UTS UNIVERSITY OF TECHNOLOGY SYDNEY

Training-Verifiable and Ownership-Protected Neural Networks

by Haiyu Deng

Thesis submitted in fulfilment of the requirements for the degree of

Doctor of Philosophy

under the supervision of Prof. Dr. Ren Ping Liu, Dr. Xu Wang

University of Technology Sydney
Faculty of Engineering and IT

February, 2026

CERTIFICATE OF ORIGINAL AUTHORSHIP

I, Haiyu Deng, declare that this thesis is submitted in fulfilment of the requirements for the award of Doctor of Philosophy, in the School of Electrical and Data Engineering, Faculty of Engineering and Information Technology at the University of Technology Sydney.

This thesis is wholly my own work unless otherwise referenced or acknowledged. In addition, I certify that all information sources and literature used are indicated in the thesis.

This document has not been submitted for qualifications at any other academic institution.

This research was supported by an Australian Government Research Training Program (RTP) Scholarship doi.org/10.82133/C42F-K220.

Signature: Production Note:
Signature removed prior to publication.

Date: February 22, 2026

Thesis by Compilation Declaration				
Paper Title	**List of Authors**	**Current Status**	**Student's Contribution**	**Related Chapter**
PoLO: Proof-of-Learning and Proof-of-Ownership at Once with Chained Watermarking	Haiyu Deng Yanna Jiang Guangsheng Yu Qin Wang Xu Wang Baihe Ma Wei Ni Ren Ping Liu	Submit to the 33rd ACM Conference on Computer and Communications Security (CCS).	Student takes around 60% of tasks to complete the proposed idea, survey, simulations, paper writing, and revision. Guangsheng Yu and Yanna Jiang contributes 30% of tasks, including the concept refining, paper writing and visualization. Other co-authors provided comments, and technical support to improve the paper.	Chapter 3
NNFMAC: A Neural Network Fingerprinting -based Model Authentication Code Scheme	Haiyu Deng Xu Wang Guangsheng Yu Wei Ni Ying He Tanzeela Altaf Ren Ping Liu	Published on ACM Transactions on Multimedia Computing, Communications, and Applications 10.1145/3778121.	Student takes around 90% of tasks to complete the proposed idea, survey, simulations, paper writing, and revision. Co-authors provided comments, and technical support to improve the paper.	Chapter 4
Client-Cooperative Split Learning	Haiyu Deng Yanna Jiang Guangsheng Yu Qin Wang Xu Wang Shiping Chen Wei Ni Ren Ping Liu	Submit to IEEE Transactions on Services Computing	Student takes around 90% of tasks to complete the proposed idea, survey, simulations, paper writing, and revision. Co-authors provided comments, and technical support to improve the paper.	Chapter 5
FedNIFW: Non-Interfering	Haiyu Deng Xiaocui Dang Yanna Jiang	Published on IEEE 23rd International	Student takes around 90% of tasks to complete the proposed idea, survey,	Chapter 5

Fragmented Watermarking for Federated Deep Neural Networ	Xu Wang Guangsheng Yu Wei Ni Ren Ping Liu	Conference on Trust, Security and Privacy in Computing and Communications (TrustCom).	simulations, paper writing, and revision. Co-authors provided comments, and technical support to improve the paper.	

	Name	Signature	Date
Student	Haiyu Deng	Production Note: Signature removed prior to publication.	5/12/2025
Co-authors	Ren Ping Liu	Production Note: Signature removed prior to publication.	5/12/2025
	Wei Ni	Production Note: Signature removed prior to publication.	5/12/2025
	Xu Wang	Production Note: Signature removed prior to publication.	5/12/2025
	Guangsheng Yu	Production Note: Signature removed prior to publication.	5/12/2025
	Baihe Ma	Production Note: Signature removed prior to publication.	5/12/2025
	Yanna Jiang	Production Note: Signature removed prior to publication.	5/12/2025
	Qin Wang	Production Note: Signature removed prior to publication.	5/12/2025
	Xiaocui Dang	Production Note: Signature removed prior to publication.	5/12/2025
	Shiping Chen	Production Note: Signature removed prior to publication.	5/12/2025
	Ying He	Production Note: Signature removed prior to publication.	5/12/2025
	Tanzeela Altaf	Production Note: Signature removed prior to publication.	5/12/2025

Abstract

As Artificial Intelligence (AI) gains popularity across wide applications, AI models are increasingly shared, outsourced, and commercialized, necessitating robust verification of both provenance (Proof-of-Learning, PoL) and ownership (Proof-of-Ownership, PoO).

In existing PoL schemes, verifiers must replay or retrain from the recorded snapshots, causing verification costs to scale with model size. This approaches the full cost of training, making the approach impractical for large or high-capacity models. Fingerprinting and watermarking are two primary methods for protecting the model PoO. However, current watermarking alters model weights and can degrade performance, while fingerprinting often merely verifies uniqueness or requires heavy computation. These limitations become even more prohibitive in collaborative learning such as Split Learning (SL) and Federated Learning (FL), which involve multiple parties and strict privacy requirements.

To address the drawback of existing PoL, this thesis proposes PoLO, a unified framework that simultaneously achieves PoL and PoO through chained watermarks. PoLO offers more efficient and privacy-preserving verification than gradient-based PoL methods that risk exposing training data, while providing stronger ownership guarantees than traditional watermark-based PoO methods. To address PoO limitations, a Neural Network Fingerprinting-based Model Authentication Code (NNFMAC) is presented to verify both model uniqueness and ownership without degrading performance. NNFMAC extracts trained model key weights, applies median-based binarization to derive a unique binary fingerprint used as a codebook, and encodes ownership via a newly designed index-based function, producing reliable authentication codes. Furthermore, PoO and label expansion are instantiated for multi-client cooperative SL (CliCooper), where model segments are distributed across training clients to furnish multi-party training verification and copyright attestation while preserving data privacy along the SL pipeline. For FL, a non-interfering fragmented watermarking strategy (FedNIFW) assigns client-specific segments of network layers for embedding, thereby protecting ownership and eliminating cross-client watermark conflicts.

The extensive experiments demonstrate 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 PoL. NNFMAC maintains model accuracy with no additional training overhead and demonstrates strong robustness against a range of attacks. Experiments further show strong privacy protection in SL: input sample reconstruction similarity drops from 0.5 to 0.03, and the PoLO-based instantiation enables secure, efficient multi-party training verification. In FL, client-specific segmented watermarking prevents cross-client conflicts without degrading model accuracy.

Acknowledgements

I am deeply grateful to my supervisor, Dr. Xu Wang, whose patient and generous mentorship has guided every step of this work. He discussed ideas at length, challenged weak arguments, and carefully edited drafts line by line. His steady encouragement helped me through academic setbacks, and his advice often arrived at exactly the moments I needed it most. Beyond academic, he has been a true friend—taking me to the badminton court and reminding me to keep a healthy balance between study and life. My sincere thanks go to my principal supervisor, Prof. Ren Ping Liu, for granting me the opportunity to pursue a Ph.D. at UTS and for his sustained support throughout the journey. He closely followed my progress, urged me to aim higher, and helped resolve practical constraints, including shortages of research hardware. His trust and vision created the conditions under which this thesis could be completed. I also wish to thank Prof. Wei Ni. His breadth of knowledge and rigorous academic standards have been a constant inspiration. From him, I learned to demand precision from my reasoning and clarity from my writing. His encouraging, constructive style of guidance greatly strengthened my confidence.

I am fortunate to have collaborated with Saber Yu, Yanna Jiang, and Qing Wang. Brainstorming with you guys, running experiments together, and co-writing papers were both efficient and enjoyable. I hope we will continue to work together in the future. I am indebted to my co-authors, Ying He, Baihe Ma, Tanzeela, and Prof. Shiping Chen, for insightful suggestions, careful revisions, and guidance on our joint papers. Your contributions sharpened the technical depth and clarity of my research.

I am profoundly grateful to my family. Your unwavering support and encouragement gave me the courage to pursue my studies in Australia. Whenever I faced difficulties in life, you offered patient advice and practical solutions that helped me overcome each hurdle. Your faith in me has been a constant source of strength, and this work would not have been possible without your love. I acknowledge support from the University of Technology Sydney's International Research Scholarship (IRS) and the China Scholarship Council (CSC), whose assistance enabled this research.

Finally, to everyone who offered feedback, encouragement, and practical help. Your support made this thesis possible and enriched my doctoral experience beyond measure.

Haiyu Deng
February 22, 2026
Sydney, Australia

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Deep neural networks (DNNs) are now pervasive across vision, language, and speech applications [1], [2], [3], and their computational demands continue to grow with architectural complexity. Model quality hinges on the richness and scale of the training data and on judicious model selection with well-tuned hyperparameters. Building datasets, from collection and preprocessing to manual annotation, is labor-intensive and entails substantial financial investment. Consequently, training a complete and effective AI model is memory-intensive [4]. Beyond carefully designing suitable architectures and curating labor-intensive labeled datasets, high-performance hardware such as GPUs [5], TPUs [6], and FPGAs [7] is required to enable fast and efficient training.

Training AI models is difficult and costly. Upon completion, practitioners must provide PoL to demonstrate that they have indeed performed the requisite optimization and that the process was correct—especially when training is outsourced to third parties [8]. To safeguard the intellectual property and copyright of model developers/owners [9], protective mechanisms such as watermarking [10] and fingerprinting [11] are required. When training data must remain private, collaborative learning paradigms, notably FL [12] and SL [13], are employed. FL and SL rely on multi-party collaborative training, where participant honesty and trustworthiness are difficult to ascertain. In practice, parties may be heterogeneous and partially untrusted, making their compliance ex ante unverifiable. In such settings, it is necessary not only to preserve data privacy but also to ensure training correctness and to protect model ownership.

The demand for PoL and PoO arises wherever models are trained, traded, or deployed across organisational boundaries. In Machine Learning as a Service (MLaaS) [14] and outsourced [15] training, PoL enables a customer to verify that a provider actually performed the promised optimisation—supporting integrity assurances, fair payment, and auditability without re-training the model from scratch [16], [17], [18], [19]. PoO underpins Intellectual Property (IP) enforcement in model marketplaces and licensing pipelines by establishing who owns a delivered artifact, deterring theft or resale, and resolving disputes (e.g., takedown or counter-notice workflows) [9], [20], [21], [22]. It also helps prevent false ownership claims and supports provenance/lineage requirements emerging in governance regimes for high-impact AI systems [23], [24], [25]. Beyond model IP, PoO/PoL principles extend to dataset or contribution attribution in modern pre-training pipelines and to raising the cost of model extraction attacks in commercial APIs [26], [27].

In collaborative learning, these guarantees become operational. In Split Learning, PoL can

certify that the compute-holding party executed training correctly without revealing raw data, while PoO can attribute rights to the institutions contributing private data or compute in regulated domains such as healthcare [28], [29]. In Federated Learning, PoO mechanisms (e.g., federated watermarking/fingerprinting) and PoL-style accountability support traceability, ownership verification, and incentive-compatible participation across many partially trusted clients and platforms [30], [31], [32].

1.1 Motivation

Machine Learning (ML) models have transitioned from academic artifacts to valuable digital assets [33], [34] traded in marketplaces [20], outsourced to third-party trainers, and deployed across collaborative platforms. Trust in this emerging economy hinges on two guarantees: that a model was legitimately trained PoL and that it is rightfully owned, PoO [26]. Treating these two problems separately leaves systemic gaps: adversaries can forge one proof at low cost and manipulate the other for illegitimate claims [35], [36], undermining attribution, reward, and recourse in real deployments. A unified approach is therefore required to attest both training effort and ownership in one coherent framework. However, existing PoL techniques struggle with cost and privacy [17], [35], [37]. Hash-based [19] or replay-style PoL [17] asks verifiers to retrain from intermediate snapshots, making verification scale with model size and requiring access to data, which is impractical and privacy-risking; even small nondeterminism in optimization breaks exact matching, rendering verification unreliable. Zero-knowledge PoL [18] mitigates disclosure but still exposes raw batches through repeated proofs and incurs heavy proof generation overheads (often minutes per iteration), again limiting real-world use.

On the ownership side, traditional watermarking embeds signals into AI model weights to claim PoO [10], but may alter weight distributions [38] or interact poorly with subsequent training and compression; conversely, many fingerprinting methods verify uniqueness without providing an explicit, verifiable ownership claim. In federated settings, client watermarks often conflict at aggregation when multiple parties embed in overlapping regions [31], degrading detection as clients or watermark length grow. In the SL setting, frameworks typically assume that a data-owning client has limited computational resources, whereas a single powerful server possesses sufficient capacity to train the model independently [39]. Such a setting imposes a heavy computational requirement on the server and presumes that it can single-handedly complete model training. What's more, clients in the proposed environment are only partially trusted [39], making it essential to rethink data privacy [40], [41], [42], model security, and intellectual property protection [43].

Guided by these gaps, this thesis pursues a three-part agenda:

- Existing PoL mechanisms inherit three structural limitations. **Cost**, replay/retrain-based verification scales with model size and training horizon, so the verifier's work can approach the cost of original training; this makes routine audits impractical for modern, high-capacity networks [17], [35], [37]. **Fragility**, gradient-trace or snapshot-matching approaches rely on brittle optimization determinism; benign nondeterminism (e.g., different hardware, seeds, libraries) breaks exact matching and yields false negatives, while over-permissive tolerances open room for forgeries [35], [36]. **Privacy exposure**, many schemes presume access to training data or intermediate states, which risks leaking sensitive information and conflicts with data-minimization requirements. To overcome these constraints, the PoL problem should be reframed so that training emits compact, tamper-evident evidence and verification becomes

challenge-response rather than re-execution. This reframing naturally leads to two central Research Questions (**RQs**): **RQ1:** *How can* PoO *be extended to certify training effort for* PoL *by chaining ownership evidence across the entire training trajectory rather than only the final model?* **RQ2:** *Under a rational-attacker model, how can* PoO*-based* PoL *make removal-plus-re-embedding no cheaper than honest training, thus deterring forgery?* This thesis proposes PoLO to partition training into shards and chain each shard to its predecessor via a watermark derived from the prior model state; a verifier can then sample shards to jointly validate learning correctness and provenance with cost decoupled from full retraining, while avoiding disclosure of raw data or full trajectories.

- Recent watermarking methods [10], [44], [45], [46], [47], [48] commonly augment the training objective with an embedding regularizer, thereby altering the model's weights. Although this enables ownership verification, the intrusive modification introduces several practical challenges, such as accuracy degradation, limited applicability, training inefficiency, and security vulnerabilities, that hinder deployment. On the other hand, many fingerprinting techniques merely attest uniqueness (i.e., that a model is distinguishable) without encoding a verifiable claim of ownership; several variants also impose heavy computation or require retraining [49], [50], [51], which is incompatible with downstream procurement checks and compliance audits. This motivates two concrete **RQs**: **RQ3:** *How can model ownership be bound without intrusive weight modifications or training-time changes?* **RQ4:** *How can we authenticate model uniqueness and verify explicit ownership claims while remaining robust to standard post-training edits and attacks?* NNFMAC, proposed in this thesis, is motivated by this gap: extract a stable binary fingerprint from critical weights via median-based binarization and use it as a codebook to encode ownership with an index-based map and redundancy. This design decouples the ownership payload from the authentication codes, while exploiting the complementary strengths of binary weight patterns and ownership information.
- In SL, conventional pipelines expose label information during gradient computation and allow clustering or inversion on the exchanged smashed representations, so privacy can leak even when raw data are not shared [52], [53]. In multi-client SL, participants are only partially trusted [39], which compels a fresh examination of data privacy [40], [41], [42], model security, and model ownership [43]. Motivated by these challenges, this thesis targets three questions: **RQ5** (data privacy): *How can the data-providing client preserve the privacy of raw inputs and labels while still enabling effective SL?* **RQ6** (layer ownership): *How can trainer clients present cryptographically anchored ownership claims for their contributions to access compensation?* **RQ7** (unauthorized-use defense): *How can the collaboratively trained model resist unauthorized use?*
  In FL, prior watermarking schemes have notable limitations: watermarks inserted by different clients can interfere in later rounds, depressing detection rates [31]. The conflict stems from multiple clients embedded in overlapping parameter regions, so aggregation superposes their marks and partially cancels or overwrites them. To deal with this issue, this thesis proposes FedNIFW to hinge on a prior agreement that allocates disjoint embedding regions to the clients. With client-specific, non-overlapping regions, watermarks are embedded independently, eliminating mutual interference.

1.2 Research Contributions and Overview

This thesis aims to remedy key shortcomings of current proof of learning and ownership-protection schemes, including high computation cost, insufficient privacy guarantees, accuracy degradation, and limited applicability. What's more, this thesis develops effective methods to

address these limitations and demonstrate their practicality by deploying them in SL and FL. The key research points and overall structure of this paper are illustrated in Figure 1.1.

Chapter 2 presents a comprehensive review of existing approaches to PoL and PoO for deep neural networks and analyzes their limitations. We begin by introducing the Proof-of-Anything (PoX) perspective, which provides a general lens for reasoning about verifiable computation. Building on this, we motivate the need for PoL in practice and survey representative PoL techniques, outlining their assumptions, verification costs, privacy implications, and failure modes. We then examine two mainstream strategies for PoO: model watermarking and model fingerprinting. For each, we summarize the core mechanisms, typical threat models, and robustness to post-processing, and we highlight unresolved issues such as optimization interference, removability, and the lack of explicit, decodable ownership claims. Finally, we turn to collaborative learning paradigms—SL and FL—and review the state of the art in privacy protection and ownership attribution under partial trust, heterogeneous data, and communication constraints. We identify where current solutions fall short, particularly in achieving privacy-preserving PoL and non-interfering PoO that scale to multi-client deployments. This review motivates the unified designs developed in subsequent chapters.

Chapter 3 establishes a unified lens for training provenance and ownership by formalizing the PoX view of verification. Building on this, we design PoLO, which converts the learning trajectory into a tamper-evident record: training is partitioned into shards, and a watermark state is iteratively updated via a hash over partial weights, cryptographically chaining each shard to its predecessor. This construction allows a verifier to sample arbitrary shards and validate PoL and PoO in a single challenge–response procedure without accessing training data or reproducing gradient traces. We prove that any adversary who removes or alters a single watermark cannot pass verification unless it reconstructs a consistent chain, making forgery as costly as honest training under a rational-attacker model. We analyze verifier complexity and show that PoLO decouples verification cost from full retraining, enabling lightweight audits for large models. We also study privacy: since only hashed commitments and watermark states are revealed, PoLO avoids exposing raw examples or intermediate gradients. An end-to-end implementation demonstrates practicality across diverse architectures and datasets; ablations confirm the roles of shard size, hash scheduling, and challenge rate. Empirical results indicate $\approx$99% ownership detection with verification costs reduced to $\sim$1.5–10% of replay-style PoL, while maintaining main-task accuracy and robustness to post-training operations such as fine-tuning and pruning. Collectively, this chapter delivers (i) a principled formulation for joint PoL/PoO, (ii) a chained-watermark mechanism that binds provenance to ownership, (iii) a privacy-preserving, sampling-based verifier protocol, and (iv) comprehensive evidence that PoLO offers scalable, economical, and secure verification for real-world deployments.

