PoLO: Proof-of-Learning and Proof-of-Ownership at Once with Chained Watermarking

Haiyu Deng[1], Yanna Jiang[1], Guangsheng Yu[1], Qin Wang[1,2], Xu Wang[1], Baihe Ma[1], Wei Ni[1,2], Ren Ping Liu[1]
[1] *University of Technology Sydney* | [2] *CSIRO Data61, Australia*

ABSTRACT

Machine learning models are increasingly shared and outsourced, raising requirements of verifying training effort (Proof-of-Learning, PoL) to ensure claimed performance and establishing ownership (Proof-of-Ownership, PoO) for transactions. When models are trained by untrusted parties, PoL and PoO must be enforced *together* to enable protection, attribution, and compensation. However, existing studies typically address them separately, which not only weakens protection against forgery and privacy breaches but also leads to high verification overhead.

We propose PoLO, a unified framework that simultaneously achieves PoL and PoO using *chained watermarks*. PoLO splits the training process into fine-grained training shards and embeds a dedicated watermark in each shard. Each watermark is generated using the hash of the preceding shard, certifying the training process of the preceding shard. The chained structure makes it computationally difficult to forge any individual part of the whole training process. The complete set of watermarks serves as the PoL, while the final watermark provides the PoO. PoLO offers more efficient and privacy-preserving verification compared to the vanilla PoL solutions that rely on gradient-based trajectory tracing and inadvertently expose training data during verification, while maintaining the same level of ownership assurance of watermark-based PoO schemes. Our evaluation shows that PoLO achieves **99%** watermark detection accuracy for ownership verification, while preserving data privacy and cutting verification costs to just **1.5–10%** of traditional methods. Forging PoLO demands **1.1–4×** more resources than honest proof generation, with the original proof retaining over **90%** detection accuracy even after attacks.

1 INTRODUCTION

As Machine Learning (ML) evolves, models are increasingly developed across distributed settings, exchanged between parties, and outsourced to third-party providers [1]. This introduces a critical need for verifying both the legitimacy of training efforts and rightful model ownership. For instance, in incentive-driven distributed learning [2, 3], participants must prove that models are properly trained to receive fair rewards. In ML marketplaces [4], buyers need confidence that models are genuinely trained and transferable without dispute. Similarly, in outsourced training [5, 6], organizations must ensure that externally developed models are both authentic and securely attributed.

The shift introduces a fundamental challenge:

How do we verify a model's legitimacy in ML ecosystems?

Legitimacy in this context involves two facets: verifying training effort and establishing rightful ownership [7]. This requires two complementary mechanisms. Proof-of-Learning (PoL) ensures the claimed computational effort was genuinely invested, deterring fraudulent claims. Proof-of-Ownership (PoO), often implemented via watermarking, embeds verifiable ownership information to create a tamper-resistant link to the rightful owner. Without PoL, training claims are unverifiable; without PoO, ownership can be misused, transferred, or stolen without recourse.

Figure 1: Why at once? PoL verifies training effort but lacks ownership tracking, while PoO ensures ownership but fails to justify training efforts. Separating PoL and PoO creates attribution risks and ownership conflicts.

In this paper, we stress that *PoL and PoO are intertwined and must be jointly addressed to achieve a complete proof of legitimacy* (cf. Fig.1). Existing solutions have long viewed PoL and PoO as separate verification mechanisms, leading to PoL becoming the weakest link throughout the process:

- *Heightened vulnerability*, since existing PoL methods not only require exposing intermediate training states and training data [8–10], compromising privacy, but also remain prone to forgery, where attackers can fabricate plausible but fraudulent training trajectories to deceive verification [9, 10];
- *High verification costs*, as existing PoL schemes demand re-training at each recorded snapshot [8–10], imposing substantial computational overhead and making large-scale verification impractical.

Simply stacking them, such as combining a conventional PoL method with a standalone watermark, fails to provide a unified, secure, and privacy-preserving proof of both training effort and ownership.

Our goal is to integrate PoL and PoO into a unified framework by validating training efforts with persistent ownership at once. This presents a series of technical challenges with corresponding research questions (RQs):

- **RQ1:** *How can a PoO method be leveraged to achieve PoL?* Existing PoO methods rely on a single mature watermark embedded in the final model, offering no protection during earlier training stages and leaving intermediate versions unverified.
- **RQ2:** *How can PoO-based PoL remain resilient to removal and forgery?* Existing watermarking schemes are vulnerable to be

forged, so a robust design must ensure verifiable, tamper-resistant ownership throughout training and in the final model.

To address these RQs, we propose a novel design, PoLO that embeds *chained watermarks* throughout the training process. Each watermark is deterministically derived using a hash function over partial model weights and auxiliary parameters from the previous training shard. This chaining securely links training phases and embeds ownership information at every stage. Unlike static watermarking, which tags only the final model, our method accumulates verifiable ownership across the entire training trajectory. By replacing gradient-based tracking with chained watermarks, PoLO ensures training effort and ownership verification while enhancing tamper resistance, preserving data privacy, and eliminating redundant model recomputation.

We achieve it in a stepwise manner:

- *We formalize the concept of Proof-of-Anything (PoX)* (§2–§3) as a unified framework for analyzing PoL and PoO. Inspired by Proof-of-Work (PoW), PoX provides a structured basis for systematically evaluating PoL methods in design, efficiency, and security. Through this lens, we identify key limitations in existing PoL approaches, including inefficient proof generation, high verification costs, fragmented security, and privacy risks. Our findings highlight the need for an integrated solution that verifies both training effort (i.e., PoL) and ownership (i.e., PoO) efficiently and securely, motivating the design of PoLO.
- *We introduce PoLO* (§4), a novel method that unifies PoL and PoO through chained watermarks. The model trainer embeds watermarks throughout training, updating them iteratively using a hash function with partial weights. This creates a tamper-proof chain encoding both training effort and ownership, allowing verifiers to validate PoL and PoO in a single step. The final watermark preserves the cumulative training history, enabling verification without separate ownership checks or data exposure. By replacing PoL's reliance on gradient trajectories with cryptographically secure hashing, PoLO enhances privacy, eliminates dataset access, and reduces computational costs by removing redundant verifications. Verifiers can efficiently confirm both training integrity and ownership by checking any point in the watermark chain, providing a scalable and practical model verification solution.
- *We conduct extensive experiments* (§5) using ResNet, VGG, and BERT models on CIFAR10, TinyImageNet, and AG News datasets. We implement watermarking schemes with varying sizes to generate PoLO proofs, compare verification overhead with traditional PoL methods, and test resilience against two novel proof forgery attacks. Results show that PoLO achieves **99%** watermark detection accuracy for ownership verification while fully preserving training data privacy. Verification requires only **1.5–10%** of computational overhead compared to traditional PoL methods. Forging PoL proofs against PoLO demands **1.1–4 times** more resources than legitimate PoLO generation, with the original proof maintaining over **90%** detection accuracy even after attacks.

2 FORMALIZING CONCURRENT WORKS

We provide the formalization of PoX as a unified framework for analyzing PoL and PoO, outlining their core concepts and concurrent works. Tab.5 summarizes the notations.

Table 1: Comparison with existing PoL.

	Ownership	Security			Privacy		Low Ov.	
	①	②	③	④	⑤	⑥	⑦	⑧
PoL (GD, Vanilla) [8–10]	✗	✓	✗	✗	✗	✗	✗	✗
PoL (hash) [11]	✗	✓	✓	✓	✗	✗	✗	✗
PoL (zkp) [12]	✗	✓	✓	✓	✗	✓	✗	✓
PoLO (ours)	✓	✓	✓	✓	✓	✓	✓	✓

Notations: ✓ for attack-resistance/property-held; ✗ vice versa; **Ov.** for overhead.
① Avoid unauthorized use? ② (Prevent) Reverse reconstruction attacks?
③ Synthetic trajectory attacks? ④ Verification loophole attacks?
⑤ Avoid data sharing? ⑥ Defend against gradient leakage?
⑦ Proof generation? ⑧ Verification?

2.1 Proof-of-Anything and Proof-of-Learning

Definition 1 (Proof-of-anything, PoX)**.** *A prover $\mathcal{P}$ sends a proof $\mathbb{P}$ to a verifier $\mathcal{V}$, where the resources required for verifying whether $\mathcal{P}$ satisfies a certain condition Ψ is negligible compared to the computational overhead incurred during the generation of $\mathbb{P}$. The core of this verification process is a function Θ, which is irreversible in the sense that its output cannot be practically reproduced without complete knowledge of its input χ, nor can a proof $\mathbb{P}(\chi', \Psi, \Theta) = \mathbb{P}(\chi, \Psi, \Theta)$ be forged when $\chi' \neq \chi$.*

Proof-of-work. PoW [13, 14] adheres to the definition of PoX and predates its broader abstraction. PoW was one of the earliest frameworks widely used in decentralized ledger technologies (DLT) [15, 16] to demonstrate the feasibility of proofs tied to computationally expensive tasks, such as solving cryptographic puzzles, that satisfy a specific condition Ψ in terms of difficulty. In PoW, the function Θ is a cryptographic hash that transforms input (e.g., the t-th block body) into an output that is computationally infeasible to invert, ensuring irreversibility. In contrast, verifying the proof is efficient, requiring only one evaluation of Θ to check whether the output satisfies the condition Ψ.

Proof-of-(more). In addtion to computational resources (i.e., puzzles in PoW), a variety of physical or virtual resources can also serve as measurable bases for proofs, including holdings (proof-of-stake, PoS) [17, 18], time (proof-of-elapsed-time, PoET) [19, 20], storage (proof-of-space/bandwith) [21, 22], and repuatation (proof-of-authority, PoA) [23, 24]. In each of these cases, the verification function Θ typically aligns closely with its input χ.

Definition 2 (Proof-of-Learning, PoL)**.** *A prover $\mathcal{P}$ constructs a PoL proof $\mathbb{P}$ by recording the complete training trajectory of machine learning model W_T with $\mathbb{P} := (W_t, \mathbf{B}_t, A_t)_{t=0}^{T}$ where W_t denotes model weights at the t-th epoch, $\mathbf{B}_t \subseteq \mathcal{D}$ the training batches, and A_t auxiliary training parameters. The verification condition Ψ requires $\|W_{t+1} - \Theta(W_t, \mathbf{B}_t)\| \leq \epsilon$ for all t, where Θ is the model training function.*

Gradient-based PoL. Gradient-based PoL (Vanilla PoL) [8] instantiates Definition 2 through iterative parameter updates via $\Theta(W_t, \mathbf{B}_t) = W_t - \eta \nabla \mathcal{L}(W_t, \mathbf{B}_t)$, where η is learning rate and $\mathcal{L}$ the loss function. The proof $\mathbb{P}$ consists of consecutive weight snapshots (W_t, W_{t+1}) and data batches $\mathbf{B}_t$ used for each update. Subsequent studies have revealed vulnerabilities in this approach including synthetic trajectory attacks [9] and gradient matching exploits [10], driving the need for enhanced verification mechanisms.

Hash-based PoL. To enhance computational efficiency and reduce communication complexity, Zhao et al. [11] introduce an authentication and verification protocol that achieves a better balance

between computational overhead and security. In this scheme, the prover $\mathcal{P}$ generates a proof $\mathbb{P}$ during the model training process by saving the intermediate weight parameters W_t as the input χ to the function Θ. Here, Θ is instantiated as a hash function $h(W_t)$ to ensure irreversibility. During verification, the verifier $\mathcal{V}$ recalculates the hash value $h(W_t)$ from the intermediate weights W_t provided by $\mathcal{P}$ and compares it to the original hash in the proof. To confirm the training progression, $\mathcal{V}$ retrains the model from W_{t-1} and checks whether the resulting state matches W_t by evaluating the hash value, which serves as the condition Ψ.

This method preserves proof integrity and mitigates synthetic trajectory attacks by enforcing exact hash matching for intermediate states. However, it remains essentially an enhanced version of vanilla PoL and lacks the ability to verify current ownership. Like vanilla PoL, it depends on sharing data samples $\mathbf{B}$ for retraining in order to regenerate intermediate weights, raising serious privacy concerns and incurring significant computational overhead. Moreover, it suffers from a key limitation: the recomputed weights are unlikely to exactly match those produced during original training. As a result, even with correct retraining, hash mismatches may occur, rendering the verification process unreliable.

ZK-PoL. Zero-knowledge cryptographic commitments have been integrated into PoL to enhance privacy. ZK-PoL methods [12, 25] verify the training process in zero knowledge, ensuring the model is derived from the training data and a random seed. This requires the prover to work proportionally to the number of iterations, proving training efforts and resource use, thus promoting fairness for parties with limited resources. By verifying the entire process, the verifier ensures adherence to the training procedure. The protocol can be expressed as $\mathbb{P}((\sigma_{W_t}, \sigma_{W_{t+1}}, \mathbf{B}_t, p_{\mathbf{B}_t}, \pi_t, \pi_{t+1}), \Psi, \Theta)$, where $\mathbf{B}_t$, the raw data batch, is revealed to the verifier along with its Merkle proof $p_{\mathbf{B}_t}$. While model weight commitments σ_{W_t} and $\sigma_{W_{t+1}}$ help avoid exposing raw weights, verifying $p_{\mathbf{B}_t}$ necessitates disclosing $\mathbf{B}_t$ itself. Although dataset inclusion and sumcheck proofs are performed separately, they jointly complete the verification cycle. As a result, raw data batches are repeatedly exposed during verification, raising significant privacy concerns. With randomized data partitioning, prolonged verification over many iterations may gradually leak large portions or even the entirety of the committed dataset $\mathcal{D}$, compromising data confidentiality.

While this ZK-PoL effectively demonstrates the training process and resource consumption, it faces significant privacy and efficiency challenges. Revealing raw data batches over multiple iterations allows the verifier to incrementally reconstruct the entire committed dataset $\mathcal{D}$, breaching data confidentiality. With prolonged verification, the random selection of partitions by the prover further increases the risk of exposing the entirety of $\mathcal{D}$. Additionally, proof generation for each iteration often exceeds 15 minutes, making the protocol impractical for large-scale or frequent verifications. Although ZK-PoL aligns with our PoL objectives by validating training effort, its reliance on sharing raw data and high computational overhead severely limit its scalability and real-world applicability.

Existing PoL methods face several critical challenges (cf. Tab.1). One significant limitation is the high verification cost, as verifying training correctness requires recomputing intermediate model states or retraining from stored snapshots, making large-scale verification impractical. Privacy concerns are also pressing, as the recomputing process in existing PoL methods require sharing training data during verification. On the security front, existing PoL remains susceptible to various forgery attacks, such as synthetic trajectory and adversarial example-base verification loopholes, which allow adversaries to manipulate proofs and bypass verification.

