similar to traditional software components, breeding a dense and rapidly evolving model supply chain. However, while enjoying the benefits of model reuse, developers also inherit supply chain risks ranging from legal liabilities to security threats. To mitigate these risks, a well-established model dependency graph can significantly benefit supply chain risk governance. Unfortunately, although model hosting platforms offer mechanisms for dependency disclosure, such declarations are optional and frequently missing.

To address this challenge, we propose a novel model dependency recovering framework TensorLock. It works in two phases; i.e., (1) model clustering, and (2) type-aware dependency identification within these clusters. In the first phase, TensorLock performs connectivity-based clustering to accommodate the open-ended dependency topology, grouping models with dependency relations. In the second phase, TensorLock

*Both authors contributed equally to this work.

Authors' Contact Information: Susheng Wu, College of Computer Science and Artificial Intelligence and Shanghai Key Laboratory of Data Science, Fudan University, Shanghai, China, 24110240094@m.fudan.edu.cn; Ziqian Chen, College of Computer Science and Artificial Intelligence and Shanghai Key Laboratory of Data Science, Fudan University, Shanghai, China, 25213050138@m.fudan.edu.cn; Chengyuan Li, College of Computer Science and Artificial Intelligence and Shanghai Key Laboratory of Data Science, Fudan University, Shanghai, China, 22307130304@m.fudan.edu.cn; Kaifeng Huang, School of Software Engineering, Tongji University, Shanghai, China, kaifengh@tongji.edu.cn; Zekai Chen, College of Computer Science and Artificial Intelligence and Shanghai Key Laboratory of Data Science, Fudan University, Shanghai, China, 25213050135@m.fudan.edu.cn; Yijian Wu, College of Computer Science and Artificial Intelligence and Shanghai Key Laboratory of Data Science, Fudan University, Shanghai, China, wuyijian@fudan.edu.cn; Bihuan Chen, College of Computer Science and Artificial Intelligence and Shanghai Key Laboratory of Data Science, Fudan University, Shanghai, China, bhchen@fudan.edu.cn; Yiheng Cao, College of Computer Science and Artificial Intelligence and Shanghai Key Laboratory of Data Science, Fudan University, Shanghai, China, caoyh23@m.fudan.edu.cn; Zhuotong Zhou, College of Computer Science and Artificial Intelligence and Shanghai Key Laboratory of Data Science, Fudan University, Shanghai, China, zhouzt23@m.fudan.edu.cn; Yiheng Huang, College of Computer Science and Artificial Intelligence and Shanghai Key Laboratory of Data Science, Fudan University, Shanghai, China, yihenghuang23@m.fudan.edu.cn; Xin Peng, College of Computer Science and Artificial Intelligence and Shanghai Key Laboratory of Data Science, Fudan University, Shanghai, China, pengxin@fudan.edu.cn.



ACM 2994-970X/2026/10-ARTISSTA028
https://doi.org/10.1145/3832119

employs a divide-and-conquer strategy, leveraging distinct type-specific fingerprints to first identify data-free dependencies (*Quantization* and *Merging*), and then resolve data-driven *Fine-Tuning* dependencies. Our evaluation demonstrates that TensorLock substantially outperforms state-of-the-art approaches, achieving an ARI of 0.96 in clustering and a $DF1$ of 0.82 in dependency identification, improving over the best baselines by at least 39% and 193%, respectively. Additionally, we apply TensorLock to 289 supposedly isolated models and recover 189 previously missing model dependencies, with 42 model authors confirming our findings.

CCS Concepts: • **Computing methodologies** → **Artificial intelligence**; • **Software and its engineering** → *Software maintenance tools*; • **Computer systems organization** → *Neural networks*; • **Security and privacy** → **Systems security**.

Additional Key Words and Phrases: model supply chain, model dependency recovery, model fingerprinting, model provenance

ACM Reference Format:
Susheng Wu, Ziqian Chen, Chengyuan Li, Kaifeng Huang, Zekai Chen, Yijian Wu, Bihuan Chen, Yiheng Cao, Zhuotong Zhou, Yiheng Huang, and Xin Peng. 2026. TensorLock: Recovering Model Dependency for Model Supply Chain. *Proc. ACM Softw. Eng.* 3, ISSTA, Article ISSTA028 (October 2026), 23 pages. https://doi.org/10.1145/3832119

1 Introduction

The public models on the model hosting platforms (e.g., Hugging Face) have undergone exponential growth, allowing developers to build upon existing models rather than training from scratch [4, 24, 27]. These models are then continuously reused, modified, and re-distributed similar to traditional software components, which breeds a dense and rapidly evolving model supply chain [19, 42]. However, while enjoying the benefits of reusing models, the risks are also inherited [8, 12, 19, 42], from legal liabilities (e.g., intellectual property violations) to security threats [15, 23, 24] (e.g., backdoors, jailbreaks, and prompt injection). For example, Jia et al. [23] demonstrate that backdoors embedded in pre-trained encoders can propagate to downstream models, and Greshake et al. [15] show that prompt injection attacks can compromise downstream models.

A well-established model dependency can significantly benefit the model supply chain risk governance. From an upstream perspective, it allows model providers to track model modification and distribution, aiding the detection of unauthorized use and license violations. From a downstream perspective, it enables application developers to trace model provenance and perform timely impact analysis when upstream security threats (e.g., model poisoning [41, 47]) emerge. However, recovering model dependencies is non-trivial. Unlike traditional software components, which contain explicit symbolic references that can be readily traced to their origins (e.g., config files like *package-lock.json*), the models primarily consist of weights and architectural information, providing no inherent indicators of their dependency. Although model hosting platforms offer mechanisms for dependency disclosure (e.g., the base model field in Hugging Face model cards), such declarations are optional and frequently absent. For example, Horwitz et al. [17] show that models on Hugging Face form a directed acyclic graph (DAG), yet only 15.4% of models explicitly declare their dependencies [36], and more than 60% dependent models lack dependency information altogether [18].

Existing Approaches. Model watermarking identifies model dependencies by embedding specific patterns into the weights during the initial training phase. Because these patterns remain unerasable after subsequent modifications, they serve as a unique identifier of the model's origin. However, the watermark can only be verified by the model owner who possesses the secret key, limiting the watermark's applicability in third-party or community-wide dependency analysis.

In contrast, model fingerprinting extracts distinctive features (e.g., parameter statistics or output behaviors) to infer model dependencies, making it a promising non-intrusive approach for identifying model dependency. Based on the identified dependency granularity, existing fingerprinting

approaches can be categorized into pair-wise [3, 6, 7, 16, 25, 30, 32, 34, 35, 39, 45, 48, 50, 52, 53, 56, 57], cluster-wise [43], and tree-wise [18]. Pair-wise approaches, such as Reef [56], rely on a pre-specified base model to compute similarity against other suspicious models using the aforementioned features. A high similarity score with these comparisons suggests a dependency between the base and the suspicious models. Cluster-wise approaches (e.g., TensorGuard [43]) group models into the same cluster based on pair-wise similarity. While clustering is a crucial step for tracking model dependency, it does not distinguish the relative positions or directional relationships among models within a cluster. To this end, tree-wise approaches extend cluster-wise methods by identifying intra-cluster dependencies, organizing them around base models within the cluster. For instance, MoTHer [18] models dependencies as a tree structure rooted at a single source model. Nevertheless, we identify two major challenges in existing approaches.

Challenge 1: Open-Ended Model Dependency Topology. From an ecosystem-wide perspective, base models form an open-ended population rather than a pre-specified set. However, existing cluster-wise approaches perform clustering with a pre-defined number of clusters (i.e., base models), where incorrectly estimated cluster numbers (i.e., underestimating or overestimating the number of base models) can lead to misclassification (i.e., grouping unrelated models together or separating related models apart). Moreover, cluster-wise approaches assume that all models directly derive from the base model, and thus measure similarity in a *flat* manner by comparing each model against the base model. However, in practice, model dependencies often follow a *chain-like* topology where models are iteratively modified and redistributed across multiple hops (i.e., transitive dependencies). In such cases, dissimilarities accumulate with each hop, causing distant derivatives to fall outside the similarity threshold and be excluded from the correct cluster.

Challenge 2: Mixed Dependency Types. Real-world model dependencies form a directed acyclic graph (DAG) with diverse dependency types (e.g., *Fine-Tuning*, *Merging*, *Quantization*), where *Merging* introduces multi-parent dependencies. Unfortunately, the state-of-the-art tree-wise approach MoTHer assumes each model has a single parent, focusing exclusively on *Fine-Tuning* dependencies and employing distance-based and direction-based metrics. Such metrics are inherently tailored to *Fine-Tuning* and fail to capture the distinct weight-space characteristics of *Merging* and *Quantization* dependencies.

Empirical Study and Goal. We begin with an empirical study to understand the dependency prevalence and dependency types of model reuse. Our study reveals that model reuse is a dominant but poorly managed practice, largely centered on transformer-based models and complicated by transitive dependency topology and diverse dependency types. To bridge this gap, our goal is to construct a *type-aware model dependency graph* (MDG), which facilitates effective model supply chain governance by enabling impact analysis and root-cause tracking.

Our Framework. To achieve this goal, we propose TensorLock, a novel model dependency recovering framework that takes a set of candidate transformer-based models as input, identifies and outputs their dependencies as a type-aware MDG. TensorLock operates in two phases: (1) model clustering to handle open-ended dependency topology (addressing Challenge 1), and (2) type-aware dependency identification within these clusters (addressing Challenge 2). *In Phase-1*, TensorLock adopts connectivity-based clustering. Instead of assuming a fixed number of base models, it adaptively determines cluster numbers based on density connectivity. The connectivity is quantified using statistical features (e.g., standard deviation, skewness, kurtosis) derived from the Multi-Head Self-Attention layer weights. This enables TensorLock to accurately determine the number of clusters and effectively trace chain-like dependencies through intermediate neighbors. *In Phase-2*, TensorLock adopts a divide-and-conquer strategy in identifying model dependencies and their types to avoid misclassification. As data-free dependencies (e.g., *Quantization* and *Merging*) preserve distinct weight-space characteristics between parent and child weights, TensorLock

prioritizes their identification. For *Quantization*, which inherently collapses continuous weight distributions onto discrete weight distributions, TensorLock measures the statistical alignment via Normalized Mutual Information that is robust to discretization noise. For *Merging*, TensorLock verifies whether the candidate's weights can be reconstructed via *linear interpolation* or *sparse task vector aggregation* from parent candidates. Once *Quantization* and *Merging* dependencies are determined, TensorLock then resolves the remaining *Fine-Tuning* dependencies by l_2*-norm based distance* and *singular value entropy-based direction*. Finally, the identified *Quantization*, *Merging*, and *Fine-Tuning* dependencies jointly form a unified type-aware MDG.

Evaluation. We first construct MDGBench, a comprehensive benchmark comprising 137 real-world models and 131 dependency edges across multiple domains with three dependency types (*Fine-Tuning*, *Quantization*, *Merging*), which is the first benchmark supporting both cluster-wise and graph-wise evaluation. For effectiveness evaluation, TensorLock achieves an *ARI* of 0.96 in clustering, outperforming the best baseline HuRef (0.69) by 39%. For dependency identification, TensorLock $^+$ achieves a *DF*1 of 0.82, outperforming MoTHer$^+$ (0.28) by 193%. For usefulness evaluation, we apply TensorLock to 289 supposedly isolated models that miss the model dependency. TensorLock identifies that 189 (65%) of them actually have dependencies with existing model clusters. To date, 42 developers have confirmed and updated our identified dependencies. Notably, for 38 identified dependencies, both the parent and child models rank within the top 1,000 most-downloaded on Hugging Face, demonstrating the community-wise contribution of TensorLock.

Contributions. This paper makes the following contributions.

- **Framework.** We propose TensorLock, the first framework that constructs type-aware model dependency graph by first conducting connectivity-based clustering and then leveraging type-specific fingerprints to accurately identify model dependencies.
- **Benchmark.** We construct MDGBench, a comprehensive benchmark comprising 137 real-world models across multiple domains with three dependency types. MDGBench supports both cluster-wise and graph-wise evaluation with diverse model formats (e.g., safetensors, GGUF, GPTQ).
- **Evaluation.** We conduct comprehensive evaluation to demonstrate TensorLock's superior effectiveness and usefulness in recovering model dependencies.