Chapter 4 defines a post-training ownership-authentication problem for neural networks and develops NNFMAC, a non-intrusive scheme that verifies both model uniqueness and ownership without retraining. The method first selects salient weights across layers and applies median-based binarization to derive a stable, model-specific binary fingerprint. Treating this fingerprint as a codebook, we design an index-based encoding with expansion that maps ownership bits to positions in the fingerprint, decoupling the ownership payload from the authentication codes and enabling reliable decoding under perturbations. We analyze sensitivity and error tolerance, showing how redundancy mitigates bit flips caused by fine-tuning, pruning, weight shifting, and other post-processing. Comprehensive experiments on diverse architectures and datasets demonstrate that NNFMAC maintains the original task accuracy, imposes negligible overhead, and supports fast verification. Comparisons with loss-regularized watermarking con-

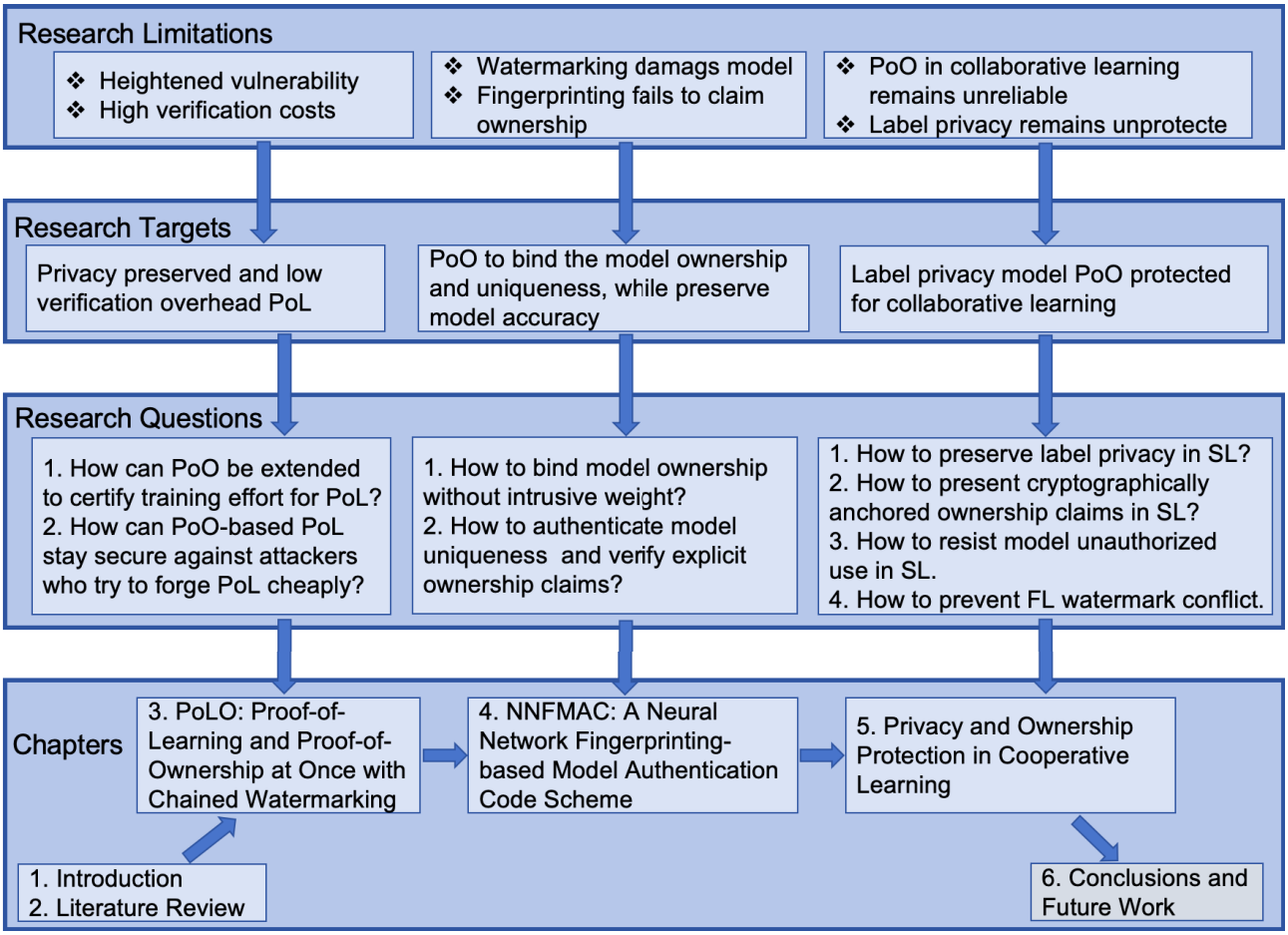


Figure 1.1: Research targets, research questions, and chapters.

firm that NNFMAC avoids optimization interference and remains robust after standard model edits, while outperforming prior fingerprinting baselines that only test uniqueness or require heavy computation. Collectively, this chapter delivers: (i) a precise post-training formulation of ownership authentication; (ii) a fingerprint-as-codebook design that produces reliable authentication codes without modifying training; and (iii) empirical evidence that the scheme is accurate, efficient, and resilient for practical model procurement, auditing, and marketplace scenarios.

Chapter 5 establishes a privacy-first, verifiable collaboration protocol under partial trust. In the SL setting, we propose CliCooper, which masks label information by label expansion, hiding both semantics and cardinality from trainer clients; it further applies a DP mechanism to smashed activations to suppress clustering and inversion on shared representations. Along the training pipeline, chained watermarks bind successive stages, so provenance and ownership (PoL + PoO) can be audited through lightweight challenges without revealing raw data or gradients. The result is an SL workflow that preserves model utility while offering explicit, verifiable guarantees on both data privacy and training correctness.

We also propose FedNIFW to resolve watermark collisions that arise when multiple clients embed signals in overlapping parameters. The method designates watermark layers and partitions them into client-exclusive segments; during local training, each client embeds only in its own segment and freezes others. At the server, exclusive aggregation is applied to watermark regions, while standard FedAvg is used elsewhere. This non-interfering fragmented watermark-

ing cleanly attributes ownership to specific clients, scales with the number of participants and watermark length, and maintains the accuracy and stability of the global model. Together, CliCooper and FedNIFW provide complementary, deployable solutions that bring privacy, verifiability, and conflict-free copyright protection to the two principal forms of collaborative learning.

Chapter 6 summarizes the thesis and outlines the future work, and Chapter 7 lists the key publications.

Chapter 2

Literature Review

We first provide the formalization of PoX as a unified framework for analyzing PoL and PoL, outlining their core concepts and concurrent works. PoL lines of research include gradient-trajectory matching, hash-based replay, and zero-knowledge proofs; they differ in privacy, security, and overhead profiles, but none simultaneously deliver low verifier cost, strong privacy, and ownership guarantees. Table 2.1 summarizes state-of-the-art(SoTA) PoL methods. For model ownership protection, watermarking and fingerprinting represent two major approaches. Watermarking, widely used for safeguarding multimedia such as images [54], text [55], and videos [56], was first adapted to NNs by Uchida et al. [10] who embedded ownership information directly into model weights. In contrast, fingerprinting [11], [49], [50], [51], [57], [58] identifies unique model characteristics, analogous to human fingerprints, without altering the model. These methods extract distinctive features or behaviors to verify ownership while preserving the model's original properties. Table 2.2 summarizes and compares the two approaches across SoTA schemes, highlighting their characteristics and trade-offs.

Both SL and FL are distributed training paradigms, yet their progress on copyright protection is uneven. In Split Learning, prior art documents that raw data are hidden, yet smashed activations and labels remain vulnerable; attacks include feature/label inference, clustering, inversion, and surrogate extraction, prompting defenses such as keeping label-sensitive layers on clients and adding local DP noise. What's more, SL—despite active research on attacks and defenses for privacy and robustness—still lacks dedicated, end-to-end schemes that establish and verify model ownership. In FL, a growing body of work has explored watermarking and attribution under server–client aggregation, yielding practical mechanisms for ownership verification and tamper evidence. However, these FL model watermarking schemes are still not fully optimized. They present some drawbacks, for example, the watermarks embedded by different client nodes can interfere with each other, reducing the watermark detection rate in the later stages [31]. Watermark conflicts are likely to occur because clients may embed their watermarks in the same region of the model. When this situation arises, during the model aggregation process, the watermarks from different clients can overlap, resulting in conflicts among the embedded watermarks.

2.1 Proof-of-Anything and Proof-of Learning

2.1.1 Proof-of-Anything

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 PoW [59] 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) 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). Beyond computational effort (as in PoW), various physical and virtual resources can serve as proof bases, including holdings (proof-of-stake, PoS) [60], time (proof-of-elapsed-time, PoET) [61], storage and bandwidth (proof-of-space) [62], and reputation (proof-of-authority, PoA) [63]. In each case, the verification function Θ is typically closely tied to its input χ.

2.1.2 Proof-of-Learning

Definition 2 (PoL)**.** *A prover $\mathcal{P}$ constructs a* PoL *proof $\mathbb{P}$ by recording the complete training trajectory of ML 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, $\mathbf{B}_t \subseteq \mathcal{D}$ stands for the training batches, and A_t denotes 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) [17] 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 the learning rate and $\mathcal{L}$ is the loss function. The proof $\mathbb{P}$ comprises consecutive weight snapshots (W_t, W_{t+1}) and data batches $\mathbf{B}_t$ used for each update. Studies have revealed vulnerabilities in this approach, including synthetic trajectory attacks [35] and gradient matching exploits [37], driving the need for enhanced verification mechanisms.

Hash-based PoL. To enhance computational efficiency and reduce communication complexity, Zhao et al. [19] introduced 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.

Table 2.1: Comparison of existing PoL.

	Ownership	Security			Privacy		Low Ov.	
	①	②	③	④	⑤	⑥	⑦	⑧
PoL (GD, Vanilla) [17], [35], [37]	✗	✓	✗	✗	✗	✗	✗	✗
PoL (hash) [19]	✗	✓	✓	✓	✗	✗	✗	✗
PoL (zkp) [18]	✗	✓	✓	✓	✗	✓	✗	✓

Notation: ✓ for attack-resistance/property-held; ✗ vice versa; **Ov.** for overhead. Prevent:
① Unauthorized use?
② 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?

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 generated during the original training process due to inherent randomness in optimization and hardware-level non-determinism. As a result, even if retraining is performed correctly, the resulting model may differ slightly, leading to hash mismatches and making the verification process unreliable.

ZK-PoL. Zero-knowledge cryptographic commitments have been integrated into PoL to enhance privacy. ZK-PoL methods [18], [64], [65] 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 $\mathbf{B}_t$. 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 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 limits its scalability and real-world applicability.

Existing PoL methods face several critical challenges (cf. Table 2.1). 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 requires sharing training data during verification. On the security front, existing PoL remains susceptible to various forgery attacks, such as synthetic trajectory and adversarial example-based verification loopholes, which allow adversaries to manipulate proofs and bypass verification.

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 a legal construct governed by intellectual property laws, granting developers exclusive rights to control the model's usage, distribution, and modification [9]. A complete ownership proof can provide a more efficient means to address such disputes [66], complementing PoL's technical validation to ensure comprehensive protection for machine learning models against both attribution disputes and unauthorized use.

Definition 3 (PoL)**.** *For a model owner acting as the prover $\mathcal{P}$, a valid ownership proof is denoted $\mathbb{P}_o(W_t, \mathcal{D}, A, \Lambda, \Psi)$. 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.*

Two main approaches have emerged for protecting DNN ownership: watermarking and fingerprinting. Watermarking, widely used for safeguarding multimedia such as images [54], text [55], and videos [56], was first adapted to NNs by Uchida et al. [10] who embedded ownership information directly into model weights. In contrast, fingerprinting [11], [49], [50], [51], [57], [58] identifies unique model characteristics, analogous to human fingerprints, without altering the model. These methods extract distinctive features or behaviors to verify ownership while preserving the model's original properties. Table 2.2 summarizes and compares the two approaches across SoTA schemes, highlighting their characteristics and trade-offs.

2.2.1 Watermarking Approaches

EWDNN [10] proposes the first scheme to embed a watermark $\{0,1\}^t$ into the host model (the model to be protected), where t is the length of the watermark. Subsequently, various improved watermarking schemes based on weight modification are proposed. The existing schemes are relatively easy to implement and exhibit moderate robustness. However, the majority of existing schemes rely on embedding regularizers, which alter the model weights. This alteration can

Table 2.2: A summary of state-of-the-art model IP protection schemes.

Category	Scheme	Brief summary	Ownership Verification	Performance preserved	No extra training	Adapt to various tasks
Water-marking	EWDNN [10] WSFCL [46]	Embed-based	✓	✗	✗	✓
	RIGA [45] HufuNet [47] DeepMarks [48]	Extra NN-based	✓	✗	✗	✓
Finger-printing	IPGuard [11] pHash [57] TAFA [49]	Feature-based	✗	✓	✓	✗
	MetaV [51] GNNfingers [58] InFIP [50]	Classifier-based	✗	✓	✗	✓

impact the model performance and diminish model training convergence efficiency. Wang et al. [38] demonstrate that most of these schemes are insufficiently covert because the watermark embedding deforms the weight distribution of the host model.

RIGA [45] redesigns the loss function proposed in [10] to measure the difference between non-watermarked and watermarked models. DeepMarks [48] embedded the watermark into the density function of the host model weights, comparable to the method utilized by Uchida et al. Wang et al. [44] improve the method of Uchida et al. by independently training a neural network to replace the key X, which is used to embed the watermark into the host model. Liu et al. [67] enhanced the robustness of the watermark embedding method by greedily selecting important weights to embed watermarks. However, this approach significantly increases computational overhead, as it requires training an additional non-watermarked model prior to the main training process.

HufuNet [47] presents a watermarking scheme using a dedicated NN architecture split into encoder and decoder components. During host model training, the encoder is embedded, and for IP verification, it is extracted and paired with the decoder; high combined performance is used to confirm ownership. The performance change is quantified by $\Delta PCC = \left|\frac{\Delta loss}{\tau}\right|$, where τ is the crash threshold, with $\Delta PCC = 1$ marking the verification point. WSFCL [46] avoids loss-based embedding by exploiting multiple local minima in converged models, selecting n fully connected layer weights and constraining their updates to embed signatures. DeepMarks [48] introduces the first end-to-end collusion-secure fingerprinting framework, which enables ownership verification and user identification in DNNs.

Research by Lukas et al. [68] reveals significant vulnerabilities in watermarking-based approaches, particularly their susceptibility to weight shifting attacks - a prevalent technique in watermark tampering. Furthermore, the use of embedding regularizers in weight modification schemes introduces notable drawbacks such as degraded accuracy and impaired training convergence. To address these limitations, we propose NNFMAC, a novel approach that generates robust authentication codes by encoding ownership information within the model's unique fingerprint.

2.2.2 Model Fingerprinting

To address the limitations of weight modification approaches, [11], [50], [51], [57], [58] have proposed model fingerprinting schemes as an alternative for IP protection, preserving model performance and training efficiency. IPGuard [11] introduces an innovative approach that leverages data points near decision boundaries to create unique fingerprints for DNN classifiers. This design leverages the fact that a model's decision boundary is inherently determined by its learned parameters, and different models—even when trained on the same dataset—typically exhibit distinct decision boundaries. While this method effectively balances robustness and uniqueness, its application is constrained to classification tasks only. Thus, Pan et al. propose TAFA [49] and its enhanced successor MetaV [51]. TAFA introduces a fingerprinting method exploiting the properties of DNNs with ReLU activations, while MetaV uses a classifier-based approach to detect various post-processed versions of the host model. AFA's reliance on ReLU limits applicability, while MetaV's versatility incurs high overhead from generating and training on numerous simulated suspect models. Similarly, GNNFingers [58] adopts a classifier-based strategy but is restricted to GNNs and suffers from low efficiency. InFIP [50] leverages intrinsic model features but demands typically sampling $N = 400$ images from the dataset for key instance generation, incurring substantial storage costs.

These methods face two key challenges: they either have limited applicability across different NN architectures or require substantial computational and storage resources for fingerprint generation. Moreover, when confronted with increasingly sophisticated attacks, model owners must recalibrate classification thresholds and retrain their detection systems, a process that incurs significant computational overhead and time delays.

Chen et al. [57] propose pHash, a method for detecting model copying by comparing hash codes. The method operates by encoding both the host and suspect models into unique hash codes and evaluating their similarity using Hamming distance measurements. Specifically, pHash [57] implements a three-step process: it first identifies and selects critical weights from the model architecture, then computes their Normal Test Statistics (NTS), and finally generates a distinctive hash code from these statistics. The verification process involves calculating the Hamming distance between the hash codes of the original and suspicious models to determine potential infringement. However, pHash exhibits several limitations that impact its robustness and security. First, its reliance on significant weights without incorporating randomness in the selection process makes it vulnerable to targeted attacks. Second, the absence of hash code shuffling further weakens its resilience against various attack vectors. These architectural choices can lead to elevated false negative rates during verification, potentially allowing manipulated models to evade detection. Additionally, pHash's use of XOR operations for hash generation introduces a critical vulnerability: the same key cannot be safely reused across multiple hash computations due to XOR's mathematical properties. Specifically, if an adversary obtains two hashes generated with the same key, they may recover the underlying model fingerprints through a simple XOR operation of the hashes.

2.3 Collaborative Learning

2.3.1 Split Learning

Users are increasingly unwilling to disclose sensitive information. SL [29] addresses this by enabling outsourced training without sharing raw data. SL allows multiple entities to collabo-

ratively train a model while keeping local data private. We show two approaches:

- In *vanilla SL* [13], [28], [69], the client encodes local data and sends the resulting smashed data to the server, which performs most of the computation and, during gradient descent, requires access to label information [39].
- *U-shaped SL* [70], [71], [72] is tailored to unmet label-privacy requirements in vanilla SL settings. A client will not share labels with the server. By preprocessing activations after the split on the client side, the client preserves label confidentiality while maintaining the integrity.

SplitNN [13] demonstrated SL for medical diagnostics, enabling collaboration between centralized and local institutions while protecting patient privacy. Poirot et al. [28] extended SL to multi-center healthcare settings with restricted data sharing. Hu [69] highlighted the temporal nature of clinical data and proposed grouping patient samples by time to better capture sequential dependencies in SL. Gupta et al. [70] placed the output layer on the client to prevent label exposure. Lyu et al. [72] added parallel computing and an optimized resource allocation strategy to improve edge efficiency. UVSL [71] combined the U-shaped design with differential privacy to protect both labels and raw data.

Both assume a server with sufficient compute to train the model independently. In Vanilla SL, clients share labels for gradient computation, which exposes sensitive information. In contrast, U-shaped SL moves gradient computation to the client side, thereby preserving label privacy. However, this design increases communication overhead during the forward and backward passes and adds extra local computation. Consequently, both variants are unsuitable for collaborative learning where clients have limited computational or network capacity.

2.3.2 SL Privacy Threats and Defenses

SL is a privacy-preserving alternative to centralized training, but it remains vulnerable to label and feature inference attacks.

Abuadbba et al. [73] showed that 1D CNNs trained with SL can match non-split accuracy yet still leak privacy, as servers can infer input features from intermediate outputs, constituting the first feature inference attack in SL. RecoNN [74] trains a reconstruction network using shadow models and shadow samples; at attack time, target weights are input to reconstruct images of the target's training data. Unsplit [52] jointly searches the input and parameter spaces to reconstruct inputs and produce a functional clone whose outputs approximate the client's smashed data. FORA [53] uses public auxiliary data for feature-level transfer, building a surrogate client to train an attack model that reconstructs private inputs efficiently and covertly. Pasquini et al. [75] further verified across settings that a malicious server can reconstruct a client's private dataset via feature inference.

To mitigate these threats, several defenses have been explored. UVSL [71] keeps label-sensitive layers on the client and adds local DP noise to smashed activations, limiting label inference and input reconstruction. Lyu et al. [72] place the model tail on the client and perform local backpropagation, preserving label confidentiality. U-shaped designs reduce label exposure but increase communication, since each iteration requires at least two round-trip between client and server. Ngoc et al. [76] propose Binarized Split Learning (B-SL), which binarizes local layers and applies differential privacy to mixed smashed data. Mao et al. [77] introduce a randomized activation, R3eLU, to perturb forward smashed data and backward local gradients

while integrating a DP mechanism. Both B-SL and R3eLU reduce accuracy by up to 6%, underscoring the utility–privacy trade-off.

2.3.3 Federated Learning

The goal of FL is to enable multiple clients to collaboratively train a global model without disclosing each client dataset [78] while ensuring that the global model can adapt to the specific tasks of each client. An FL system consists of n clients, where each client trains a local model M^i on its private dataset D_i. The local models M^i are then sent to a central server for aggregation as the global model M^G. One of the most popular aggregation algorithms is the FedAvg algorithm [12], which operates by performing a weighted average of the parameters of each client's local model during each communication round and then assigning these averaged parameters to the global model.

$$M^G = \frac{\sum |D_i|\, M^i}{\sum |D_i|}, \tag{2.1}$$

where M^G is global model and M^i is the local model of the i-th client respectively, and $|D_i|$ is the size of local dataset. After aggregation, the server distributes the global model back to the clients for further local training. The process of client-side local model training and server-side global model aggregation is repeated for multiple rounds until the global model converges.

2.3.4 FL Watermarking

DNNs model watermarking can be seamlessly integrated with FL to protect the IP of federated models. Federated model watermarking schemes can be categorized into server- and client-side-based watermarking schemes.

Server-side-based FL Watermarking

WAFFLE [79] represents the first federated model watermarking scheme and also the first server-side-based watermarking approach. In this scheme, Tekgul et al. introduce an additional global model training step during aggregation to embed a backdoor-based watermark [80]. The backdoor watermark embeds a trigger set with special labels into the training data. Since the server lacks training data, WAFFLE [79] develops a data-free trigger set algorithm known as WafflePattern. Although this method successfully embedded a backdoor-based watermark into the global model, the accuracy of the watermarked global model significantly decreased compared to the non-watermarked model due to the lack of end-to-end retraining of the trigger set. FedTracker [81] is a more advanced federated model watermarking scheme that not only verifies the IP of federated models but also tracks potential leaks of the global model by clients. This scheme embeds a backdoor-based watermark into the global model using Continual Learning (CL) [82] before model aggregation. The backdoor watermark serves to protect the IP of the global model. In the subsequent phase, different parameter-based watermarks are embedded into the watermarked global model and distributed to different clients. The purpose of the second watermark is to track leaks for the global model. This dual-watermarking scheme, however, performs poorly against model pruning attacks, with the success rate of watermark extraction dropping to 50% when the pruning rate is more than 0.5.