Our method enhances PoL in efficiency, privacy, and security by embedding hash-based chained watermarks during training. This allows for verification through watermark validation alone, eliminating the need to share training data and reducing computational overhead. The design also strengthens resistance against adversarial attacks, making it harder to forge PoL proofs. Additionally, our method uses watermarks to verify training efforts and establish model ownership within the same framework, addressing a crucial gap in existing PoL methods, as explored below.

2.2 Proof-of-Ownership

PoL ensures authenticity and traceability in model development. However, in a model marketplace, PoL alone cannot establish legal ownership, which is essential for determining "who actually owns the model." As a result, PoL fails to resolve the "whom to pay" issue in model trading. Authorship disputes require a full PoL verification process, imposing significant computational overhead. In contrast, *model ownership* is legal constructs governed by intellectual property laws, granting developers exclusive rights to control the model's usage, distribution, and modification [26]. A complete ownership proof can provide a more efficient means to address such disputes, complementing PoL's technical validation to ensure comprehensive protection for machine learning models against both attribution disputes and unauthorized use.

Definition 3 (Proof-of-Ownership (PoO))**.** *A valid ownership proof is defined as $\mathbb{P}_o(W_t, \mathcal{D}, A, \Lambda, \Psi)$ for a model owner acting as the prover $\mathcal{P}$. This is constructed using a neural network with weights W_t at epoch t, trained on dataset $\mathcal{D}$ under specified training settings A (including hyperparameters, model architecture, optimizer, and loss functions). During training, the prover $\mathcal{P}$ embeds personalized ownership information Λ into the model. To verify ownership, a condition Ψ is evaluated on the model weights W_T at the final epoch T.*

The embedded ownership information Λ needs to exhibit a high degree of robustness, ensuring that Λ remains extractable or detectable for verification even after undergoing adversarial modifications, such as model fine-tuning, pruning, and overwriting attacks [27, 28]. This robustness guarantees the integrity of ownership verification, making it resistant to common attacks aimed at removing or obfuscating embedded ownership information.

Watermarking. Watermarking, is a type of PoO technique developed to protect multimedia content such as images [29], text [30], and videos [31] by embedding unique identifiers into the content. Uchida et al. [32] propose embedding watermarks into model weights via a regularization term added to the training loss, forming the EDNN scheme. The watermark Λ is embedded into weights W during training, and later extracted as $\hat{\Lambda}$ for verification by checking if $\Delta(\Lambda, \hat{\Lambda})$ satisfies condition Ψ. HufuNet [33] embeds a tailored autoencoder into the DNN, using the encoder as the watermark

and preserving the decoder for ownership verification. RIGA [34] introduces a GAN-based approach where the model acts as a generator producing watermarked weights, aided by a discriminator and an embedder network. FedIPR [35] extends these ideas to federated learning, allowing all clients to embed ownership into the global model for copyright protection [36].

Our method is designed to be compatible with all embedded watermarking methods for achieving PoO. By embedding a given watermark into the model parameters, we strongly associate the watermark with the model's training process, allowing for the verification of PoL through watermark verification.

3 SYSTEM OVERVIEW

Problem definitions. Arguably, *existing PoL methods fail to address scenarios where models are outsourced or exchanged for monetary assets.* Current PoL methods assume that the entity providing the PoL proof is the model owner, making them unsuitable for outsourced training where ownership and training efforts may belong to different entities [5]. In ML marketplaces, current PoL methods cannot support ownership transfer, which prevents PoL from verifying rightful ownership when models are bought, sold, or reassigned [4].

Existing PoL methods become the weakest link when simply stacking PoL and PoO. Those solutions [9, 12] are increasingly inefficient. The effort invested outweighs practical outcomes threefold.

- The training function Θ in PoL is inherently less reliable than the cryptographic hash functions used as Θ in PoW. Adversarial patterns may compromise the irreversibility of the training process [9], enabling spoofing attacks to forge training paths.
- The condition Ψ in PoL is defined as the model distance between intermediate weights, weaker than the difficulty metric in PoW.
- Maintaining a dynamic distance threshold to counter spoofing attacks adds further complexity. While techniques like ZKPs have been explored for securely sharing inputs [12], their real-world application remains constrained by high resource costs. It necessitates sharing the dataset with verifiers in most designs.

Our method instantiates the irreversible function Θ using a cryptographically secure hash function to chain the training path, transforming it from a simple GD trajectory into a rigorously linked sequence of watermarks–leveraging PoO to achieve PoL. Each step is securely linked to the previous one through model watermarks, establishing a structured verification path. This redefines the condition Ψ–the criterion for validating a PoL proof–by incorporating the robustness of PoO, making it a more practical and resilient metric against attacks. Moreover, this design preserves data privacy by eliminating dataset sharing with verifiers, significantly reducing communication overhead. PoLO also integrates the efficiency of existing watermarking techniques [32] with the undetectability properties of more advanced schemes [34].

Definition 4. *(PoLO): A PoLO proof for a prover $\mathcal{P}$ is defined as $\mathbb{P}((W_{x-1}, W_x), \Psi, \Theta)$, which verifies the training of the x-th shard s_x within a model iteration partitioned into S shards. The proof is considered valid if the ownership information Λ_x, extracted from the DP-protected final weight W_x of s_x, satisfies the condition Ψ. Specifically, this requires verifying that Λ_x matches the expected value $\hat{\Lambda}_x$, where $\hat{\Lambda}_x$ is computed using the hash-based function Θ over the prior shard's weight W_{x-1}.*

Architecture. PoLO features a chain-based embedded watermark structure to generate PoL with no need to share any training data and with superior resilience against gradient spoofing attacks [10]. Specifically, multiple watermarks are embedded throughout the model training iterations, forming a chain-based structure. Realizing the unique watermark embedded at a specific point in the chain requires a hash operation based on the knowledge of the model weights from the previous point. This ensures that an attacker intending to forge the watermark at any point would also need to forge all preceding watermarks in the chain, which incurs a computational cost equivalent to retraining a new model.

The amount of effort put into the learning is reflected by the number of *shards* formed throughout the iteration. The process of embedding each watermark naturally forms a shard, representing one point in the watermark chain. This indicates a sequential order between each shard. The size of a shard can vary to meet the accuracy threshold required to form a valid watermark. The weights of the final model within shard s_x can be regarded as a *checkpoint model*, reflecting the model performance.

Entities. Our PoLO consists of two primary entities:

- ***Prover ($\mathcal{P}$)*** is the original owner and trainer of a model, responsible for training the model, embedding watermarks, and generating proofs. As a prover, the prover provides verifiable evidence of both training effort and model ownership, ensuring the integrity and authenticity of the proof.
- ***Verifier ($\mathcal{V}$)*** is responsible for evaluating the validity of proofs by verifying the embedded watermarks and assessing the model's main task performance. As both the issuer and evaluator of the main task, the verifier ensures that the prover has completed the required workload (i.e., PoL) and retains legitimate ownership of the model (i.e., PoO). To enforce fair verification, the verifier maintains a public test dataset for performance assessment and sets a watermark detection rate threshold. Based on the verification results, $\mathcal{V}$ determines rewards for successful validation and penalties for fraud proofs.

Workflow in sketch. We focus on the procedures of model training and model challenging by presenting the interactive steps between a prover and a verifier.

- ***Step-1*** **(§4.1).** A prover $\mathcal{P}$ trains his model during which a number of watermarks are embedded to form a watermark-chain. This indicates that the published version of the model is bound to be watermarked by the final point of the chain.
- ***Step-2*** **(§4.2.1).** A verifier $\mathcal{V}$ issues a challenge to the prover, requesting verification of a randomly selected shard s_x.
- ***Step-3*** **(§4.2.1).** $\mathcal{V}$ requires to send the model weights of the checkpoint models of shards s_x and s_{x-1}.
- ***Step-4*** **(§4.2.2).** $\mathcal{V}$ computes the expected embedded watermark for s_x using the information shared from the owner, and then verifies that whether the checkpoint model of s_x is watermarked by the computed watermark.
- ***Step-5*** **(§4.2.3).** The verification result is sent back to $\mathcal{P}$. $\mathcal{V}$ decides whether to repeat the same process for further preceding shards, depending on the requirement of the security level.

Player model. Three types of players are considered.

- ***Rational prover.*** The prover functions as both the model *owner*, holding full rights to its creation and usage, and the PoL *prover*,

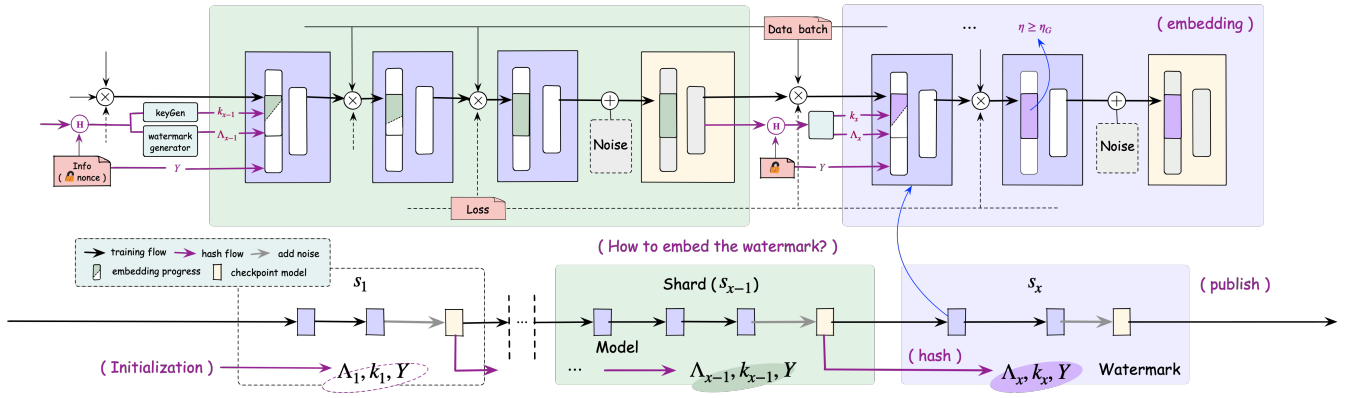


Figure 2: PoLO design: The verifier shares a secret nonce with the prover to initialize watermark parameters (Λ_1, k_1, Y) for the first shard s_1. Prior to watermark embedding, the model owner computes the watermark Λ_{x-1} and its corresponding embedding key k_{x-1} for shard s_x using a hash function $\mathbb{H}(\cdot)$ over the previous model W_{x-1}, auxiliary information, and the secret nonce. During training, Λ_{x-1} is embedded into the model using k_{x-1}, while monitoring the watermark detection rate η. Once η exceeds the threshold η_G, DP noise is applied to enhance robustness against inference attacks. Training proceeds to the next shard with new Λ_x and k_x. The process continues until the model converges.

responsible for proving the authenticity and integrity of the training process. A rational prover acts in a way that maximizes their utility. While they do not compromise the integrity of model training or watermark embedding, they may attempt to falsify reported workloads to gain additional rewards.

- ***Honest-but-curious verifier.*** The verifiers act as auditors, responsible for challenging and validating the PoL provided by the prover to assess whether they deserve rewards based on their contributions to model training. To achieve this, the verifiers use a public test dataset to evaluate the model performance on the main training task. They also manage system settings, such as accuracy thresholds for watermarks in each shard and the size of the embedded watermark. Although the verifiers operate honestly and do not interfere with the verification process, they are curious and may attempt to infer the provers' private training data from the information contained within PoL.
- ***Malicious attackers.*** The attackers seek to exploit the system through various attacks, including:
  - *Training forgery (addressed in §4.3.1).* The attacker may fabricate a fraudulent PoL to falsely represent a legitimate training process, such as by forging training trajectories or interpolating intermediate states, aiming to falsely claim training efforts.
  - *Ownership theft (addressed in §4.3.2).* The attacker may attempt to steal or misuse the model by targeting its watermark through attacks such as fine-tuning [37], pruning [38], removal [39], or overlapping [40], aiming to falsely claim model ownership.
  - *Inference attacks (addressed in §4.3.3).* The attacker may exploit the information contained in the published model to infer the provers' private training data.

4 DESIGN OF POLO

PoLO is a unified framework that integrates PoL and PoO through a chained watermarking mechanism. This section outlines its design and verification process (cf. Figs.2–3), and explains how PoLO enables fair workload validation, strong ownership protection, and robustness against adversarial attacks, while preserving the integrity of the model's primary task performance.

4.1 Training with Chained Watermarking

The core of PoLO is the *chained watermarking* mechanism, which ensures progressive proof accumulation while maintaining model integrity, directly addressing **RQ1** by binding ownership verification to the training process. Our framework links each training phase to its previous state, preventing forgery or tampering.

In PoLO, a shard is a verifiable segment of the training process that comprises one or more consecutive epochs. While an epoch represents a full pass over the training data, a shard is defined based on the successful embedding of a watermark, which may require multiple epochs to complete. This design ensures that verification is aligned with meaningful training effort rather than fixed epoch boundaries. Let T_x denote the set of epochs contained in the x-th shard s_x. PoLO partitions training into shards, enabling verifiable proof of training effort. The prover's training process with chained watermarking in PoLO consists of the following key steps:

Chained watermark generation. To ensure cryptographic linkage between successive training phases, PoLO derives the watermark Λ_x for shard s_x from the final weights W_{x-1} of the previous shard s_{x-1}. This chained watermarking mechanism guarantees that each watermark is uniquely bound to its predecessor, preventing adversaries from fabricating or modifying individual watermarks without reconstructing the entire sequence.

Initialization begins with a verifier-provided nonce, which the prover uses to generate the initial watermark parameters: the first watermark Λ_1, the embedding key k_1 for shard s_1, and a selection matrix Y that specifies the embedding positions within model weights. The matrix Y is fixed and reused across all shards to ensure consistency and verifiability. For each shard s_x, both Y and the embedding key k_x are deterministically derived from the verifier-provided secret nonce μ:

$$\begin{aligned} Y &= \mathsf{WMPosition}(\mu), \\ \Lambda_x,\ k_x &= \mathsf{WMGen}(\mathcal{H}_x),\ \mathsf{KeyGen}(\mathcal{H}_x) \text{ with} \\ \mathcal{H}_x &= \mathbb{H}(W_{x-1},\ x,\ \mu,\ id_{\mathcal{P}}), \end{aligned} \tag{1}$$

where $\mathsf{WMPosition}(\cdot)$, $\mathsf{WMGen}(\cdot)$ and $\mathsf{KeyGen}(\cdot)$ are deterministic functions, and all randomness stems from the nonce μ and

identity $id_{\mathcal{P}}$. This ensures only the legitimate prover can produce valid watermarks and embedding keys, while the shard index x enforces sequential linkage across training. The use of a nonce fixed by the verifier prevents adversaries from precomputing or guessing valid parameters.