2 Empirical Study

To validate the prevalence of model dependency opacity and motivate the necessity for automated dependency identification, we first conducted a preliminary empirical study on Hugging Face model dependency. Our study is guided by the following three research questions:

- **RQ1 Dependency Prevalence & Opacity in Model Supply Chain:** How prevalent is model dependency in the model supply chain, and how well is it currently managed?
- **RQ2 Dominant Architectures in Model Supply Chain:** Which model architecture constitutes the backbone of the current model supply chain?
- **RQ3 Dependency Types in Model Supply Chain:** What types of dependencies exist in the model supply chain?

Model Dependency Identification. We focused on the top 10,000 most downloaded models on Hugging Face (as of November 30, 2025) to ensure that our analysis captured high-impact models within the supply chain. From this population, we constructed a statistically representative sample of 369 models (95% confidence level, 5% margin of error). To establish a reliable ground truth and mitigate the known incompleteness of model cards, two domain experts (each with over three years of experience in deep learning) independently identified the model dependency along with dependency type of every sampled model. This manual inspection involved cross-referencing

Table 1. Distribution of Model Dependency Type Across Domains

Domain	FFT	QT	MG	PEFT	Distill	Convert	RL	Total
NLP	100	59	16	3	2	1	1	182
CV	7	4	1	2	0	0	0	14
Audio	12	0	0	0	0	1	0	13
Multimodal	5	3	1	2	0	0	0	11
Other	2	0	0	0	0	0	0	2
Total	**126**	**66**	**18**	**7**	**2**	**2**	**1**	**222**

model cards (READMEs), associated research papers, repository commit histories, and training configuration files, achieving a Cohen's Kappa of 0.971; all discrepancies were resolved through adjudication by a third expert. During this process, we identified 119 undeclared dependencies linking sampled models to their parents and corrected 2 erroneous dependency declarations in the model cards. Notably, both of the 2 erroneous dependencies are *Merging* mislabeled as *Fine-Tuning*. This mislabeling resulted in missing multi-parent dependencies. Based on the identified dependency, we stratified models into three structural roles; i.e., *Dependent Models* M_d (nodes with an in-degree > 0) obtained by reusing upstream parents; *Base Models* M_b (nodes with an out-degree > 0) acting as the source for downstream models; and *Isolated Models* M_i (nodes with no in-degree or out-degree), which exhibit no connectivity to other models in the sampled supply chain.

2.1 RQ1: Dependency Prevalence & Opacity in Model Supply Chain

81.8% (302/369) of the sampled models participate in model reuse. Among these, 64.9% (196/302) serve as M_b, providing the foundation for downstream reused models, while 73.5%(222/302) function as M_d. Most notably, 38.1% (115/302) of models simultaneously occupy both roles, acting as a dependent child of an upstream parent while serving as a base for further reused models. This significant overlap confirms that the model supply chain is not merely a collection of isolated pair-wise interactions but a complex dependency network. On the one hand, 53.6% of identified dependencies were absent from model cards. On the other hand, among the remaining 67 (18.2%) models classified as M_i, the lack of explicit knowledge makes it impossible even for our experts to manually determine whether they are involved in the supply chain. The poor management of model dependency leads to the fragmentation of model supply chains for the provenance of high-impact models, creating significant risks for both upstream and downstream stakeholders.

Finding 1: Model reuse is pervasive (81.8%) with complex transitive topologies, yet 53.6% of these model dependencies are undeclared and poorly managed.

2.2 RQ2: Dominant Architectures in Model Supply Chain

As detailed in Table 1, an overwhelming 96.4% (214/222) of all identified dependency edges occur between transformer-based models, underscoring their central role in the model supply chain. In the NLP domain, this dominance is absolute: all 182 dependency edges involve transformer architectures, driven by the widespread adoption of LLMs. Only 8 edges involve non-transformer models; i.e., 7 in CV and 1 in multimodal while all dependencies in other domains remain transformer-based.

Finding 2: Transformer-based models dominate (96.4%) the model supply chain.

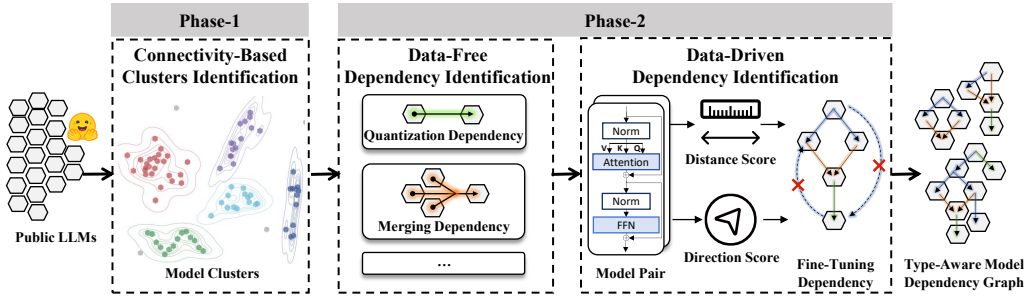


Fig. 1. Overview of TensorLock

2.3 RQ3: Dependency Types in Model Supply Chain

We categorized all M_d by domain and analyzed the specific dependency types employed during their creation. As detailed in Table 1, our analysis reveals that the vast majority of identified dependencies (97.7%) are concentrated in three primary categories; i.e., *Fine-Tuning* (i.e., *Fully Fine-Tuning* and *Parameter-Efficient Fine-Tuning*, hereafter *FFT* and *PEFT*), *Quantization* (*QT*), *Merging* (*MG*). Notably, *Merge* can introduce *multi-parent* dependencies, transforming simple model trees into complex DAGs. PEFT accounts for a relatively small proportion within the top 10,000 models, as PEFT-derived models typically reside at the periphery of the supply chain (i.e., leaf nodes with lower download counts). The remaining dependencies include *Distillation* (*Distill*), *Convert*, and *Reinforcement Learning* (*RL*), which, while less frequent, represent specialized dependency types.

Finding 3: Model dependency types are diverse and complex, with a predominant percentage of 97.7% concentrated in *Fine-Tuning*, *Quantization*, and *Merging*.

Summary. These results confirm that model reuse is a dominant but poorly managed practice, largely centered on transformer-based models and complicated by transitive dependency topology and diverse dependency types. This raises the emergent need of automated model dependency identification, especially for transformer-based models.

3 Framework

Inspired by the empirical findings, we designed TensorLock, a model dependency recovering framework that takes a set of candidate transformer-based models as input, identifies and outputs their dependencies as a type-aware MDG. TensorLock constructs the MDG through two phases; i.e., identifying disjoint model clusters (Sec. 3.2), and identifying dependency within these clusters (Sec. 3.3 and 3.4) as shown in Fig. 1.

3.1 Assumptions and Definitions

Assumption of Accessible Model Weights: We assume that the analyst has access solely to the published model weight files (e.g., `.safetensors`, `.bin`, `.pt`, `.gguf`), which are typically distributed via public platforms like Hugging Face. This assumption aligns with standard white-box fingerprinting paradigms. Original training data and source code are often inaccessible, and configuration files are frequently poorly managed, rendering metadata-based identification inadequate [18].

Assumption of a Shared Base Model: We restrict model merging to scenarios where all parent models share a common pre-trained ancestor. This restriction aligns with the prerequisite of current model merging techniques [21, 22, 46, 49]; i.e., merging is effective only when parent models are fine-tuned variants of the same base model or the base model itself, as their weight distributions remain sufficiently similar to enable meaningful interpolation or aggregation.

Definition of the Model Weights: Formally, given a model m from a set of candidate models $m \in \mathcal{M}$, we define its weight space as $\Theta_m = \{\mathcal{L}^{(l)}\}_{l=1}^{L}$, where L denotes the total number of layers. Each layer $\mathcal{L}^{(l)}$ is decomposed into two distinct weight sets: the Multi-Head Self-Attention (MSA) and the Feed-Forward Network (FFN), denoted as $\mathcal{L}^{(l)} = \mathbf{W}^{(l)}_{\text{MSA}} \cup \mathbf{W}^{(l)}_{\text{FFN}}$. The MSA components capture contextual dependencies via attention mechanisms, while the FFN components stores factual knowledge and performing non-linear transformations [14].

Definition of Data-free and Data-driven Dependencies: We categorize model dependency types into two groups based on whether additional training data is required for the transformation. *Data-free dependencies* arise from deterministic weight transformations that do not involve training data, such as *Quantization* and *Merging*. These dependencies preserve distinct relationships between parent and child model weights, enabling precise identification through weight distribution analysis. *Data-driven dependencies* arise from training-based transformations that require external data, such as *Fine-Tuning*. Since the original training data is typically inaccessible, these dependencies induce non-deterministic weight shifts and thus can only be inferred through relative distance and directional analysis.

3.2 Connectivity-Based Model Clusters Generation

Existing approaches (e.g., TensorGuard [43]) require pre-specifying the number of clusters and rely on flat comparisons that fail to capture chain-like transitive dependencies. To address this, TensorLock adopts a connectivity-based clustering approach utilizing HDBSCAN [5]. Formally, we denote the resulting cluster set as $C = \{\mathbf{C}_1, \mathbf{C}_2, \ldots, \mathbf{C}_n\}$, where n is automatically determined by the connectivity structure rather than pre-specified. Each cluster $\mathbf{C}_i$ contains a subset of models such that all clusters are disjoint ($\mathbf{C}_i \cap \mathbf{C}_j = \emptyset$ for any $i \neq j$), and their union covers the entire model set ($\bigcup_{i=1}^{n} \mathbf{C}_i = \mathcal{M}$). The clustering process consists of two steps; i.e., distance quantification and connectivity-based clustering.

First, TensorLock computes the pair-wise distances among the models. TensorLock exclusively utilizes the MSA weights ($\mathbf{W}^{(l)}_{\text{MSA}}$) rather than the complete layer weights ($\mathcal{L}^{(l)}$) for distance computation. We exclude FFN weights as they serve as domain-sensitive key-value memories, undergoing substantial weight adaptation to accommodate new factual knowledge during model reuse (e.g., fine-tuning [14, 31]). Conversely, MSA encodes generalizable contextual patterns and maintains higher invariance and stability across the same lineage [40]. Specifically, TensorLock extracts the MSA weights for each model and aligns their dimensions by truncating to the minimum depth $L_{\min} = \min(L_i, L_j)$. Each MSA layer consists of four components $\kappa \in \{Q, K, V, O\}$, corresponding to the Query, Key, Value, and Output projection matrices. For each component κ at layer l, TensorLock computes three statistical features; i.e., standard deviation $\text{sd}^{(l)}_{\kappa}$, skewness $\text{sk}^{(l)}_{\kappa}$, and kurtosis $\text{ku}^{(l)}_{\kappa}$. The features at each layer are concatenated to form feature vectors $\mathbf{f}^{\kappa}_{\text{sd}} = [\text{sd}^{(1)}_{\kappa}, \ldots, \text{sd}^{(L_{\min})}_{\kappa}]$, and similarly for $\mathbf{f}^{\kappa}_{\text{sk}}$ and $\mathbf{f}^{\kappa}_{\text{ku}}$. The similarity is computed by averaging the Spearman rank correlations $\rho(\cdot, \cdot)$ across all statistical features and MSA components, formally defined as follows:

$$s_{\text{MSA}}(m_i, m_j) = \frac{1}{4} \sum_{\kappa \in \{Q,K,V,O\}} \frac{1}{3} \left(\rho(\mathbf{f}^{\kappa}_{\text{sd},i}, \mathbf{f}^{\kappa}_{\text{sd},j}) + \rho(\mathbf{f}^{\kappa}_{\text{sk},i}, \mathbf{f}^{\kappa}_{\text{sk},j}) + \rho(\mathbf{f}^{\kappa}_{\text{ku},i}, \mathbf{f}^{\kappa}_{\text{ku},j}) \right). \tag{1}$$

Second, to derive the pairwise distance for clustering, TensorLock constructs a connectivity graph $\mathcal{G}_{conn} = (\mathcal{M}, \mathcal{E}_{conn})$, where an edge $(m_i, m_j) \in \mathcal{E}_{conn}$ exists if and only if $s_{\text{MSA}}(m_i, m_j) \geq \tau_{\text{conn}}$. The distance $d_{\text{MSA}}(m_i, m_j)$ is defined as the shortest hop count between m_i and m_j in $\mathcal{G}_{conn}$:

$$d_{\text{MSA}}(m_i, m_j) = \min_{p \in \mathcal{P}(m_i, m_j)} |p|, \tag{2}$$

where $\mathcal{P}(m_i, m_j)$ denotes the set of all paths between m_i and m_j in $\mathcal{G}_{\text{conn}}$, and $|p|$ is the number of hops along path p. $d_{\text{MSA}}(m_i, m_j) = \infty$ if m_j is not reachable from m_i. This connectivity-based distance offers two advantages over direct similarity; i.e., (1) the threshold τ_{conn} filters out low-confidence connections, reducing spurious edges; (2) hop-count captures transitive relationships, allowing models with low direct similarity to be grouped via intermediate connections (e.g., $m_i \rightarrow m_j \rightarrow m_k$). These distances are subsequently arranged into a connectivity matrix $\mathcal{A}_{\text{conn}} \in \mathbb{R}^{|\mathcal{M}|\times|\mathcal{M}|}$, in which each entry $d(m_i, m_j)$ corresponds to the distance between models m_i and m_j. Given $\mathcal{A}_{\text{conn}}$, TensorLock applies HDBSCAN to construct a cluster hierarchy based on the connectivity and extracts stable clusters through hierarchical density analysis. This connectivity-based formulation automatically determines cluster numbers and preserves arbitrary cluster shapes, avoiding the fragmentation of chain-like or sibling-like dependencies, yielding $C = \{\mathbf{C}_1, \mathbf{C}_2, \ldots, \mathbf{C}_n\}$.