Table 2.3: Existing privacy and copyright protection methods for SL and FL.

Method \ Protection	Label Privacy	Activation Data	Performance Preserved	Copyright & Train Trace
Vanilla SL [13], [28]	✗	✗	✓	✗
U-shaped SL [70], [71], [72]	✓	✗	✓	✗
B-SL [76]	✗	✓	✗	✗
R3eLU [77]	✗	✓	✗	✗
WAFFLE [79]	✓	✗	✓	✓
FedTracker [81]	✓	✗	✗	✓
FedIPR [31]	✓	✗	✗	✓

Client-side-based FL Watermarking

Liu et al. [83], [84] and Li et al. [31] propose client-side-base watermarking schemes. Liu et al. [83] introduce a scheme for model IP verification by enhancing data privacy. Specifically, the representative clients embed a backdoor-based watermark into the federated model with a trigger set consisting of pre-designed noise patterns. Li et al. [31] posited that, as all clients are involved in federated model training, they can embed a watermark into the federated model to validate their IP. They propose the FedIPR watermarking scheme, where all clients embed their respective watermarks into specified model layers. However, when the number of clients increases or the watermark length is extensive, conflicts between watermarks may occur, resulting in a decreased watermark detection rate. To address the issue of watermark conflicts, we propose a Non-Interfering Fragmented Watermark Embedding scheme.

Table 2.3 provides a side-by-side comparison of representative privacy- and ownership-protection methods for SL and FL. We evaluate each method along four dimensions—label privacy, activation-data protection, performance preservation, and copyright/training-traceability—so that the trade-offs and complementarities are immediately visible and consistent with the narrative survey.

Chapter 3

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

3.1 Introduction

As machine learning (ML) models evolve from academic artifacts into economically valuable digital assets [33], [34], their exchange and commercialization through marketplaces, federated platforms, and outsourced pipelines have become increasingly common. This transformation makes it essential to verify both the legitimacy of training efforts and rightful model ownership, as these guarantees underpin trust, attribution, and compensation in real-world ML trade. Some examples are: (i) *incentive-driven distributed learning* [21], [32]: participants prove that models are properly trained to receive fair rewards; (ii) *ML marketplaces* [20]: buyers need confidence that models are genuinely trained and transferable without dispute; (iii) *outsourced training* [16], [85]: organizations must ensure that externally developed models are both authentic and securely attributed. All this introduces a fundamental challenge:

How can a recipient verify the legitimacy of a model?

Legitimacy in this context involves two facets: verifying training effort and establishing rightful ownership [26]. This requires two 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.

In this chapter, we argue that PoL *and* PoO *are intertwined and must be addressed jointly to establish a complete proof of legitimacy.* This is not merely a design choice; it is reinforced by existing regulatory regimes that require provenance and ownership checks for listing, billing, and dispute resolution in real distribution channels (e.g., cloud providers [86], model hubs [25], marketplaces [87]). Platforms operate under notice-and-takedown and transparency duties (e.g., U.S. DMCA safe harbor [22], EU Digital Services Act [88], EU AI Act Art. 53 [89]), which effectively mandate verifying both training effort and ownership together [24]. However, existing solutions treat PoL and PoO as separate mechanisms. When naively stacked, the overall system degrades in security and effectiveness, failing primarily due to the weaknesses of

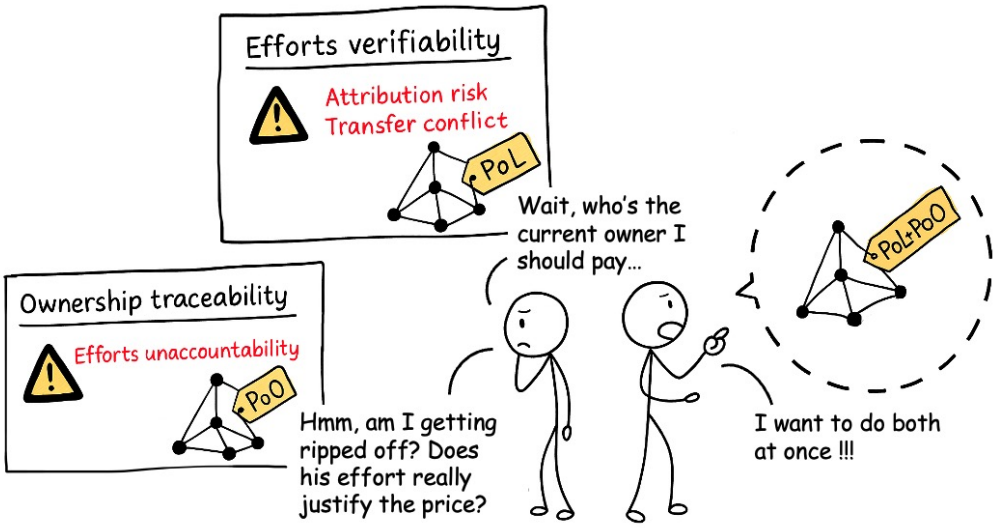


Figure 3.1: **Why at once?** PoL verifies training effort but lacks ownership tracking. PoO ensures ownership but fails to justify training efforts. Separating them creates attribution risks and ownership conflicts.

current PoL approaches:

- *Heightened vulnerability*: As first framed by Jia et al. (S&P'21 [17]), existing PoLs require exposing intermediate training states and training data; follow-ups by Zhang et al. (S&P'22 [35]) and Thudi et al. (USENIX'22 [36]) showed these artifacts can be manipulated or fabricated to forge plausible trajectories.
- *High verification costs*: In the Jia et al. formulation and its variants [17], [35], [37], verifiers must replay or retrain from recorded snapshots, causing verification costs to scale with model size, dataset size, and update count. This approaches the full cost of training, making the approach impractical for large or high-capacity models.

Simply combining PoL and PoO on top of existing PoL methods is ineffective, as the entire system just inherits PoL's vulnerabilities and overhead without providing stronger legitimacy.

Our goal is to *redesign* PoL *from the ground up to accommodate regulatory requirements*, embedding it with PoO to provide a single, seamless framework that validates training effort and ownership together under *two core assumptions*:

- Any attack that costs more than full retraining is considered *economically unviable* and thus irrational.
- Models lacking verifiable ownership are deemed invalid and unmonetizable in practical settings, making attacks that *only* remove watermarks [68], [90] *economically irrational* and irrelevant for security considerations.

In economically motivated settings, participants aim to earn rewards, making ownership proof essential for monetization in ML marketplaces or contractual training [91], and—most importantly—this requirement is reinforced by existing regulatory frameworks. Simply removing a legitimate watermark offers no financial benefit, as models without verifiable ownership cannot be claimed, licensed, or traded. Establishing a new ownership claim requires embedding a replacement watermark, which entails computational effort comparable to full retraining. Thus,

any forgery matching the cost of legitimate effort is regarded as a failed attack. This presents the technical challenges of "PoL+PoO" and motivates the following research questions (RQs):

- **RQ1:** *How can* PoO *be extended to certify training effort for* PoL*?* Existing PoO schemes embed a watermark in the final model, which cannot reflect the training trajectory and makes them incompatible with PoL under the premise that only complete and traceable effort can be rewarded.
- **RQ2:** *How can* PoO*-based* PoL *remain secure against rational adversaries aiming to forge ownership with minimal cost?* Since attackers must embed their own watermark to monetize models, the design must ensure that forging ownership requires computational effort comparable to training from the scratch, making low-cost attacks economically unviable and deterring them from happening.

To answer these RQs, we propose a novel design, PoLO, which 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 cryptographically links training phases and embeds ownership information at every stage, rather than relying on gradient-based tracking that has been proven prone to forgery [35], [37]. More importantly:

> **PoLO traps attackers in an economic dilemma:** The attackers would either incur a cost comparable to retraining the full model, making forgery *economically impractical*, or suffer degraded model performance that fails verification (i.e., *accuracy drop*). In either case, the attacker gains no benefit.

This insight highlights the necessity of PoLO's chained watermarking: It shifts the defense from protecting isolated watermarks to securing the full training trajectory. By cryptographically linking each shard, PoLO ensures that any ownership claim must reflect genuine cumulative effort. Attacks targeting a single watermark are ineffective, as the proof depends on consistent verifiability across all stages, aligning security with rational incentives.

We achieve this in a stepwise manner:

- *We formalize the concept of Proof-of-Anything (*PoX*)* (§2.1 & §3.2) 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. We identify key limitations in existing PoL methods, 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 (§3.3), 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 one go. 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* (§3.4) 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 **1.1x–4.0x** more resources than legitimate PoLO generation, with the original proof maintaining over **90%** detection accuracy even after attacks.

3.2 System Model

Problem definitions. Arguably, *existing* PoL *methods fail to address scenarios where models are outsourced or exchanged for monetary assets.* They 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 [16]. In ML marketplaces, current PoL methods cannot support ownership transfer, which prevents PoL from verifying rightful ownership when models are bought, sold, or reassigned [20].

Existing PoL *methods become the weakest link when simply stacking* PoL *and* PoO . Those solutions [18], [35] 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 [35], allowing 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 [18], 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 (i.e., 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 [10] with the undetectability properties of more advanced schemes [45].

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 obfuscation-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 [37]. 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. 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. 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 describe the procedures of model training and model challenging by presenting the interactive steps between a prover and a verifier.

- ***Step-1*** (§3.3.1). 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*** (§3.3.2). A verifier $\mathcal{V}$ issues a challenge to the prover, requesting verification for a randomly selected shard s_x.
- ***Step-3*** (§3.3.2). $\mathcal{V}$ requires sending the model weights of the checkpoint models of shards s_x and s_{x-1}.
- ***Step-4*** (§3.3.2). $\mathcal{V}$ computes the expected embedded watermark for s_x using the information shared from the owner, and then verifies whether the checkpoint model of s_x is watermarked by the computed watermark.
- ***Step-5*** (§3.3.2). 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.

Threat model. Three types of players are considered.

- ***Economically rational adversaries.*** We consider that attackers operate under rational, market-aligned incentives. In our EMM setting, only models with a valid, registered ownership watermark (incumbent or approved adjunct) are treated as valuable assets. A watermark-

free model is automatically illegitimate: it cannot be listed, licensed, or rewarded, even if its accuracy is high. Specifically, in a setting where model ownership is essential for fair exchange and compensation (e.g., ML marketplaces or outsourced training), an attacker would have no incentive to only perform a *watermark removal attack* [68], [90] without embedding their own watermark. Pure removal offers no economic benefit and only invalidates the model's ownership claim, rendering it unsellable or unusable in a system that relies on PoO for legitimacy. Therefore, PoLO focuses on defending against rational adversaries who aim to replace or forge ownership by consuming necessary resources, rather than those who engage in self-defeating, destructive behavior with no strategic gain. The attackers seek to exploit the system through various attacks, including:

  - *Training forgery (addressed in §3.3.3).* 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. If with full knowledge of the embedding key k_S and selection matrix Y, an attacker may forge ownership of the final checkpoint.
  - *Ownership theft (addressed in §3.3.3).* The attacker may attempt to steal or misuse the model by targeting its watermark through attacks such as fine-tuning [92], pruning [93], or overlapping [94], aiming to falsely claim model ownership. Watermark removal attacks [95] alone yield no benefit in trusted marketplaces, where only models with verifiable ownership are accepted. Such removal is only meaningful when combined with the embedding of a forged watermark.
  - *Inference attacks (addressed in §3.3.3).* The attacker may exploit the information contained in the published model to infer the provers' private training data.

- ***Rational prover.*** The prover functions as both the model *owner*, holding full rights to its creation and usage, and the PoL *prover*, 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. They may also try to embed the watermark in an "inactive" layer that has little impact on performance, allowing extraction without meaningful training. What's more, they may try to forge k_x to finalize a shard without performing actual training, inflating the number of claimed shards for higher 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.

3.3 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. 3.2–3.3), 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

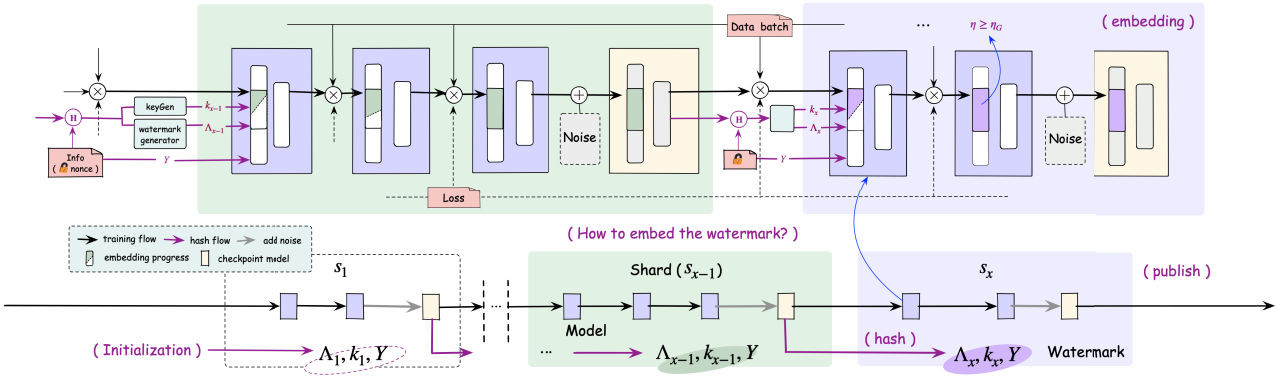


Figure 3.2: **PoLO design**: The verifier $\mathcal{V}$ shares a secret nonce with the prover $\mathcal{P}$ to initialize watermark parameters (Λ_1, k_1, Y) for the first shard s_1. Prior to watermark embedding, the model owner computes the watermark Λ_{x-1} and its corresponding embedding key k_{x-1} 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, Gaussian 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.

task performance.

3.3.1 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 unit of training that spans one or more consecutive epochs. Unlike fixed epoch boundaries, a shard is defined by the successful embedding of a watermark, which may take multiple epochs. This aligns verification with actual training effort. Let T_x denote the set of epochs in the x-th shard s_x. By partitioning training into shards, PoLO enables verifiable proof of effort. The prover's training process with chained watermarking involves the following 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

Algorithm 1 PoLO generation

Require: $\mathbb{D}$ (dataset), μ/id (secret nonce/information), σ (privacy budget of Gaussian noise), t_{max} (maximum training iterations), key: Z (selection matrix for weights subject to Gaussian noise), η_G (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** ▷ until convergence
  $W_{t_x} = \arg\min_W \left(l_w(W) + \lambda l_\Lambda(W)\right) |_{W_0 = W_{x-1}} + \delta W$
    where $\delta W = \mathbb{E}(W_{x-1}, \Lambda_x, k_x, Y)$ ▷ update 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**
    Save $W_x = W_{t_x} + \mathbb{G}(\sigma, Z, W_{t_x})$ ▷ apply Gaussian noise
    $\Lambda_{x+1}, k_{x+1} = \mathsf{WMGen}(\mathcal{H}_x), \mathsf{KeyGen}(\mathcal{H}_x)$
    $x+ = 1$
  else $W_{x-1} = W_{t_x}$ ▷ advance to the next epoch in shard s_x
  end if
end while
RETURN $\mathbb{P} \in \{W_1, W_2, ..., W_S\}$

secret nonce μ:

$$\begin{aligned} Y &= \mathsf{WMPosition}(\mu), \\ \Lambda_x,\ k_x &= \mathsf{WMGen}(\mathcal{H}_x),\ \mathsf{KeyGen}(\mathcal{H}_x) \text{ with} \\ \mathcal{H}_x &= \mathbb{H}(W_{x-1},\ x,\ \mu,\ id_\mathcal{P}), \end{aligned} \tag{3.1}$$

where $\mathsf{WMPosition}(\cdot)$, $\mathsf{WMGen}(\cdot)$, and $\mathsf{KeyGen}(\cdot)$ are deterministic functions, with all randomness derived from the verifier-fixed nonce μ and prover identity $id_\mathcal{P}$. This ensures only the legitimate prover can generate valid watermarks and keys, while the shard index x enforces sequential linkage. The fixed nonce prevents adversaries from precomputing or guessing valid parameters.

Since modifying any individual shard would invalidate all subsequent watermarks, attackers cannot selectively alter 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:

$$W_{\bar{t}_x} = \arg\min_W \left(l_w(W) + \lambda l_\Lambda(W)\right) \Big|_{W_0 = W_{x-1}} + \delta W, \tag{3.2}$$

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 calculation based on Hamming distance [57]:

$$\eta = 1 - \frac{\sum_i^n (\mathbb{C}(W_{t_x}, Y, k_x)[i] \neq \Lambda_x[i])}{n}, \tag{3.3}$$

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.

Obfuscation-based privacy protection and shard formation. PoLO applies Gaussian noise by randomly selecting a subset of weights from the non-watermarked region $W_{t_x} \setminus \{W_{t_x} Y\}$ and injecting obfuscation noise. This selective Gaussian mechanism introduces statistical uncertainty, protecting sensitive training information in non-watermarked areas from gradient leakage [96], while preserving the ownership integrity carried by the embedded watermark. The point of successful watermark embedding, denoted as $W_{\bar{t}_x}$, yields a obfuscation-protected version $W_{\bar{t}_x}^{\text{Gaussian}}$, 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.

$$W_x = W_{\bar{t}_x}^{\text{Gaussian}} = W_{\bar{t}_x} + \mathbb{G}(\sigma, Z, W_{\bar{t}_x}), \tag{3.4}$$

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 $\mathbb{G}(\sigma, Z, W_{\bar{t}_x})$ denotes the addition of σ-Gaussian noise to the subset of weights in $W_{\bar{t}_x}$ specified by a selection matrix Z for Gaussian. 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 modification attacks impossible, 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 resistant to tampering or synthetic trajectory attacks.

Ownership transfer. In scenarios requiring ownership transfer, such as ML marketplaces [20] and outsourced training [16], 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.

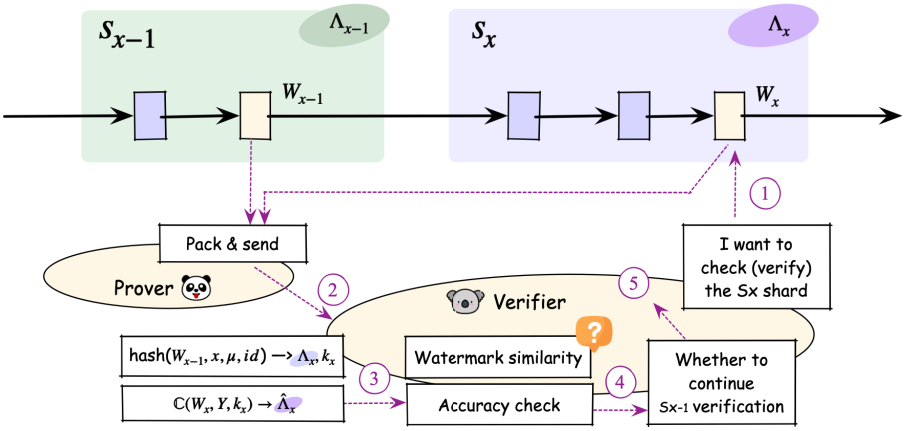


Figure 3.3: **The verification of chained watermarking for PoLO:** ① (verifier) requests verification for shard x; ② (prover) sends checkpoints W_{x-1} and W_x; ③ (verifier) derives the selection matrix Y from the locally recorded nonce μ associated with the prover, if Y has not already been obtained. 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.

3.3.2 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 obfuscation-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 overclaim 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.

Generating proof

When the verifier $\mathcal{V}$ requests validation for shard s_x, the prover $\mathcal{P}$ submits the obfuscation-protected weights W_x and W_{x-1} from shards s_x and s_{x-1}, respectively, along with the hash function $\mathbb{H}(\cdot)$ used to derive the watermark Λ_x and key k_x from W_{x-1} using the verifier-recorded nonce μ. Moreover, $\mathcal{P}$ also provides the function that deterministically derives the selection matrix Y from μ. These elements enable $\mathcal{V}$ to verify both model integrity and the sequential linkage of watermarks (cf. Algorithm 1).

Unlike existing PoL [17], [18], [19], [35], [37] that requires 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 allowing the verifier to assess both training effort and ownership authenticity.

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(threshold of similarity between the hashed and extracted watermarks).
Ensure: "**Fail**" or "**Success**".

Choose $i \in \{S-1, S-2, ..., 1\}$ ▷ $\mathcal{V}$ selects the shard at which to terminate 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 $\hat{Acc}_{main}$ not meet expectation **or** $\eta_x < \eta_G$ **then**
        RETURN "**Fail**", **break**
end for
RETURN "**Success**"

Verifying proof

Verification begins by checking whether the checkpoint W_x meets the required accuracy on a public test set. This ensures the model has achieved meaningful task-specific performance, reflecting real-world expectations where only accurate models hold value in ML marketplaces [20]. If W_x fails to meet the threshold, the proof is immediately rejected and the prover $\mathcal{P}$ is penalized, discouraging the embedding of watermarks in inadequately trained models.