Since modifying any single shard would invalidate all subsequent watermarks, attackers cannot selectively alter or remove a watermark without regenerating the entire chain, nor can they forge valid watermarks without reconstructing the full training sequence. This chained construction establishes a strong cryptographic binding, guaranteeing the authenticity of PoLO and ensuring that the training process remains verifiable and resistant to manipulation.

Watermark embedding in model training. During the training of s_x, the prover $\mathcal{P}$ selects specific layers in the model to embed a unique watermark Λ_x. The watermark Λ_x is embedded into all model weights $W_{t_x}, \forall t_x \in T_x$, gradually forming the weights $W_{\bar{t}_x}$ of the final epoch in s_x where Λ_x has been successfully embedded. This process uses a secret key k_x to perform the embedding:

$$W_{\bar{t}_x} = \arg\min_{W} \left(l_w(W) + \lambda l_\Lambda(W)\right)\Big|_{W_0 = W_{x-1}} + \delta W, \tag{2}$$

where $\delta W = \mathbb{E}(W_{x-1}, \Lambda_x, k_x, Y)$ is a watermark embedding perturbation using the previous final weights W_{x-1}, the watermark Λ_x, a key k_x, and a selection matrix Y that determines the positions for embedding the watermark Λ_x, is added to W_{trained}. This final adjustment yields the watermarked weights $W_{\bar{t}_x} = W_{\text{trained}} + \delta W$.

This embedding process is dynamically monitored, with training continuing until the watermark detection rate η, which is computed using the following based on Hamming distance [41]:

$$\eta = 1 - \frac{\sum_i^n (\mathbb{C}(W_{t_x}, Y, k_x)[i] \neq \Lambda_x[i])}{n}, \tag{3}$$

reaches a predefined threshold η_G (i.e., $\eta \geq \eta_G$). Therein, $\mathbb{C}(\cdot)$ is the watermark extraction function and n is the size of Λ_x. By enforcing this controlled, sufficient embedding process, PoLO ensures ownership traceability while preserving the main task's performance.

Differential privacy protection and shard formation. PoLO applies differential privacy (DP) by randomly selecting a subset of weights from the non-watermarked region $W_{t_x} \setminus \{W_{t_x} Y\}$ and injecting noise. This selective DP mechanism introduces statistical uncertainty, protecting sensitive training information in non-watermarked areas from gradient leakage [42], while preserving the ownership integrity carried by the embedded watermark. The point of successful watermark embedding, denoted as $W_{\bar{t}_x}$, yields a DP-protected version $W_{\bar{t}_x}^{\text{DP}}$, which marks the completion of shard s_x. Therefore, we refer to this checkpoint model as W_x that serves as a secure and verifiable training checkpoint, ensuring tamper resistance and preserving privacy throughout each phase.

$$W_x = W_{\bar{t}_x}^{\text{DP}} = W_{\bar{t}_x} + \mathbb{DP}(\varepsilon, Z, W_{\bar{t}_x}), \tag{4}$$

where $\bar{t}_x = \max_{t_x \in T_x}(t_x)$ and $W_{\bar{t}_x}$ represents the model weights at the last epoch of s_x. The term $\mathbb{DP}(\varepsilon, Z, W_{\bar{t}_x})$ denotes the addition of ε-DP noise to the subset of weights in $W_{\bar{t}_x}$ specified by a selection matrix Z for DP. The storage of W_x marks the completion of shard s_x, allowing the training process to transition to the next phase.

Iterative training with watermark propagation. The new watermark Λ_{x+1}, derived from $\mathbb{H}(\cdot)$, is then embedded into the training of shard s_{x+1}, continuing the chained watermarking process. This cycle repeats until the model either converges to a stable state or meets the performance objectives of the main task. With each iteration in PoLO, the previous watermark propagates forward, ensuring strong sequential dependency across shards. This incremental accumulation of watermarks makes removal or modification attacks impractical, as any tampering would require reconstructing the entire sequence while preserving model performance. By enforcing hash-based chaining and progressive watermark propagation, PoLO provides a cryptographically verifiable proof of both training effort and ownership, making it highly resistant to tampering, removal, or synthetic trajectory attacks.

Ownership transfer. In scenarios requiring ownership transfer, such as ML marketplaces [4, 43] and outsourced training [5], an additional fine-tuning shard is appended to the training process to reassign ownership. Specifically, a new or modified owner identifier $id_{\mathcal{P}'}$ is used to generate a fresh watermark Λ_{S+1} based on the latest model W_S. This watermark is then embedded through fine-tuning, producing a new model instance W_{S+1} that remains tied to the rightful owner while preserving the integrity of existing PoLO proofs. The resulting PoLO not only verifies the legitimacy of the final model owner but also retains the entire historical chain of PoO and PoL, ensuring full traceability.

4.2 Verification in PoLO

The verification mechanism in PoLO is designed to ensure training effort accountability and ownership authenticity by validating both the watermark chain and the model performance. The verifier $\mathcal{V}$ assesses the prover's PoLO proof $\mathbb{P}$ by sequentially verifying the stored DP-protected model checkpoints and their extracted watermarks from the latest shard to the earliest. In PoLO, verification is performed at the *shard level*, rather than per epoch, ensuring that provers cannot exaggerate their training workload while maintaining a provable, tamper-resistant sequence of training. The verifier can selectively challenge any shard to verify both training efforts and ownership.

4.2.1 Generating proof. When the verifier $\mathcal{V}$ requests validation for shard s_x, the prover $\mathcal{P}$ provides the necessary components, including the DP-protected model weights W_x and W_{x-1} from shards s_x and s_{x-1}, respectively. Additionally, $\mathcal{P}$ provides the hash function $\mathbb{H}(\cdot)$ to derive the watermark Λ_x and the secret key k_x from W_{x-1} using the verifier-recorded secret nonce μ, along with the function that deterministically derives the watermarked weight selection matrix Y from the same nonce. These components allow the verifier to validate both the model's integrity and the sequential linkage of watermarks (cf. Algorithm 1 in **Appendix** B).

Unlike existing PoL [8–12] that require the prover $\mathcal{P}$ to disclose training data for verification, PoLO ensures that verification can be conducted without exposing any training data to the verifier $\mathcal{V}$. Our design significantly enhances data privacy, eliminating the risk of unintended data leakage while still allowing the verifier to assess both training effort and ownership authenticity.

4.2.2 Verifying proof. The verification begins by assessing if the model checkpoint W_x meets the expected performance criteria on the public test dataset associated with the main task. The verifier

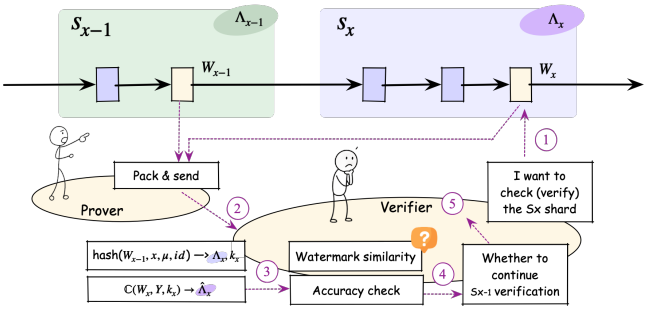


Figure 3: The verification of chained watermarking for PoL: ① (verifier) requests verification for shard x; ② (prover) sends checkpoints W_{x-1} and W_x; ③ (verifier) derives the selection matrix Y from the locally recorded nonce μ associated with the prover, if Y has not already been obtained. Using the received checkpoints, the verifier then derives the watermark Λ_x and the extraction key k_x from W_{x-1}, also based on μ, and finally extracts the watermark $\hat{\Lambda}_x$ from W_x using k_x and Y; ④ (verifier) validates the watermark and main task accuracy; ⑤ (verifier) returns the result and repeats for earlier shards if needed.

$\mathcal{V}$ evaluates the model against the test dataset to ensure that the training process has led to meaningful task-specific improvements, which aligns with the practical requirement that only sufficiently accurate models are considered valuable in ML marketplaces [4]. If W_x fails to meet the expected accuracy threshold, the proof is rejected immediately, and the prover $\mathcal{P}$ incurs penalties. This prevents provers from embedding watermarks in arbitrarily generated models without properly completing the training process.

Once the model passes the main task performance evaluation, the verifier $\mathcal{V}$ proceeds to chained watermark verification. First, the verifier reconstructs the watermarked weight selection matrix Y, expected watermark Λ_x and the key k_x for shard s_x using the stored DP-protected weights from shard s_{x-1} (cf. Equation 1). This confirms that the watermark Λ_x must have been generated from W_{x-1} and cannot be arbitrarily forged or altered.

The verifier $\mathcal{V}$ then extracts $\hat{\Lambda}_x$ from the checkpoint model W_x using the selection matrix Y and the secret key k_x:

$$\hat{\Lambda}_x = \mathbb{C}(W_x, Y, k_x). \tag{5}$$

By comparing $\hat{\Lambda}_s$ with the watermark Λ_s derived from Equation 1, the verifier $\mathcal{V}$ determines whether the watermark detection rate exceeds the predefined threshold η_G. The watermark detection rate of the x-th shard is computed based on the following:

$$\eta_x = 1 - \frac{\sum_i^n (\hat{\Lambda}_x[i] \neq \Lambda_x[i])}{n}. \tag{6}$$

If the extracted watermark satisfies $\eta_x \geq \eta_G$, the PoLO proof is accepted. Otherwise, the verification fails due to potential tampering, watermark removal, or inconsistency in the PoLO proof (cf. Algorithm 2 in **Appendix** B).

4.2.3 Cumulating security confidence. To reinforce the integrity of the PoLO proof, the verifier $\mathcal{V}$ may conduct backward verification by recursively validating previous shards, moving from s_x to s_{x-1}, s_{x-2}, and so forth, until a sufficiently verifiable proof chain is established. Since each watermark is cryptographically linked to its predecessor, backward verification prevents partial proof forgery. No malicious prover can manipulate only recent checkpoints while leaving earlier training shards unverifiable.

Upon successful verification, the verifier $\mathcal{V}$ can determine the correct rewards for the prover based on the number of validated shards S and the performance on the main task. A higher final model accuracy on the test set results in greater rewards, incentivizing provers to optimize model performance continuously. Moreover, each shard serves as a verifiable unit of training effort, making the total compensation directly proportional to the number of recognized shards S. This shard-based reward mechanism prevents provers $\mathcal{P}$ from inflating their reported training workload by falsely claiming an excessive number of epochs for watermark embedding. Such manipulation would not only fail to contribute to the main task but also undermine the fairness of reward allocation, making an epoch-based system impractical.

The number of shards S primarily depends on the number of training epochs required for watermark embedding. A higher learning rate enables faster watermark embedding, leading to more shards. However, an excessively high learning rate in later training stages can hinder the model's ability to further improve its accuracy on the main task. Since rewards are determined by both the number of validated shards and the final model accuracy, a rational prover will strategically balance the learning rate to maximize both shard count and accuracy, achieving optimal rewards.

The verification mechanism in PoLO provides strong security guarantees. The chained watermarks ensure that forging a valid PoLO proof requires reconstructing the entire training sequence while preserving the model performance, making forgery computationally infeasible. Modifying any single shard invalidates all subsequent watermarks, rendering selective tampering impractical. Moreover, the shard-based verification strategy ensures fairness by tying rewards to the number of validated shards, preventing provers $\mathcal{P}$ from inflating their workload claims. This security foundation also extends naturally to *collaborative scenarios involving multiple users* (cf. **Appendix** G), where PoLO can cryptographically attribute and verify the contribution of each party in relay-style training pipelines.

4.3 Security and Privacy Analysis

By combining hash-based watermark chaining, backward verification, and shard-level performance checks, PoLO offers a robust framework where training effort and ownership are provable, tamper-resistant, and cryptographically verifiable. Given the adversarial models (cf. §3), we now address **RQ2** by showing how PoLO unifies PoL and PoO while preserving security and privacy. Key takeaways are summarized in **Appendix** F.

4.3.1 Training efforts. PoLO addresses a range of adversarial behaviours that aim to manipulate or falsify the training process.

Forging PoLO. PoLO introduces a chained watermarking framework, where each shard's watermark is securely linked to the previous shard's output via a hash function $\mathbb{H}(\cdot)$. This chaining enforces that any tampering or forgery attempt requires reconstructing the entire sequence of valid watermarks while maintaining main task performance. Leveraging the immutability of cryptographic hashes, PoLO ensures that proof forgery is computationally infeasible, thereby preserving the integrity of verification.

Exaggerated workload claims. A rational prover might attempt to inflate training workload by embedding excessive watermarks to artificially increase the number of reported epochs. PoLO counters this by enforcing verification at the shard level, where each shard represents a complete, verifiable training unit defined by successful watermark embedding. The number of shards depends on the learning rate: higher rates speed up embedding, reducing epochs per shard and increasing shard count, but may harm main task performance. Since rewards are based on both the number of verified shards and model accuracy, the prover is incentivized to balance these factors by choosing an optimal learning rate.

A rational prover may attempt to exaggerate training effort by inflating the number of reported shards without performing the required computation. With full access to model parameters, the prover could forge watermarked weights by either (1) anticipating the expected watermark Λ_{x+1} for shard s_{x+1} and crafting a corresponding embedding key k_{x+1} without actual training, or (2) embedding known watermarks into low-impact weights using a manipulated selection matrix Y, allowing extraction without genuine model updates. To prevent forgery, PoLO derives k_{x+1} from a hash of the previous shard's output and a verifier-provided nonce μ, ensuring only genuine training and access to the nonce can yield valid keys. The selection matrix Y is also deterministically generated from this nonce, binding watermark placement to verifier-specified positions. This tightly links watermark embedding to real training effort, making shard inflation both computationally infeasible and economically unviable.

4.3.2 Ownership. PoLO is compatible with existing embedded watermarking techniques and does not compromise their inherent security guarantees. On the contrary, the chaining structure introduced by PoLO can enhance robustness against certain attack vectors, such as watermark forgery, by cryptographically binding each watermark to prior authenticated states.