3.3 Data-Free Dependency Identification

As established in the definition of data-free and data-driven dependencies, data-free dependencies preserve deterministic mathematical feature between parent and child weights, enabling precise identification through their distinct transformational paradigms. Therefore, TensorLock prioritizes the identification of these dependencies over data-driven ones. Based on the extensive usage indicated from our empirical analysis, this module primarily targets *Quantization* and *Merging*. Nevertheless, TensorLock can be readily extended to other data-free dependencies.

3.3.1 Quantization Dependency Identification. TensorLock first identifies the set of *Quantization* dependency edges, denoted as $\mathcal{E}_{\text{QT}} = \{\langle m_p, m_q\rangle \mid m_p, m_q \in \mathbf{C}\}$, where m_p is the parent model and m_q is the quantized model. This process involves two steps; i.e., *Candidate Quantized and Parent Models Identification*, and *Quantization Dependency Identification*.

Candidate Quantized and Parent Models Identification. To minimize computational overhead, TensorLock employs a heuristic detection strategy. Initially, TensorLock performs a lightweight model format scan to identify models with explicit *Quantization* suffixes (e.g., `.gguf`). For general formats (e.g., `.safetensors`) without explicit quantization indicators, TensorLock identifies the weight files for internal *Quantization* signatures, such as specific weight names (e.g., `qweight`, `scales`) or low-precision data types (e.g., `int4`, `int8`). The identified models by our heuristic strategy are treated as candidate quantized models $\mathcal{M}_q$ and models that exhibit neither of these traits are classified as candidate parent models $\mathcal{M}_p = \mathcal{M} \setminus \mathcal{M}_q$.

Quantization Dependency Identification. Quantization reduces model storage and computational costs by representing weights with lower bit-width integers (e.g., INT4, INT8), enabling efficient deployment on resource-constrained devices. This bit-width reduction inherently imposes a unique weight transformation. i.e., collapsing the continuous weight distribution onto a discrete, uniformly spaced grid. While specific implementations vary, ranging from activation-aware scaling (AWQ) to Hessian-based error compensation (GPTQ), we observe that they all adhere to a generalized affine formulation [10–13, 29]; i.e., $\Theta_c \approx S \cdot \text{round}\left((\Theta_p + \epsilon)/S + Z\right)$. In this paradigm, the scale factor S and zero-point Z (which is often set to zero in GPTQ and AWQ) dictate the structure of the target value space, while ϵ denotes method-specific calibration offsets that compensate for quantization error (e.g., Hessian-based adjustments in GPTQ or activation-aware scaling in AWQ). This operation imposes a unique *Quantization* weight transformation. i.e., it collapses the continuous weight distribution of the parent model onto a discrete, uniformly spaced grid, which makes it distinguishable from other dependency type. Leveraging this uniqueness, TensorLock computes the statistical alignment between a candidate model's and the parent's shared discrete weight distribution as a verifiable fingerprint of their dependency.

To quantify the structural alignment between the parent and candidate models, we propose the Quantization Provenance Score (*QPS*). Given a candidate parent model $m_p \in \mathcal{M}_p$ and a candidate quantized model $m_q \in \mathcal{M}_q$, TensorLock first grids the flattened weight vector of each layer l into discrete probability distributions $P^{(l)}$ and $Q^{(l)}$ using histogram binning with B bins, where $|B|$ denotes the number of bins. Fewer bins reduce distributional details, whereas excessive bins yield sparse counts that compromise statistical reliability. Unlike standard geometric metrics (e.g., Euclidean distance) which are sensitive to the absolute numerical shifts and noise introduced by the rounding operation, we employ Normalized Mutual Information (NMI). NMI is robust to scaling and shifting, focusing instead on the statistical dependence between distributions. In our context, it measures how accurately the candidate's weight distribution preserves the information structure of the parent's original grid. The *QPS* is defined as the layer-averaged NMI:

$$QPS(m_p, m_q) = \mathrm{NMI}(P, Q) = \frac{1}{L} \sum_{l=1}^{L} \frac{2 \cdot I(P^{(l)}; Q^{(l)})}{H(P^{(l)}) + H(Q^{(l)})} \tag{3}$$

where $H(\cdot)$ denotes Shannon entropy, and $I(P^{(l)}; Q^{(l)})$ is the mutual information quantifying the statistical dependence between the parent's distribution $P^{(l)}$ and the candidate's distribution $Q^{(l)}$. A high *QPS* indicates that the candidate retains the distribution of the parent's quantization grid. TensorLock identifies the true parent by selecting the candidate that maximizes *QPS*, yielding $\mathcal{E}_{\mathrm{QT}}$.

3.3.2 Merging Dependency Identification. TensorLock then identifies the set of *Merging* dependency edges, denoted as $\mathcal{E}_{\mathrm{MG}} = \{\langle m_{p_1}, m_{p_2}, \ldots, m_{p_n}, m_m \rangle \mid m_{p_1}, m_{p_2}, \ldots, m_{p_n}, m_m \in C\}$, where $m_{p_1}, m_{p_2}, \ldots, m_{p_n}$ are parent models and m_m is the merged model. This process involves *Candidate Merged and Parent Models Identification* and *Merging Dependency Identification*.

Candidate Merged and Parent Models Identification. Unlike *Quantization* with explicit signatures, merged models inherit their parents' format and weights, making heuristic-based detection ineffective. To address this, we exploit the inherent characteristic of model merging, where a merged model inherits parameters from multiple parent models simultaneously and shares a high similarity with its parent [49]. Based on the observation, TensorLock re-leverages *MSA-Based Similarity* (Eq. 1) and the threshold τ_{conn} to identify candidate merged models and their parents. For each model m, we define its *candidate parent set* as $\mathcal{M}_p(m) = \{m' \in \mathbf{C} \mid s_{\mathrm{MSA}}(m, m') \geq \tau_{\mathrm{conn}}\}$. If $|\mathcal{M}_p(m)| \geq 2$, the model m is identified as a *candidate merged model*, and the models within $\mathcal{M}_p(m)$ are treated as the initial set of *candidate parent models*.

Merging Dependency Identification. The candidate identification may introduce false positives, as other high-connectivity models (e.g., the root model or models in the middle of a chain) can also satisfy the $|\mathcal{M}_p(m)| \geq 2$. To distinguish true merged models from such false positives and identify their actual parents, we categorize current *Merging* operations into *Interpolation* and *Task Vector Aggregation* according to Yang et al. [49].

- *Linear Interpolation.* Interpolation, e.g., Linear Interpolation (Lerp) and Spherical Linear Interpolation (SLERP), is inherently a dual-parent paradigm that blends the weights of exactly two parent models. These methods adhere to a generalized interpolation formulation; i.e., $\Theta_{m_m} = \mathcal{F}(\Theta_{p_1}, \Theta_{p_2}, t)$, where Θ_{p_1} and Θ_{p_2} denote the weights of the two parent models, and $t \in [0, 1]$ is the mixing coefficient that controls their relative contribution. If a candidate model m_m is indeed interpolated from parents m_i and m_j, there exists a specific t such that Θ_{m_m} exhibits high directional alignment with the interpolated weights of m_1 and m_2 under the mixing coefficient t. To this end, TensorLock exploits this paradigm, treating the directional alignment between a candidate's weights and the interpolation coefficient of m_1 and m_2 as a verifiable fingerprint of *Interpolation*. To quantify the alignment, we propose the Interpolation Provenance Score (IPS). Unlike Euclidean distance that is sensitive to magnitude differences between Lerp and

SLERP, we employ Cosine Similarity as a unified metric, which measures directional alignment while being invariant to scalar variations. For each layer l and component type $\kappa \in \{\text{MSA}, \text{FFN}\}$, TensorLock searches the optimal mixing coefficient $t_\kappa^{*(l)} \in [0,1]$ that maximizes the Cosine Similarity between $\mathbf{W}_{\kappa,c}^{(l)}$ and the interpolated weights $(1-t)\mathbf{W}_{\kappa,i}^{(l)} + t \cdot \mathbf{W}_{\kappa,j}^{(l)}$. The IPS is then defined as the layer-averaged maximum similarity, where a higher IPS over the threshold τ_{mg} indicates that the candidate resides on the interpolation path between the two parents.

$$\text{IPS}(m_m, m_i, m_j) = \frac{1}{2L} \sum_{l=1}^{L} \sum_{\kappa} \max_{t \in [0,1]} \text{CosSim}\big(\mathbf{W}_{\kappa,c}^{(l)}, (1-t)\mathbf{W}_{\kappa,i}^{(l)} + t \cdot \mathbf{W}_{\kappa,j}^{(l)}\big) \tag{4}$$

- *Sparse Task Vector Aggregation.* Unlike *Interpolation* which operates directly on model weights, Sparse Task Vector Aggregation techniques (e.g., TIES [46], DARE [51], DARE-TIES) manipulate the weight shifts $\Delta = \Theta - \Theta_{\text{base}}$, termed *task vectors*. These techniques adhere to a generalized sparse aggregation formulation; i.e., $\Theta_{m_m} = \Theta_{\text{base}} + \sum_i \alpha_i \Delta_i$, where Δ_i denotes the sparsified task vector. This paradigm imposes a unique weight transformation; i.e., the merged model largely preserves the base weights, with deviations confined to a sparse subspace spanned by parent task vectors. If a candidate model m_m is indeed aggregated from parents, its residual $\Theta_{m_m} - \Theta_{\text{base}}$ should be linearly decomposable into sparse combinations of parent task vectors. To this end, TensorLock exploits this paradigm, treating the sparse linear decomposability of the candidate's residual as a verifiable fingerprint of *Sparse Task Vector Aggregation*. To verify this sparse linear decomposability, given a candidate merged model m_m and its candidate parent set $\mathcal{M}_p(m_m)$, TensorLock first identifies the *base model* m_{base} as the model with the highest connectivity in $\mathcal{M}_p(m_m)$, since the shared pre-trained ancestor typically exhibits the strongest connections to all derived models. The remaining models form the candidate task model set $\mathcal{M}_{\text{task}} = \mathcal{M}_p(m_m) \setminus \{m_{\text{base}}\}$. However, $\mathcal{M}_{\text{task}}$ may include false positives, as high-connectivity models (e.g., sibling fine-tuned variants) can exhibit strong similarity without being actual parents. To identify true parents, TensorLock employs LASSO regression [38] to reconstruct the candidate's residual $\Delta_{m_m} = \Theta_{m_m} - \Theta_{m_{\text{base}}}$ as a sparse linear combination of task vectors $\{\Delta_i = \Theta_{m_i} - \Theta_{m_{\text{base}}} \mid m_i \in \mathcal{M}_{\text{task}}\}$. The L1 regularization inherent in LASSO shrinks irrelevant coefficients to exactly zero, automatically filtering out non-parent models. Models with non-zero coefficients form the identified parent set $\mathcal{M}'_{\text{task}} = \{m_i \in \mathcal{M}_{\text{task}} \mid \alpha_i \neq 0\}$. We then propose the Task Vector Provenance Score (TPS) to quantify reconstruction quality, where $\Delta_{m_m,\kappa}^{(l)}$ and $\Delta_{i,\kappa}^{(l)}$ denote the task vectors at layer l for component κ. A high TPS indicates that the candidate's residual is well explained by the identified parent task vectors. TensorLock confirms the merge if TPS $> \tau_{mg}$, yielding the final parent set $\mathcal{M}'_{\text{task}} \cup \{m_{\text{base}}\}$.