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, the expected watermark Λ_x and the key k_x for shard s_x using the stored obfuscation-protected weights from shard s_{x-1} (cf. §(3.1)). This confirms that the watermark Λ_x must have been generated from W_{x-1} and cannot be arbitrarily forged or altered.

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

$$\hat{\Lambda}_x = \mathbb{C}(W_x, Y, k_x). \tag{3.5}$$

By comparing $\hat{\Lambda}_s$ with the watermark Λ_s derived from (3.1), 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 as:

$$\eta_x = 1 - \frac{\sum_i^n (\hat{\Lambda}_x[i] \neq \Lambda_x[i])}{n}. \tag{3.6}$$

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

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. Hash-based chaining combined with cumulative verification ensures that the cost of forgery scales with depth and exceeds the rational benefit, making partial manipulation economically and computationally impractical.

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 the 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 the 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 improve the main task but also distort fair reward distribution, making it impractical to assign credit or rewards based solely on individual epochs.

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 improve its accuracy on the main task. As PoLO ties rewards directly to measurable contributions, it introduces a natural incentive mechanism that aligns computational effort with economic return. 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 the provers $\mathcal{P}$ from inflating their workload claims.

3.3.3 Security and Privacy Analysis

PoLO combines hash-based watermark chaining, backward verification, and shard-level performance checks to ensure training effort and ownership are provable, tamper-resistant, and cryptographically verifiable. Building on the threat models in §3.2, we address **RQ2** by demonstrating how PoLO unifies PoL and PoO under economically rational and dual-legitimacy premises.

Training efforts

PoLO counters adversarial attempts 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 the main task performance. Leveraging the immutability of cryptographic hashes, PoLO ensures that honest training remains the only

viable path, as attackers must either bear retraining costs or the failed verification, making all shortcuts ineffective and unprofitable.

Exaggerated workload claims. A rational prover may attempt to inflate training workload by embedding excessive watermarks to fraudulently 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 shard count depends on the learning rate: Higher rates accelerate embedding, producing more shards but risking performance loss. Since rewards reflect both verified shards and model accuracy, the prover is incentivized to balance efficiency and fidelity by tuning the learning rate.

A rational prover may also 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.

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 [45] and FedIPR [31], PoLO ensures that embedded watermarks remain stealthy, attack-resistant, and robust against adversarial modifications. These techniques provide strong ownership proof in PoLO, deterring unauthorized distribution and tampering while maintaining the integrity of the model.

Ownership evasion and impersonation. 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 contractual training, where models are exchanged or transferred across entities [91]. Models without verifiable watermark traces are considered illegitimate and thus ineligible for use, resale, or transfer. As a result, pure removal attacks like weight shifting [68] or structural obfuscation [90] give no gain.

What matters is *not* the removal itself, but the required *re-embedding*: To reclaim ownership and profit, an attacker must embed a new watermark that both preserves task performance and passes watermark verification by retraining or fine-tuning. The effort and computational cost involved in crafting such a profitable forgery watermark match those of training the model from the scratch, even attackers attempt to reduce cost by freezing parts of the model parameters, as they are constrained to adopt a smaller learning rate and more training epochs to meet both

performance preservation and watermark verification requirements (cf. Table 3.3, Figs. 3.4–3.5). This makes ownership impersonation through watermark forgery and replacement economically impractical.

Dataset privacy

PoLO enhances data privacy by removing the need for provers to share training data during verification. Unlike prior PoL methods that rely on dataset disclosure, PoLO achieves verifiability through obfuscation-protected model checkpoints and chained watermarks. To mitigate inference risks from gradient leakage, Gaussian noise is applied at the end of each shard, ensuring the prover's dataset remains secure [97].

3.4 Experiments

We 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).

3.4.1 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 micro-server infrastructure, providing a standardized and controlled environment for consistent model evaluation and comparison.

Implementation environment. The system features dual Intel Xeon Gold 6126 CPUs (12 cores each), 192GB of 2666MHz ECC DDR4 RAM across six channels, and 2×1.2TB 10,000 RPM SAS II drives in RAID 1. It is powered by two NVIDIA Tesla V100 GPUs with 5,120 CUDA cores, 640 Tensor cores, and 32GB memory each, running on 64-bit Ubuntu 18.04.

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 adopt a diverse set of neural architectures: *AlexNet*, *ResNet18/34*, *WideResNet*, *VGG16*, *TextCNN*, and *MiniBert*. *AlexNet* and *VGG16* offer strong baselines for vision tasks; *ResNet* introduces residual connections for deeper training; *WideResNet* boosts accuracy via increased layer width. For NLP, *TextCNN* captures local features through one-dimensional convolutions, while *MiniBert* provides efficient contextual embedding via transformer-based pretraining.

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 that the attacker $\mathcal{A}$ has access to all checkpoint models W_x, they may attempt to forge a PoLO proof $\mathbb{P}'$. To assess PoLO's robustness, we evaluate two representative attack strategies.

- *Overhaul Fine-tuning Attack (OFA) with training data.* OFA attempts to replace the original watermark and falsely assert ownership by fine-tuning [92], [98]. Given access to checkpoint models but not the legitimate keys (k_x, Y), the attacker $\mathcal{A}$ forges illicit proofs $\mathbb{P}'$ by fine-tuning all model weights to replace the authentic proof $\mathbb{P}$. 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) without training data.* WMA is analogous to pruning [93] and watermark overlap [94] 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 an attacker $\mathcal{A}$ can generate a forged proof $\mathbb{P}'$ with lower computational cost than a legitimate proof $\mathbb{P}$ while successfully corrupting the original watermarks Λ_x, the forgery may pass verification.

Evaluation metrics. We consider three metrics:

- *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* measures computational efficiency in two aspects: (1) the additional cost of generating PoLO should be minimal relative to the overall training workload, and (2) the cost of verifying PoLO should be significantly lower than its generation, ensuring practical deployment. In our experiments, time costs are also converted into monetary values (USD) based on a cost-based pricing model [99].
- *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.

Experiments are based on the above settings to validate the superiority of PoLO.

3.4.2 Compatibility

The PoLO framework exhibits strong versatility by supporting diverse watermarking schemes, including RIGA [45], EDNN [10], and FedIPR [31]. This flexibility allows for generating PoLO proofs $\mathbb{P}$ while establishing ownership using different watermarking strategies. During training,

Table 3.1: Comparison of shard rate $|s|$, model performance Acc_{main}, and proof 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 [45]	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 [10]	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 [31]	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.

the convergence of embedded watermarks reflects training progress, since the watermark convergence is directly governed by the model's weight updates during optimisation. Each watermark shard quantifies the training effort, serving as verifiable evidence of computation invested. To assess the computational efficiency of PoLO, we use the shard rate $|s|$ as the 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.3.1, $|s|$ varies 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 exhibits higher $|s|$ under all watermarking configurations. Notably, the Acc_{main} values remain stable across different watermarking approaches, with minimal deviation from baseline model performance. Furthermore, the total computational time for training and $\mathbb{P}$ generation remains relatively unchanged across different watermarking schemes, indicating that computational overhead remains consistent regardless of the chosen watermarking method.

Table 3.2: Comparison between PoLO and existing PoL in terms of the time consumption of proof generation (+training) and verification.

Watermark **size** (bit)	**Time(s)**	**Methods**	CIFAR-10 AlexNet	CIFAR-10 ResNet18	CIFAR-100 ResNet18	CIFAR-100 WideResNet	TinyImageNet VGG16	TinyImageNet ResNet34	AG News TextCNN	AG News MiniBert
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) [19] follows a similar PoL process to that of PoL (Vanilla) [17], [35], [37], with the key difference being the hash computation, which takes only a negligible amount of time.
• PoL (zkp) [18] 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 $\sim$ 20s on VGG-16 with TinyImageNet.

Takeaway (high compatibility). PoLO supports a wide range of watermarking schemes and can maintain stable main task accuracy and consistent computational overhead, enabling seamless integration into different implementations without compromising performance.

3.4.3 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 PoLO framework, we conduct comparative analyses against traditional PoL schemes [17], [18], [19], [35], [37] 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.3.2 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 training–a 3.11% increase. In contrast, conventional PoL methods require generation time nearly identical to baseline training.

PoLO's efficiency is most evident during verification, which takes far less time than proof generation. As shown in Tab.3.2, the lowest verification-to-generation ratio occurs with VGG16 on TinyImageNet, requiring only 1.3–1.8% of the generation time, and remains under 13% across all models. In contrast, conventional PoL methods require verification times nearly as long as generation, with overheads reaching 98%. PoLO, by embedding watermarking into training, adds only minimal overhead to $\mathbb{P}$ generation. Though it slightly extends the runtime over baseline training, the increase is small and well justified. Traditional PoL, which stores

Table 3.3: Time consumption, legitimate watermark detection rate η, Acc_{main}, and monetary cost of launching attacks against PoLO.

Watermark **size** (bits)	**PoLO forgery attacks**		CIFAR-10 AlexNet	CIFAR-10 ResNet18	CIFAR-100 ResNet18	CIFAR-100 WideResNet	TinyImageNet VGG16	TinyImageNet 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} / Monetary cost	PoLO	89.22% / $0.17	91.89% / $0.28	74.81% / $0.30	72.83% / $1.66	73.00% / $4.45	72.46% / $1.30	91.04% / $0.19	91.88% / $0.45
		launch OFA (cost ↑)	87.47% / $0.30	91.72% / $0.48	70.00% / $0.42	69.84% / $2.75	73.91% / $20.22	71.94% / $5.41	90.37% / $0.40	91.45% / $1.41
		launch WMA (Acc ↓)	47.52% / $0.01	80.37% / $0.01	36.84% / $0.01	46.56% / $0.04	43.25% / $0.06	39.81% / $0.03	58.69% / $0.01	55.20% / $0.01
1024	Time(s)	PoLO	1066.78	1861.62	1681.42	8968.80	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} / Monetary cost	PoLO	88.94% / $0.18	91.96% / $0.33	74.93% / $0.39	72.96% / $1.59	73.33% / $4.43	73.31% / $1.28	90.94% / $0.19	92.06% / $0.47
		launch OFA (cost ↑)	87.45% / $0.20	91.68% / $0.37	69.15% / $0.32	69.38% / $1.66	73.65% / $10.92	72.00% / $3.16	90.36% / $0.24	91.62% / $0.81
		launch WMA (Acc ↓)	39.07% / $0.01	79.30% / $0.01	34.35% / $0.01	40.52% / $0.04	45.31% / $0.06	31.09% / $0.02	61.19% / $0.01	58.96% / $0.01
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} / Monetary cost	PoLO	89.54% / $0.17	92.22% / $0.31	75.18% / $0.32	72.64% / $1.50	73.91% / $4.52	73.22% / $1.20	90.65% / $0.18	91.82% / $0.44
		launch OFA (cost ↑)	87.49% / $0.19	91.80% / $0.33	69.61% / $0.34	69.41% / $1.66	73.76% / $10.94	72.17% / $3.26	90.38% / $0.24	91.60% / $0.81
		launch WMA (Acc ↓)	40.50% / $0.01	79.51% / $0.01	38.88% / $0.01	40.86% / $0.03	37.97% / $0.06	38.41% / $0.03	65.39% / $0.01	65.19% / $0.01

• 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).

intermediate models and datasets, incurs generation times comparable to full training.

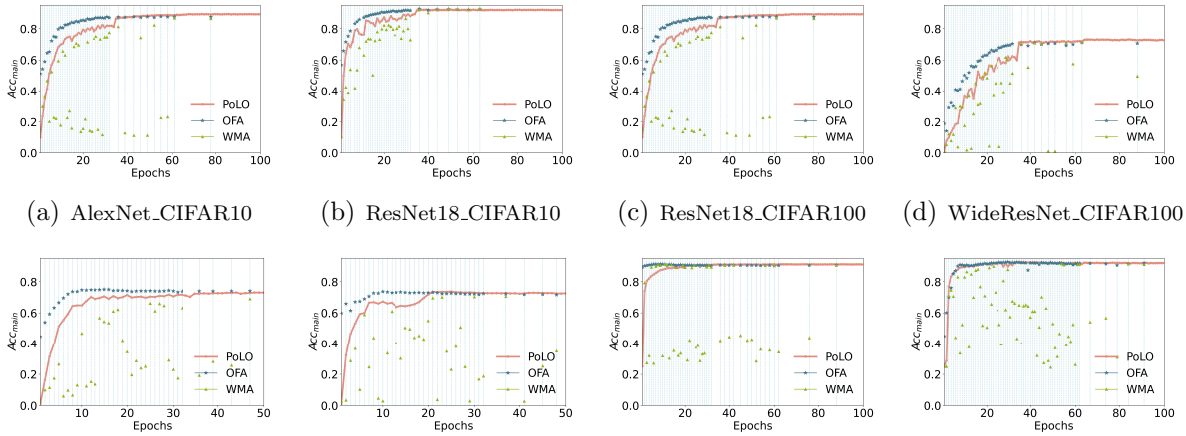
Takeaway (faster and privacy-preserving verification). With similar proof generation time, PoLO reduces verification time to as low as 1.3–1.8%, compared to up to 98% in conventional PoL schemes. By eliminating the need to retrain intermediate models, PoLO achieves efficient, scalable, and privacy-preserving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

3.4.4 Unforgeability

To assess PoLO's resistance to forgery, we evaluate two attack strategies: OFA and WMA. All attacks are run under identical computational conditions. The legitimate model embeds a 2048-bit watermark, while attackers attempt to insert illicit watermarks of 2048, 1024, and 512 bits. Effectiveness is measured by three metrics: computational overhead, post-attack detection rate η of the legitimate watermark, and main task accuracy Acc_{main}. In OFA, attackers apply RIGA-based embedding and fine-tune all model weights with a reduced learning rate [80] (10% of the original learning rate) to embed their watermark while disrupting the original. In WMA, attackers construct and insert weight values based on their illicit watermark and key, selectively overwriting specific model parameters.

Fig.3.4 shows the impact of PoLO forgery attacks across different shard stages. The watermark embedding speed (shard changing rate) is sensitive to the learning rate. Fig.3.4 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 10 [100]. 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.3.3 demonstrates the economic impracticality of forging PoLO under rational constraints. OFA, which attempts to erase and re-embed watermarks through full-model fine-tuning, is highly resource-intensive. For instance, on VGG16 with TinyImageNet, OFA takes



(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 3.4: 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.

4,703.23 seconds and costs $0.88, nearly five times higher than the 993.12 seconds and $0.17 required for legitimate PoLO proof generation. Despite this high cost, the legitimate watermark remains largely intact, with detection rates (η) ranging from 64.27% to 98.47%. WMA is more efficient in terms of runtime and cost, ranging from 11.32 seconds ($0.01) to 344.94 seconds ($0.06), yet it proves equally ineffective. Detection of the original watermark remains consistently above 90%, reaching up to 99.81%. Additionally, as shown in Fig.3.4, WMA introduces severe instability in the main task performance, causing models to fail verification.

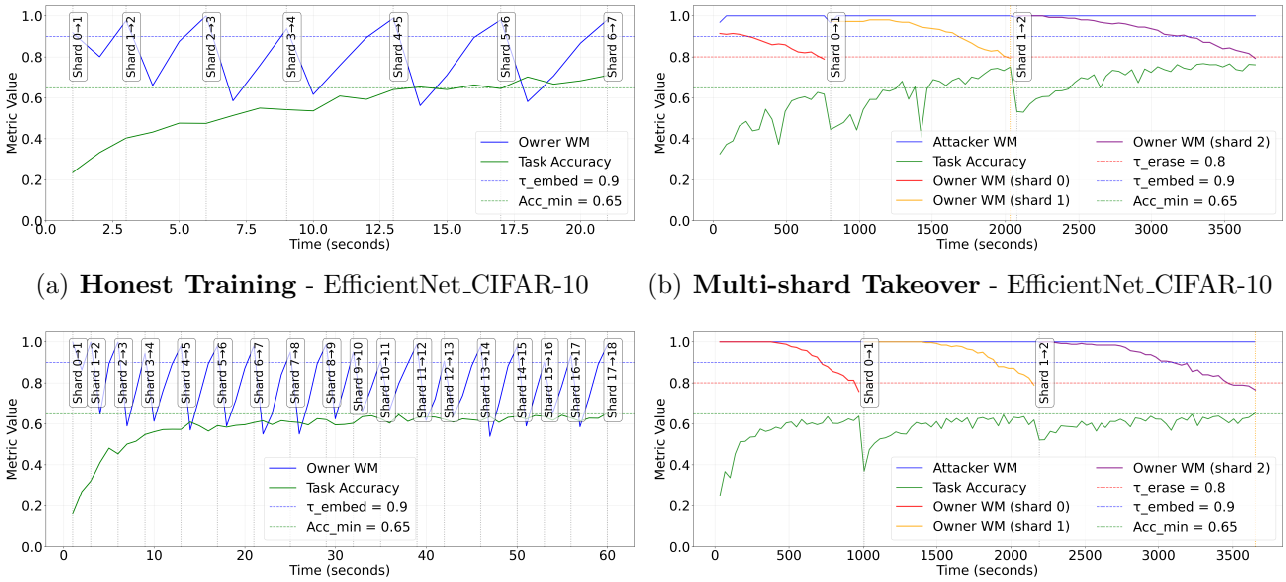
These results reflect the fundamental limitations of both attack strategies. OFA requires extensive fine-tuning across epochs, incurring computational and monetary costs comparable to or exceeding those of honest training, which violates the assumption of economic rationality and renders the attack impractical. WMA avoids training by directly modifying weights, but its inability to precisely target embedded watermarks and its tendency to degrade model performance undermine its effectiveness. In both cases, attackers either waste resources without benefit or produce unverifiable models, making forgery economically and functionally unviable under rational incentives.

Takeaway (robust against forgery with practical cost barriers). Both OFA and WMA fail to compromise the legitimacy of PoLO: OFA incurs up to 5× the cost of proof generation while still yielding detection rates between 64.27% and 98.47%, and WMA, though cheaper, remains ineffective with detection consistently above 90%.

3.4.5 Supplementary Evaluation

Multi-shard Takeover: OFA Analysis

To further examine the practicality of ownership forgery under PoLO, we provide a supplementary analysis of a shard-wise instantiation of the OFA attack. Unlike conventional single-shot fine-tuning attack that targets only the final model, the OFA attack is a multi-shard takeover that applies fine-tuning sequentially across shard checkpoints $\{W_x\}$, aiming to progressively erase the owner's chained watermarks $\{\Lambda_x\}$ and embed attacker-controlled watermarks while



(a) **Honest Training** - EfficientNet_CIFAR-10

(b) **Multi-shard Takeover** - EfficientNet_CIFAR-10

(c) **Honest Training** - ResNet18_CIFAR-100

(d) **Multi-shard Takeover** - ResNet18_CIFAR-100

Figure 3.5: Honest training showing owner watermark embedding and accuracy progression across shards. Multi-shard attack showing sequential takeover attempts with per-shard owner watermark erasure, attacker watermark embedding, and maintained accuracy.

preserving main-task accuracy. This setting directly stress-tests whether OFA remains viable once ownership claims are cryptographically bound to the full training trajectory.

We consider an attacker who has access to all obfuscation-protected checkpoint models W_x but does not possess the legitimate watermark keys and embedding position information (k_x, Y). At each shard, the attacker fine-tunes the model to jointly suppress the owner's watermark and inject a forged watermark Λ'_x. The attack objective is formalized as the following optimization problem:

$$l_{\text{ofa}} = l_w + \lambda_{\Lambda'} l_{\Lambda'} + \lambda_{\text{blind}} l_{\text{pert}}, \tag{3.7}$$

where l_w denotes the main task loss, $l_{\Lambda'}$ enforces the attacker's target watermark, and l_{pert} penalizes excessive deviation from the shard-initial parameters to maintain model fidelity. Specifically,

$$l_{\text{pert}} = \sum_{w \in W'_x} \mathbb{E}\left[\left\|w - w^{(0)}\right\|_2^2\right], \tag{3.8}$$

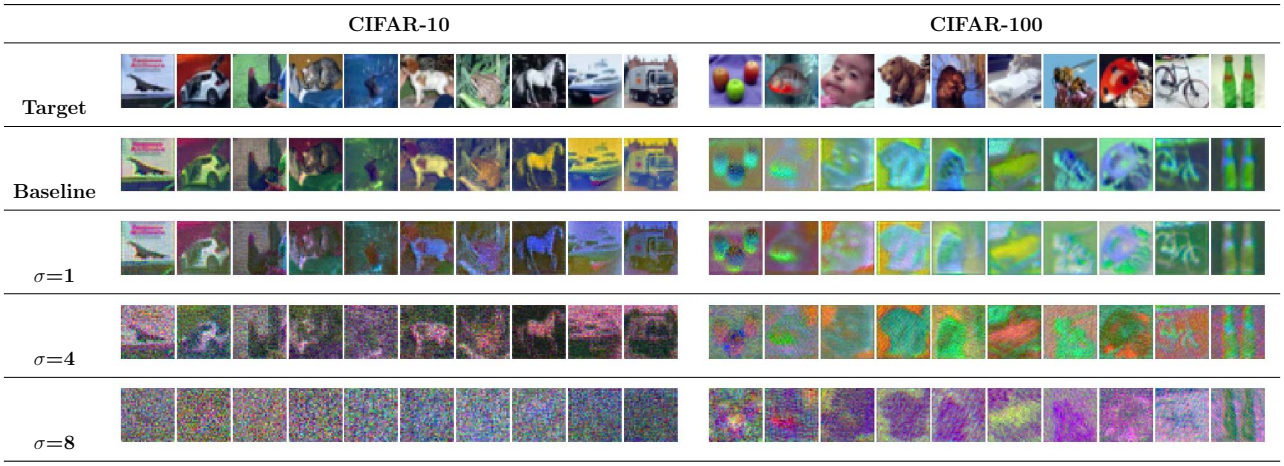
where $w^{(0)}$ denotes the checkpoint parameters at the beginning of the shard W_x, while W'_x represents all trainable weights in the attack process. Here, we set $\lambda_{\text{blind}} = 0.1$, following standard OFA practice to balance watermark suppression and task accuracy preservation.