Unauthorized use and model theft. To counter unauthorized use and model theft, PoLO integrates seamlessly with various model watermarking techniques, enabling strong ownership verification and resilience against watermark-related attacks. By incorporating advanced watermarking methods, such as RIGA watermarking [34] and FedIPR [35], PoLO ensures that embedded watermarks remain stealthy, attack-resistant, and robust against removal attempts and adversarial modifications. These techniques provide strong ownership proof in PoLO, effectively deterring unauthorized distribution and tampering while maintaining the integrity of the model.

Ownership evasion via watermark stripping. In PoLO, the presence of a valid watermark is a prerequisite for establishing model ownership. This is particularly critical in scenarios such as ML marketplaces or outsourced training, where models are exchanged or transferred across entities. In our setting, models without verifiable watermark traces are treated as illegitimate and thus ineligible for use, resale, or transfer. As a result, simple watermark removal attacks become ineffective—stripping the watermark severs the model's ownership proof, rendering it unverifiable and valueless.

4.3.3 Dataset privacy. PoLO inherently enhances data privacy by eliminating the need for provers to share their training data with verifiers during the validation process. Unlike existing PoL mechanisms that rely on dataset disclosures for proof verification, PoLO ensures that verifiability is achieved solely through DP-protected model checkpoints and chained watermarks. To further safeguard against privacy inference attacks related to gradient leakage, PoLO incorporates DP mechanisms at each shard's completion. By applying DP protection to model weights, PoLO effectively prevents verifiers and adversaries from inferring sensitive training data, preserving the privacy of the prover's dataset.

5 EXPERIMENTS

We implement and evaluate PoLO to demonstrate its practical viability. Our experiments focus on three key aspects: *compatibility* (broad applicability), *overhead* (low computational costs), and *unforgeability* (strong resistance to forgery).

5.1 Experimental Settings

To ensure comprehensive evaluation and broad applicability, we conduct experiments across diverse model architectures and multiple datasets. All experiments are performed on a dedicated microserver infrastructure, providing a standardized and controlled environment for consistent model evaluation and comparison.

Implementation environment. The hardware configuration comprises two Intel Xeon Gold 6126 processors, each with 12 cores. The server is equipped with 192GB of 2666MHz ECC DDR4-RAM, organized in six channels. Storage is handled by 2x1.2TB 10,000 RPM SAS II hard drives configured in Raid 1. The computational power is enhanced by two NVIDIA Tesla V100 GPUs, each featuring 5,120 CUDA cores, 640 Tensor cores, and 32GB of dedicated memory. The operating system running on this setup is 64-bit Ubuntu 18.04. This robust hardware environment supports extensive data processing and complex computational tasks required for the research.

Datasets. Our evaluation includes four datasets: three focused on image classification tasks and one dedicated to text classification.

- *CIFAR-10* is a dataset with 60,000 color images (32×32 pixels) across 10 classes, split into 50,000 training and 10,000 test images.
- *CIFAR-100* includes 60,000 color images (32×32 pixels) across 100 classes, with 50,000 for training and 10,000 for testing. Each image has fine and coarse labels.
- *TinyImageNet* has 200 classes, each with 500 training, 50 validation, and 50 test images, totaling 100,000 training images, all resized to 64×64 pixels.
- *AG News* is a text classification dataset with 120,000 training samples and 7,600 test samples in 4 categories.

Model architectures. We employ a variety of established deep neural networks: *AlexNet* [44], *ResNet18* and *ResNet34* [45], *WideResNet* [46], *VGG16* [47], *TextCNN* [48], and *MiniBert* [49]. *AlexNet* offers simplicity and effectiveness for image classification; *ResNet* models leverage residual connections to train deeper networks; *WideResNet* improves performance by increasing layer width. *VGG16* is known for its depth and strong performance in visual tasks. *TextCNN* captures n-gram features via 1D convolutions for text classification, while *MiniBert*, a lightweight transformer, learns contextualized word embeddings through masked language modeling and next sentence prediction.

Table 2: The comparison of shard rate $|s|$, main task performance Acc_{main}, and PoLO generation time (+training), using different watermarks.

Watermark **methods**	Watermark **size**(bits)	**Training**	CIFAR-10 AlexNet	CIFAR-10 ResNet18	CIFAR-100 ResNet18	CIFAR-100 WideResNet	TinyImageNet VGG16	TinyImageNet ResNet34	AG News TextCNN	AG News MiniBert
RIGA [34]	2048	$\|s\|$(%)	37.07	37.62	38.61	35.83	74.51	73.08	43.64	64.15
		Acc_{main}(%)	89.22	91.89	74.81	72.83	73.00	72.46	91.04	91.88
		Time(s)	990.74	1618.78	1707.59	9357.71	25051.72	7332.32	1070.12	2573.38
	1024	$\|s\|$(%)	64.04	64.60	71.29	63.48	96.15	96.15	71.29	93.07
		Acc_{main}(%)	88.94	91.96	74.93	72.96	73.33	73.31	90.94	92.06
		Time(s)	1066.78	1861.62	1681.42	8968.80	24934.03	7238.95	1067.23	2687.89
	512	$\|s\|$(%)	87.74	92.08	86.79	87.74	96.15	96.15	74.29	95.05
		Acc_{main}(%)	89.54	92.22	75.18	72.64	73.91	73.22	90.65	91.82
		Time(s)	993.12	1783.50	1798.00	8433.43	25442.66	6775.54	1051.52	2486.51
EDNN [32]	2048	$\|s\|$(%)	35.64	35.64	35.64	36.27	70.00	70.00	37.62	47.06
		Acc_{main}(%)	89.88	92.07	74.32	72.65	73.61	71.89	90.29	91.71
		Time(s)	880.91	1518.95	1689.89	8423.67	24492.53	6955.89	896.11	2333.33
	1024	$\|s\|$(%)	39.17	45.54	45.54	44.12	83.00	78.00	46.60	60.00
		Acc_{main}(%)	89.67	91.55	73.98	72.86	73.91	72.01	90.78	91.99
		Time(s)	923.03	1493.05	1708.98	7876.67	24342.67	7501.34	956.65	2435.32
	512	$\|s\|$(%)	57.27	60.95	52.00	54.24	96.00	96.00	61.39	73.79
		Acc_{main}(%)	89.89	91.62	73.89	72.16	73.30	71.88	90.56	91.89
		Time(s)	949.75	1711.73	1665.76	7966.58	24689.07	7683.83	998.02	2534.32
FedIPR [35]	2048	$\|s\|$(%)	40.78	40.59	58.00	32.67	70.00	80.39	65.69	35.64
		Acc_{main}(%)	88.92	91.79	74.67	72.93	73.23	71.99	90.32	91.43
		Time(s)	993.91	1779.48	1594.15	9330.49	25089.67	6941.68	1050.88	2691.72
	1024	$\|s\|$(%)	74.26	86.27	67.33	88.12	92.31	96.15	71.29	69.31
		Acc_{main}(%)	89.38	91.43	74.30	71.54	74.30	71.76	90.37	91.87
		Time(s)	996.87	1693.35	1643.65	8901.83	24810.68	6891.76	994.08	2499.38
	512	$\|s\|$(%)	92.08	93.14	81.37	91.18	96.15	96.15	94.12	93.14
		Acc_{main}(%)	89.04	92.10	74.09	72.01	73.87	72.01	90.50	91.87
		Time(s)	1004.12	1750.65	1674.89	9354.67	25102.65	7354.67	1059.54	2652.31

• A larger shard rate $|s|$ means less time spent embedding watermarks, indicating weaker security but reduced waste of intra-shard training effort, as effort is accounted for at the shard level regardless of how much is invested within each shard.

We use AlexNet and ResNet18 for CIFAR-10, ResNet18 and WideResNet for CIFAR-100, VGG16 and ResNet34 for TinyImageNet, and TextCNN and MiniBERT for AG News. All models are trained with SGD (momentum 0.9), using dataset-specific learning rates: 0.1 for CIFAR-10/100, 0.05 for TinyImageNet, and 0.005 for AG News. A uniform batch size of 256 is used across all experiments.

Attack settings. Assuming the attacker $\mathcal{A}$ has access to all checkpoint models W_x, they could potentially forge a PoLO proof $\mathbb{P}'$. To evaluate the robustness of PoLO against forgery attempts, we implemented two distinct attack strategies.

- *Overhaul Fine-tuning Attack (OFA).* OFA combines fine-tuning [37] and watermark removal [39]. Given access to checkpoint models but not the legitimate keys (k_x, Y), the attacker $\mathcal{A}$ attempts to forge illicit proofs $\mathbb{P}'$ by fine-tuning all model weights to erase the authentic proof $\mathbb{P}$. Since watermark locations are unknown, full-model fine-tuning is necessary–partial updates are insufficient to disrupt the chained watermark structure [37]. When training data is available, $\mathcal{A}$ additionally embeds fake watermarks Λ_x' during fine-tuning to overwrite legitimate ones. To balance cost and impact, five fine-tuning rounds are applied for 2048-bit watermarks, and three rounds for 1024-bit and 512-bit versions.
- *Weight Manipulation Attack (WMA).* WMA is analogous to pruning [38] and watermark overlap [40] techniques. In the absence of training data access, the attacker $\mathcal{A}$ may attempt to manipulate specific weight parameters (similar to pruning-based attacks) to embed their illicit watermarks Λ_x' into each intermediate model. The $\mathcal{A}$ predetermines their Λ_x' and embedding keys, manipulates the appropriate weight values, and replaces certain weights in the intermediate model with these values.

If the attacker $\mathcal{A}$ can forge a proof $\mathbb{P}'$ with lower computational overhead than generating a legitimate proof $\mathbb{P}$ while successfully corrupting the original watermarks Λ_x, such a forged proof $\mathbb{P}'$ could potentially pass the verification process.

Evaluation metrics. We consider the following three metrics, in which comprehensive comparisons with baselines are conducted.

- *Compatibility* evaluates the applicability of various watermarking schemes for PoLO implementation by assessing the work transition efficiency, measured as the *shard rate* $|s| = \frac{\text{shards}}{\text{epochs}}$. An optimal shard rate falls within the range $0 < |s| < 1$, indicating efficient shard generation relative to training epochs.
- *Overhead* evaluates computational efficiency in two aspects: (1) the additional computational cost to generate PoLO should be minimal compared to the intensive demands of the model training process itself, and (2) the computational overhead required to verify a PoLO's correctness must be substantially lower than that needed to generate it, ensuring practical verification.

Table 3: The comparison between PoLO and existing PoL in terms of the time consumption of PoLO generation (+training) and verification.

Watermark **size**(bit)	**Time** (s)	**Methods**	CIFAR-10 AlexNet	CIFAR-10 ResNet18	CIFAR-100 ResNet18	CIFAR-100 WideResNet	TinyInageNet VGG16	TinyInageNet ResNet34	AG News TextCNN	AG News Bert
2048	Training	Baseline	970.45	1598.37	1691.38	9340.34	25031.45	7312.89	1052.38	2551.45
	Training+$\mathbb{P}$ gen (v.s.) $\mathbb{P}$ gen	PoLO	990.74/20.29	1618.78/20.41	1707.59/16.21	9357.91/17.57	25051.72/20.27	7332.32/19.43	1070.12/17.74	2573.38/21.93
		PoL (Vanilla)	975.33/4.88	1603.47/5.10	1695.58/4.20	9343.83/3.49	25035.45/4.00	7316.12/3.23	1055.23/2.85	2556.15/4.70
		PoL (hash)	979.13/8.68	1604.67/6.30	1696.78/5.40	9347.43/7.09	2504.65/11.20	7317.42/4.53	1056.63/4.25	2557.55/6.10
	$\mathbb{P}$ verify	PoLO	83.37	75.96	82.73	211.06	342.17	135.43	44.54	56.29
		PoL (Vanilla)	891.99	1530.55	1612.91	9132.82	24693.31	7180.73	1010.71	2495.89
		PoL (hash)	895.79	1531.75	1614.10	9136.42	24700.51	7182.03	1012.11	2497.29
1024	Training	Baseline	1043.32	1832.45	1653.78	8945.56	24910.32	7215.56	1040.67	2653.75
	Training+$\mathbb{P}$ gen (v.s.) $\mathbb{P}$ gen	PoLO	1066.78/23.46	1861.62/29.17	1681.42/27.64	8968.80/23.24	24934.03/23.71	7238.95/23.39	1067.23/26.56	2687.89/34.14
		PoL (Vanilla)	1047.57/4.25	1837.13/4.68	1658.59/4.81	8948.85/3.29	24914.52/4.20	7219.32/3.76	1044.31/3.64	2658.57/4.82
		PoL (hash)	1051.37/8.05	1838.32/5.88	1659.78/6.00	8952.45/6.89	24921.71/11.40	7220.61/5.57	1045.71/5.04	2659.97/6.22
	$\mathbb{P}$ verify	PoLO	157.60	149.07	149.63	380.86	456.77	172.15	67.90	78.57
		PoL (Vanilla)	887.03	1688.14	1509.04	8566.07	24455.79	7045.22	974.43	2580.03
		PoL (hash)	890.83	1689.34	1510.24	8569.67	24462.98	7046.51	975.83	2581.43
512	Training	Baseline	963.13	1754.67	1763.65	8402.83	25419.39	6751.64	1023.67	2451.54
	Training+$\mathbb{P}$ gen (v.s.) $\mathbb{P}$ gen	PoLO	993.12/29.99	1783.50/28.83	1798.00/34.35	8438.43/35.60	25442.66/23.27	6775.54/23.90	1051.52/27.85	2486.51/34.97
		PoL (Vanilla)	967.13/4.00	1756.94/2.27	1768.85/5.20	8407.34/4.51	25425.03/5.64	6754.39/2..75	1026.55/2.88	2456.71/5.17
		PoL (hash)	970.93/7.80	1758.14/3.47	1770.05/6.40	8410.94/8.11	25432.23/12.84	6755.69/4.05	1027.95/4.28	2458.43/6.57
	$\mathbb{P}$ verify	PoLO	193.79	213.85	188.58	495.44	445.18	167.15	76.69	81.88
		PoL (Vanilla)	768.41	1541.18	1580.35	7912.02	24975.89	6587.29	949.88	2374.87
		PoL (hash)	722.21	1542.38	1581.55	7915.62	24983.09	6588.59	951.28	2376.27

• The baseline in this table refers to pure model training without applying any PoL or PoO techniques.
• PoL (hash) [11] follows a similar PoL process to that of PoL (Vanilla) [8–10], with the key difference being the hash computation, which takes only a negligible amount of time.
• PoL (zkp) [12] incurs around 15 mins of proof generation time per iteration on VGG-11 with CIFAR-10 (excluding training time); in contrast, PoLO completes proof generation for all iterations in ~ 20s on VGG-16 with TinyImageNet.