$$\text{TPS}(m_m, \mathcal{M}_{\text{task}}) = \frac{1}{2L} \sum_{l=1}^{L} \sum_{\kappa \in \{\text{MSA,FFN}\}} \left(1 - \frac{\|\Delta_{m_m,\kappa}^{(l)} - \sum_{m_i \in \mathcal{M}'_{\text{task}}} \alpha_i \Delta_{i,\kappa}^{(l)}\|_2^2}{\|\Delta_{m_m,\kappa}^{(l)}\|_2^2}\right) \tag{5}$$

3.4 Data-Driven Dependency Identification

Identifying dependencies in data-driven *Fine-Tuning* (including *FFT* and *PEFT*) is inherently challenging compared to data-free paradigms, as the inaccessibility of training data precludes deterministic verification. Inspired by MoTHer which infers *Fine-Tuning* dependencies via parameter relative distance and direction, TensorLock first calculates the pair-wise *Fine-Tuning* distance and direction matrices given each cluster $\mathbf{C}_i \in \mathcal{C}$ (Sec. 3.4.1). Based on the pre-deterministic data-free edges recognized, TensorLock then refines the distance and direction matrices and identifies the *Fine-Tuning* dependency edges (Sec. 3.4.2). After traversing all clusters, TensorLock yields the final Model Dependency Graph $\mathcal{G} = \{\mathcal{M}, \mathcal{E}\}$, where $\mathcal{E}$ represents the complete set of dependency edges.

3.4.1 Model Distance and Direction Calculation. Given a cluster of models $\mathbf{C}_i$, TensorLock outputs a pair-wise distance matrix $\mathcal{A}_{dis}$ and a binary direction matrix $\mathcal{A}_{dir}$.

To calculate model distance for identifying fine-tuning dependencies, we additionally account for FFN layers, which encode task-specific factual knowledge acquired during fine-tuning [14]. Accordingly, for each pair of models (m_i, m_j), TensorLock computes the fine-tuning distance as the layer-wise l_2-norm averaged over both MSA and FFN components. This yields the pair-wise distance matrix $\mathcal{A}_{dis} \in \mathbb{R}^{|C_i|\times|C_i|}$.

$$\mathcal{A}_{dis}(m_i, m_j) = \frac{1}{2L}\sum_{l=1}^{L}\left(\|\mathbf{W}^{(l)}_{\text{MSA},i} - \mathbf{W}^{(l)}_{\text{MSA},j}\|_2 + \|\mathbf{W}^{(l)}_{\text{FFN},i} - \mathbf{W}^{(l)}_{\text{FFN},j}\|_2\right) \tag{6}$$

To determine dependency directions, we leverage spectral analysis. This approach differs from [18], which uses kurtosis to detect weight "specialization" based on the observation that weight outliers increase during pre-training but decrease during fine-tuning. However, the kurtosis-based calculation becomes unstable for models with large-scale parameters (e.g., LLMs), as the massive parameter space statistically drowns out the subtle outlier shifts caused by fine-tuning [2]. Intuitively, as a model specializes for a specific task, it activates additional expressive dimensions to encode task-specific knowledge, leading to an increase in the effective rank and higher spectral entropy [18]. Based on this insight, we compute a layer-wise spectral entropy difference as the directionality indicator. Concretely, for each weight matrix $\mathbf{W}$, we perform Singular Value Decomposition (SVD) as $\mathbf{W} = \mathbf{U}\Sigma\mathbf{V}^\top$, where $\Sigma = \text{diag}(\sigma_1, \sigma_2, \ldots, \sigma_r)$ contains the singular values. We normalize these singular values to form a probability distribution $p_i = \sigma_i / \sum_{j=1}^{r}\sigma_j$, from which we compute the spectral entropy $H_\sigma(\mathbf{W}) = -\sum_{i=1}^{r} p_i \log p_i$ (analogous to the Shannon entropy in Sec. 3.3). Given two models m_i and m_j, the directionality score is computed as:

$$\mathcal{S}(m_i, m_j) = \frac{1}{2L}\sum_{l=1}^{L}\sum_{\kappa\in\{\text{MSA,FFN}\}}\left(H_\sigma(\mathbf{W}^{(l)}_{\kappa,j}) - H_\sigma(\mathbf{W}^{(l)}_{\kappa,i})\right) \tag{7}$$

Finally, we construct the binary direction matrix $\mathcal{A}_{dir} \in \{0,1\}^{|C|\times|C|}$, where $\mathcal{A}_{dir}[i,j] = 1$ if and only if $\mathcal{S}(m_i, m_j) > 0$, indicating a valid dependency direction from the lower-entropy generalist parent m_i to the higher-entropy specialized child m_j.

TensorLock then updates $\mathcal{A}_{dis}$ and $\mathcal{A}_{dir}$ with the identified deterministic dependencies $\mathcal{E}_{\text{fixed}} = \mathcal{E}_{\text{QT}} \cup \mathcal{E}_{\text{MG}}$. For every identified child model m_c involved in a fixed dependency (either $\langle m_{p_1}, m_c\rangle \in \mathcal{E}_{\text{QT}}$ with a single parent, or $\langle m_{p_1}, \ldots, m_{p_n}, m_c\rangle \in \mathcal{E}_{\text{MG}}$ with multiple parents), TensorLock performs two topological corrections. First, it strictly enforces directionality by updating the direction matrix entries $\mathcal{A}_{dir}[m_{p_i}, m_c] = 1$ for all parents m_{p_i}. Second, TensorLock sets the distance of these validated edges to zero (i.e., $\mathcal{A}_{dis}(m_{p_i}, m_c) \leftarrow 0$) and effectively prunes all other potential parents $m_k \notin \{m_{p_1}, \ldots, m_{p_n}\}$ by setting their distance to infinity (i.e., $\mathcal{A}_{dis}(m_k, m_c) \leftarrow \infty$). This constraint injection process yields the refined matrices $\mathcal{A}'_{dir}$ and $\mathcal{A}'_{dis}$.

3.4.2 Fine-Tuning Dependency Identification. Unlike *Merging* which has multiple parent dependencies, *Fine-Tuning* inherently follows a single-parent paradigm. This structural implies that the fine-tuning dependencies within a cluster naturally form a tree topology. Consequently, identifying fine-tuning dependencies can be formulated as finding a directed spanning tree that minimizes the total edge weight (i.e., parameter distance) while respecting the directionality constraints encoded in $\mathcal{A}'_{dir}$. i.e., the Minimum Directed Spanning Tree (MDST) problem.

Based on $\mathcal{A}'_{dir}$ and $\mathcal{A}'_{dis}$, TensorLock identifies the Model Dependency Graph via Chu-Liu-Edmonds' algorithm for MDST [9, 18]. However, a fundamental topological mismatch arises. The deterministic dependencies in $\mathcal{E}_{\text{fixed}}$ inherently induce a DAG structure with multi-parent nodes (e.g.,

Merging). When MDST incorporates the zero-distance edges from $\mathcal{A}'_{dis}$, it greedily satisfies these multi-parent constraints but leaves the parent nodes as isolated roots, fracturing the dependency of their upstream models. To this end, TensorLock adopts a *detach-resolve-reattach* strategy that generalizes to any multi-parent dependency type captured in $\mathcal{E}_{\text{fixed}}$. First, TensorLock temporarily detaches all multi-parent child nodes (along with their downstream subgraphs) from the initial MDST. Second, TensorLock re-executes the MDST algorithm on the remaining subgraph, which now correctly resolves the shared upstream lineage of the previously isolated parent nodes. Finally, TensorLock reattaches the detached subgraphs using the deterministic zero-distance edges in $\mathcal{A}'_{dis}$, preserving both the validated multi-parent relationships and the identified upstream ancestry.

After traversing all clusters in C, TensorLock aggregates the identified dependency edges from both data-free ($\mathcal{E}_{\text{QT}}$, $\mathcal{E}_{\text{MG}}$) and data-driven ($\mathcal{E}_{\text{FT}}$) phases, yielding the final Model Dependency Graph $\mathcal{G} = \{\mathcal{M}, \mathcal{E}\}$, where $\mathcal{E} = \mathcal{E}_{\text{QT}} \cup \mathcal{E}_{\text{MG}} \cup \mathcal{E}_{\text{FT}}$ represents the complete set of dependency edges.

4 Evaluation

We first establish a comprehensive model dependency benchmark MDGBench. Building on this benchmark, we evaluate the effectiveness of TensorLock. Additionally, we apply this framework to automatically identify dependencies for real-world isolated models. All experiments were uniformly executed on a standardized hardware setup consisting of four NVIDIA A100 80GB GPUs. Our evaluation is structured around the following five research questions (RQs):

- **RQ4: Effectiveness Evaluation.** How effective is TensorLock in recovering model dependencies across different domains and dependency types compared to state-of-the-art approaches?
- **RQ5: Ablation Study.** How is each component's contribution to TensorLock's effectiveness?
- **RQ6: Sensitivity Analysis.** How do hyperparameters affect the effectiveness of TensorLock?
- **RQ7: Efficiency Evaluation.** What is TensorLock's time cost in identifying dependencies?
- **RQ8: Usefulness Evaluation.** What is the usefulness of TensorLock?

4.1 Benchmark Construction

Existing approaches for model dependency identification often construct dedicated datasets for evaluation. As summarized in Table 2, we assess these datasets or benchmarks along three dimensions: (1) *Characteristics*, including whether models are collected from real-world repositories, whether multiple domains are covered, and whether multiple dependency types are supported; (2) *Granularity*, indicating whether the benchmark supports cluster-level grouping and graph-level dependency reconstruction; and (3) *Scale*, measured by the number of models, clusters, edges, and parameter sizes. Most existing datasets or benchmarks are limited in scale, typically containing only a dozen to a few tens of models, and focus exclusively on pair-wise comparisons without cluster or graph-level evaluation. While LeafBench provides 149 real-world models with over 1T parameters, yet it remains restricted to pair-wise comparisons and lacks the granularity needed for complex and transitive dependency topology. MoTher, while containing approximately 500 models and supporting cluster-level evaluation, relies on self-trained dependent models rather than real-world artifacts, and is limited to the vision domain with relatively small model sizes. To address these gaps, we curate MDGBench, a comprehensive benchmark consisting of 137 real-world models across domains of NLP, CV, Audio (AD) and Multimodal (MD) with three dependency types (e.g., *Fine-Tuning*, *Quantization*, *Merging*), supporting both cluster-level and graph-level evaluation for model dependency identification.