Fig.3.5 visualizes both honest training and multi-shard takeover OFA on EfficientNet with CIFAR-10 and ResNet18 with CIFAR-100. Under honest training, the owner's watermark detection rate η exhibits a clear shard-wise embedding pattern while the main-task accuracy increases steadily. In contrast, during a multi-shard takeover, the attacker must repeatedly fine-tune each shard checkpoint for extended durations in order to reduce the legitimate watermark below the erasure threshold and embed a forged one. Because multi-shard takeover incurs substantially higher wall-clock cost than honest training, we only snapshot the first three shards in the attack plots for clarity.

More importantly, the attack fails to achieve a favorable trade-off between watermark erasure and task fidelity. As shown in Figs.3.5(b) and 3.5(d), suppressing the owner's legitimate watermark inevitably requires aggressive fine-tuning that either degrades the main-task accuracy or demands prolonged optimization comparable to, or exceeding, honest shard training. Across all settings, attempts to maintain main task accuracy above the threshold leave detectable remnants of the owner's watermark, while successful erasure of the watermark leads to noticeable accuracy degradation that violates the verifier's acceptance criteria.

These observations reinforce the core security intuition of PoLO. Because each shard watermark Λ_x is deterministically derived from the preceding checkpoint, multi-shard takeover forces the attacker to reconstruct the training process shard by shard. Any attempt to shortcut this process either fails watermark verification or violates the main task performance constraints. Consequently, multi-shard OFA becomes economically irrational in the EMM setting, requiring cost comparable to or exceeding honest training while offering no reliable path to successful ownership forgery.

Table 3.4: Visual comparison of reconstructed samples under different Gaussian noise levels (σ) for CIFAR-10 and CIFAR-100.



• Reconstructions from unprotected checkpoints exhibit relatively high similarity with the private data ($SSIM = 0.49$ on CIFAR-10 and 0.25 on CIFAR-100). When Gaussian noise is injected at release time, reconstruction quality degrades substantially: on CIFAR-10, $SSIM$ decreases to 0.39, 0.26, and 0.06; on CIFAR-100, it decreases to 0.18, 0.14, and 0.09 for σ=1,4,8, respectively.

Data Inference Attackers

To demonstrate that Gaussian noise injection can effectively protect model weights against reconstruction attempts, we have evaluated a white-box, RecoNN-based reconstruction membership inference attack (MIA) [74]. In this setting, the adversary is assumed to know the training pipeline and architecture, has access to the released checkpoint, and holds a reference dataset from the same distribution, but cannot directly observe the private training data. The adversary's goal is to reconstruct the private training samples from the released checkpoint. Following a shadow-model & reconstructor approach, shadow models are first trained on the reference dataset, and a lightweight convolutional decoder is then trained to map model weights and outputs to image reconstructions. At test time, the released checkpoint is fed into the reconstructor to generate guesses of the private training samples.

Structural Similarity Index ($SSIM$) is adopted to quantify the similarity between reconstructed and original images. As shown in Tab.3.4, visual comparisons further confirm that increasing the noise strength renders reconstructed samples almost indistinguishable from random noise.

Table 3.5: Effect of learning rate (LR) and reference threshold (η_G) on shard rate and main-task accuracy. Values are averaged.

		ResNet18, CIFAR-10			TextCNN, AG News		
	Metric	**0.01**	**0.1**	**0.2**	**0.0005**	**0.005**	**0.01**
LR	epochs	120	113	101	105	101	101
	shards	44	73	79	67	72	90
	$\lvert s\rvert$ (%)	36.67	64.61	78.21	63.81	71.28	88.51
	Acc_{main} (%)	91.88	91.66	92.11	90.53	90.98	88.51
	Metric	**0.85**	**0.95**	**0.99**	**0.85**	**0.95**	**0.99**
η_G	epochs	105	101	113	100	102	101
	shards	91	78	73	95	94	72
	$\lvert s\rvert$ (%)	86.67	77.22	64.61	95.00	92.25	71.28
	Acc_{main} (%)	92.21	91.89	92.02	90.76	90.31	90.98

These results demonstrate that Gaussian perturbation of released checkpoints provides an effective defense against data reconstruction MIAs.

Configuration of Shard

To examine how learning rate LR and watermark detection threshold η_G influence shard formation, we conducted experiments on two benchmark datasets: CIFAR-10 with ResNet18, and AG News with TextCNN. In particular, we varied LR and η_G, and measured their impact on shard count, shard rate $|s|$, and the main task accuracy Acc_{main}.

The results are reported in Tab.3.5. We observe that increasing the LR consistently raises the $|s|$; for instance, on CIFAR-10, $|s|$ grows from 0.36 at LR=0.01 to 0.78 at LR=0.2, and on AG News, from 0.63 at LR=0.0005 to 0.89 at LR=0.01. In contrast, increasing η_G reduces $|s|$, as stricter requirements slow down watermark embedding and result in fewer validated shards. For example, on CIFAR-10, $|s|$ drops from 0.86 at η_G=0.85 to 0.64 at η_G=0.99. Importantly, across all LR and η_G configurations, Acc_{main} ($\approx$0.90–0.92), indicating that PoLO preserves task performance while allowing shard dynamics to be flexibly tuned.

3.5 Concluding Remarks

We introduced PoLO, a unified framework that integrates PoL and PoO via chained watermarking. For **RQ1**, we transformed PoO into PoL by embedding evolving watermarks throughout training, enabling verifiable effort tracking across the full process. For **RQ2**, we used cryptographic hash chains in place of gradient-based proofs, ensuring tamper resistance and forgery resilience. In economically driven settings where ownership enables monetization and forgery must be cost-effective, PoLO raises an economic barrier: forgery demands effort comparable to honest training or results in unverifiable models. Unlike other methods that expose data or require recomputation, PoLO enables efficient, privacy-preserving verification. Experiments show PoLO achieves **99%** detection accuracy, reduces verification cost to **1.5%–10%**, and demands **1.1x–4.0x** more effort to forge than to generate legitimately, while retaining over **90%** accuracy after attacks.

Chapter 4

NNFMAC: A Neural Network Fingerprinting-based Model Authentication Code Scheme

4.1 Introduction

Deep Neural Networks (DNNs) have become ubiquitous in various applications [1], [2], [3], and the required computational resources are increasing along with the complexity of the models. AI development and deployment practices, such as model exchanges in federated learning [32], [101], [102] and sharing in open-source communities, have made model sharing increasingly common and essential. However, this accessibility creates opportunities for malicious users to claim ownership [103] or plagiarize models by copying their structure and parameters [9], raising serious IP concerns [104]. Model authentication is necessary for: (i) Protecting against unauthorized model copying [9], theft, and plagiarism attempts. (ii) Validating legitimate ownership claims and intellectual property rights of model contributors [105]. (iii) Maintaining trust and accountability in model sharing ecosystems [106]. Unlike traditional message authentication schemes that only verify exact matches, neural network authentication methods such as watermarking [10], [45], [46], [47], [48] and fingerprinting [11], [49], [50], [51], [57] are specifically designed to confirm model uniqueness and detect theft and plagiarism while tolerating minor model modifications. Recent research has focused on optimizing the trade-off between attack resilience and model uniqueness verification, while minimizing interference on model training and performance.

Among various model authentication approaches, watermarking has emerged as a promising solution for model IP protection by embedding verifiable information into models. Recent watermarking schemes [10], [44], [45], [46], [47], [48] typically modify the model's loss function by incorporating an embedding regularizer during training. While watermarking enables ownership verification, the intrusive nature of modifying weights introduces critical challenges that limit its practical adoption:

- *Accuracy degradation*: The embedding regularizer forces the model to converge to suboptimal solutions, causing noticeable accuracy drops [10], [47], for example, accuracy drops of up to 1.53% [10]. Such degradation is unacceptable for safety-critical applications like autonomous driving or medical diagnosis, where model reliability is paramount.

- *Limited applicability and training inefficiency*: Since watermarking requires integration with the training process, it can only be applied during model training or fine-tuning scenarios, limiting its applicability for protecting pre-trained models or when access to the training process is restricted. The additional regularization term of watermarking substantially increases training complexity and prolongs convergence. With studies showing watermarked models requiring up to twice the training time compared to non-watermarked versions [44], [45].
- *Security vulnerabilities*: The explicit modification of weight distributions creates detectable patterns that sophisticated adversaries may identify and exploit [38]. This undermines the fundamental security requirement of watermarking schemes to be concealed and tamper-resistant.

To overcome these drawbacks, model fingerprinting [11], [49], [50], [51], [57] provides a non-intrusive method for protecting model IP by encoding the model's unique and inherent features as its distinctive fingerprint. While this approach avoids modifying weights, existing fingerprinting methods face several critical limitations that hinder their practical adoption:

- *Lack of ownership verification*: Fingerprinting schemes verify uniqueness but lack ownership validation, allowing potential false ownership claims through fingerprint matching.
- *Task limitations*: Some fingerprinting methods [11] only work for classification tasks by using classification boundaries, preventing their use in other NN models.
- *High computation cost*: Existing schemes [49], [50], [51] that support multiple tasks require significant computational resources for fingerprint generation and verification.
- *Attack vulnerability*: Public fingerprinting algorithms allow attackers to modify weights while preserving model function but avoiding detection.

In this paper, we propose a novel Neural Network (NN) Fingerprinting-based Model Authentication Code (NNFMAC) scheme that simultaneously authenticates model uniqueness and verifies model ownership. NNFMAC motivates the two research questions:

- **RQ3:** *How can model ownership be bound without intrusive weight modifications or training-time changes?*
- **RQ4:** *How can we authenticate model uniqueness and verify explicit ownership claims while remaining robust to standard post-training edits and attacks?*

NNFMAC can be applied to any NN model in a post-training manner without interfering with the training process. NNFMAC first leverages critical model weights to generate a unique bit sequence that serves as a fingerprint to identify the trained model and resist plagiarism. This fingerprint is then used as a codebook to encode ownership information. By combining model-specific fingerprinting with ownership encoding, NNFMAC provides a comprehensive solution that enhances both security and reliability of authentication while maintaining the model's integrity.

NNFMAC consists of three key components. First, critical model weights are identified and randomly selected across NN layers. The selected weights serve as the foundation for generating unique model fingerprints. Then, we design a median-threshold approach to convert these numerical weights into ordered binary sequences, creating robust fingerprints resilient to various attacks while concealing the actual weight values. Third, we develop a new redundant–index-based encoding scheme that leverages the weight binary sequences as the codebook to encode ownership information. We propose redundant encoding that enhances robustness by repeating each ownership bit multiple times, thereby enhancing robustness. In addition, we design an index-based encoding mechanism that strategically maps the redundant ownership bits to

specific position indexes using the binary weight sequences. This ensures independence between the ownership information and verification codes, while effectively combining the unique properties of both the binary weight patterns and the ownership data. Through this process, NNFMAC generates authentication codes that simultaneously verify both model ownership and uniqueness.

The contributions are summarized as follows.

- We propose NNFMAC, a new non-intrusive NN authentication scheme that leverages model fingerprinting to verify model uniqueness and encode ownership information. Unlike existing watermarking approaches that modify weight distributions and degrade performance, or fingerprinting methods that lack ownership validation and are task-specific, NNFMAC generates efficient fingerprints from model weights and utilizes them as codebooks to encode ownership information. NNFMAC achieves robust authentication while fully preserving the original model performance, offering a comprehensive solution for NN IP protection.
- We develop a median-based binarization algorithm that transforms NN weights into distinctive binary sequences, which serve as robust fingerprints and codebooks for encoding ownership information. This binarization algorithm effectively masks the actual weight values while demonstrating strong resilience against various attacks. Through the strategic random selection of significant model weights and the median-based binarization, NNFMAC achieves robust protection against pruning, weight shifting, and adaptive weight perturbation attacks.
- We design a new authentication code generation function that combines fingerprints with ownership information through an index-based sequential encoding way. This function demonstrates high sensitivity to minor fingerprint variations, enabling precise and reliable model authentication. By implementing redundant encoding of ownership data across distributed fingerprint positions, NNFMAC achieves strong resilience against weight perturbation attacks while ensuring reliable ownership verification.

We comprehensively evaluate our scheme, focusing on its effectiveness, efficiency, and robustness through the comparison with state-of-the-art studies [10], [11], [45], [46], [47], [48], [57]. Extensive experiments show that NNFMAC maintains model accuracy without introducing additional training overhead, while the other watermark schemes sacrifice accuracy by 0.36% to 1.53%. NNFMAC exhibits strong robustness against various attacks, with bit error rates of 0.12 under the weight perturbation attack, 0.03 under the fine-tuning attack, 0.08 under the pruning attack, and 0.09 under the weight shifting attacks, compared to the bit error rates of 0.51, 0.49, 0.46, and 0.22 from [57] under the same attack settings, respectively. Experimental results further confirm the superiority of the proposed scheme over the other state-of-the-art (SoTA) methods [11], [45], [46], [47], [48], [57].

4.2 Problem Statement

Our objective is to design a fingerprinting-based model authentication scheme that encodes model ownership information using a unique model fingerprint to safeguard the model IP. The fingerprinting-based design preserves model performance while ensuring high security and robustness against diverse attacks.

4.2.1 Requirement for Model Authentication

Following existing works on model IP protection [10], [48], [80], the requirements for model authentication schemes can be summarized as follows:

- **Robustness.** The model authentication scheme should demonstrate strong resilience against various plagiaristic attacks, including model fine-tuning and pruning, to ensure continued protection of the model's IP.
- **Ownership Verification.** The model ownership information should be verifiable with high confidence when the correct authentication key is provided.
- **Model Uniqueness.** The model authentication scheme should confirm the model uniqueness and not be linked to other models falsely.
- **Generality.** The model authentication scheme should be adaptable to different host models and NNs.
- **Fidelity.** The model authentication scheme should preserve the original model performance.
- **Time Complexity.** The model authentication scheme should maintain the original training efficiency without affecting the model's convergence rate.
- **Concealment.** The model authentication scheme should remain undetectable and not reveal model parameters.

Balancing these requirements simultaneously can be challenging, yet it is essential to seek the optimal trade-off [107].

4.2.2 Threats

In model authentication schemes, attackers may target either the model itself or the embedded ownership information in order to steal or plagiarize it. We consider the white-box setting [107] where attackers can *adaptively* leverage various attacks in the sense that they are aware of the design details of the protection method being used, and are able to target specific weaknesses in an adaptive way. The attackers' goal is to mislead authentication verification by manipulating the model or the associated ownership information. Take the white-box watermark schemes as an example, attackers have access to the internal structure and parameters of the host model, enabling them to employ various methods to detect the presence of a watermark. For instance, they may compare weight distributions with non-watermarked models or inject noise to erase the watermark once detected.

Possible threats to model authentication schemes are outlined below.

Weight perturbation. The attackers can plagiarize by perturbing the model weight with noise, aiming to bypass the model authentication. Adding Gaussian noise can serve as an effective means of weight perturbation attacks [68]. However, excessive noise addition may compromise model accuracy.

Model fine-tuning. Model fine-tuning trains a given model with specific datasets to train the model on new tasks [92]. Model fine-tuning can be considered as an attack method against the watermark. Attackers may fine-tune the model to change model weights and fingerprint to bypass model authentication.

Model pruning. Model pruning removes unimportant neurons or weights to compress models for lightweight devices. Attackers can employ model pruning to change the model fingerprint and then bypass model authentication [108], [109].

Weight shifting. Weight shifting is an adaptive attack way in the white-box setting [68], which introduces perturbations to all the convolutional kernels of the host model and subsequently fine-tunes the perturbed host model to restore its test accuracy. It is notably effective and efficient against EWDNN [10] and DeepMarks [48] watermarking methods [68].

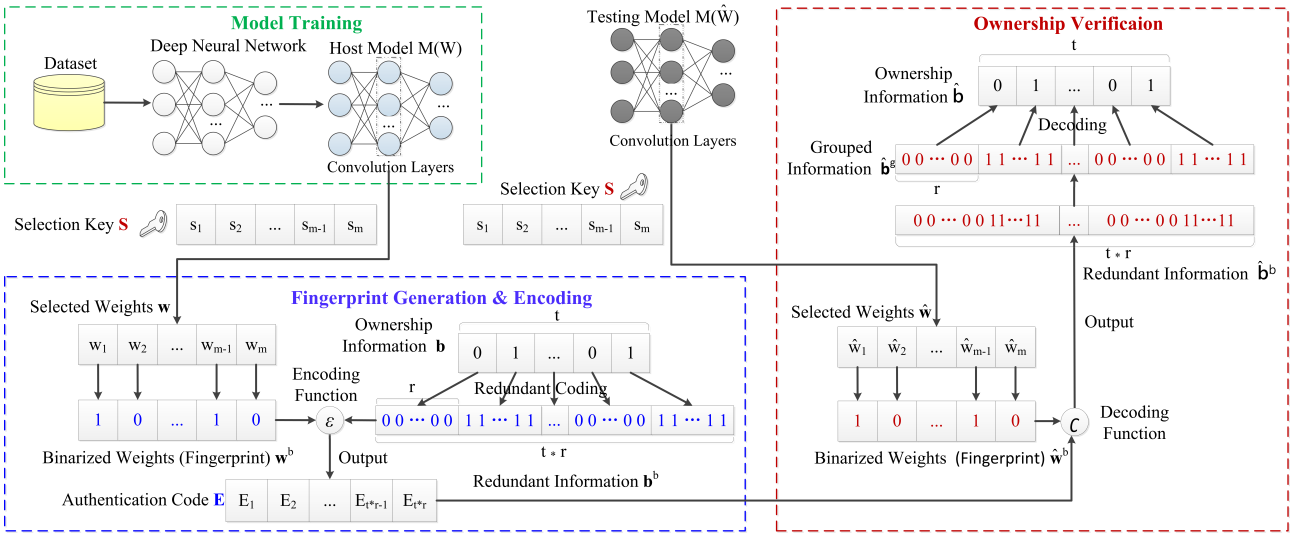


Figure 4.1: **NNFMAC design**: In the fingerprint generation and ownership information coding process, the selection key **S** (private) is utilized to filter the most crucial weights **w** (private), which are then binarized to the binary weight $\mathbf{w}^b$ as the fingerprint (private), while the ownership information **b** (public) undergoes redundant coding. The encoding function 4.3 outputs the authentication code **E** (private) by coding the fingerprint $\mathbf{w}^b$ and the redundant ownership information $\mathbf{b}^b$ (private). During the ownership verification, a similar process is applied to obtain the fingerprint $\hat{\mathbf{w}}^b$ (private). The redundant ownership information $\hat{\mathbf{b}}^b$ (private) is extracted using the authentication code **E** (private) and the decoding function 4.5, followed by a redundant decoding operation to retrieve the ownership information $\hat{\mathbf{b}}$ (public).

4.3 NN Fingerprinting-based Model Authentication Coding Design

We present NNFMAC, which addresses accuracy degradation, convergence slowdown, and weak concealment commonly observed in conventional watermarking approaches while surpassing model fingerprinting with greater adaptability across training tasks and reduced time consumption. The scheme leverages model fingerprints, generated from random critical model weights, to encode ownership information. The core concept is to binarize model weights using a threshold, producing a binary sequence that serves both as a fingerprint and as a codebook for ownership encoding. We then design a novel index-based ownership information encoding function to generate authentication code by coding the ownership information with the codebooks. Additionally, we propose spreading the ownership information using redundant encoding, allowing each bit of the ownership information to correspond to multiple model weights. It enhances the resilience of the model authentication against tampering and various attacks.