- *Unforgeability* evaluates whether the computational overhead required to forge a PoLO proof $\mathbb{P}'$ surpasses that of generating a legitimate proof $\mathbb{P}$, while simultaneously verifying that any forgery attempt results in the destruction of the legitimate proof $\mathbb{P}$. A forged proof $\mathbb{P}'$ can only pass verification if both these conditions are satisfied.

5.2 Experimental Results

Experiments are based on three main metrics–compatbility, overhead, and unforgeability (more minor metrics are discussed in **Appendix** C and D) to validate the superiority of PoLO compared with existing PoL methods under the OFA and WMA attack settings.

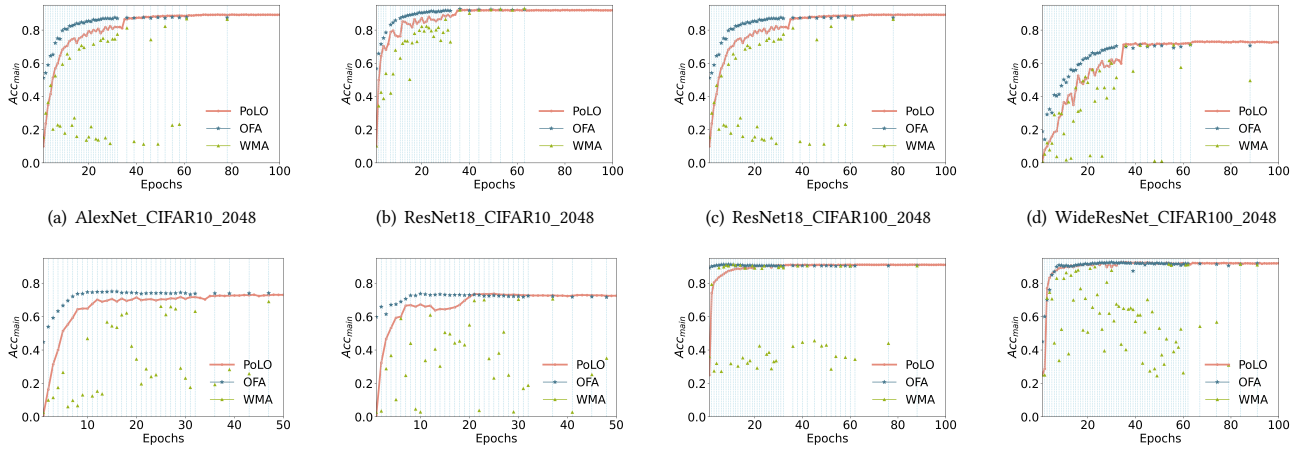
5.2.1 Compatibility. The PoLO framework exhibits strong versatility by supporting diverse watermarking schemes, including RIGA [34], EDNN [32], and FedIPR [35]. This flexibility allows for generating PoLO proofs $\mathbb{P}$ while establishing ownership using different watermarking strategies. During training, the convergence of embedded watermarks reflects training progress, with their convergence speed closely tied to weight update dynamics. Each watermark shard quantifies the training effort, serving as verifiable evidence of computation invested. To assess the computational efficiency of PoLO, we use shard rate $|s|$ as the key metric. To evaluate watermarking compatibility, we conduct experiments by using three distinct watermarking schemes to generate PoLO proof $\mathbb{P}$. For each scheme, we test three watermark sizes: 2048 bits, 1024 bits, and 512 bits. Our evaluation metrics included the shard rate $|s|$, main task accuracy Acc_{main}, and the total computational time required for training and $\mathbb{P}$ generation.

As shown in Tab.2, $|s|$ varied significantly, ranging from 32.67% to 96.15%. A clear inverse relationship is observed between watermark size and $|s|$, where a shorter watermark size results in a proportionally higher $|s|$ across all watermarking schemes. This trend is particularly evident in the TinyImageNet dataset, which consistently exhibited higher $|s|$ under all watermarking configurations. Notably, the Acc_{main} values remained stable across different watermarking approaches, with minimal deviation from baseline model performance (cf. Fig.6 in **Appendix** D). Furthermore, the total computational time for training and $\mathbb{P}$ generation remained relatively unchanged across different watermarking schemes, indicating that computational overhead remains consistent regardless of the chosen watermarking method.

> **Takeaway (high compatibility).** PoLO supports a wide range of watermarking schemes, while maintaining stable main task accuracy and consistent computational overhead, enabling seamless integration into different implementation without compromising performance.

5.2.2 Overhead. A critical consideration in our evaluation is ensuring that $\mathbb{P}$ generation does not impose substantial computational overhead beyond baseline model training costs. Furthermore, for practical implementation, the verification overhead must be significantly lower than $\mathbb{P}$ generation costs. To validate the efficiency of our proposed PoLO framework, we conduct comparative analyses against traditional PoL schemes [8–12] and baseline training approaches. Our experimental setup incorporates three distinct watermark sizes for PoLO generation. To maintain experimental rigor and ensure fair comparison, we standardize computational resources and parameter settings across all trials. The verification process is designed to validate each intermediate $\mathbb{P}$, providing a comprehensive assessment of the framework's efficiency.

Tab.3 compares the computational overhead of PoLO with conventional PoL methods, focusing on $\mathbb{P}$ generation and verification time. PoLO incurs only modest overhead when integrated into training; for example, on CIFAR-10 with AlexNet and a 512-bit watermark, it takes 993.12 seconds versus 963.13 seconds for baseline



(a) AlexNet_CIFAR10_2048 (b) ResNet18_CIFAR10_2048 (c) ResNet18_CIFAR100_2048 (d) WideResNet_CIFAR100_2048

(e) VGG16_TinyImageNet_2048 (f) ResNet34_TinyImageNet_2048 (g) TextCNN_AG News_2048 (h) MiniBert_AG News_2048

Figure 4: The Acc_{main} comparison of the two $\mathbb{P}$ forge attacks with baseline - PoLO. The number followed by the dataset is the watermark size. In each subplot, the vertical lines perpendicular to the x-axis represent the completion times for each shard.

training–a 3.11% increase. In contrast, conventional PoL methods require generation time nearly identical to baseline training.

The efficiency of PoLO is most apparent during verification, which takes significantly less time than generation. As shown in Tab.3, the lowest verification-to-generation ratio is achieved with VGG16 on TinyImageNet, requiring only 1.3–1.8% of the generation time. Across all models, this ratio stays below 13%. In contrast, conventional PoL methods require verification times nearly equal to generation, with overheads reaching up to 98%. Furthermore, PoLO introduces only modest overhead during $\mathbb{P}$ generation by integrating watermarking into training. While this slightly extends runtime compared to baseline training, the increase is minimal and justified. In contrast, traditional PoL methods store intermediate models and datasets separately, keeping $\mathbb{P}$ generation time roughly equal to standard training time.

Takeaway (faster and privacy-preserving verification). With similar proof generation time, PoLO reduces verification time to as low as 1.3–1.8%, compared to up to 98% in conventional PoL schemes. By eliminating the need to retrain intermediate models, PoLO achieves efficient, scalable, and privacy-preserving verification.

5.2.3 Unforgeability. To evaluate resistance against forgery attacks, we conduct experiments using both OFA and WMA strategies. For consistency, identical computational resources are allocated across all attack scenarios. The legitimate PoLO-protected model embeds 2048-bit watermarks, while attackers attempted to embed illicit watermarks of varying sizes (2048, 1024, and 512 bits). We assess the effectiveness of these attacks based on three key metrics: computational overhead, post-attack detection rate η of legitimate watermark, and main task accuracy Acc_{main}. For the OFA strategy, attackers utilized the RIGA embedding methodology to embed illicit watermarks while attempting to compromise legitimate ones. The attack process involved fine-tuning all model weights with a reduced learning rate [50] (0.1 × the original) to simultaneously embed the illicit watermark and disrupt the legitimate one. For the WMA strategy, attackers adopted a more precise strategy by constructing specific weights based on their illicit watermark and key parameters, then selectively replacing targeted weights in the model with these computed values.

Fig.4 shows the impact of PoLO forgery attacks across different shard stages, while Fig.7 in **Appendix** E provides additional results for illicit watermark sizes of 1024 and 512 bits. The watermark embedding speed (shard changing rate) is sensitive to the learning rate. Fig.4 shows that the shard indices lines (vertical blue lines) vary in density every 30 epochs due to the scheduled learning rate decay by a factor of ten [51]. Models under WMA exhibit noticeable instability, with Acc_{main} fluctuating significantly during the attack. This degradation causes the forged proof $\hat{\mathbb{P}}$ to fail verification. In contrast, OFA maintains a comparable Acc_{main} to that of the legitimate PoLO-protected model.

However, Tab.4 shows that OFA is highly resource-intensive, especially for models with high $|s|$. For instance, on VGG16 with TinyImageNet, OFA takes nearly five times longer than generating a legitimate PoLO. Despite the high cost, the legitimate watermark remains largely intact, with detection rates (η) ranging from 64.27% to 98.47%, making OFA-based forgery both costly and ineffective. In contrast, WMA incurs much lower computational overhead–ranging from 11.32 to 344.94 seconds–compared to the 990.74 to 25,442.66 seconds required for legitimate PoLO proof generation. However, it is equally ineffective. The detection rate of legitimate watermarks remains consistently above 90%, reaching as high as 99.81%. As shown in Fig.4, the resulting performance instability further indicates that WMA fails to compromise watermark integrity, rendering it an ineffective PoLO forgery method.

These results reflect the intrinsic nature: OFA demands full fine-tuning across epochs, leading to costs comparable to or exceeding legitimate training, which undermines its economic viability. Attackers face a trade-off between effectiveness and efficiency: increasing fine-tuning enhances watermark removal but inflates cost, while reducing it saves resources at the expense of attack success.

Table 4: The time consumption, legitimate watermark detection rate η, and Acc_{main} of launching attacks under different watermark sizes.

Watermark **size**(bits)	**PoLO forgery attacks**		CIFAR-10 AlexNet	CIFAR-10 ResNet18	CIFAR-100 ResNet18	CIFAR-100 WideResNet	TinyInageNet VGG16	TinyInageNet ResNet34	AG News TextCNN	AG News MiniBert
2048	Time(s)	PoLO	990.74	1618.78	1707.59	9357.91	25051.72	7332.32	1070.12	2573.38
		launch OFA	1696.91	2723.17	2381.83	15498.77	113771.31	30420.92	2275.26	7954.11
		launch WMA	56.51	62.04	64.37	218.91	344.94	158.55	11.33	30.46
	η(%)	PoLO	99.24	99.02	99.07	99.07	99.46	99.71	99.21	99.41
		launch OFA	93.62	77.82	77.52	96.83	86.37	75.27	69.84	78.09
		launch WMA	98.31	94.17	94.75	99.78	95.78	95.75	91.48	96.61
	Acc_{main}(%)	PoLO	89.22	91.89	74.81	72.83	73.00	72.46	91.04	91.88
		launch OFA	87.47	91.72	70.00	69.84	73.91	71.94	90.37	91.45
		launch WMA	47.52	80.37	36.84	46.56	43.25	39.81	58.69	55.20
1024	Time(s)	PoLO	1066.78	1861.62	1681.42	8968.8	24934.03	7238.95	1067.23	2687.89
		launch OFA	1136.86	2106.95	1802.52	9355.78	61437.63	17808.86	1355.37	4545.51
		launch WMA	52.21	62.98	63.66	208.04	343.46	148.24	11.32	30.33
	η(%)	PoLO	99.02	99.12	99.51	92.21	100.00	100.00	99.22	99.32
		launch OFA	96.93	82.28	75.91	98.43	92.23	84.11	73.52	64.53
		launch WMA	97.01	94.34	95.41	99.81	95.08	94.21	92.69	96.11
	Acc_{main}(%)	PoLO	88.94	91.96	74.93	72.96	73.33	73.31	90.94	92.06
		launch OFA	87.45	91.68	69.15	69.38	73.65	72.00	90.36	91.62
		launch WMA	39.07	79.30	34.35	40.52	45.31	31.09	61.19	58.96
512	Time(s)	PoLO	993.12	1783.50	1798.00	8433.43	25442.66	6775.54	1051.52	2486.51
		launch OFA	1077.32	1839.35	1967.88	9369.91	61578.36	18359.39	1355.33	4575.48
		launch WMA	52.71	66.92	63.92	206.76	341.63	144.61	11.39	30.42
	η(%)	PoLO	99.20	99.22	99.21	99.41	100.00	100.00	100.00	99.81
		launch OFA	96.98	75.03	75.99	98.47	84.24	63.52	79.67	64.27
		launch WMA	96.35	93.25	92.84	99.79	96.08	93.95	91.15	93.47
	Acc_{main}(%)	PoLO	89.54	92.22	75.18	72.64	73.91	73.22	90.65	91.82
		launch OFA	87.49	91.80	69.61	69.41	73.76	72.17	90.38	91.60
		launch WMA	40.50	79.51	38.88	40.86	37.97	38.41	65.39	65.19

• For launching attacks, a smaller η indicates greater attack effectiveness against the legitimate watermark, while a higher Acc_{main} reflects better fidelity to the main task performance (i.e., less impact on accuracy).

WMA avoids training entirely by directly altering weights, but suffers from imprecise targeting of watermark positions and destabilizing effects on Acc_{main} due to untrained modifications.

Takeaway (robust against forgery with practical cost barriers). Both OFA and WMA fail to remove PoLO's legitimate watermark–OFA incurs up to 5× the proof generation cost with detection rates still between 64.27% and 98.47%, while WMA is cheaper but also ineffective, with detection rates consistently above 90%.

6 CONCLUDING REMARKS

We introduced PoLO, a unified framework that integrates PoL and PoO through chained watermarking, addressing key limitations of existing approaches. In response to **RQ1**, we leveraged PoO to achieve PoL by embedding evolving watermarks throughout training, ensuring ownership is persistently linked to the entire training process rather than just the final model. For **RQ2**, we replaced gradient-based PoL proofs with cryptographic hash chains, making the proof tamper-resistant and resilient to fine-tuning, pruning, removal, and overlap attacks. Unlike traditional methods that expose training data or require costly recomputation, PoLO supports efficient, privacy-preserving verification. Experiments show that PoLO achieves **99%** watermark detection accuracy, reduces verification overhead to **1.5–10%** of conventional PoL methods, and requires **1.1–4×** more effort to forge than to generate legitimately–while retaining over **90%** detection accuracy even after attacks. These results demonstrate PoLO's effectiveness as a secure, efficient, and privacy-aware solution for model verification.