4.1.1 Model Dependency Graph Construction. Considering the rapid emergence of LLMs since 2022, we first identified the top 5 most downloaded transformer-based text-generation models released annually between 2022 and 2025. This initial set of 20 candidate models served as the cluster seeds

Table 2. Comparison of MDGBench with Existing Model Dependency Benchmarks

	Characteristics			Granularity		Scale			
Benchmark	**Real-world†**	**Multi-Dom.**	**Multi-Dep.**	**Cluster**	**Graph**	**# Models**	**# Clusters**	**# Edges**	**Param Size**
Reef	✗	NLP	✓	✗	✗	38	–	–	< 500B
PDF	✓	NLP	✗	✗	✗	12	–	–	< 500B
HuRef	✗	NLP	✗	✗	✗	69	–	–	< 500B
LeafBench	✓	NLP	✓	✗	✗	149	–	–	> 1T
LLMMap	✓	NLP	✗	✗	✗	36	–	–	< 100B
MET	✓	NLP	✓	✗	✗	25	–	–	< 500B
Trap	✓	NLP	✗	✗	✗	38	–	–	< 500B
TensorGuard	✓	NLP	✗	✓	✗	58	8	50	< 500B
MoTHer	✗	CV	✓	✓	✗	~ 500	5	~ 500	< 500B
MDGBench	✓	NLP,CV,AD,MD	✓	✓	✓	**137**	**17**	**131**	> 1**T**

for identifying major model families. Although some models like `petals-team/StableBeluga2` exhibit high download counts, they have few documented derived models. Therefore, to ensure MDGBench provides sufficient complexity for dependency analysis, we filtered these families to retain only those containing more than 100 recorded dependent models in model cards and we ultimately identified 14 core cluster seeds. To demonstrate the versatility of our proposed method across domains, we extended our scope beyond text-generation models. Using the same download-based ranking, we selected the top 1 model from the CV, Audio, and Multi-model domains (as of November 30, 2025), which are all transformer-based. This expanded our dataset to a total of 17 distinct model cluster seeds. We then adopted a sampling strategy informed by our empirical findings, which indicate a typical ratio of approximately 1:4 between root models and their downstream derivatives. Therefore, we randomly sampled 68 downstream dependent models from the 17 model families (e.g., `Qwen/Qwen2.5-1.5B, meta-llama/Llama-3.2-1B, openai-community/gpt2, Qwen/Qwen2-7B, google/flan-t5-base`). Since these sampled models may reside multiple hops away from their corresponding seed models (e.g., a model fine-tuned from a quantized variant), we further iteratively traced the complete dependency path from each seed model to each sampled descendant, which introduced 52 additional intermediate models. As a result, the final benchmark contains 137 models $\mathcal{M}$ and 131 dependency edges $\mathcal{E}$ (i.e., 32 $\mathcal{E}_{QT}$, 22 $\mathcal{E}_{MG}$, and 77 $\mathcal{E}_{FT}$). Each dependency edge was verified by cross-referencing model cards, in-text documentation, associated publications, commit histories, and training configuration files, and was independently annotated by two domain experts, achieving a Cohen's Kappa of 0.931. Any disagreement was further resolved by a third expert.

4.1.2 Evaluation Metrics for Model Dependency Identification. We design six metrics from cluster-wise and graph-wise perspectives to evaluate the effectiveness in identifying model dependency.

Cluster-Wise Metrics. We adopt the Adjusted Rand Index (*ARI*) [20] to quantify the accuracy of model clustering. Given the model corpus $\mathcal{M}$, let $C = \{\mathrm{C}_1, \ldots, \mathrm{C}_n\}$ denote the identified clusters, and $C^* = \{\mathrm{C}_1^*, \ldots, \mathrm{C}_k^*\}$ denote the ground-truth clusters. *ARI* measures pair-wise consistency; i.e., for any two models $m_i, m_j \in \mathcal{M}$, whether they are correctly grouped together (both in the same cluster) or correctly separated (in different clusters). *ARI* ranges from −1 to 1 (perfect match).

Graph-Wise Static Metrics. Given a model corpus $\mathcal{M}$, let $\mathcal{G}_{pred} = \{\mathcal{M}, \mathcal{E}_{pred}\}$ denote the predicted Model Dependency Graph identified by automated approaches, and $\mathcal{G}_{gt} = \{\mathcal{M}, \mathcal{E}_{gt}\}$ denote the ground-truth graph. We adopt the dependency edge precision (*DPre*), recall (*DRec*), and F1 score (*DF*1) to quantify the accuracy of dependency identification:

$$DPre = \frac{|\mathcal{E}_{pred} \cap \mathcal{E}_{gt}|}{|\mathcal{E}_{pred}|}, \quad DRec = \frac{|\mathcal{E}_{pred} \cap \mathcal{E}_{gt}|}{|\mathcal{E}_{gt}|}, \quad DF1 = \frac{2 \cdot DPre \cdot DRec}{DPre + DRec} \tag{8}$$

where each edge $e = \langle m_p, m_c \rangle \in \mathcal{E}$ represents a directed dependency from a parent model m_p to its child model m_c. An edge is considered correct only if both parent and child models match exactly.

Graph-Wise Dynamic Metrics. While static metrics evaluate the structural correctness of the identified graph, real-world supply chain risk often concentrates on specific vulnerable points and propagates dynamically. When a model is identified with risks, all downstream models are potentially affected by the risk. To quantify this traceability, we introduce *Reachability Recall*:

$$RchRec = \frac{1}{|\mathcal{M}|} \sum_{m \in \mathcal{M}} \frac{|\mathcal{R}_{pred}(m) \cap \mathcal{R}_{gt}(m)|}{|\mathcal{R}_{gt}(m)|} \tag{9}$$

where $\mathcal{R}_{gt}(m)$ and $\mathcal{R}_{pred}(m)$ denote the sets of downstream models reachable from m in the ground-truth and predicted graphs, respectively. A low *RchRec* indicates bad traceability. Since *RchRec* is recall-oriented and does not penalize falsely predicted reachable nodes, we further introduce a complementary metric, *Reachability Precision*:

$$RchPre = \frac{1}{|\mathcal{M}|} \sum_{m \in \mathcal{M}} \frac{|\mathcal{R}_{pred}(m) \cap \mathcal{R}_{gt}(m)|}{|\mathcal{R}_{pred}(m)|} \tag{10}$$

where *RchPre* denotes the proportion of predicted reachable downstream models that are truly reachable in the ground-truth graph. A low *RchPre* indicates that the predicted graph introduces many spurious risk propagation paths.

4.1.3 Model Format Alignment. Real-world models are often distributed in diverse quantized formats (e.g., GGUF, AWQ, GPTQ), which poses significant challenges for White-box fingerprinting approaches due to interface incompatibility. These approaches typically require gradients, activations, or unpacked weights with specific layer interfaces. We design a unified de-quantization pipeline that leverages format-specific loaders (e.g., `llama-cpp-python` for GGUF, `transformers` and `autoawq` for AWQ/GPTQ) and projects quantized weights back to full-precision representations, enabling fair and consistent evaluation of all White-box approaches on MDGBench.

4.2 RQ4: Effectiveness Evaluation

RQ4 Setup for Baseline Selection. Our baselines are designed to evaluate the framework across two core tasks; i.e., *Clustering Evaluation* and *Dependency Identification Evaluation*. For clustering evaluation, we first select TensorGuard and MoTHer, two of the few existing works that support cluster-level model grouping. Since pair-wise dependency identification approaches can be adapted for clustering by coupling with clustering algorithms, we further extend our baselines using state-of-the-art pair-wise approaches. We consider two scenarios; i.e., (1) the number of model families is known, using K-means; and (2) using HDBSCAN to automatically determine cluster counts. Following the pair-wise SoK [33], we selected seven representative approaches: three White-box approaches (i.e., REEF, PDF, and HuRef) which require access to internal model parameters or gradients, and four black-box approaches (i.e., SEF, LLMMap, MET, and TRAP) which rely exclusively on model outputs or API responses. As black-box approaches are originally designed for LLMs and require prompts, making them inapplicable to vision, audio and multimodal models.

For dependency identification evaluation, we adopt MoTHer as our primary baseline, which is the only existing work that addresses model dependency tree construction. Furthermore, Horwitz et al. [36] suggest that for 99.5% of models, the parent model is released earlier than its derivatives. Inspired by this finding, we implement ChronChain, which utilizes the release timestamps to reconstruct dependency relationships under the assumption of linear temporal evolution. We consider two scenarios; i.e., (1) neither cluster membership nor the number of clusters is known (i.e.,

Table 3. Clustering Evaluation Results on MDGBench

Approach	Type	$\|\mathcal{M}\|$	K-means	HDBSCAN	Approach	Type	$\|\mathcal{M}\|$	K-means	HDBSCAN
SEF	Black	121	0.12	0.10	TensorGuard	White	137	0.31	0.36
LLMMap	Black	121	0.10	0.12	MoTHer	White	137	0.50	0.32
MET	Black	121	0.04	0.06	Reef	White	137	0.13	0.29
TRAP	Black	121	0.01	0.02	PDF	White	137	0.62	0.52
HuRef	White	137	0.27	0.69	**TensorLock**	White	137	**0.91**	**0.96**

Table 4. Dependency Identification Evaluation Results on MDGBench ($|\mathcal{M}| = 137$)

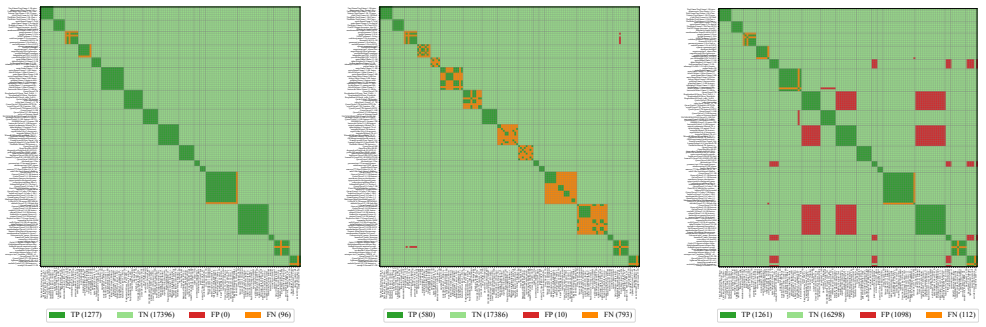
Metric	MoTHer	MoTHer$^+$	ChronChain	ChronChain$^+$	TensorLock	TensorLock $^+$
DPre	0.04	0.29	0.13	0.29	0.81	**0.83**
DRec	0.04	0.26	0.13	0.27	0.75	**0.81**
*DF*1	0.04	0.28	0.13	0.28	0.78	**0.82**
RchRec	0.05	0.47	0.97	0.97	0.77	**0.81**
RchPre	0.22	0.47	0.02	0.31	0.77	**0.79**

TensorLock, MoTHer and ChronChain) and (2) the ground-truth clusters are known, identifying dependencies within each cluster (i.e., TensorLock $^+$, MoTHer$^+$ and ChronChain$^+$).

Overall Results. Table 3 and Table 4 present the overall effectiveness of TensorLock and baseline approaches on MDGBench. For clustering, TensorLock achieves the highest ARI of 0.91 with K-means and 0.96 with HDBSCAN, substantially outperforming all baselines. Black-box approaches generally underperform White-box approaches, with the highest ARI of only 0.12 achieved by SEF, indicating that input and output-based fingerprinting is insufficient for accurate model clustering. Among White-box approaches, directly applying clustering algorithms on pair-wise features yield suboptimal results. PDF achieves at most 0.62 (K-means), while HuRef reaches 0.69 (HDBSCAN), indicating that raw pair-wise similarity fails to capture complex dependency structures. In contrast, TensorLock employs a connectivity-based distance formulation (Sec. 3.2), achieving consistently high ARI with both K-means (0.91) and HDBSCAN (0.96), outperforming the best baseline by 47% and 39%, respectively.

For dependency identification, TensorLock $^+$ achieves the highest precision of 0.83 and recall of 0.81, with an increase in precision by 0.54 (186%) and in recall by 0.55 (212%) compared to MoTHer$^+$. The corresponding *DF*1 score of 0.82 represents an improvement of 0.54 (193%) over MoTHer$^+$'s *DF*1 of 0.28. Even without prior knowledge of cluster membership, TensorLock achieves a *DPre* of 0.81 and *DRec* of 0.75, demonstrating robust end-to-end dependency identification capability. In contrast, MoTHer achieves only 0.04 in both *DPre* and *DRec*, highlighting its sensitivity to clustering errors. ChronChain achieves a low precision of 0.13 with comparable recall of 0.13, as its linear temporal assumption introduces substantial false positives by incorrectly linking unrelated models that happen to be released sequentially.