During the verification process, the model fingerprint is first obtained by applying the fingerprint generation process on the testing model. Subsequently, the redundant ownership information is extracted using the corresponding decoding function. Finally, the original ownership

Algorithm 3 Fingerprinting and Encoding

Require: **b** (ownership information), **S** (selection key), $M(W)$ (host model).
Ensure: authentication code **E**.

$\mathbf{w} \xleftarrow{\mathbf{S}} M(W)$ ▷ Get selected weight **w** using selection key **S**
$\mathbf{w}^b = [1 \text{ if } \mathbf{w}_i \geq \mathbf{w}_a \text{ else } 0 \text{ for } \mathbf{w}_i \in \mathbf{w}]$ ▷ Binarize the selected weights (Fingerprint obtained)
$\mathbf{b}^b = [[b_j] \times r \text{ for } b_j \in \mathbf{b}]$ ▷ Redundant coding of ownership information **b**
$\mathbf{E} = []$
for $\mathbf{b}^b_j \in \mathbf{b}^b$ **do** ▷ Encode the redundant ownership information $\mathbf{b}^b$
    for $i, \mathbf{w}^b_i \in \text{enumerate}(\mathbf{w}^b)$ **do**
        if $\mathbf{b}^b_j == \mathbf{w}^b_i$ and $i \notin \mathbf{E}$ **then**
            $\mathbf{E}.append(i)$; **break**
        end if
    end for
end for
RETURN E

information is recovered through redundant decoding for verification purposes. This complete process is illustrated in Fig. 4.1, which provides a schematic overview of the NNFMAC framework.

4.3.1 Model Training (pre-phase)

NNFMAC takes a trained model as input. This phase marks a departure from traditional watermarking methods, where watermark embedding occurs concurrently with model training. As a result, NNFMAC does not hinder the convergence rate of the model, in contrast to traditional watermarking methods that often result in a slowdown in convergence. NNFMAC can be applied to any type of NNs, including Convolutional NN (CNN), Recurrent NN (RNN), and Residual Network (ResNet) [110]. We denote the trained model as $M(W)$ and represent its weights as W. In this context, bias terms are not considered, as utilizing the weights alone to code the ownership information is adequate.

4.3.2 Fingerprinting and Ownership Encoding

Prior to initiating the ownership information encoding process, it is crucial to meticulously select the weights from the host model to generate fingerprints. To enhance concealment, we opt for weights from various convolutional layers and select a fraction of the weights from each convolutional layer. This selection process entails excluding weights from the output layer and those with smaller absolute values. This precaution is necessary due to the distinct methodologies employed in model fine-tuning, which primarily focus on retraining the weights of the output layer, and model pruning, which mainly involves removing weights or neurons with minimal absolute values. Therefore, we randomly mark m locations where the weights with significant absolute values from all convolutional layers and use the absolute values of the weights, denoted by **w**, for the next step. The locations of **w** servers as the selection key $\mathbf{S} \in \mathbb{R}^m$, which must be kept secret and used in both authentication code generation and verification. Using **S**, weights **w** can be extracted from the host model. Selecting significant absolute values helps defend against pruning attacks, since attackers typically remove less significant weights. Meanwhile, the random selection strategy can defend against adaptive weight perturbation attacks, where attackers might target specific weights to evade detection.

To establish a connection between the binary ownership information and the selected weights **w**,

we binarize $\mathbf{w}$ to $\mathbf{w}^b$ in the next step. The binarized weights $\mathbf{w}^b$ serves as the model fingerprint and also the codebook to encode the ownership information. Using the median $\mathbf{w}_a$ of $\mathbf{w}$ as the threshold, weights greater than $\mathbf{w}_a$ are assigned to 1 and weights less than $\mathbf{w}_a$ are assigned to 0. The processing of $\mathbf{w}^b$ is given by

$$\mathbf{w}^b \in \{0,1\}^m \overset{\text{binarize}}{\longleftarrow} \begin{cases} 0, & \text{if } \mathbf{w}_i < \mathbf{w}_a; \\ 1, & \text{if } \mathbf{w}_i \geq \mathbf{w}_a. \end{cases} \tag{4.1}$$

To enhance the robustness of the ownership information, Redundant Coding (RC) is used to spread the ownership information $\mathbf{b} \in \{0,1\}^t$ to the redundant ownership information $\mathbf{b}^b$. In other words, each bit of the ownership information is expanded to r bits. Multiple bits of the model fingerprint are used to encode a single bit of the ownership information, as depicted in Fig. 4.1 and (4.2). The adjustable parameter r is defined as the redundancy rate.

$$\mathbf{b}^b \in \{0,1\}^{t\times r} \leftarrow \{\underbrace{[0,\cdots,0]}_{r},\underbrace{[1,\cdots,1]}_{r}\}^t \overset{\text{RC}}{\leftarrow} \mathbf{b} \in \{0,1\}^t. \tag{4.2}$$

The redundant ownership information $\mathbf{b}^b$ is then encoded with binarized weights $\mathbf{w}^b$ to generate the authentication code $\mathbf{E} \in \mathbb{R}^{t\times r}$ using the new encoding function $\mathcal{E}$, as follows

$$\mathbf{E} = \mathcal{E}(\mathbf{b}^b, \mathbf{w}^b). \tag{4.3}$$

The encoding function $\mathcal{E}$ scans $\mathbf{b}^b$ bit by bit, it finds the same bit value with unused index in $\mathbf{w}^b$, and then appends the index of the identified bit in $\mathbf{w}^b$ to $\mathbf{E}$. To be more specific, if the j-th bit of $\mathbf{b}^b$, i.e., $\mathbf{b}^b_j$, is equal to the i-th bit of $\mathbf{w}^b$, i.e., $\mathbf{w}^b_i$, the index i is appended to the authentication code $\mathbf{E}$. Each $\mathbf{w}^b$ bit index can only be appended once. If an index has been appended, the corresponding bit element of $\mathbf{w}^b$ skips the matching. This process repeats over the $t \times r$ rounds until all bits of $\mathbf{b}^b$ are matched with bits in $\mathbf{w}^b$. Since $\mathbf{w}^b$ contains approximately half of its elements to be 1 bits and the other half to be 0 bits, m should be no less than $2(t \times r)$, i.e., $m \geq 2tr$, such that all bits in $\mathbf{b}^b$ can be encoded.

Algorithm 3 illustrates the entire model authentication code generation process, including model fingerprinting and ownership information encoding process. In Step 1, the selection key $\mathbf{S}$ is utilized to select the crucial weights from convolution layers. Step 2 is the weight binarization process, which is the fingerprint generation process shown in (4.1). Step 3 involves the process of RC, aiming at bolstering the error correction capability of the extracted ownership information, ideally to withstand various attacks. In this step, each bit of the ownership information $\mathbf{b}^b$ is extended to r bits, which is introduced in (4.2). The larger r is, the stronger the error correction ability of this scheme and the stronger robustness against fine-tuning attacks, but more weights are needed to code the redundant ownership information $\mathbf{b}^b$. Step 4 is the ownership information encoding process using the index-based encoding function. In this process, $\mathbf{b}^b_j$ is matched with $\mathbf{w}^b_i$; if they are equal, the index of $\mathbf{w}^b_i$ is appended to the authentication code $\mathbf{E}$; see (4.3). Each index of $\mathbf{w}^b_i$ is appended only once. The ownership information encoding process exclusively relies on coding and mapping operations, without any modifications to the weights, thereby ensuring the preservation of accuracy.

4.3.3 Ownership Verification

In the scenario where ownership verification of a contentious host model $M(\hat{W})$ is required, the extraction of the ownership information from the testing model $M(\hat{W})$ becomes necessary. If

Algorithm 4 Ownership Information Extraction

Require: $\mathbf{E}$ (authentication code), $\mathbf{S}$ (selection key), $M(\hat{W})$ (testing model).
Ensure: ownership information $\hat{\mathbf{b}}$.

1: $\hat{\mathbf{w}} \overset{\mathbf{S}}{\leftarrow} M(\hat{W})$ ▷ Obtain the selected weight $\hat{\mathbf{w}}$ guided by $\mathbf{S}$
2: $\hat{\mathbf{w}}^b \leftarrow [1 \text{ if } \hat{\mathbf{w}}_i \geq \hat{\mathbf{w}}_a \text{ else } 0 \text{ for } \hat{\mathbf{w}}_i \in \hat{\mathbf{w}}]$ ▷ Process $\hat{\mathbf{w}}$ to get the binarized weight $\hat{\mathbf{w}}^b$
3: $\hat{\mathbf{b}}^b \leftarrow [\hat{\mathbf{w}}^b[\mathbf{E}_i] \text{ for } \mathbf{E}_i \in \mathbf{E}]$ ▷ Using decoding function $\mathcal{C}$, collect the bits to constitute the redundant ownership information $\hat{\mathbf{b}}^b$
4: $\hat{\mathbf{b}}^g \leftarrow [\hat{\mathbf{b}}^b[i : i + r] \text{ for } i \text{ in range}(0, \text{len}(\hat{\mathbf{b}}^b), r)]$ ▷ Divide $\hat{\mathbf{b}}^b$ into groups $\hat{\mathbf{b}}^g$
5: $\hat{\mathbf{b}} \leftarrow [1 \text{ if } \hat{\mathbf{b}}_i^g.\text{count}(1) > \hat{\mathbf{b}}_i^g.\text{count}(0) \text{ else } 0 \text{ for } \hat{\mathbf{b}}_i^g \in \hat{\mathbf{b}}^g]$ ▷ Use RC decoding to decode the grouped ownership information $\hat{\mathbf{b}}^g$
6: **RETURN** $\hat{\mathbf{b}}$

ownership information can be extracted from $M(\hat{W})$ with an acceptable Bit Error Rate (BER), it enables verifying the ownership of the testing model. BER is defined by dividing the number of error-extracted ownership information bits by the total ownership information bits, as given by

$$BER = \frac{b^e}{b^t}, \tag{4.4}$$

where b^e is the number of erroneous bits, and b^t is the total number of the ownership information bit.

As described in **Algorithm** 4, ownership information extraction is the inverse of the ownership information encoding. The extraction of ownership information from the testing model $M(\hat{W})$ and the authentication code $\mathbf{E}$ requires the selection key $\mathbf{S}$. The initial step in ownership information extraction from the host model $M(\hat{W})$ involves selecting the weights, for which the selection key $\mathbf{S}$ is required to identify the chosen weights. Step 2 binarizes the selected weights to generate a fingerprint; see (4.1). The purpose of this step is to get the codebooks to extract the ownership information. The third step is the ownership information extraction: the bits of $\hat{\mathbf{b}}^b$ are gathered from $\hat{\mathbf{w}}^b \in (0, 1)^m$ using the decoding function $\mathcal{C}$, as follows

$$\hat{\mathbf{b}}^b = \mathcal{C}(\mathbf{E}, \hat{\mathbf{w}}^b). \tag{4.5}$$

The decoding function $\mathcal{C}$ goes through $\hat{\mathbf{w}}^b$ one bit after another, finds the E_i-th index in $\hat{\mathbf{w}}^b$, and then appends $\hat{\mathbf{w}}^b[\mathbf{E}_i]$ in $\hat{\mathbf{w}}^b$ to $\hat{\mathbf{b}}^b$. Step 4 divides the redundant ownership information $\hat{\mathbf{b}}^b$ of length $r \times t$ into groups, each containing r bits, forming $\hat{\mathbf{b}}^g$,

$$\hat{\mathbf{b}}^g \in \{\underbrace{[0, \cdots, 0]}_{r}, \underbrace{[1, \cdots, 1]}_{r}\}^t \overset{\text{grouping}}{\longleftarrow} \hat{\mathbf{b}}^b \in \{0, 1\}^{t \times r}. \tag{4.6}$$

In Step 5, $\hat{\mathbf{b}}^g$ is decoded to $\hat{\mathbf{b}}$, as follows

$$\hat{\mathbf{b}} \in \{0, 1\}^t \overset{\text{RC decoding}}{\longleftarrow} \hat{\mathbf{b}}^g \in \{\underbrace{[0, \cdots, 0]}_{r}, \underbrace{[1, \cdots, 1]}_{r}\}^t. \tag{4.7}$$

In each r-bit group of $\hat{\mathbf{b}}^g$, the character with higher frequency is appended to $\hat{\mathbf{b}}$. For example, if '1' appears more frequently than '0', then '1' is appended to $\hat{\mathbf{b}}$. If the frequencies are equal, one of the values is selected randomly. In other words, this redundancy maintains authentication accuracy even when up to 50% of the corresponding fingerprint bits are corrupted. Therefore, the redundant encoding strategy enhances the robustness of NNFMAC by mitigating the impact of bit-level distortions in the model fingerprint. To be more specific, if the host model

is not attacked, each r-bit element of $\hat{\mathbf{b}}^g$ consists solely of '0' or '1', and $\mathbf{b} = \hat{\mathbf{b}}$. However, under attacks such as fine-tuning or pruning, some bits of $\hat{\mathbf{b}}^g$ may be altered. Due to the redundant coding scheme, the encoding algorithm can tolerate minor changes on each r-bit element of $\hat{\mathbf{b}}^g$. This is the error correction process of ownership information extraction, which shows the benefit of the redundant coding scheme.

NNFMAC achieves strong robustness and reliable ownership verification by combining median-based weight binarization with a redundant–index-based encoding mechanism. The median-based binarization generates stable binary model fingerprints that remain resilient to weight distribution shifts caused by fine-tuning, pruning, and noise. Building on the fingerprints, the sequential index-based encoding mechanism achieves high sensitivity to model fingerprint variations through a cascading effect: when a single bit in the fingerprint changes state, it propagates modifications throughout all subsequent index positions until encountering an opposing bit flip. This carefully designed sensitivity serves two key purposes - it enables precise model differentiation while making it computationally challenging for adversaries to reverse-engineer the original fingerprint from encoded outputs. We further strengthen the NNFMAC's robustness through redundant encoding, where each ownership bit is encoded using multiple fingerprint bits. This redundancy maintains authentication accuracy even when up to 50% of the corresponding fingerprint bits are corrupted. By combining sensitive index-based encoding with redundant error tolerance, NNFMAC provides robust authentication that withstands various model modification attacks while preserving the integrity of the verification process.

4.4 Experiments and Discussions

This section evaluates the effectiveness of the proposed NNFMAC scheme through comprehensive experiments. The evaluation compares NNFMAC with existing model IP protection approaches, including watermarking schemes [10], [45], [46], [47], [48] and fingerprint schemes [11], [57]. The experimental implementation includes both publicly available and self-implemented methods. For schemes without open-source code [11], [46], [57], implementations are developed based on their publications. Official implementations are used for methods with available source code [10], [45], [47], [48]. To ensure fairness, all methods are evaluated under consistent experimental settings. The evaluation framework primarily adopts the BER as the robustness metric, aligning with several baseline methods [10], [45], [46], [47]. For comparison with HufuNet [47], the evaluation analyzes ΔPCC trends under equivalent threshold settings. The BER for IPGuard [11] is calculated as the ratio between misclassified fingerprint data points and total fingerprint samples.

4.4.1 Experimental Setup

The experiments utilize several widely-adopted machine learning benchmark datasets: **MNIST**, **MNISTLetter**, **CIFAR10**, **CIFAR100** and **TinyImageNet**. The MNIST dataset comprises 60,000 training images and 10,000 testing images across ten distinct classes, while the MNISTLetters dataset contains 62,400 training samples and 10,400 testing samples distributed among 26 letter classes. Both CIFAR10 and CIFAR100 datasets contain 60,000 images, with 50,000 allocated for training and 10,000 for testing. CIFAR10 comprises 10 distinct classes, while CIFAR100 encompasses 100 different categories. TinyImageNet represents a compact variant of the ImageNet dataset, containing 200 classes. Each class includes 500 training images, 50 validation images, and 50 test images.

All experiments are conducted on a laptop equipped with an i7-12700 CPU, RTX 4060 GPU, 32 GB RAM, and a 512 GB hard disk. The host networks include **CNN**, **ResNet** [110], **VGG** [111] and **VIT** [112], which are trained on MNIST, CIFAR10, CIFAR100 and TinyImageNet datasets to generate several diverse host models. The ownership information $\mathbf{b} \in \{0,1\}^{128}$ is a randomly generated binary string. When training CNN, the following settings are made: learning rate = 0.01, batch_size = 512, epoch = 50, momentum = 0.5, RMSprop acts as the optimizer, and CrossEntropyLoss acts as the loss function. Adam serves as the optimizer with the following settings to train ResNet, VGG, and VIT: learning rate = 0.001, batch_size = 128, epoch = 80, weight_decay= 0.0001, and the loss function is CrossEntropyLoss. In the embedding experiment, each ownership information bit is redundantly coded into 8 bits, i.e., $r = 8$, unless otherwise specified.

4.4.2 Fidelity and Time Complexity

Table 4.1: Running time (in minutes) for model training, NNFMAC code generation, and NNFMAC code verification under different ownership information lengths across various models.

Watermark Length	128 bits	256 bits	512 bits	1024 bits
CIFAR10 (ResNet)	127.9/3.2/1.2	128.3/5.3/2.4	129.1/8.3/4.0	128.9/10.4/5.3
CIFAR10 (CNN)	90.3/1.2/0.7	90.4/2.3/1.0	91.0/3.8/2.0	90.9/5.9/3.1
CIFAR100 (VGG)	134.5/5.5/2.3	135.2/8.3/4.1	134.1/11.3/5.6	135.6/15.3/7.5
CIFAR100 (ResNet)	139.3/5.1/2.7	140.2/8.3/4.4	140.2/11.5/5.3	141.1/15.3/7.4
TinyImageNet (VIT)	410.1/5.6/3.2	411.1/8.9/4.3	410.5/13.4/7.0	411.2/18.5/9.3

NNFMAC operates without an embedding regularizer, thereby preserving both model performance and training efficiency. In contrast, existing watermarking schemes [10], [45], [47] alter model weights during embedding, which constrains the models from reaching their optimal local minima. This weight modification leads to measurable degradation in model accuracy compared to non-watermarked versions. For instance, EWDNN [10] modifies weights during embedding, which adversely affects the host model's convergence speed. This necessitates additional computational time for both model training and watermark embedding processes. Similarly, HufuNet [47] and RIGA [45] introduce further complexity by requiring auxiliary neural networks for their operations. Their weight-modification approaches significantly impact training convergence rates. The cumulative effect of these requirements typically results in a twofold increase in the total training duration for watermarked models.

The time efficiency of NNFMAC scales well with increasing encoding capacity. Modern NN can encode substantial ownership information due to their millions of parameters. The maximum encoding capacity is $T_{max} = \frac{N_{weight}}{r}$, where N_{weight} is the total number of parameters and r is the redundancy rate.

Table 4.1 presents results with ownership information sizes ranging from 128 to 1024 bits, showing negligible overhead for NNFMAC code generation and verification. The largest relative overhead appears in CIFAR100 (VGG) with a 1024-bit watermark, where code generation and ownership verification account for 10.29% and 5.53% of training time, respectively. All other settings exhibit smaller overhead. These results highlight the efficiency and practicality

of NNFMAC, demonstrating a negligible impact on training time even with large ownership information sizes.

Table 4.2 shows that NNFMAC retains 100% model accuracy, while the benchmarks of watermarking schemes degrade by 0.36% to 1.53%, e.g. EWDNN [10], RIGA [45], and HufuNet [47] degrade by 0.95% to 1.53%, 0.36% to 0.83%, 0.44% to 0.89%, respectively. When achieving the same model accuracy, Table 4.3 illustrates that NNFMAC incurs a minimum time overhead with up to 0.05 training time, whereas the watermarking schemes range from 1.36 to 2.04 times, e.g., HufuNet [47] for 1.36 times, RIGA [45] for 2.04 times. The results of the experiment show that NNFMAC preserves a high accuracy in the host model, has no impact on the model convergence rate, and requires a shorter time. Overall, NNFMAC maintains model performance while introducing negligible computational overhead.

Table 4.2: The model accuracy with and without ownership information. NNFMAC preserves the accuracy while other watermarking schemes downgrade the model accuracy.

Models		Proposed scheme	EWDNN [10]	RIGA [45]	HufuNet [47]
CIFAR10 (ResNet)	Authenticated	87.61%	87.35%	85.33%	87.18%
	Original	87.61%	88.76%	86.18%	87.62%
	Ratio	**1**	0.984	0.990	0.994
CIFAR10 (CNN)	Authenticated	85.30%	84.32%	86.53%	85.53%
	Original	85.30%	85.55%	86.89%	86.26%
	Ratio	**1**	0.985	0.995	0.991
CIFAR100 (VGG)	Authenticated	65.67%	63.90%	63.15%	63.18%
	Original	65.67%	65.43%	63.98%	63.72%
	Ratio	**1**	0.976	0.987	0.991
CIFAR100 (ResNet)	Authenticated	72.45%	69.20%	77.50%	68.73%
	Original	72.45%	70.15%	78.14%	69.62%
	Ratio	**1**	0.986	0.991	0.986

Table 4.3: Comparison of various schemes in training time. The existing watermarking schemes need more time to train the host model and embed the watermark.