REFERENCES

[1] Joost Verbraeken, Matthijs Wolting, Jonathan Katzy, Jeroen Kloppenburg, Tim Verbelen, and Jan S Rellermeyer. A survey on distributed machine learning. *ACM Computing Surveys (CSUR)*, 53(2):1–33, 2020.

[2] Peng Sun, Xu Chen, Guocheng Liao, and Jianwei Huang. A profit-maximizing model marketplace with differentially private federated learning. In *IEEE Conference on Computer Communications (INFOCOM)*, pages 1439–1448, 2022.

[3] Guangsheng Yu, Xu Wang, Caijun Sun, Qin Wang, Ping Yu, Wei Ni, and Ren Ping Liu. Ironforge: An open, secure, fair, decentralized federated learning. *IEEE Transactions on Neural Networks and Learning Systems (TNNLS)*, 36(1):354–368, 2025.

[4] Jiasi Weng, Jian Weng, Chengjun Cai, Hongwei Huang, and Cong Wang. Golden grain: Building a secure and decentralized model marketplace for mlaas. *IEEE Transactions on Dependable and Secure Computing (TDSC)*, 19(5):3149–3167, 2022.

[5] Lingchen Zhao, Qian Wang, Cong Wang, Qi Li, Chao Shen, and Bo Feng. Veriml: Enabling integrity assurances and fair payments for machine learning as a service. *IEEE Transactions on Parallel and Distributed Systems (TPDS)*, 2021.

[6] Guangsheng Yu, Yanna Jiang, Qin Wang, Xu Wang, Baihe Ma, Caijun Sun, Wei Ni, and Ren Ping Liu. Split unlearning. *ACM SIGSAC Conference on Computer and Communications Security (CCS)*, 2025.

[7] Yuechen Xie, Jie Song, Mengqi Xue, Haofei Zhang, Xingen Wang, Bingde Hu, Genlang Chen, and Mingli Song. Dataset ownership verification in contrastive pre-trained models. *International Conference on Learning Representations (ICLR)*, 2025.

[8] Hengrui Jia, Mohammad Yaghini, Christopher A Choquette-Choo, Natalie Dullerud, Anvith Thudi, Varun Chandrasekaran, and Nicolas Papernot. Proof-of-learning: Definitions and practice. In *IEEE Symposium on Security and Privacy (SP)*, pages 1039–1056. IEEE, 2021.

Table 5: List of Notations

Notation	Definition
$\mathbb{P}$	Proof-of-Anything (PoX) proof
$\mathcal{P}/\mathcal{V}/\mathcal{A}$	Prover/Verifier/Attacker
Θ	Irreversible function with input χ
Ψ	Verification condition required for proof validity
t	The t-th training epoch out of T total epochs
W_t	Model weights at epoch t
$\mathbf{B}_t$	Data batch sampled from dataset $\mathcal{D}$ at epoch t
H_t, A_t	Signature and auxiliary info at epoch t
Λ	Ownership information embedded in the model
s_x	The x-th shard out of S shards
T_x	Set of epochs in s_x, $\bigcup_{x=1}^{S} T_x = \{1, 2, \ldots, T\}$, and $T_x \cap T_{x'} = \emptyset$ for $x \neq x'$
t_x	The t-th training epoch, which belongs to shard s_x
$\tilde{t}_x$	The t-th training epoch, which is the final epoch of shard s_x
W_x	The checkpoint model of shard s_x
Λ_x	Embedded watermark in s_x
k_x	Embedding key for Λ_x
Y	Selection matrix for weights in W_{t_x}, $t_x \in T_x$, used for Λ_x
Z	Selection matrix for weights subject to the DP obfuscation
μ_x	Unique secret nonce used in hash computation
η	Watermark detection rate
$\mathbb{E}(\cdot)/\mathbb{C}(\cdot)$	Watermark embedding/extraction function
$l_w(\cdot)$	The loss function for main task
$l_\Lambda(\cdot)$	The loss function for embedding ownership information
ε	The privacy budget of differential privacy (DP)

[9] Rui Zhang, Jian Liu, Yuan Ding, Zhibo Wang, Qingbiao Wu, and Kui Ren. "adversarial examples" for proof-of-learning. In *IEEE Symposium on Security and Privacy (SP)*, 2022.

[10] Congyu Fang, Hengrui Jia, Anvith Thudi, Mohammad Yaghini, Christopher A. Choquette-Choo, Natalie Dullerud, Varun Chandrasekaran, and Nicolas Papernot. Proof-of-learning is currently more broken than you think. In *IEEE European Symposium on Security and Privacy (EuroSP)*, 2023.

[11] Zishuo Zhao, Zhixuan Fang, Xuechao Wang, Xi Chen, and Yuan Zhou. Proof-of-learning with incentive security. *arXiv preprint arXiv:2404.09005*, 2024.

[12] Kasra Abbaszadeh, Christodoulos Pappas, Jonathan Katz, and Dimitrios Papadopoulos. Zero-knowledge proofs of training for deep neural networks. In *ACM SIGSAC Conference on Computer and Communications Security (CCS)*, page 4316–4330, 2024.

[13] Markus Jakobsson and Ari Juels. *Proofs of Work and Bread Pudding Protocols (Extended Abstract)*, pages 258–272. 1999.

[14] Juan Garay, Aggelos Kiayias, and Nikos Leonardos. The bitcoin backbone protocol: Analysis and applications. *Annual International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT)*, pages 281–310, 2015.

[15] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. *Satoshi Nakamoto*, 2008.

[16] Vitalik Buterin et al. Ethereum white paper. *GitHub repository*, 1:22–23, 2013.

[17] Peter Gaži, Aggelos Kiayias, and Dionysis Zindros. Proof-of-stake sidechains. In *IEEE Symposium on Security and Privacy (SP)*, pages 139–156, 2019.

[18] Aggelos Kiayias, Alexander Russell, Bernardo David, and Roman Oliynykov. Ouroboros: A provably secure proof-of-stake blockchain protocol. In *Annual International Cryptology Conference (CRYPTO)*, pages 357–388, 2017.

[19] Lin Chen, Lei Xu, Nolan Shah, Zhimin Gao, Yang Lu, and Weidong Shi. On security analysis of proof-of-elapsed-time (poet). In *Stabilization, Safety, and Security of Distributed Systems*, pages 282–297, 2017.

[20] Huibo Wang, Guoxing Chen, Yinqian Zhang, and Zhiqiang Lin. Multi-certificate attacks against proof-of-elapsed-time and their countermeasures. In *Network and Distributed System Security (NDSS) Symposium*, 2022.

[21] Stefan Dziembowski, Sebastian Faust, Vladimir Kolmogorov, and Krzysztof Pietrzak. Proofs of space. In *Annual International Cryptology Conference (CRYPTO)*, pages 585–605, 2015.

[22] Tal Moran and Ilan Orlov. Simple proofs of space-time and rational proofs of storage. In *Annual International Cryptology Conference (CRYPTO)*, pages 381–409, 2019.

[23] Parinya Ekparinya, Vincent Gramoli, and Guillaume Jourjon. The attack of the clones against proof-of-authority. In *Network and Distributed System Security (NDSS) Symposium*, 2020.

[24] Qin Wang, Rujia Li, Qi Wang, Shiping Chen, and Yang Xiang. Exploring unfairness on proof of authority: Order manipulation attacks and remedies. In *ACM on Asia Conference on Computer and Communications Security (AsiaCCS)*, pages 123–137, 2022.

[25] Gefei Tan, Adrià Gascón, Sarah Meiklejohn, Mariana Raykova, Xiao Wang, and Ning Luo. Founding zero-knowledge proofs of training on optimum vicinity. *Cryptology ePrint Archive*, 2025.

[26] Yilin Sai et al. Is your AI truly yours? leveraging blockchain for copyrights, provenance, and lineage. *arXiv preprint arXiv:2404.06077*, 2024.

[27] Xinyun Chen, Wenxiao Wang, Chris Bender, Yiming Ding, Ruoxi Jia, Bo Li, and Dawn Song. REFIT: A unified watermark removal framework for deep learning systems with limited data. In *ACM Asia Conference on Computer and Communications Security (AsiaCCS)*, page 321–335, 2021.

[28] Huajie Chen, Tianqing Zhu, Chi Liu, Shui Yu, and Wanlei Zhou. High-frequency matters: Attack and defense for image-processing model watermarking. *IEEE Transactions on Services Computing (TSC)*, 17(4):1565–1579, 2024.

[29] Jie Zhang, Dongdong Chen, Jing Liao, Zehua Ma, Han Fang, Weiming Zhang, Huamin Feng, Gang Hua, and Nenghai Yu. Robust model watermarking for image processing networks via structure consistency. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 2024.

[30] Aiwei Liu, Leyi Pan, Yijian Lu, Jingjing Li, Xuming Hu, Xi Zhang, Lijie Wen, Irwin King, Hui Xiong, and Philip Yu. A survey of text watermarking in the era of large language models. *ACM Computing Surveys (CSUR)*, 57(2):1–36, 2024.

[31] Xiyang Luo, Yinxiao Li, Huiwen Chang, Ce Liu, Peyman Milanfar, and Feng Yang. Dvmark: A deep multiscale framework for video watermarking. *IEEE Transactions on Image Processing (TIP)*, 2023.

[32] Yusuke Uchida, Yuki Nagai, Shigeyuki Sakazawa, and Shin'ichi Satoh. Embedding watermarks into deep neural networks. In *International Conference on Multimedia Retrieval (ICML)*, page 269–277, 2017.

[33] Peizhuo Lv, Pan Li, Shengzhi Zhang, Kai Chen, Ruigang Liang, Hualong Ma, Yue Zhao, and Yingjiu Li. A robustness-assured white-box watermark in neural networks. *IEEE Transactions on Dependable and Secure Computing (TDSC)*, 2023.

[34] Tianhao Wang and Florian Kerschbaum. Riga: Covert and robust white-box watermarking of deep neural networks. In *Proceedings of the Web Conference (WWW)*, 2021.

[35] Bowen Li, Lixin Fan, Hanlin Gu, Jie Li, and Qiang Yang. Fedipr: Ownership verification for federated deep neural network models. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 45(4):4521–4536, 2022.

[36] Shuo Shao, Wenyuan Yang, Hanlin Gu, Zhan Qin, Lixin Fan, and Qiang Yang. Fedtracker: Furnishing ownership verification and traceability for federated learning model. *IEEE Transactions on Dependable and Secure Computing (TDSC)*, 2024.

[37] Xinyun Chen, Wenxiao Wang, Chris Bender, Yiming Ding, Ruoxi Jia, Bo Li, and Dawn Song. Refit: A unified watermark removal framework for deep learning systems with limited data. In *ACM Asia Conference on Computer and Communications Security (AsiaCCS)*, page 321–335, 2021.

[38] Wenwen Gu. Watermark removal scheme based on neural network model pruning. In *International Conference on Machine Learning and Natural Language Processing (MLNLP)*, page 377–382, 2023.

[39] Tianhao Wang and Florian Kerschbaum. Attacks on digital watermarks for deep neural networks. In *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2019.

[40] Yiyang Luo, Ke Lin, Chao Gu, Jiahui Hou, Lijie Wen, and Ping Luo. Lost in overlap: Exploring watermark collision in LLMs. *arXiv preprint arXiv:2403.10020*, 2024.

[41] Haozhe Chen, Hang Zhou, Jie Zhang, Dongdong Chen, Weiming Zhang, Kejiang Chen, Gang Hua, and Nenghai Yu. Perceptual hashing of deep convolutional neural networks for model copy detection. *ACM Transactions on Multimedia Computing, Communications and Applications (TOMM)*, 19(3):1–20, 2023.

[42] Wenqi Wei and Ling Liu. Gradient leakage attack resilient deep learning. *IEEE Transactions on Information Forensics and Security (TIFS)*, 17:303–316, 2022.

[43] Guangsheng Yu, Qin Wang, Caijun Sun, Lam Duc Nguyen, HMN Bandara, and Shiping Chen. Maximizing NFT incentives: References make you rich. *arXiv preprint arXiv:2402.06459*, 2024.

[44] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Advances in Neural Information Processing Systems (NeuIPS)*, 2012.

[45] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016.

[46] Sergey Zagoruyko. Wide residual networks. *arXiv preprint arXiv:1605.07146*, 2016.

[47] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[48] Yoon Kim. Convolutional neural networks for sentence classification. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1746–1751, 2014.

[49] Jacob Devlin. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

Algorithm 1 PoLO generation

Require: $\mathbb{D}$ (dataset), μ/id (secret nonce and identity information), ε (privacy budget of DP), t_{max} (maximum training iterations), key: Z (Selection matrix for weights subject to the DP method), η_G (the threshold for successful watermark embedding), l_Λ (watermark embedding regularizers).
Ensure: PoLO proof $\mathbb{P}$

W_0, μ, x ▷ initial model, and shard number.
Initialize $\Lambda_1, k_1 = \mathsf{WMGen}(\mathcal{H}_1), \mathsf{KeyGen}(\mathcal{H}_1)$,
  where $\mathcal{H}_x = \mathbb{H}(W_x, x+1, \mu, id_{\mathcal{P}})$, and $x = 1$.
$Y = \mathsf{WMPosition}(\mu)$. ▷ selection matrix for weights used for Λ
while ($\neg$ converging) $\vee$ (t_{max} is reached) **do** ▷ loop until convergence
  $W_{t_x} = \arg\min_W (l_w(W) + \lambda l_\Lambda(W)) \mid_{W_0 = W_{x-1}} + \delta W$
    where $\delta W = \mathbb{E}(W_{x-1}, \Lambda_x, k_x, Y)$ ▷ update the model weights
  $\eta = 1 - \frac{\sum_i^n (\mathbb{C}(W_{t_x}, Y, k_x)[i] \neq \Lambda_x[i])}{n}$ ▷ update watermark detection rate
  if $\eta \geq \eta_G$ **then**
    $W_x = W_{t_x} + \mathbb{DP}(\varepsilon, Z, W_{t_x})$ ▷ apply DP noise
    Save W_x
    $\Lambda_{x+1}, k_{x+1} = \mathsf{WMGen}(\mathcal{H}_x), \mathsf{KeyGen}(\mathcal{H}_x)$
    x+ = 1
  else $W_{x-1} = W_{t_x}$ ▷ move to the next epoch within the current shard s_x
  end if
end while
RETURN $\mathbb{P} \in \{W_1, W_2, ..., W_S\}$

Algorithm 2 PoLO verification

Require: $\mathbb{P} \in \{W_1, W_2, ..., W_S\}$ (PoLO proof), μ/id (secret nonce and identity information), $\mathbb{D}^t$ (public test dataset), η_G(the threshold of similarly between the hashed and extracted watermarks).
Ensure: "**Fail**" or "**Success**".