Notably, although ChronChain exhibits highly *RchRec*, its *RchPre* is only 0.02, meaning that 98% of its identified reachable dependencies are actually unreachable. This indicates that its seemingly high *RchRec* is driven largely by over-approximated reachability rather than meaningful dependency tracing. We argue that structural accuracy is a prerequisite for meaningful reachability analysis, since otherwise high reachability recall merely amplifies false positives. This issue arises at both the cross-family and in-family levels. For cross-family false reachability, under ChronChain's temporal chain, `openai-community/gpt2` correctly reaches its 6 ground-truth downstream models, but simultaneously marks 130 unrelated models as reachable solely based on release dates. In a risk propagation scenario, such a graph would flag nearly the entire benchmark as potentially compromised, making security triage infeasible. Even when restricted to ground-truth clusters, false reachability can still cascade within a family. For example, `Qwen/Qwen2.5-0.5B-GGUF` is a



(a) TensorLock (HDBSCAN) (b) PDF (HDBSCAN) (c) PDF (K-means)

Fig. 2. Pair-wise Visualized Confusion Matrix of TensorLock and State-of-the-Art Approaches

leaf node in the ground truth, with an empty reachable set. However, ChronChain+ incorrectly assigns its parent as `Qwen/Qwen2.5-0.5B-Instruct` instead of the true parent `Qwen/Qwen2.5-0.5B`, causing it to falsely inherit 13 downstream models. Similarly, timestamp-based ordering can also miss true dependencies: `Qwen/Qwen2-VL-2B` (parent, 2024-09-05) has a later timestamp than its child `Qwen2-VL-2B-Instruct` (2024-08-28) due to a re-upload, causing the true dependency to be missed while unrelated models are incorrectly linked. In contrast, TensorLock achieves a more meaningful trade-off between *RchRec* and *RchPre*, as its reachable paths are grounded in identified dependency edges rather than temporal heuristics alone. As a result, TensorLock not only preserves substantially better traceability, but also avoids the excessive false reachability that makes temporal baselines unreliable for practical risk propagation analysis.

In-Depth Analysis of Effectiveness w.r.t Clustering. To further investigate the clustering effectiveness, we decompose the ARI metric into pair-wise confusion matrices, as shown in Fig. 2. For any pair of models (m_i, m_j), we examine whether they are correctly grouped (TP/TN) or incorrectly associated (FP/FN). We select PDF as the representative baseline since it achieves the competitive performance on both HDBSCAN and K-means among all baselines. PDF achieves a low FP of 10 but suffers from a high FN of 793 with K-means, indicating that many models from the same clusters are scattered due to the fixed cluster count constraint. When switching to HDBSCAN, the FN drops to 112, but the FP surges to 1,098, revealing that the PDF's fingerprinting struggles to maintain clear cluster boundaries in both K-means and HDBSCAN without considering the dependency topology. In contrast, TensorLock achieves 0 FP, and only 96 FN. The absence of FP indicates that TensorLock never incorrectly groups unrelated models into the same cluster. This precision stems from the combination of MSA-based distance with connectivity-based clustering. Nevertheless, the 96 FN pairs originate from six models being isolated into singleton clusters, including three quantized, two fine-tuned, and one merged model. The majority of these cases stem from quantization. This is because, while quantization preserves the statistical distribution of weights, the aggressive discretization that decrease the connectivity to its model cluster.

In-Depth Analysis of Effectiveness w.r.t Dependency Identification. To further investigate the dependency identification effectiveness, we also decompose the precision and recall metrics into FP and FN edges, as illustrated in Fig. 3. MoTHer exhibits high error rates with 120 FP edges and 126 FN edges out of 131 ground-truth edges. 29 FN and 35 FP edges originate from the clustering. 97 FN and 85 FP edges originate from the dependency identification component. In contrast, TensorLock achieves significantly lower errors with only 23 FP edges and 33 FN edges. We trace the sources of these errors across the three-phase pipeline. 7 FN and 0 FP edges originate from the 6 isolated models. The data-free phase contributes 12 FP and 10 FN edges, while the data-driven *Fine-Tuning* phase causes 11 FP and 16 FN edges.

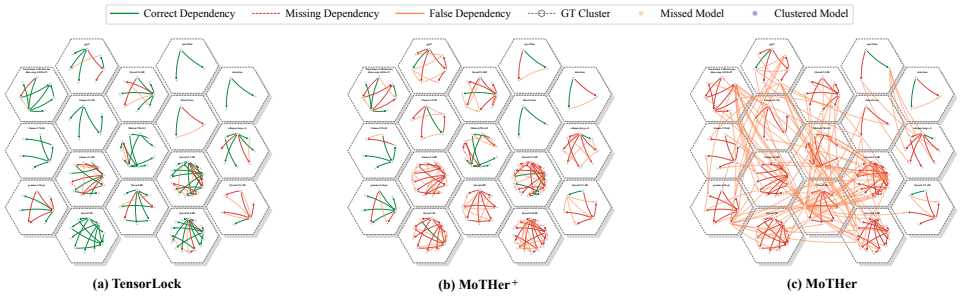


Fig. 3. Dependency Recovering Result of TensorLock and State-of-the-Art Approaches

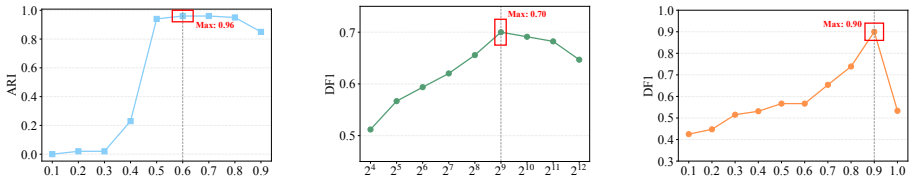
Table 5. Ablation Study Results of TensorLock on MDGBench

	Clustering	Dependency Identification													
		DPre				*DRec*				*DF*1				*RchRec*	*RchPre*
Approach	ARI	$\mathcal{E}_{QT}$	$\mathcal{E}_{MG}$	$\mathcal{E}_{FT}$	$\mathcal{E}$	$\mathcal{E}_{QT}$	$\mathcal{E}_{MG}$	$\mathcal{E}_{FT}$	$\mathcal{E}$	$\mathcal{E}_{QT}$	$\mathcal{E}_{MG}$	$\mathcal{E}_{FT}$	$\mathcal{E}$		
w/ $\mathcal{W}_{FFN}$ (Δ)	**-0.21**	**-0.05**	**±0**	-0.07	-0.07	**-0.07**	**-0.37**	**-0.18**	-0.18	**-0.06**	**-0.27**	-0.13	-0.13	-0.27	**-0.26**
w/ $\mathcal{W}_{MSA}$&$\mathcal{W}_{FFN}$ (Δ)	-0.05	-0.05	±0	-0.04	-0.04	-0.07	-0.14	-0.18	-0.14	-0.06	-0.09	-0.11	-0.10	-0.18	**-0.26**
w/o Ku. (Δ)	±0	±0	±0	**-0.12**	-0.07	±0	±0	-0.16	-0.09	±0	±0	**-0.14**	-0.08	-0.03	-0.07
w/o D.F. (Δ)	±0	N/A	N/A	N/A	**-0.17**	N/A	N/A	N/A	**-0.19**	N/A	N/A	N/A	**-0.18**	**-0.28**	-0.18
TensorLock	0.96	0.75	1.00	0.77	0.81	0.66	0.82	0.79	0.75	0.70	0.90	0.78	0.78	0.77	0.77

4.3 RQ5: Ablation Study

RQ5 Setup for Ablation Component. We conduct the ablation study of TensorLock. We design four ablation variants; i.e., (1) w/ $\mathcal{W}_{FFN}$, which uses only FFN weights for clustering instead of MSA weights; (2) w/ $\mathcal{W}_{MSA}$&$\mathcal{W}_{FFN}$, which uses both MSA and FFN weights for clustering; (3) w/o Ku., which replaces Spectral Entropy with Kurtosis for directionality estimation in Fine-Tuning dependency identification; and (4) w/o D.F., which removes the Data-Free dependency identification module for *Quantization* and *Merging*. The full TensorLock serves as the reference baseline.

Ablation Study Results. Table 5 presents the ablation study results, where values indicate the relative performance change compared to the full TensorLock. The "±0" denotes metrics that remain unchanged for the specific ablation variant, while "N/A" indicates metrics that cannot be evaluated as the corresponding module is removed. Δ w/ $\mathcal{W}_{FFN}$ exhibits the most severe degradation with ARI dropping by 0.21, confirming our design rationale that FFN layers undergo drastic weight shifts during fine-tuning to encode task-specific factual knowledge, thereby disrupting the connectivity patterns essential for accurate clustering. This cascading effect propagates to downstream dependency identification, resulting in consistent performance drops across all metrics (*DF*1: -0.13, *RchRec*: -0.27, *RchPre*: -0.26). When incorporating both MSA and FFN weights (Δ w/ $\mathcal{W}_{MSA}$&$\mathcal{W}_{FFN}$), the degradation is mitigated (ARI: -0.05) since MSA weights still contribute stable fingerprints. However, the inclusion of FFN weights introduces noise that inflates inter-model distances within the same cluster, leading to moderate performance decline across dependency identification metrics (*DF*1: -0.10). The results of Δ w/ Ku. show notable degradation in Fine-Tuning dependency identification, with $\mathcal{E}_{FT}$ recall dropping by 0.16 and F1 by 0.14. This confirms our hypothesis that Kurtosis-based detection becomes unstable for models with large parameter scales, as the massive parameter space statistically drowns out subtle outlier shifts caused by fine-tuning. Δ w/o D.F. removes the dedicated modules for *Quantization* and *Merging* dependency identification, forcing all dependencies to be resolved by the Fine-Tuning module via MDST. This variant exhibits the most significant overall degradation, with *DF*1 dropping by 0.18 and *RchRec* plummeting by 0.28.



(a) τ_{conn} (b) $|B|$ (c) τ_{mg}

Fig. 4. Sensitivity Analysis Results

Table 6. Partial Cases of Our Recovered Dependencies ("Conf." indicates confirmed (✓), or pending (○))

Dependency (Parent → Child)	Downloads	Types	Conf.
google/flan-t5-base → Babelscape/t5-base-summarization-claim	3,564K	*Fine-Tuning*	✓
meta-llama/Meta-Llama-3-8B → NousResearch/Meta-Llama-3-8B-Instruct	1,625K	*Fine-Tuning*	✓
mlabonne/Daredevil-8B-abliterated → mlabonne/NeuralDaredevil-8B-abliterated	316K	*Fine-Tuning*	✓
meta-llama/Meta-Llama-3-8B → winninghealth/WiNGPT2-Llama-3-8B-Chat	279K	*Fine-Tuning*	✓
meta-llama/Llama-2-7b-hf → lmsys/vicuna-7b-v1.5	11,481K	*Fine-Tuning*	○
NousResearch/Nous-Hermes-Llama2-13b, etc. → Undi95/ReMM-SLERP-L2-13B	54K	*Merging*	✓
TinyLlama/TinyLlama-1.1B-intermediate → TinyLlama/TinyLlama-1.1B-Chat-v1.0	30,812K	*Fine-Tuning*	○
mistralai/Mistral-7B-v0.1 → Q-bert/Optimus-7B	37K	*Fine-Tuning*	✓
google/gemma-7b-it → lemon-mint/gemma-ko-7b-it-v0.40	23K	*Fine-Tuning*	✓
google-t5/t5-small → ml6team/keyphrase-generation-t5-small-inspec	12K	*Fine-Tuning*	✓
mistralai/Mistral-7B-Instruct-v0.2, etc. → MaziyarPanahi/finetuned-mistral-7b-Mistral-7B-Instruct-v0.2-slerp	2K	*Merging*	✓
meta-llama/Llama-3.2-3B → NousResearch/Hermes-3-Llama-3.2-3B	872K	*Fine-Tuning*	✓
Qwen/Qwen2-1.5B → Qwen/Qwen2-1.5B-Instruct	7,350K	*Fine-Tuning*	○
stabilityai/stable-code-3b → stabilityai/stable-code-instruct-3b	63K	*Fine-Tuning*	✓
meta-llama/Meta-Llama-3-8B-Instruct → QuantFactory/Meta-Llama-3-8B-Instruct-GGUF	762K	*Quantization*	✓
google/flan-t5-base → lizhuang144/flan-t5-base-VG-factual-sg	570K	*Fine-Tuning*	✓
TinyLlama/TinyLlama-v1.1 → unsloth/tinyllama-bnb-4bit	278K	*Quantization*	○
mlabonne/Daredevil-8B → mlabonne/Daredevil-8B-abliterated	176K	*Fine-Tuning*	✓
SUSTech/SUS-Chat-34B, etc. → cloudyu/Mixtral_34Bx2_MoE_60B	126K	*Merging*	✓
Qwen/Qwen2.5-0.5B-Instruct → taobao-mnn/Qwen2.5-0.5B-Instruct-MNN	119K	*Fine-Tuning*	✓

4.4 RQ6: Sensitivity Analysis

RQ6 Setup for Sensitivity Dimensions. Three parameters are configurable in TensorLock; i.e., the similarity threshold τ_{conn} for clustering, the $|B|$ for Quantization dependency identification, and the provenance threshold τ_{mg} for *Merging* dependency identification. We evaluate the sensitivity of TensorLock to these parameters by varying one parameter while keeping the others fixed.