Models		Proposed scheme	EWDNN [10]	RIGA [45]	HufuNet [47]
CIFAR10 (ResNet)	Authenticated	131 min	685 min	532 min	384 min
	Original	128 min	490 min	265 min	272 min
	Ratio	**1.02**	1.40	2.00	1.41
CIFAR10 (CNN)	Authenticated	91 min	138 min	178 min	132 min
	Original	90 min	75 min	88 min	91 min
	Ratio	**1.01**	1.82	2.04	1.45
CIFAR100 (VGG)	Authenticated	139 min	1,190 min	205 min	205 min
	Authenticated	134 min	793 min	101 min	107 min
	Ratio	**1.04**	1.50	2.03	1.92
CIFAR100 (ResNet)	Authenticated	145 min	1,788 min	235 min	171 min
	Original	139 min	1192 min	115 min	126 min
	Ratio	**1.04**	1.50	2.04	1.36

Takeaway–greater fidelity and lower overhead. NNFMAC preserves 100% model accuracy with minimal time overhead (1.05 times) compared to other watermarking schemes that degrade accuracy and require up to twice the training time.

4.4.3 Generality and Uniqueness

To validate the ownership verification process, we encode a 128-bit ownership signature into 1024 binarized weights ($1024 = t \times r = 128 \times 8$), randomly selected from convolutional layers to enhance concealment. The ownership information is perfectly extracted ($BER = 0$) from unmodified models, demonstrating the scheme's reliable encoding and extraction capabilities across various NN architectures.

To evaluate the uniqueness of NNFMAC, we test its ability to reject false ownership claims by training seven models using identical settings and datasets as the original host model. These models serve as potential false claimants since they share maximum similarity with the host model.

We extract ownership information from these models using the original selection key **S** and authentication code **E**. As shown in Fig. 4.2, the BER ranges from 0.359 to 0.476, with higher and more stable values observed in complex networks like ResNet, VGG, and VIT.

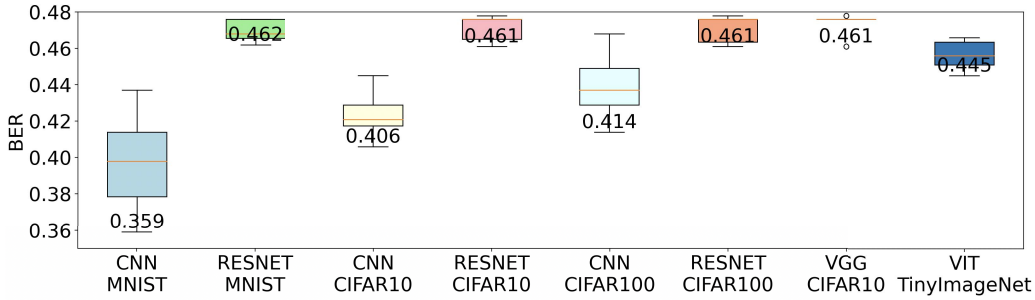


Figure 4.2: The BER of verification from the false positive models, and the minimum BER is more than 0.359.

NNFMAC successfully prevents false ownership claims by maintaining distinct signatures even when models share identical training parameters and datasets. This robustness stems from neural networks converging to different local minima with unique weight configurations [113]. Our experiments show a minimum BER of 0.359 with $r = 8$, leading us to establish a conservative verification threshold of $BER = 0.25$. Any BER value exceeding this threshold indicates potential tampering, ensuring reliable ownership verification.

Takeaway–uniqueness guaranteed. NNFMAC achieves perfect ownership extraction from unmodified models, while maintaining robust verification through a 0.25 BER threshold that identifies false claims ($BER > 0.359$).

4.4.4 Security Analysis

Ownership information Detection

Our fingerprinting-based authentication method avoids modifying weights, making it inherently more resistant to detection and removal attempts, unlike watermarking schemes that change weights and can be detected [68].

Key Sensitivity

We evaluate key sensitivity by testing how modifications to keys affect ownership information extraction across different protection schemes. We evaluate sensitivity by perturbing the keys with noise and scrambling their order.

Fig. 4.3 shows the results, with key modification degree on the x-axis. The left y-axis shows BER for EWDNN and our scheme, while the right shows ΔPCC (defined as $\Delta PCC = \left|\frac{\Delta loss}{\tau}\right|$) for HufuNet - both measure extraction accuracy under modified keys. Steeper curves indicate higher key sensitivity. Both metrics increase monotonically with the degree of key modification. NNFMAC exhibits superior sensitivity, with BER exceeding the verification threshold at only 25% key modification. In contrast, HufuNet [47] and EWDNN [10] only become sensitive after 45% and 55% modification, respectively, indicating they can still extract watermarks with substantially incorrect keys.

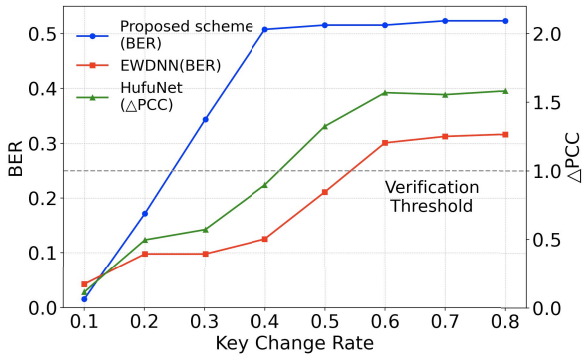


Figure 4.3: The comparison of the ownership information extraction with changed keys. The x-axis is the ratio of the keys that are replaced with random numbers. The left y-axes are BER EWDNN and NNFMAC, and ΔPCC on the right-hand side for HufuNet. The two y-axes are aligned using the thresholds of NNFMAC ($BER = 0.25$) and HufuNet ($\Delta PCC = 1$).

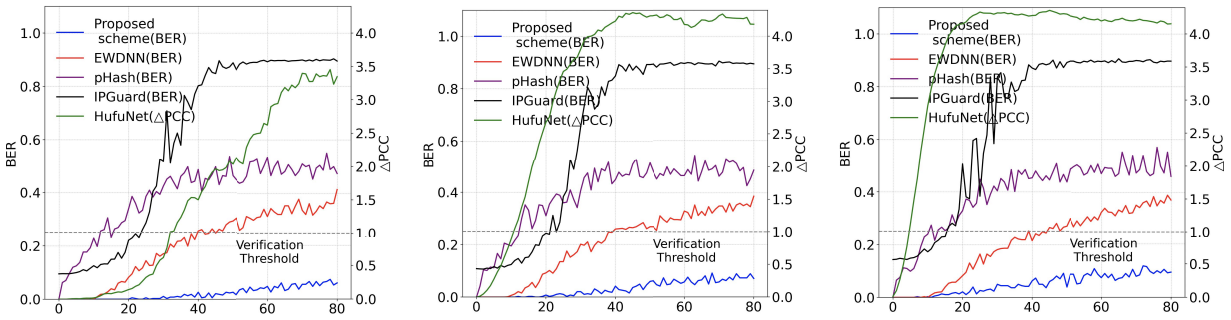
Takeaway–higher concealment and key sensitivity. NNFMAC provides superior concealment of ownership information through preserved weight distributions and enhanced key sensitivity - failing verification at just 25% key modification compared to 55% for other methods.

4.4.5 Robustness Analysis

Robustness Against Weight Perturbation Attacks

To evaluate the resilience against weight perturbation attacks, we conducted experiments using Gaussian noise with varying parameters. The noise with means of 0, 0.001, and 0.002 was applied, with variance scaled from 0 to 80 relative to average model weights.

Figs. 4.4(a) to 4.4(c) compare how different model protection methods perform under weight perturbation attacks. Each method shows distinct behavior: HufuNet fails when noise variance exceeds 30 times the average weight value, with its ΔPCC crossing the threshold. Its weakness intensifies with noise; at mean=0.002, it fails at variance=5. pHash is sensitive to noise, failing ($BER > 0.32$) at variance=20 due to its reliance on weight segment characteristics that are easily disrupted by Gaussian noise. The BER of IPGuard rises from 0.1 to 0.85 after the variance exceeds 15. NNFMAC and EWDNN show the best noise resistance, remaining stable



(a) BER: mean=0 (b) BER: mean=0.001 (c) BER: mean=0.002

Figure 4.4: The comparison of robustness against weight perturbation attacks. In this test, the Gaussian noise is added to the models, and the noise means in the figures are 0, 0.001, and 0.002, respectively. The x-axes represent the ratio of the variance of the Gaussian noises to the average model weights, from 0 to 80. The y-axes of Figs. 4.4(a) to 4.4(c) denote BER on the left-hand side for EWDNN and NNFMAC and ΔPCC on the right-hand side for HufuNet. The two y-axes are aligned using the thresholds of the proposed scheme ($BER = 0.25$) and HufuNet ($\Delta PCC = 1$).

up to variance=50. Even under severe noise (mean=0.02, variance=80), our scheme maintains a low BER of 0.12, while EWDNN and HufuNet degrade significantly (BER=0.4, ΔPCC=4.3) beyond their thresholds.

The experimental results indicate that methods lacking redundant error-correction mechanisms show increased vulnerability to noise perturbations. The proposed NNFMAC scheme, through its robust design, demonstrates consistent resilience against weight perturbation attacks across various noise conditions.

Robustness Against Fine-Tuning Attacks

Table 4.4: The BER under fine-tuning and transfer learning scenarios. The arrow (→) indicates the direction of transfer, where the left side represents the source dataset used for initial training and the right side represents the target dataset used for fine-tuning or transfer learning.

Epochs	MNISTLetter → MNIST	MNIST → MNISTLetter	CIFAR100 → CIFAR10	CIFAR10 → CIFAR100	TinyImageNet → CIFAR10
10	0	0	0	0	0
30	0	0	0.0156	0.012	0.023
50	0.109	0.007	0.0156	0.024	0.052
70	0.109	0.031	0.0156	0.031	0.089
90	0.109	0.031	0.0156	0.031	0.102
100	0.109	0.031	0.0156	0.031	0.124

The evaluation of robustness against fine-tuning attacks involves experiments across multiple datasets and model architectures. The experimental setup consists of five host models trained on MNIST, MNISTLetter, CIFAR10, CIFAR100, and TinyImageNet datasets. For each model, the authentication codes are generated before fine-tuning, and verification tests are performed on the fine-tuned models to assess the robustness of the authentication scheme.

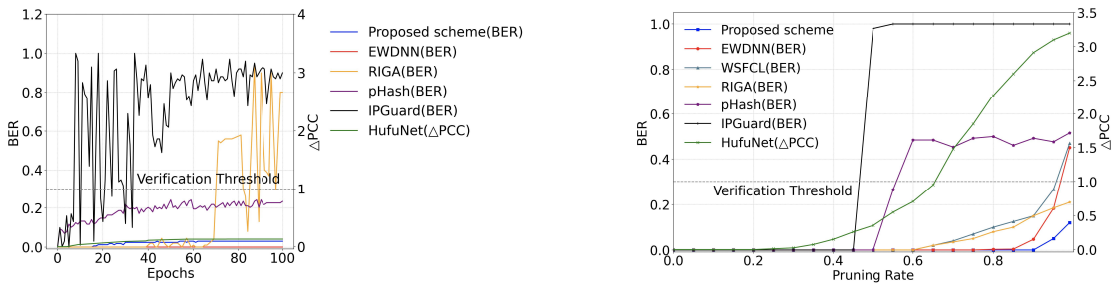
The experimental parameters are configured following the settings by Adi et al. [80]. Specifically, we maintain a learning rate of 0.0002 for non-output layers (consistent with the final training epoch) and 0.01 for the output layer (matching the initial training epoch). The fine-tuning processes are executed for 100 epochs.

The experimental results of NNFMAC against fine-tuning and transfer learning, presented in Table 4.4, demonstrate that the BER remains consistently low, ranging from 0 to 0.109 for up to 90 epochs. Even after 100 epochs of fine-tuning or transfer learning, the ownership information remains reliably extractable with BER well below the verification threshold (0.25). These quantitative results provide strong evidence for NNFMAC's resilience against fine-tuning and transfer learning attacks.

Fig. 4.5(a) compares the robustness of different model IP protection schemes against fine-tuning attacks. Most methods maintain a BER of 0 during the first 10 epochs, with two exceptions. pHash [57] exhibits an immediate increase in BER from the start of fine-tuning, reaching a stable value of approximately 0.23 after 80 epochs, though remaining below its verification threshold of 0.32. IPGuard [11] demonstrates significant instability, with its BER showing notable fluctuations that exceed the verification threshold at epochs 20, 60, and near 100. RIGA [45] maintains relative stability between epochs 40 and 60, but experiences substantial fluctuations after epoch 90, reaching a peak BER of 0.9. In contrast, EWDNN [10] demonstrates exceptional stability, maintaining a BER near 0 throughout the fine-tuning process. The proposed scheme shows strong resilience, with only a minor increase in BER to 0.03, where it stabilizes. HufuNet [47]'s ΔPCC measurements closely parallel the performance of the proposed scheme.

Among all methods, EWDNN exhibits the strongest resistance to fine-tuning attacks, while RIGA and IPGuard show significant vulnerabilities. NNFMAC and HufuNet demonstrate comparable robustness, as indicated by their similar performance curves. However, a notable distinction exists in their response timing: HufuNet's ΔPCC begins to increase around epoch 5, while NNFMAC's BER remains stable until approximately epoch 15.

Robustness Against Pruning Attacks



(a) fine-tuning attack

(b) pruning attack

Figure 4.5: The comparison of robustness against fine-tuning and pruning attacks, where x-axis is the fine-tuning epoch or pruning rate. The y-axes are aligned using the thresholds of NNFMAC ($BER = 0.25$) and HufuNet ($\Delta PCC = 1$).

To assess NNFMAC's resilience against model pruning attacks, we train CNN and ResNet models on MNIST and CIFAR10 datasets, and VIT on the TinyImageNet dataset, creating five host models. The ownership information is encoded before pruning and subsequently extracted

Table 4.5: The BER of ownership information extracted from various pruned models remains consistently low. Regardless of the dataset and network used to train the host model, when the pruning rate is lower than 0.95, the BER is 0. This indicates that NNFMAC effectively withstands model pruning attacks.

Pruning Rate	CIFAR10 (CNN)	CIFAR10 (ResNet)	MNIST (CNN)	MNIST (ResNet)	TinyImageNet (VIT)
0.1	0	0	0	0	0
0.5	0	0	0	0	0
0.95	0	0	0	0	0
0.98	0.078	0.01	0.344	0.01	0.086
0.99	0.352	0.12	0.476	0.476	0.478

from the pruned models. Table 4.5 illustrates the BER for these models across pruning rates from 0.1 to 0.99. The BER remains at 0 for pruning rates below 0.95, only increasing to 0.476 at the extreme pruning rate of 0.99.

Fig. 4.5(b) presents a comparative analysis of robustness against pruning attacks across different schemes. NNFMAC maintains superior performance under pruning attacks, demonstrating lower BER values compared to other methods at equivalent attack intensities. The scheme maintains perfect extraction ($BER = 0$) until the pruning rate reaches 0.95, while other methods begin to show a BER raise at lower thresholds: HufuNet at 0.3, EWDNN at 0.85, RIGA and WSFCL at 0.6, pHash at 0.5, and IPGuard at 45. At an extreme pruning rate of 0.99, NNFMAC maintains a relatively low BER of 0.12, while other methods show more significant degradation: RIGA [45] reaches a BER of 0.21, EWDNN [10] and WSFCL [46] approach 0.5 (BER of 0.48 and 0.49 respectively), and HufuNet [47] exhibits a ΔPCC of 3.2. The pHash method shows particular sensitivity, with its BER rapidly increasing to and stabilizing at approximately 0.5 once the pruning rate exceeds 0.5. IPGuard shows a sharp rise in BER, reaching about 0.99 after the pruning rate exceeds 0.45. These results demonstrate NNFMAC's enhanced resilience to pruning attacks across the full range of pruning intensities.

Robustness Against Weight Shifting Attacks

Weight shifting attack is an effective attack method in white box watermarking [68]. To evaluate the weight shifting attack, we formalize the attack process as $\mathbb{A}(w; \lambda_1, \lambda_2)_i$, where w denotes the convolutional layer set, and hyperparameters λ_1 and λ_2 control the attack strength, as defined by:

$$\mathbb{A}(w; \lambda_1, \lambda_2)_i = w_i - \frac{\lambda_1}{n} \sum_{j=1}^{n} w_j - \lambda_2 A, \tag{4.8}$$

where n is the total number of convolutional layers, and A represents a random matrix matching the dimensions of convolutional layer w_i. The matrix A is generated with zero mean and a variance matching that of all convolutional layers. For experimental consistency, we set $\lambda_1 = 1.5$ and $\lambda_2 = 1.0$. Following the initial attack, we conduct 10 iterations of fine-tuning to comprehensively assess the attack's effectiveness on model robustness.

NNFMAC exhibits resistance against such weight shifting attacks, while EWDNN [10], DeepMarks [48], IPGuard [11] and pHash [57] are susceptible to this attack. The comparison result

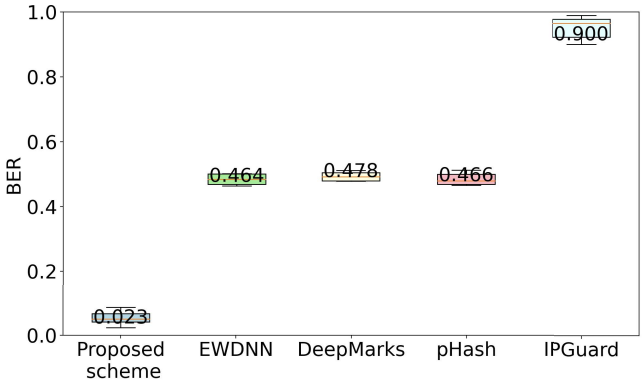


Figure 4.6: Comparison of the ability to resist weight shifting attacks. The BER of different schemes under the weight shifting attack is tested independently five times.

Table 4.6: Comparison of HufuNet, IPGuard, pHash, and NNFMAC against functionality-equivalent attacks (the smaller the value, the stronger the robustness).

	Attack Methods	HufuNet(ΔPCC)	IPGuard(BER)	pHash(BER)	**NNFMAC** (BER)
VGG	Parameter Adjustment	7.755	0.051	0.515	0.054
	Channel Reordering	4.491	0.992	0.531	0.476
	Channel Expansion	7.210	0.680	0.484	0.112
ResNet	Parameter Adjustment	7.000	0.063	0.523	0.075
	Channel Reordering	4.934	0.987	0.521	0.487
	Channel Expansion	-	0.700	0.501	0.123

is shown in Fig. 4.6, where five independent attacks are conducted on five tested models with the same shift setting. In five identical weight shifting attack scenarios, EWDNN, DeepMarks, pHash, and IPGuard exhibit an average BER of 0.464, 0.478, 0.466, and 0.900, respectively, which means that the extracted ownership information is reduced to almost random. However, the BER of NNFMAC remains an average of 0.023. This highlights the resilience of NNFMAC against weight shifting attacks compared to EWDNN, DeepMarks, pHash, and IPGuard.

Robustness Against Functionality-equivalent Attack

Functionality-equivalent attacks, introduced by Lv et al. [47], covers three main attack methods: Parameter Adjustment which scales weights of neurons in adjacent layers by reciprocal constants to maintain functional equivalence, Channel Reordering which modifies the sequence of channels in consecutive convolutional layers while preserving functionality, and Channel Expansion which increases output dimensions of one layer with corresponding adjustments to subsequent layers. To evaluate robustness against these attacks, we implement test configurations matching those in HufuNet [47], using either available open-source implementations or developing code based on their published algorithms. The evaluation focuses on measuring error rates in ownership information extraction across different methods, including HufuNet [47], IPGuard [11], and pHash [57], without applying restoration processes.

The experimental results presented in Table 4.6 demonstrate varying levels of robustness across different methods. The proposed NNFMAC scheme maintains low BER values under Parameter Adjustment (0.054) and Channel Expansion (0.112) attacks. In comparison, alternative

schemes show higher vulnerability: HufuNet exhibited ΔPCC values between 4.491 and 7.775, while IPGuard and pHash show BER values ranging from 0.484 to 0.992. These results suggest that existing schemes face significant challenges in maintaining robustness against functionality-equivalent attacks.

Takeaway–strong robustness against attacks. NNFMAC shows strong robustness against weight perturbation, fine-tuning, pruning, weight shifting and functionality-equivalent attacks, maintaining significantly lower BER than others and demonstrating superior resilience in protecting ownership information under adversarial conditions.