Choose $i \in \{S-1, S-2, ..., 1\}$ ▷ $\mathcal{V}$ choose the shards to end the verification
$\mathcal{V}$ receives $\mathbb{P} \in \{W_1, W_2, ..., W_S\}$, and μ/id from $\mathcal{P}$
$Y = \mathsf{WMPosition}(\mu)$. ▷ recover the selection matrix for watermark
for $x \leftarrow S, S-1, ..., i$ **do** ▷ verification loop
  $\hat{Acc}_{main} = \mathsf{Test}(W_x, \mathbb{D}^t)$ ▷ test the model main task accuracy
  $\Lambda_x, k_x = \mathsf{WMGen}(\mathcal{H}_{x-1}), \mathsf{KeyGen}(\mathcal{H}_{x-1})$
    where $\mathcal{H}_{x-1} = \mathbb{H}(W_{x-1}, x, \mu, id_{\mathcal{P}})$ ▷ calculate the watermark from the previous model by $\mathbb{H}(\cdot)$
  $\hat{\Lambda}_x = \mathbb{C}(W_x, Y, k_x)$ ▷ extract the current watermark
  $\eta_x = 1 - \frac{\sum_i^n (\hat{\Lambda}_x[i] \neq \Lambda_x[i])}{n}$ ▷ calculate the watermarks similarity
  if $\hat{Acc}_{main}$ does not meet the expectation **then**
    RETURN "**Fail**", **break**
  if $\eta_x < \eta_G$ **then**
    RETURN "**Fail**", **break**
end for
RETURN "**Success**"

[50] Yossi Adi, Carsten Baum, Moustapha Cisse, Benny Pinkas, and Joseph Keshet. Turning your weakness into a strength: Watermarking deep neural networks by backdooring. In *USENIX Security Symposium (USENIX Sec)*, 2018.

[51] Xidong Wu, Feihu Huang, Zhengmian Hu, and Heng Huang. Faster adaptive federated learning. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, volume 37, pages 10379–10387, 2023.

APPENDIX

A NOTATION

Tab.5 summaries the notations used throughout the paper.

B ALGORITHM OF THE FRAMEWORKS

The proof generation process and verification process of PoLO (§4.1–§4.2) can be found from **Algorithms** 1&2.

Proof generation. Algorithm 1 details the procedure for PoLO proof generation. The algorithm starts with an initial model W_0 and computes the first watermark Λ_1 and embedding key k_1 using a hash function $\mathbb{H}(\cdot)$, which incorporates the prover's identity $id_{\mathcal{P}}$, a secret verifier-provided nonce μ, and the shard index x.

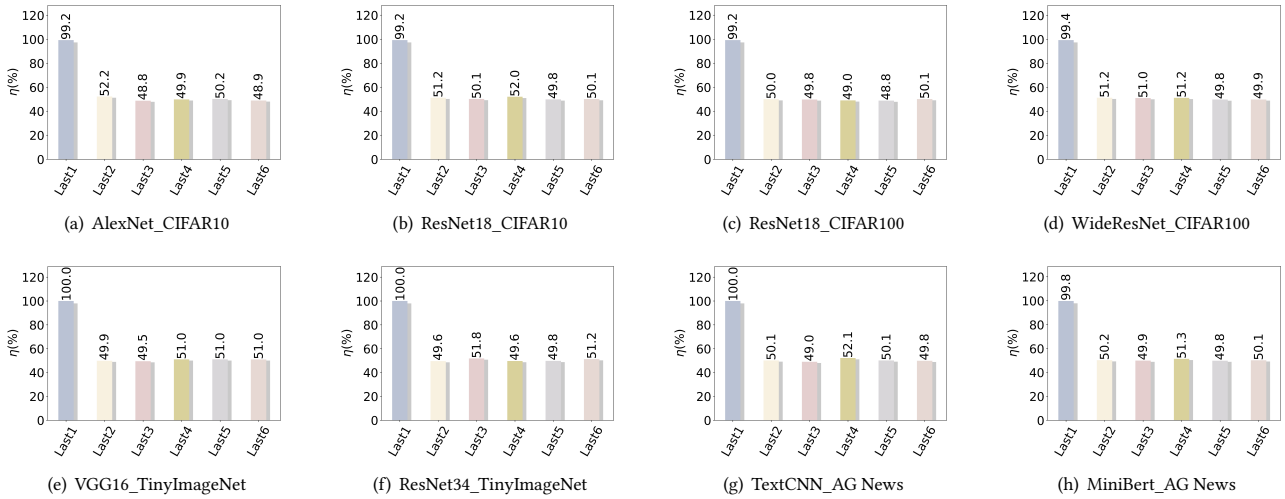
Training proceeds in a loop over epochs, where in each iteration, the model is updated through standard training combined with watermark embedding. The watermark Λ_x is embedded into selected weights of W_{t_x} at each epoch $t \in T$ using a secret embedding key k_x and a selection matrix Y. The watermark detection rate η is dynamically computed after each epoch to assess whether the embedded watermark has reached the desired robustness threshold η_G. Once satisfied, differential privacy protection is selectively applied to a subset of the non-watermarked weights using another selection matrix Z, and the resulting model W_x is stored as the final checkpoint of shard s_x. A new watermark Λ_{x+1} for the next shard is then computed using the DP-protected model W_x and the chained hashing mechanism, ensuring that each watermark is cryptographically linked to its predecessor. This process continues iteratively until the max training epoch t_{max} and watermark quality η_G are simultaneously satisfied.

In our implementation, we set $\eta_G = 0.99$, ensuring strong watermark robustness while maintaining training efficiency. The final output of the algorithm is a PoLO proof $\mathbb{P}$. This guarantees traceable and tamper-resistant proof of training effort and model ownership.

Proof verification. Algorithm 2 outlines the verification process of PoLO, which aims to validate both training efforts and model ownership by ensuring the integrity of the chained watermark sequence and the model's performance Acc_{main}. The verifier $\mathcal{V}$ operates at the shard level, rather than at the epoch level, enabling efficient and tamper-resistant assessment. The $\mathcal{V}$ initiates the verification process by selecting a stopping shard index $i \in \{S-1, S-2, ..., 1\}$ from the submitted PoLO proof $\mathbb{P}$, which contains a sequence of DP-protected models $\{W_1, W_2, ..., W_S\}$, and some information to generate the watermark Λ and embedding key k.

- **Main task accuracy validation.** The model W_x is evaluated on the public test dataset $\mathbb{D}^t$ to measure the main task accuracy $\hat{Acc}_{main}$. If the $\hat{Acc}_{main}$ does not meet the expectation, the verification fails, and the proof is rejected.
- **Watermark consistency verification.** $\mathcal{V}$ reconstructs the expected watermark Λ_x and the extraction key k_x (same as the embedding key) from the prior model checkpoint W_{x-1} using the hash function $\mathbb{H}(\cdot)$, and extracts the current watermark $\hat{\Lambda}_x$ from W_x using the extraction key k_x and selection key Y. It then computes the watermark similarity score η_x by calculating the Hamming similarity between Λ_x and $\hat{\Lambda}_x$. If $\eta_x < \eta_G$, the verification fails.

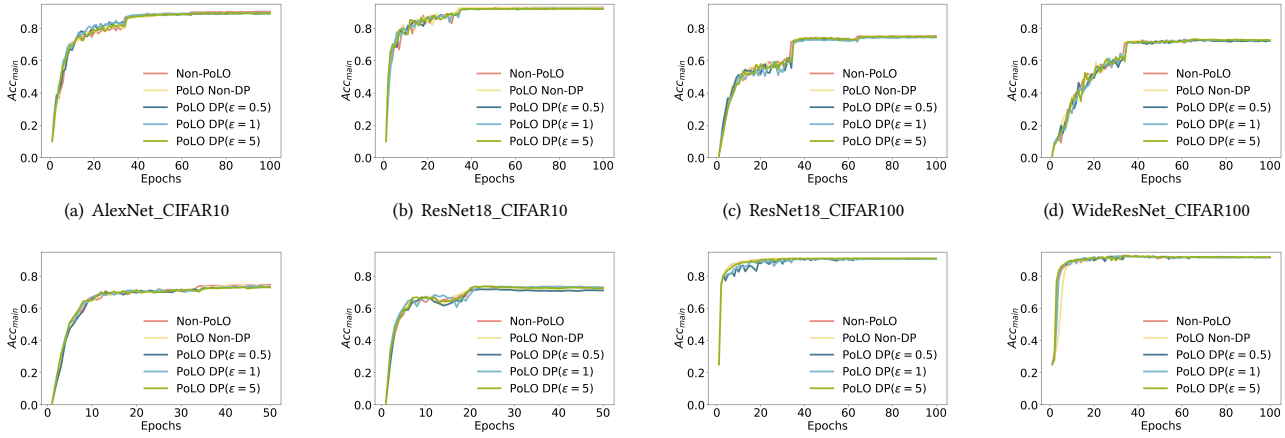
  This verification process is backward, allowing the $\mathcal{V}$ to check a sequence of earlier shards progressively (from $S-1$ to 1). This chained structure ensures that each watermark is cryptographically bound to its predecessor, preventing partial proof forgery or selective model manipulation. If all selected shards pass the checks, the proof is accepted and verification returns "**Success**". This mechanism demonstrates PoLO's resilience against tampering, ensuring verifiable training effort and ownership integrity while maintaining computational efficiency.



(a) AlexNet_CIFAR10 (b) ResNet18_CIFAR10 (c) ResNet18_CIFAR100 (d) WideResNet_CIFAR100

(e) VGG16_TinyImageNet (f) ResNet34_TinyImageNet (g) TextCNN_AG News (h) MiniBert_AG News

Figure 5: The η of the last 1-6 intermediate shards' watermark attempts to be extracted from the last shard models.



(a) AlexNet_CIFAR10 (b) ResNet18_CIFAR10 (c) ResNet18_CIFAR100 (d) WideResNet_CIFAR100

(e) VGG16_TinyImageNet (f) ResNet34_TinyImageNet (g) TextCNN_AG News (h) MiniBert_AG News

Figure 6: Comparison of the main accuracy across different architectures and datasets with and without PoLO.

C EFFORT IRRECOVERABILITY FOR ATTACKERS

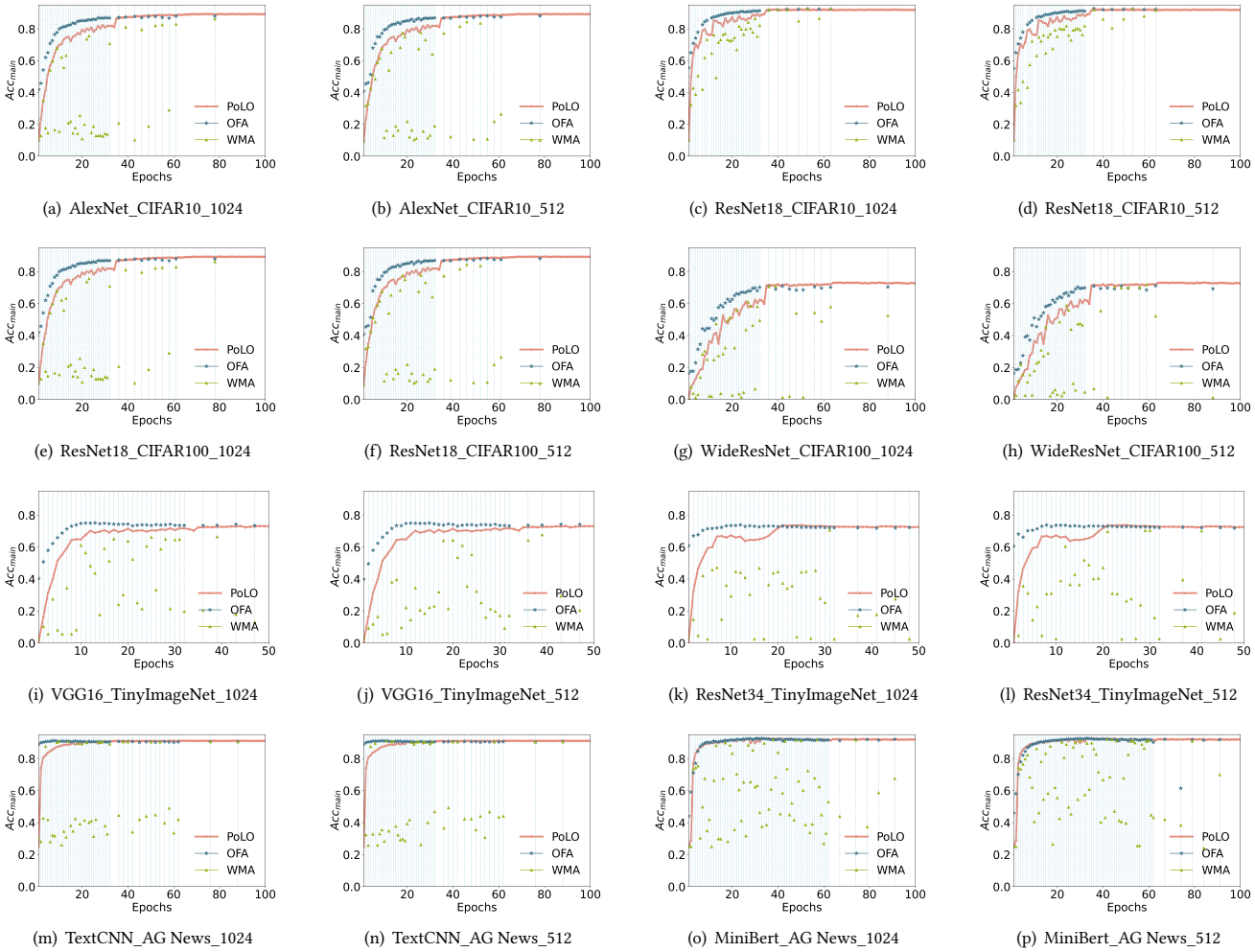
In the generation of PoLO, each new watermark shard completely overwrites the watermark embedded in the previous shard, rather than layering additional watermark information. This design ensures that watermarks from earlier shards become irretrievable from the final model—even when the same embedding keys (k_x) and selection positions are reused.