Sensitivity Results. Fig. 4 illustrates the results of our sensitivity analysis. TensorLock achieves the optimal performance when $\tau_{conn} = 0.6$, $|B| = 2^9(512)$, and $\tau_{mg} = 0.9$.

4.5 RQ7: Efficiency Evaluation

RQ7 Setup for Time Complexity. We measured the average time taken to cluster and identify dependencies of TensorGuard, MoTHer, and TensorLock in MDGBench.

Efficiency Results. TensorGuard exhibits the highest computational overhead (25,350s) due to its reliance on reverse gradient computation, which requires backpropagation through each model to extract gradient-based fingerprints. In contrast, both TensorLock and MoTHer operate on static model weights, enabling direct weight comparison without additional computation. For clustering, TensorLock achieves slightly better efficiency than MoTHer (963s vs. 1,052s). For dependency identification, TensorLock incurs additional overhead (1,659s vs. 1,041s) due to its type-aware dependency identification, which enables more accurate dependency reasoning.

4.6 RQ8: Usefulness Evaluation

RQ8 Setup for Supposedly Isolated Model Selection and Sampling. As revealed in our empirical study, approximately 20% of models lack any documentation regarding their dependency. These models appear isolated in the supply chain, making it nearly infeasible even for human experts to determine their actual dependencies. While MDGBench primarily includes models with verifiable dependency relationships. To assess the real-world usefulness of our framework in

uncovering hidden dependencies for such supposedly isolated models, we extended our analysis to a large-scale supposedly isolated model pool. We included the 67 isolated models in Sec. 2, and draw an additional 369 models from the broader Hugging Face ecosystem following the sampling methodology described in Sec. 2, resulting in an initial pool of 436 models. We then further excluded non-Transformer-based models and resulted in a final set of 289 supposedly isolated models.

Dependency Identification Results of TensorLock. We applied our framework to these 289 sampled models with the 17 established clusters in MDGBench. TensorLock identified that 189 (65.4%) of these supposedly isolated models actually have dependencies with existing model clusters. We then reached out to the respective model authors and submitted Pull Requests to update their model cards with the identified dependency information. To date, 42 developers have responded, acknowledging our findings. Notably, for 38 identified dependencies, both the parent and child models rank within the top 1,000 most-downloaded on Hugging Face, demonstrating the community-wise contribution of TensorLock. Table 6 presents partial cases identified by TensorLock, including derivatives of `GPT-2`, `Llama`, `T5`, `Mistral`, `Gemma`, and `Phi` series, underscoring the practical usefulness of TensorLock in enhancing model dependency traceability.

5 Discussion

Threats to Validity. First, our empirical study focused on the top downloaded models from Hugging Face, which may introduce popularity bias. However, high-download models constitute the backbone of the model supply chain, as they are more likely to be reused and thus have greater downstream impact when risks emerge. Second, MDGBench contains 137 models, which is fewer compared to MoTHer (~500). However, MoTHer's dataset consists of self-trained synthetic models whereas MDGBench is curated entirely from real-world models on Hugging Face. This real-world grounding ensures that our evaluation reflects the actual complexity of the model supply chain.

Limitations. First, TensorLock is specifically designed for Transformer-based models. This design choice is grounded in our empirical finding that Transformer-based models dominate the current model supply chain. Second, TensorLock covers three dependency types (i.e., *Fine-Tuning*, *Quantization*, and *Merging*). However, the framework is designed to be extensible; i.e., incorporating additional dependency types requires only identifying their characteristic weight-space fingerprinting and integrating corresponding detection modules. Third, our *Merging* dependency identification assumes that all parent models share a common pre-trained ancestor (Assumption 2). While this may not hold for cross-family merging scenarios, this constraint aligns with the technical prerequisites of current model merging techniques [49], which require parent models to have sufficiently similar weight distributions for meaningful interpolation or aggregation.

6 Related Works

Pair-Wise Model Fingerprinting. Current model fingerprinting approaches are primarily designed to verify dependencies between a pre-trained base model and a suspect model. These approaches can be broadly categorized into white-box approaches [3, 7, 35, 43, 50, 52, 53, 55–57] and black-box approaches [6, 16, 25, 30, 32, 34, 39, 45, 48]. White-box approaches can be categorized into three distinct types; i.e., static weight [50, 52, 53, 55, 57], forward-pass feature [3, 7, 35, 56], and backward-pass gradient approaches [43]. Static weight approaches analyze model parameters directly without requiring deployment, offering the lowest computational overhead among white-box approaches. Among them, MDIR [57] and Neural Lineage [52] apply full weights analysis across the fine-tuning trajectory. HuRef [53] and PDF [50] focus on weight subsets; i.e., HuRef extracts permutation-invariant directions from weight vectors, while PDF derives statistical signatures from attention weights. Forward-pass approaches (e.g., Reef [56] and DEEPJUDGE [7]) apply intermediate layer representations as their fingerprints. Compared to static weight, forward-pass approaches

require specific input data, which leads to higher latency. Backward-pass approaches (e.g., TensorGuard [43]) utilize gradient backpropagation paths to verify model distance. However, they incur substantial computational overhead while yielding limited accuracy in our evaluation. Black-box approaches typically extract unique model fingerprints through adversarial prompts [16, 45], statistical pattern generation (e.g., RAFP [39]), or Chain-of-Thought (CoT) analysis [32]. While non-invasive, black-box approaches are often sensitive to decoding strategies and generally offer lower verification confidence compared to white-box approaches. Despite their diversity, pair-wise approaches rely on a pre-specified base model to compute similarity against suspect models, which makes them difficult to scale in complex ecosystems.

Cluster-Wise and Tree-Wise Model Fingerprinting. While pair-wise approaches verify individual dependency, cluster-wise approaches aim to cluster large-scale models into groups. TensorGuard [43] utilizes gradient-based fingerprints to enable cluster-wise grouping, which struggle with *open-ended model dependency topology*. Tree-wise approaches then further identify the model dependency within each identified cluster. MoTHer [18] addresses this by extracting the layer-wise distance and direction between models within a cluster but lack support for *mixed dependency types*.

Model Watermarking. Clearstamp [26] introduces a human-visible ownership proof via transposed model training to protect the intellectual property of LLM-based code generation APIs [28]. Adi et al. [1] and Zhang et al. [54] pioneered training models to produce predefined outputs upon specific triggers, while Xu et al. [44] use similar mechanisms to detect misuses of open-source LLMs. However, the embedding process is inherently intrusive and may lead to performance degradation on primary tasks. The verification process often requires a secret key or specific trigger held exclusively by the model owner, which restricts its applicability for independent third-party analysis.

7 Conclusion

In this paper, we present TensorLock, a model dependency recovering framework that constructs type-aware model dependency graphs. We then conduct comprehensive evaluation to demonstrate TensorLock's superior effectiveness and usefulness in recovering model dependencies.

8 Data Availability

TensorLock and MDGBench are available at our website [37].

Acknowledgments

This work was supported by the National Natural Science Foundation of China (Grant No. 62332005, 62372114 and 62402342).

References

[1] Yossi Adi, Carsten Baum, Moustapha Cisse, Benny Pinkas, and Joseph Keshet. 2018. Turning your weakness into a strength: Watermarking deep neural networks by backdooring. In *27th USENIX security symposium (USENIX Security 18)*. 1615–1631.

[2] Armen Aghajanyan, Sonal Gupta, and Luke Zettlemoyer. 2021. Intrinsic dimensionality explains the effectiveness of language model fine-tuning. In *Proceedings of the 59th annual meeting of the association for computational linguistics and the 11th international joint conference on natural language processing (volume 1: long papers)*. 7319–7328. doi:10.18653/v1/2021.acl-long.568

[3] Saeif Alhazbi, Ahmed Hussain, Gabriele Oligeri, and Panos Papadimitratos. 2025. Llms have rhythm: Fingerprinting large language models using inter-token times and network traffic analysis. *IEEE Open Journal of the Communications Society* (2025). doi:10.1109/OJCOMS.2025.3577016

[4] Kushal Raj Bhandari, Pin-Yu Chen, and Jianxi Gao. 2025. Forecasting Open-Weight AI Model Growth on HuggingFace. *arXiv preprint arXiv:2502.15987* (2025). doi:10.48550/arXiv.2502.15987

[5] Ricardo JGB Campello, Davoud Moulavi, and Jörg Sander. 2013. Density-based clustering based on hierarchical density estimates. In *Pacific-Asia conference on knowledge discovery and data mining*. Springer, 160–172. doi:10.1007/978-3-642-37456-2_14

[6] Xiaoyu Cao, Jinyuan Jia, and Neil Zhenqiang Gong. 2021. IPGuard: Protecting intellectual property of deep neural networks via fingerprinting the classification boundary. In *Proceedings of the 2021 ACM asia conference on computer and communications security*. 14–25. doi:10.1145/3433210.3437526

[7] Jialuo Chen, Jingyi Wang, Tinglan Peng, Youcheng Sun, Peng Cheng, Shouling Ji, Xingjun Ma, Bo Li, and Dawn Song. 2022. Copy, right? a testing framework for copyright protection of deep learning models. In *2022 IEEE symposium on security and privacy (SP)*. IEEE, 824–841. doi:10.1109/SP46214.2022.9833747

[8] Ziqian Chen, Zekai Chen, Susheng Wu, Bihuan Chen, Wenyan Song, Yiheng Huang, Zhuotong Zhou, Yiheng Cao, and Xin Peng. 2026. A First Look at Model Supply Chain: From the Risk Perspective. In *2026 IEEE/ACM 48th International Conference on Software Engineering (ICSE)*. IEEE. doi:10.1145/3744916.3773171

[9] Yoeng-Jin Chu. 1965. On the shortest arborescence of a directed graph. *Scientia Sinica* 14 (1965), 1396–1400. https://www.cs.cmu.edu/~15850/handouts/chu-liu_1965.pdf

[10] Tim Dettmers, Mike Lewis, Younes Belkada, and Luke Zettlemoyer. 2022. Gpt3. int8 (): 8-bit matrix multiplication for transformers at scale. *Advances in neural information processing systems* 35 (2022), 30318–30332. doi:10.52202/068431-2198

[11] Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. 2023. Qlora: Efficient finetuning of quantized llms. *Advances in neural information processing systems* 36 (2023), 10088–10115. doi:10.52202/075280-0441

[12] Kazuki Egashira, Robin Staab, Mark Vero, Jingxuan He, and Martin Vechev. 2025. Mind the Gap: A Practical Attack on GGUF Quantization. *arXiv preprint arXiv:2505.23786* (2025). doi:10.48550/arXiv.2505.23786

[13] Elias Frantar, Saleh Ashkboos, Torsten Hoefler, and Dan Alistarh. 2022. Gptq: Accurate post-training quantization for generative pre-trained transformers. *arXiv preprint arXiv:2210.17323* (2022). doi:10.48550/arXiv.2210.17323

[14] Mor Geva, Roei Schuster, Jonathan Berant, and Omer Levy. 2021. Transformer feed-forward layers are key-value memories. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*. 5484–5495. doi:10.18653/v1/2021.emnlp-main.446

[15] Kai Greshake, Sahar Abdelnabi, Shailesh Mishra, Christoph Endres, Thorsten Holz, and Mario Fritz. 2023. Not what you've signed up for: Compromising real-world llm-integrated applications with indirect prompt injection. In *Proceedings of the 16th ACM workshop on artificial intelligence and security*. 79–90. doi:10.1145/3605764.3623985

[16] Martin Gubri, Dennis Ulmer, Hwaran Lee, Sangdoo Yun, and Seong Joon Oh. 2024. Trap: Targeted random adversarial prompt honeypot for black-box identification. *arXiv preprint arXiv:2402.12991* (2024). doi:10.48550/arXiv.2402.12991

[17] Eliahu Horwitz, Nitzan Kurer, Jonathan Kahana, Liel Amar, and Yedid Hoshen. 2025. We Should Chart an Atlas of All the World's Models. *arXiv preprint arXiv:2503.10633* (2025). doi:10.48550/arXiv.2503.10633

[18] Eliahu Horwitz, Asaf Shul, and Yedid Hoshen. 2024. Unsupervised model tree heritage recovery. *arXiv preprint arXiv:2405.18432* (2024). doi:10.48550/arXiv.2405.18432

[19] Qiang Hu, Xiaofei Xie, Sen Chen, Lili Quan, and Lei Ma. 2025. Large language model supply chain: Open problems from the security perspective. In *Proceedings of the 34th ACM SIGSOFT International Symposium on Software Testing and Analysis*. 169–173. doi:10.1145/3713081.3731747

[20] Lawrence Hubert and Phipps Arabie. 1985. Comparing partitions. *Journal of classification* 2, 1 (1985), 193–218. doi:10.1007/BF01908075

[21] Gabriel Ilharco, Marco Tulio Ribeiro, Mitchell Wortsman, Suchin Gururangan, Ludwig Schmidt, Hannaneh Hajishirzi, and Ali Farhadi. 2023. Editing Models with Task Arithmetic. In *International Conference on Learning Representations*.

[22] Dong-Hwan Jang, Sangdoo Yun, and Dongyoon Han. 2024. Model stock: All we need is just a few fine-tuned models. In *European Conference on Computer Vision*. Springer, 207–223. doi:10.1007/978-3-031-72784-9_12

[23] Jinyuan Jia, Yupei Liu, and Neil Zhenqiang Gong. 2022. Badencoder: Backdoor attacks to pre-trained encoders in self-supervised learning. In *2022 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2043–2059. doi:10.1109/SP46214.2022.9833644

[24] Wenxin Jiang, Nicholas Synovic, Matt Hyatt, Taylor R Schorlemmer, Rohan Sethi, Yung-Hsiang Lu, George K Thiruvathukal, and James C Davis. 2023. An empirical study of pre-trained model reuse in the hugging face deep learning model registry. In *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*. IEEE, 2463–2475. doi:10.1109/ICSE48619.2023.00206

[25] Heng Jin, Chaoyu Zhang, Shanghao Shi, Wenjing Lou, and Y Thomas Hou. 2024. Proflingo: A fingerprinting-based intellectual property protection scheme for large language models. In *2024 IEEE Conference on Communications and Network Security (CNS)*. IEEE, 1–9. doi:10.1109/CNS62487.2024.10735575

[26] Torsten Krauß, Jasper Stang, and Alexandra Dmitrienko. 2024. {ClearStamp}: A {Human-Visible} and Robust {Model-Ownership} Proof based on Transposed Model Training. In *33rd USENIX Security Symposium (USENIX Security 24)*. 5269–5286.

[27] Benjamin Laufer, Hamidah Oderinwale, and Jon Kleinberg. 2025. Anatomy of a machine learning ecosystem: 2 million models on hugging face. *arXiv preprint arXiv:2508.06811* (2025). doi:10.48550/arXiv.2508.06811

[28] Zongjie Li, Chaozheng Wang, Shuai Wang, and Cuiyun Gao. 2023. Protecting intellectual property of large language model-based code generation apis via watermarks. In *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security*. 2336–2350. doi:10.1145/3576915.3623120

[29] Ji Lin, Jiaming Tang, Haotian Tang, Shang Yang, Guangxuan Xiao, and Song Han. 2025. AWQ: Activation-aware weight quantization for on-device LLM compression and acceleration. *GetMobile: Mobile Computing and Communications* 28, 4 (2025), 12–17. doi:10.1145/3714983.3714987

[30] Nils Lukas, Yuxuan Zhang, and Florian Kerschbaum. 2019. Deep neural network fingerprinting by conferrable adversarial examples. *arXiv preprint arXiv:1912.00888* (2019). doi:10.48550/arXiv.1912.00888

[31] Kevin Meng, David Bau, Alex Andonian, and Yonatan Belinkov. 2022. Locating and editing factual associations in gpt. *Advances in neural information processing systems* 35 (2022), 17359–17372. doi:10.52202/068431-1262

[32] Zhenzhen Ren, GuoBiao Li, Sheng Li, Zhenxing Qian, and Xinpeng Zhang. 2025. CoTSRF: Utilize Chain of Thought as Stealthy and Robust Fingerprint of Large Language Models. *arXiv preprint arXiv:2505.16785* (2025). doi:10.48550/arXiv.2505.16785

[33] Shuo Shao, Yiming Li, Yu He, Hongwei Yao, Wenyuan Yang, Dacheng Tao, and Zhan Qin. 2025. Sok: Large language model copyright auditing via fingerprinting. *arXiv preprint arXiv:2508.19843* (2025). doi:10.48550/arXiv.2508.19843

[34] Shuo Shao, Haozhe Zhu, Yiming Li, Hongwei Yao, Tianwei Zhang, and Zhan Qin. 2025. Fit-print: Towards false-claim-resistant model ownership verification via targeted fingerprint. *arXiv preprint arXiv:2501.15509* (2025). doi:10.48550/arXiv.2501.15509

[35] Hae Jin Song and Laurent Itti. 2025. Riemannian-Geometric Fingerprints of Generative Models. *arXiv preprint arXiv:2506.22802* (2025). doi:10.48550/arXiv.2506.22802

[36] Trevor Stalnaker, Nathan Wintersgill, Oscar Chaparro, Laura A Heymann, Massimiliano Di Penta, Daniel M German, and Denys Poshyvanyk. 2025. The ML supply chain in the era of software 2.0: Lessons learned from hugging face. *arXiv preprint arXiv:2502.04484* (2025). doi:10.48550/arXiv.2502.04484

[37] TensorLock and MDGBench. 2026. *TensorLoc and MDGBench.* Retrieved Jan 30, 2026 from https://github.com/TensorLock/TensorLock

[38] Robert Tibshirani. 1996. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society Series B: Statistical Methodology* 58, 1 (1996), 267–288. doi:10.1111/j.2517-6161.1996.tb02080.x

[39] Yun-Yun Tsai, Jia Hao Liang, Chuan Guo, Junfeng Yang, and Laurens van der Maaten. 2025. RAFP: Identifying LLM Lineages via Rare-Region Fingerprints. *arXiv preprint arXiv:2505.12682* (2025). doi:10.48550/arXiv.2505.12682

[40] Elena Voita, David Talbot, Fedor Moiseev, Rico Sennrich, and Ivan Titov. 2019. Analyzing multi-head self-attention: Specialized heads do the heavy lifting, the rest can be pruned. *arXiv preprint arXiv:1905.09418* (2019). doi:10.48550/arXiv.1905.09418

[41] Alexander Wan, Eric Wallace, Sheng Shen, and Dan Klein. 2023. Poisoning language models during instruction tuning. In *International Conference on Machine Learning*. PMLR, 35413–35425.

[42] Shenao Wang, Yanjie Zhao, Xinyi Hou, and Haoyu Wang. 2025. Large language model supply chain: A research agenda. *ACM Transactions on Software Engineering and Methodology* 34, 5 (2025), 1–46. doi:10.1145/3708531

[43] Zehao Wu, Yanjie Zhao, and Haoyu Wang. 2025. Gradient-Based Model Fingerprinting for LLM Similarity Detection and Family Classification. *arXiv preprint arXiv:2506.01631* (2025). doi:10.48550/arXiv.2506.01631

[44] Yijie Xu, Aiwei Liu, Xuming Hu, Lijie Wen, and Hui Xiong. 2025. Mark your llm: Detecting the misuse of open-source large language models via watermarking. *arXiv preprint arXiv:2503.04636* (2025). doi:10.48550/arXiv.2503.04636

[45] Zhenhua Xu, Zhebo Wang, Maike Li, Wenpeng Xing, Chunqiang Hu, Chen Zhi, and Meng Han. 2025. RAP-SM: Robust Adversarial Prompt via Shadow Models for Copyright Verification of Large Language Models. *arXiv preprint arXiv:2505.06304* (2025). doi:10.48550/arXiv.2505.06304

[46] Prateek Yadav, Derek Tam, Leshem Choshen, Colin A Raffel, and Mohit Bansal. 2023. Ties-merging: Resolving interference when merging models. *Advances in Neural Information Processing Systems* 36 (2023), 7093–7115. doi:10.52202/075280-0310

[47] Jun Yan, Vikas Yadav, Shiyang Li, Lichang Chen, Zheng Tang, Hai Wang, Vijay Srinivasan, Xiang Ren, and Hongxia Jin. 2024. Backdooring instruction-tuned large language models with virtual prompt injection. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*. 6065–6086. doi:10.18653/v1/2024.naacl-long.337

[48] Yuliang Yan, Haochun Tang, Shuo Yan, and Enyan Dai. 2025. DuFFin: A Dual-Level Fingerprinting Framework for LLMs IP Protection. *arXiv preprint arXiv:2505.16530* (2025). doi:10.48550/arXiv.2505.16530

[49] Enneng Yang, Li Shen, Guibing Guo, Xingwei Wang, Xiaochun Cao, Jie Zhang, and Dacheng Tao. 2026. Model merging in llms, mllms, and beyond: Methods, theories, applications, and opportunities. *Comput. Surveys* 58, 8 (2026), 1–41. doi:10.1145/3787849

[50] Do-hyeon Yoon, Minsoo Chun, Thomas Allen, Hans Müller, Min Wang, and Rajesh Sharma. 2025. Intrinsic Fingerprint of LLMs: Continue Training is NOT All You Need to Steal A Model! *arXiv preprint arXiv:2507.03014* (2025). doi:10.48550/arXiv.2507.03014

[51] Le Yu, Bowen Yu, Haiyang Yu, Fei Huang, and Yongbin Li. 2024. Language models are super mario: Absorbing abilities from homologous models as a free lunch. In *International Conference on Machine Learning*.

[52] Runpeng Yu and Xinchao Wang. 2024. Neural lineage. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 4797–4807. doi:10.1109/CVPR52733.2024.00459

[53] Boyi Zeng, Lizheng Wang, Yuncong Hu, Yi Xu, Chenghu Zhou, Xinbing Wang, Yu Yu, and Zhouhan Lin. 2024. Huref: Human-readable fingerprint for large language models. *Advances in Neural Information Processing Systems* 37 (2024), 126332–126362. doi:10.52202/079017-4013

[54] Jialong Zhang, Zhongshu Gu, Jiyong Jang, Hui Wu, Marc Ph Stoecklin, Heqing Huang, and Ian Molloy. 2018. Protecting intellectual property of deep neural networks with watermarking. In *Proceedings of the 2018 on Asia conference on computer and communications security*. 159–172. doi:10.1145/3196494.3196550

[55] Jie Zhang, Jiayuan Li, Haiqiang Fei, Lun Li, and Hongsong Zhu. 2024. EasyDetector: Using Linear Probe to Detect the Provenance of Large Language Models. In *2024 IEEE 23rd International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom)*. IEEE, 2410–2417. doi:10.1109/TrustCom63139.2024.00333

[56] Jie Zhang, Dongrui Liu, Chen Qian, Linfeng Zhang, Yong Liu, Yu Qiao, and Jing Shao. 2024. Reef: Representation encoding fingerprints for large language models. *arXiv preprint arXiv:2410.14273* (2024). doi:10.48550/arXiv.2410.14273

[57] Ruichong Zhang and Daniel Goldstein. 2025. Matrix-Driven Identification and Reconstruction of LLM Weight Homology. *arXiv preprint arXiv:2508.06309* (2025). doi:10.48550/arXiv.2508.06309

Received 2026-01-30; accepted 2026-04-16