To evaluate this overwrite effect, we attempt to extract the last 1–6 shard watermarks from the final trained model using the same embedding parameters. A watermark detection rate η close to 50%–equivalent to random guessing—indicates that a watermark is no longer present in the model and thus unrecoverable. For experimental consistency, we used fixed embedding positions and a uniform 512-bit watermark size across all shards. As shown in Fig.5, watermark detection for the most recent shard (last-1) remains near-perfect ($\eta \approx 100\%$) across different models and datasets, confirming successful extraction. However, detection rates drop sharply to around 50% for earlier shards (last-2 through last-6), demonstrating that later training fully overwrites prior watermark information.

This overwrite mechanism plays a vital role in PoLO's security. Even if an attacker gains access to the watermark embedding key k and position Y in the released model, they can at most forge ownership of that specific model. They cannot extract or claim earlier wwatermarks—hence, they cannot prove the training effort. This makes PoLO resistant to forgery and enables the training effort itself to act as auxiliary evidence for ownership, further strengthening PoO verification.

D FIDELITY

Our fidelity study examines how PoLO affects various models, specifically focusing on the impact of watermarks and DP noise on model performance. We evaluate multiple neural network architectures (AlexNet, ResNet18, WideResNet, VGG16, ResNet34, TextCNN, and MiniBert) across diverse datasets (CIFAR-10, CIFAR-100, TinyImageNet, and AG News). For PoLO settings, we employ a 512-bit



(a) AlexNet_CIFAR10_1024 (b) AlexNet_CIFAR10_512 (c) ResNet18_CIFAR10_1024 (d) ResNet18_CIFAR10_512

(e) ResNet18_CIFAR100_1024 (f) ResNet18_CIFAR100_512 (g) WideResNet_CIFAR100_1024 (h) WideResNet_CIFAR100_512

(i) VGG16_TinyImageNet_1024 (j) VGG16_TinyImageNet_512 (k) ResNet34_TinyImageNet_1024 (l) ResNet34_TinyImageNet_512

(m) TextCNN_AG News_1024 (n) TextCNN_AG News_512 (o) MiniBert_AG News_1024 (p) MiniBert_AG News_512

Figure 7: The Acc_{main} comparison of the two $\mathbb{P}$ forge attacks with baseline - PoLO. The number followed by the dataset is the watermark size. In each subplot, the vertical lines perpendicular to the x-axis represent the completion times for each shard.

watermark size and test different DP noise levels ($\varepsilon = 0.5, 1, 5$) as well as no DP noise, while baseline models are trained without watermarks or DP noise.

Fig.6 presents a thorough comparative analysis of performance between PoLO and baseline models spanning various architectures and datasets. Our experimental findings reveal that incorporating watermarks and DP noise has minimal impact on model performance relative to the baselines. The most notable performance impact is observed in the ResNet34 architecture trained on TinyImageNet[1] with $\varepsilon = 0.5$, yet this decline remained modest at under 2%. Remarkably, all other model configurations maintained performance within 0.5% of their baseline counterparts, regardless of the DP noise level applied.

This minimal performance impact can be attributed to the abundance of local optima within neural networks. When watermarks and DP noise are embedded, the model's optimization process may simply converge to a different yet equally effective local optimum, maintaining near-optimal performance. The comprehensive fidelity evaluation demonstrates that our chained watermarking approach in PoLO preserves model utility while successfully implementing PoL, thus establishing PoLO as a viable and practical scheme.

[1]We use the top-5 accuracy as the metric to measure the Acc_{main} for TinyImageNet.

E UNFORGEABILITY (MORE)

Fig.7 shows how OFA and WMA attacks affect the model Acc_{main} when forging PoLO proofs using illicit watermarks of 1024 and 512 bits. Our experimental analysis reveals an intriguing pattern: regardless of the illicit watermark size, OFA-based forgery attempts initially yield higher Acc_{main} than legitimate PoLO models during early shard stages. However, this advantage diminishes in later stages, where OFA models' Acc_{main} converges to match that of legitimate PoLO models. This behavior can be attributed to OFA's fine-tuning approach. During early stages when the model hasn't fully converged, fine-tuning across all weights can enhance model Acc_{main}. As the model approaches convergence in later stages, additional fine-tuning ceases to yield performance improvements.

WMA attacks, however, exhibit markedly different characteristics, producing significant Acc_{main} fluctuations post-attack. WMA

attacks result in severe degradation of model accuracy, which consequently causes the forged PoLO $\hat{\mathbb{P}}$ proof to fail verification. This instability stems from WMA's direct weight modification strategy, where it calculates and substitutes weight values based on the illicit watermark. The model's performance stability depends critically on how well these modified weights integrate with the unchanged portions of the network. When integration is successful, performance remains stable; when it fails, the model's functionality deteriorates substantially, leading to pronounced Acc_{main} variations. These dramatic fluctuations in Acc_{main} serve as a distinctive signature of WMA attack attempts.

F TAKEAWAYS: Q&A—SECURITY THREATS AND DEFENSES IN POLO

To enhance transparency and clarity, we address common threat scenarios against the PoLO framework and explain how each is mitigated by our design.

F.1 External Malicious Attackers

Q1: What if an attacker fine-tunes the final released model and claims it as theirs? (the OFA strategy)

A1: PoLO embeds watermarks across chained shards, meaning a valid claim requires proving ownership of all previous shards. As the attacker does not know the watermark's embedding position, they must fine-tune across a wide range of model layers. The process is computationally expensive, especially since the attacker not only inserts his watermark but also erases a legitimate one. The cost of such an attack is comparable to (exceeds) the cost of legitimate training, making it economically impractical.

Q2: What if an attacker blindly inserts their watermark into the model and claims ownership? (the WMA strategy)

A2: Randomly embedding a watermark without regard for model integrity is likely to degrade performance on the main task. During verification, the verifier uses a test dataset to validate model accuracy. Any performance degradation from unauthorized watermarking will result in failed verification.

Q3: What if the attacker tries to guess the embedding key k_x that aligns with the watermark Λ_x in shard s_x to forge a PoLO proof?

A3: Both the watermark and embedding key are deterministically derived from a cryptographic hash over the previous shard, identity information, and a verifier-provided secret nonce μ. This nonce is unknown to the attacker during training and only revealed at verification. Without knowing μ, the attacker cannot precompute the correct key or watermark, rendering attacks ineffective.

Q4: What if the attacker just wants to remove the watermark without embedding their own?

A4: In our setting, only models carrying valid PoO information are considered valuable and eligible for exchange in marketplaces or for earning rewards in incentive-driven distributed learning scenarios. If an attacker clones a legitimate model and removes its watermark, the model becomes unverifiable and thus unqualified for use or trade. Meanwhile, the original, watermark-protected model remains valid and publicly available, rendering the attacker's effort meaningless and unrewarded.

Q5: Can an attacker infer training data from public checkpoints using membership inference attacks?

A5: To mitigate inference attacks, we apply DP to the final model of each shard. A randomized subset of non-watermarked weights is perturbed with DP noise, making it statistically difficult to reverse-engineer the training dataset while preserving watermark integrity and model performance.

Q6: What if an attacker gains access to the embedding key k_S and selection matrix Y for the final (released) model?

A6: Even with full knowledge of k_S and Y, the attacker can at most forge ownership of the final model checkpoint. They cannot recover or claim watermarks from earlier shards, since each new shard overwrites the previous one at fixed positions. This prevents attackers from proving training efforts, which are cryptographically chained across shards. In fact, this property allows PoLO to use effort proof as auxiliary evidence for ownership, reinforcing PoO. Moreover, it enables clean ownership transfer: a new owner can append a new shard with a fresh watermark, extending the chain.

F.2 Rational (Self-Interested) Provers

Q7: What if a prover tries to forge k_x to finalize a shard without performing actual training, inflating the number of claimed shards for higher rewards?

A7: Since k_x is deterministically derived from a hash involving a verifier-provided secret nonce μ (alongside previous weights and identity), the prover cannot arbitrarily select k_x without knowing or controlling μ. This linkage ensures that a valid watermark can only be embedded through legitimate training progression.

What if a prover picks an "inactive" layer for watermark embedding that doesn't affect performance, letting the watermark be extracted regardless of training quality?

A8: To prevent this, we do not allow provers to choose the embedding position or secret nonce arbitrarily. Instead, the verifier—considered to be honest-but-curious—generates and provides the nonce at registration time, which is then verifiably included in the hash computation. This design mimics a regulated environment, akin to obtaining a business license before model training begins, ensuring all work is bound to an approved watermarking path.

F.3 Honest-but-Curious Verifier

Q9: Can a verifier infer sensitive training data from the checkpoints provided during verification?

A9: No. The final model of each shard is protected via DP, which adds calibrated noise to non-watermarked weights. This ensures that even an honest-but-curious verifier is hard to execute membership inference attacks or extract sensitive training data.

F.4 Watermark schemes

Q10: Why PoLO uses embedded watermarks instead of backdoor-based watermarks

A10: PoLO adopts watermarking methods that embed ownership information directly into model weights, rather than relying on backdoor-based triggers. This is crucial for ensuring that watermark embedding is inherently tied to the training process—allowing watermark detection rates to reflect actual training effort. Backdoor-based schemes decouple watermarking from training. An attacker

can implant a trigger into a pre-trained model without performing meaningful training, making forgery cheap and breaking PoL guarantees. In contrast, embedded watermarking requires legitimate weight updates to achieve detectable watermark signals, ensuring that only genuine training yields a valid proof. This tight integration makes it computationally impractical to forge ownership or training effort without investing comparable resources, preserving PoLO's core security and accountability goals.

G EXTENSION: COLLABORATIVE RELAY TRAINING ACROSS MULTIPLE USERS

PoLO's chained watermarking framework naturally extends to collaborative relay training scenarios, such as split learning (SL) [6], where a model is sequentially trained by multiple users—common in research consortia, cross-institutional collaborations, and multi-party development pipelines. In these settings, each participant must be individually accountable for their training contribution, while the final model owner must demonstrate legitimate inheritance of all prior efforts.

A real-world use case arises in collaborative medical AI development, where hospitals contribute to training a shared diagnostic model on sensitive patient data. For example, Hospital A trains a base model on its dataset and passes it to Hospital B, which continues training on a different cohort, and so on. When the model is eventually deployed or sold, the chain of training must be provably authentic and tamper-resistant, and each hospital's contribution should be fairly attributed.

In the multi-user scenario, each participant $\mathcal{P}_m$ is responsible for training a specific shard s_x and embedding a user-specific watermark $\Lambda_{x,m}$. This watermark and its corresponding embedding key $k_{x,m}$ are deterministically derived using the participant's identity and prior shard information:

$$\begin{aligned} \Lambda_{x,m},\ k_{x,m} = \mathsf{WMGen}(\mathcal{H}_{x,m}),\ \mathsf{KeyGen}(\mathcal{H}_{x,m}) \quad \text{with} \\ \mathcal{H}_{x,m} = \mathbb{H}(W_{x-1,m},\ x,\ \mu_m,\ id_{\mathcal{P}_m}), \end{aligned} \tag{7}$$

where $id_{\mathcal{P}_m}$ uniquely identifies the contributor, μ_m is the verifier-provided secret nonce assigned for each $\mathcal{P}_m$, and $W_{x-1,m}$ denotes the prior model state. While the watermarked weight selection matrix Y_m for participant $\mathcal{P}_m$ is derived from the same verifier-provided secret nonce μ_m by:

$$Y_m = \mathsf{WMPosition}(\mu_m). \tag{8}$$

This design ensures that both the watermark and embedding key are tightly bound to the participant's contribution and the training trajectory. Once the watermark $\Lambda_{x,m}$ reaches sufficient robustness and the model passes differential privacy protection, the resulting model state is finalized as W_s, serving as a verifiable checkpoint in the collaborative PoLO proof chain.

The design preserves core properties in multi-user settings:

- *Effort verifiability* is ensured per user through uniquely embedded watermark shards tied to individual training contributions.
- *Ownership traceability* is inherently linked to each contributor's identity and training history, enabling precise attribution across the collaborative process.
- *Tamper resistance* is enforced through cryptographic hash chaining across shards, making it infeasible for later participants to forge or overwrite earlier contributions.
- *Forgery prevention* is achieved by deterministically generating both the watermark and embedding key via WMGen and KeyGen, with all randomness derived from a secret nonce μ_m. This nonce remains hidden during training and is only revealed during verification, making it infeasible for adversaries to fabricate valid watermarks or keys without access to the original shard's secret.

Impact. Supporting multi-user training represents a major advancement over prior single-user PoL designs. In practical scenarios such as FL/SL, multi-tenant model development, or outsourced R&D pipelines, training is often distributed across multiple entities. Conventional PoL methods treat training as a single, continuous process, lacking identity binding or shard-level isolation. Consequently, contributions are aggregated into one undifferentiated proof tied to the final model, making it impossible to attribute individual efforts. Without cryptographic linkage between contributors and training segments, downstream parties can falsely claim full authorship by presenting the final model and regenerating a PoL proof—effectively hijacking earlier contributions.

By contrast, PoLO ensures that each party's contribution is verifiably recorded and cryptographically bound to their identity. The use of chained watermarking with identity-tagged keys prevents any participant from forging or overwriting previous contributions, thus making the proof tamper-resistant and fully auditable. This elevates PoLO from a proof mechanism for isolated training to a general-purpose provenance layer for collaborative machine learning, enabling new use cases in auditability, licensing, and cross-organization model governance.

ETHICS CONSIDERATIONS AND COMPLIANCE WITH OPEN SCIENCE POLICY

Our experiments are conducted on a local testing platform using open-source libraries and public datasets, without connecting to any external or live systems. These experiments do not involve any issues related to animals, human beings, the environment, healthcare, or military factors. As of this writing, these studies have not had real-world impact as they have only been utilized in our experiments. Consequently, we have addressed numerous ethical considerations in our experimental design, strictly adhering to the ethical principles outlined in the Menlo Report.

We confirm that authors are expected to openly share their artifacts by default if this paper is settled in academic venues. This aligns with the community's commitment to transparency, reproducibility, and collaborative advancement. We recommend that anyone involved in reviewing this paper who is interested in accessing the source codes should reach out to the authors via private email. The authors are willing to provide access upon request, while maintaining the anonymity and integrity of the review process.