4.5 Conclusion

This chapter presents NNFMAC, a fingerprinting-based model authentication scheme that simultaneously verifies model uniqueness and ownership without compromising performance. By extracting key weights, applying median-based binarization to generate a unique binary fingerprint, and encoding ownership information through an index-based expansion method, NNFMAC provides comprehensive and non-intrusive IP protection. Experimental results confirm that NNFMAC preserves model accuracy without additional training overhead, in contrast to watermarking schemes that incur 0.36%–1.53% accuracy loss. It achieves BER of 0.12 under weight perturbation, 0.03 under fine-tuning, 0.08 under pruning, and 0.09 under weight shifting, which are far lower than the 0.51, 0.49, 0.46, and 0.22 reported in prior work, demonstrating superior robustness, efficiency, and security across diverse attack scenarios.

Chapter 5

Privacy and Ownership Protection in Cooperative Learning

5.1 Introduction of Client-Cooperative Split Learning

Split Learning (SL) enables resource-constrained data owners to train deep models without exposing raw data: clients compute intermediate activations and a server completes training [15], [29], [39], [69]. This paradigm assumes a single, well-provisioned server that is fully trusted for computation and provenance [114]. In practice, deployments differ: personal and edge devices offer surplus yet fragmented compute, which is insufficient individually for large models but becomes valuable when aggregated [115]. This motivates a **serverless, multi-client collaboration** [116], [117]: one client provides data but limited compute; several other clients contribute compute and jointly play the "server" role. This emerging setting becomes both feasible and desirable, better meeting cost by amortizing idle edge/GPU cycles [118], [119], avoiding centralized rental, and improving availability by enabling elastic, on-demand participation [120].

However, this multi-client collaboration setting is only partially trusted [39]. This shift raises coupled challenges that are insufficiently addressed by current SL frameworks:

- **RQ5** (data privacy): *How can the data-providing client preserve the privacy of raw inputs and labels while still enabling effective SL?*
- **RQ6** (layer ownership): *How can trainer clients present cryptographically anchored ownership claims for their contributions to access compensation?*
- **RQ7** (unauthorized-use defense): *How can the collaboratively trained model resist unauthorized use?*

Building on these RQs, we present CliCooper, a multi-client cooperative SL framework, tailored to the serverless, partially trusted, multi-client setting. CliCooper integrates secret mapping labels with Differential Privacy (DP)-guarded activations and a provenance-preserving watermark chain.

For the data-providing client, raw data does not need to be disclosed; instead, each true class is mapped to a set of pseudo-labels, which hides task semantics and label quantity while expanding the effective label space. In parallel, data are augmented to match the expanded label space, and inputs are passed through an embedder whose intermediate activations are perturbed with

calibrated DP noise before any upload. Training thus proceeds only on DP-protected activations paired with pseudo-labels, which (i) suppresses raw-input and label inference to address **RQ5**, and (ii) binds the model's practical usefulness to authorization, since only parties holding the secret true-to-pseudo mapping can faithfully interpret predictions, thereby constraining unauthorized reuse for **RQ7**.

For the trainer clients, after coordinating the execution order and model structure, each receives only the upstream activation, keeps its local parameters opaque, and forwards its output activation to the next participant. At agreed checkpoints, a chained watermark is derived from the received activation digest and then embedded into the client's segment. The resulting watermarks are cryptographically linked across segments, creating a tamper-evident lineage that ties each segment to the collaborative run and the intended task. This lineage enables auditable ownership assertion, lawful entitlement to compensation, and accountability against segment theft or task-mismatch reuse, fulfilling **RQ6** while preserving the confidentiality constraints of **RQ7**.

The key contributions can be summarized as follows:

- We address privacy risks in partially trusted, client-cooperative SL by concealing label semantics and reducing exposure of raw inputs (via label expansion and calibrated noise on intermediate activation), preventing task leakage and curtailing inversion while keeping the true–pseudo mapping under the data owner's control.
- We ensure training traceability and model ownership in multi-client settings by binding checkpoints with cryptographically chained watermarks, enabling verifiable training integrity, robust provenance, and fair compensation, and deterring fabricated training trajectories.
- We validate at scale across diverse datasets and architectures: baseline accuracy is preserved and improves by up to 2%; internal clustering attacks on client-supplied activation drop to 0%; inversion-reconstruction similarity falls from 0.50 to 0.03; and in black-box settings, model-extraction attacks fail, with surrogate models trained on API-labeled data achieving $\sim$1% accuracy (random guessing).

5.2 Technical Warm-ups

5.2.1 Split Learning Architectures

SL allows multiple entities to collaboratively train a model while keeping local data private. We show two approaches:

- In *vanilla SL* [13], [28], [69], [114], the client encodes local data and sends the resulting intermediate activation to the server, which performs most of the computation and, during gradient descent, requires access to label information [39].
- *U-shaped SL* [70], [71], [72] is tailored to unmet label-privacy requirements in vanilla SL settings. A client will not share labels with the server. By preprocessing activations after the split on the client side, the client preserves label confidentiality while maintaining the integrity.

Both assume a server with sufficient compute to train the model independently. In Vanilla SL, clients share labels for gradient computation, which exposes sensitive information. In contrast, U-shaped SL moves gradient computation to the client side, thereby preserving label privacy. However, this design increases communication overhead during the forward and backward passes

and adds extra local computation. Consequently, both variants are unsuitable for collaborative learning where clients have limited computational or network capacity.

5.2.2 Watermarking

Watermarking via modifying model parameters (weights W) was first proposed (EWDNN) by Uchida et al. [10]. A binary watermark $\Lambda \in \{0,1\}^B$ is embedded during training by augmenting the objective:

$$loss(W) = l_w(W) + \lambda l_\Lambda(W), \tag{5.1}$$

where l_w denotes the main task loss, λ is a adjustable trade-off coefficient, and l_Λ is the watermark-embedding regularizer:

$$l_\Lambda(W) = -\sum_{i=1}^{B}(\Lambda_i log(\hat{\Lambda}_i) + (1-\Lambda_i)log(1-\hat{\Lambda}_i')), \tag{5.2}$$

where B denotes the watermark length and $\hat{\Lambda}$ is the extracted watermark. During embedding, it simultaneously embeds and extracts to assess the similarity between the ground-truth watermark and the extracted watermark. The extracted watermark is computed as $\hat{\Lambda}' = \Phi(\sum kW)$, where k is the secret key used for watermark embedding, and $\Phi(z)$ is the step function defined by $\Phi(z) = \frac{1}{1-exp(-z)}$.

In general, weight-modified-based watermarking relies on the embedding regularizer. Wang et al. [38] showed that such modifications distort the weight distribution, allowing adversaries to detect the presence of a watermark. RIGA [45] subsequently redesigned the loss function to mitigate this vulnerability:

$$loss(W) = l_w(W) + \lambda l_\Lambda(W) + \beta l_\Delta(W), \tag{5.3}$$

where β is a tunable coefficient and l_Δ measures the discrepancy between the clean and watermarked model weights, constraining their divergence to deter detection of the watermark via weight-distribution tests. FedIPR [31] applied model watermarking to federated learning, using a watermarking method largely aligned with EWDNN and RIGA.

5.3 System Model of Client-Cooperative SL

We target a serverless, partially trusted, multi-client cooperative SL model with heterogeneous participants: *one data client owns the private dataset but has limited compute, while several trainer clients contribute surplus yet fragmentary compute.* The group jointly trains a split model by agreeing on split points and an execution order. Each trainer handles a local segment sized to its capacity. Parties are honest-but-curious and there may be divergent interests, so the design must (i) preserve raw input/label privacy for the data owner, (ii) support auditable ownership and lawful entitlement for trainers, and (iii) constrain unauthorized model reuse.

To this end, we integrate a privacy-and-authorization layer that couples secret-mapping label expansion with DP protection. The data client never discloses its true labels; instead, each true class is mapped to a set of pseudo-labels, obfuscating task semantics and label quantities. Data augmentation [121], [122] is introduced to align sample counts with the expanded label space. Inputs are then passed through an embedder, and the resulting intermediate activations are perturbed with DP noise before any upload, mitigating feature inversion and label inference

attacks [52], [74]. Training proceeds only on these DP-protected activations paired with pseudo-labels. For authorized users who hold the secret true↔pseudo mapping, model utility remains intact, while for unauthorized parties, model outputs degrade to an unusable level.

Furthermore, to establish segment-level provenance and enforce lawful ownership without exposing internals, CliCooper weaves a chained watermark through the collaboration process. At agreed checkpoints, each trainer derives a watermark token from a digest of the received activation and then embeds it into its segment. Tokens are cryptographically linked across segments to form a tamper-evident lineage that ties each segment to the collaborative run and intended task, supporting auditable ownership assertions, compensation eligibility, and accountability against segment theft or task-mismatch reuse.

Here, each trainer exposes only the upstream activation and keeps its local segment opaque to collaborators. Consequently, external adversaries interact with the final collaboratively trained model purely as a black box [80]. Under this visibility constraint, our chained watermarking design can preserve the confidentiality guarantees required in a partially trusted setting and is resilient to common attacks, including removal [123], overwriting [124], fine-tuning [92], and pruning [93].

Table 5.1: List of Notation for CliCooper.

Notation	Definition
$\mathcal{C}/\mathcal{T}/\mathcal{V}$	Data client/Trainer client/Verifier
$\mathbb{D}^*/\ \mathbb{D}/\ \mathbb{D}_t$	Final/Original/Test Dataset
Y/Y^*	True/Pseudo label of Dataset
q	The class number of the dataset
g_i	The expansion factor of the labels Y_i^*
γ	The average expansion factor of Y^* relative to Y
$\mathcal{G}_Y/\mathcal{G}_Y^{-1}$	The one-to-many/many-to-one mapping between true and pseudo label
$\varepsilon/\mathbb{G}(\cdot)$	DP privacy budget/DP noise injection
$\mathbb{M}_{\mathcal{T}_i}$	$\mathcal{T}_i$'s activation for $\mathcal{T}_{i+1}$
$\mathbb{M}_{\mathcal{C}}^{\mathrm{DP}}$	DP-protected activation
A	the activation of the $\mathcal{C}$
W	The whole SL model/weights
$W_{\mathcal{T}_i}$	The ith trainer's model/weights
$W_{\mathcal{C}}$	$\mathcal{C}$'s pre-trained encoder/model
Λ	Copyright information embedded in the model
$\Lambda_{\mathcal{T}_i}$	$\mathcal{T}_i$'s watermark
B	The watermark size
$k_{\mathcal{T}_i}$	Embedding key for $\Lambda_{\mathcal{T}_i}$
$Z_{\mathcal{T}_i}$	Selection matrix for weights in $W_{\mathcal{T}_i}$, used for $\Lambda_{\mathcal{T}_i}$
μ_i	Unique secret nonce used in hash computation
$ID_{\mathcal{T}_i}$	Unique $\mathcal{T}_i$ node identity
η	Watermark detection rate
$\mathbb{O}(\cdot)$	Watermark extraction function
$l_w(\cdot)/l_\Lambda(\cdot)$	The loss function for main task and embedding watermark

5.3.1 Entities

CliCooper involves three primary entities under a serverless, partially trusted, multi-client regime:

- *Data client* $\mathcal{C}$ owns the private dataset with ground-truth labels, but has only lightweight computing adequate for embedding/perturbation rather than end-to-end training. Its primary requirement is to prevent raw input and label inference, hide task semantics/label quantities from others.
- *Trainer clients* $\mathcal{T}$ contribute surplus yet fragmentary compute without proprietary data, each controlling a private model segment sized to its capacity. As rational, honest-but-curious participants, they do not sabotage the task, but may prefer minimal effort for "free-lunch" rewards and may be inquisitive about peers' resources, especially attempting to infer information about $\mathcal{C}$'s data distribution, labels, or sample membership from observed activations.
- *Verifier* $\mathcal{V}$ is fully trusted and responsible for coordinating and incentive management. It can be instantiated by the data client $\mathcal{D}$, a trainer client $\mathcal{T}$, or a neutral third party, endowed with an omniscient audit view that includes the end-to-end activation relay and all segment parameters. It has the authority to validate chained watermarks and allocate incentives or penalties, thereby enforcing fair compensation and protocol compliance without weakening the confidentiality guarantees among collaborating clients.

5.3.2 Threat Models.

We distinguish internal users and external adversaries interacting only through exposed APIs. All clients are **rational and honest-but-curious** [39]: they share the common interest of completing the main task and thus do not sabotage training, yet they may try to minimize their own effort and may be curious about others' resources.

- **Internal users**. Internal users primarily refer to the trainer clients $\mathcal{T}_i$, as the data client $\mathcal{C}$ has minimal visibility without access to others' parameters, and the verifier $\mathcal{V}$ is fully trusted. Each $\mathcal{T}_i$ has white-box access to its own segment but sees only *upstream activations* from peers. Model parameters of other segments and $\mathcal{C}$'s raw data and true labels remain hidden. The dominant risks are curiosity-driven inference and credit/ownership disputes rather than protocol sabotage. Typical inference attempts include:
  - *Clustering attack.* Even without access to $\mathcal{C}$'s raw dataset, $\mathcal{T}$ may apply clustering [125], [126] to intermediate activation to infer which samples belong to the same class.
  - *Inversion attack.* Assuming $\mathcal{T}$ has knowledge of $\mathcal{C}$'s model architecture, it may still leverage exposed intermediate information to reconstruct the original training samples or generate visually similar approximations [52], [74].

  We consider a marketplace-style collaboration, where trainer clients are rational and honest-but-curious: they share common interest in completing the main task and thus do not sabotage training but may attempt passive inference from observed activations or seek "free-lunch" rewards with minimal effort [127]. Unless otherwise stated, trainer clients are assumed non-colluding; even under collusion, trainers still only observe DP-protected activations and operate in the pseudo-label space, while the data client's raw inputs, true labels, and the secret true↔pseudo mapping remain undisclosed. In particular, reusing a pre-trained/open-source segment to bypass training is infeasible because the task semantics are hidden by secret-mapping label expansion and DP perturbation, preventing trainers from identifying a matching pre-trained model in advance.

  We assume that the verifier $\mathcal{V}$ is fully trusted and responsible for settlement: it holds an audit view (end-to-end activation relay and all segment parameters) to verify model utility and chained watermarks and then allocate incentives or penalties. Model runs failing the accuracy requirement are rejected and deemed ineligible for compensation. Notably, $\mathcal{V}$ needs no access to the data client's raw inputs, true labels, or the secret label mapping; revealing

the mapping to any untrusted third party is out of scope and should be avoided.

- **External adversaries**. In contrast, we assume that external adversaries only have black-box access to the released SL model. They cannot obtain knowledge of the SL architecture or model parameters; their sole capability is interacting with the model through its exposed APIs. Such adversaries may perform model extraction attacks [128], [129]. Specifically, by querying the API with unlabeled inputs that are similar to the training data, they can obtain the corresponding labels. The attacker exploits the target SL model as a labeling oracle and subsequently uses the labeled data to train a surrogate model with functionality similar to the original model.

5.4 Design of CliCooper

CliCooper has five components (Fig.5.1): an SL protocol, secret-mapping label expansion, DP-guarded activations, cooperative training with embedded watermark, and verification.

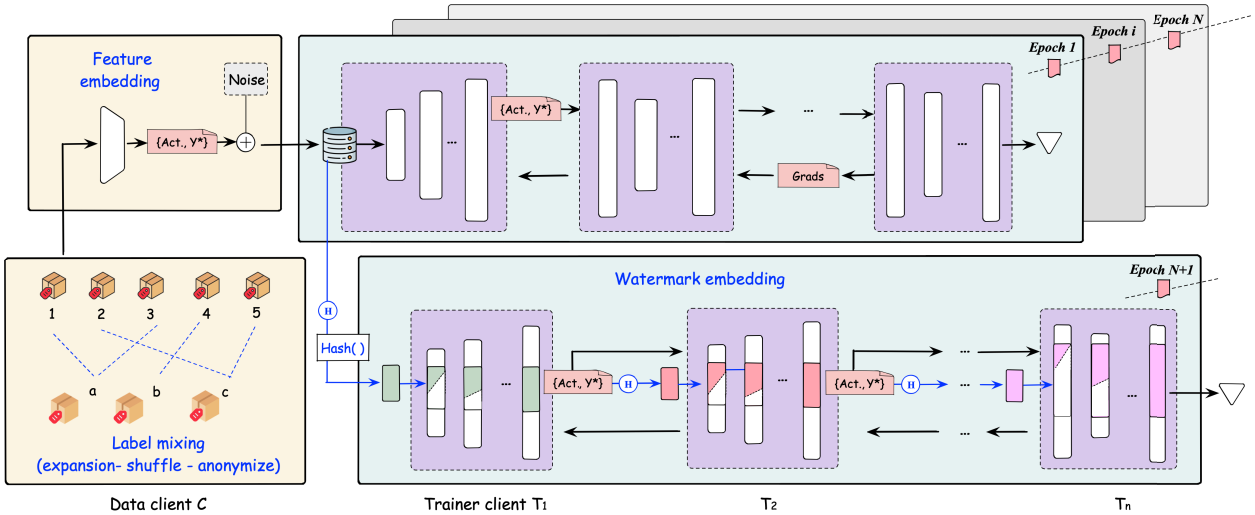


Figure 5.1: **CliCooper design:** The data client $\mathcal{C}$ holds the dataset but has very limited compute and thus collaborates with multiple trainer clients $\mathcal{T}$ for split learning. First, to disclose the ground-truth labels Y, $\mathcal{C}$ expands Y to pseudo-labels Y^* whose quantity and semantics differ from Y. Next, $\mathcal{C}$ uses a lightweight encoder to embed features and applies DP to protect activations. $\mathcal{T}$ then train the model using DP-protected activations $\mathbb{M}_{\mathcal{C}}^{\mathrm{DP}}(W_{\mathcal{C}}, \mathbb{D}^*)$ and Y^* for N epochs until convergence. In the N+1th epoch, each trainer $\mathcal{T}$ embeds a chained watermark into its assigned subnetwork in a pipelined manner, asserting ownership claim for their contributions to access compensation.

5.4.1 SL Protocol

We target a marketplace-style collaboration where heterogeneous clients provide training services in exchange for rewards. Clients with higher capacity host larger network partitions, while fragmentary devices host smaller segments. Unlike conventional SL that delegates downstream training to a single powerful server, CliCooper distributes the "server side" over multiple trainer clients, scaling the number and size of segments with model complexity and available compute.

Algorithm 5 Label Expansion & Privacy Protection

Require: $\mathbb{D}$ (original dataset), Y (true label), $\mathcal{G}_Y$ (predefined one-to-many map)), ε (privacy budget of DP noise), $\{g_i\}_{i=1}^{q}$ (expansion factor for each true label).
Ensure: $\mathbb{M}_{\mathcal{C}}^{\mathrm{DP}}(W_{\mathcal{C}}, \mathbb{D}^*)$ (DP-protected activations), Y^* (pseudo-labels).

1: **for** $Y_i \in Y$ **do** ▷ label expansion loop
2: $\quad Y_i \mapsto \{Y_{i_1}^*, \ldots, Y_{i_{g_i}}^*\}$, ▷ true labels expand to pseudo-labels
3: $\quad \{(\mathbb{D}_i, Y_i)\} \mapsto \{(\mathbb{D}_{i_1}^*, Y_{i_1}^*) \ldots (\mathbb{D}_{i_{g_i}}^*, Y_{i_{g_i}}^*)\}$ ▷ data-and-label pairing.
4: **end for**
5: $Y \xrightarrow{\mathcal{G}_Y} Y^*,\ Y^* = \{Y_1^*, \ldots, Y_{\sum_{i=1}^{q} g_i}^*\}$. ▷ shuffle randomly.
6: $\{\mathbb{D}, Y\} \mapsto \{\mathbb{D}^*, Y^*\}$. ▷ transform accordingly.
7: $\tilde{A} = \mathbb{M}_{\mathcal{C}}(W_{\mathcal{C}}, \mathbb{D}^*)$. ▷ generate client-side activations.
8: $\tilde{A} \xrightarrow{\mathsf{Clip}_S} \bar{A}$. ▷ enforce an ℓ_1 sensitivity bound.
9: $\bar{A} \xrightarrow{\mathbb{G}(\varepsilon)} \mathbb{M}_{\mathcal{C}}^{\mathrm{DP}}(W_{\mathcal{C}}, \mathbb{D}^*) = \bar{A} + \mathrm{Laplace}\big(0,\ \frac{\Delta_1}{\varepsilon}\big)$. ▷ Laplace-based DP mechanism injection.
10: **RETURN** $\{\mathbb{M}_{\mathcal{C}}^{\mathrm{DP}}(W_{\mathcal{C}}, \mathbb{D}^*),\ Y^*\}$

Pre-training negotiation. Before any data flow, the task publisher (usually $\mathcal{C}$ or $\mathcal{V}$) conducts a light-weight negotiation phase with participating trainers to (i) agree on split points and a strict execution order $\mathcal{T}_1 \to \ldots \to \mathcal{T}_n$, and (ii) size each segment to match the host's compute/memory budget and the end-to-end latency target. The agreed plan fixes interface tensor shapes and optimizer/hyper-parameter responsibilities for each segment.

Training flow. Let $\mathbb{D}$ denote the task dataset held exclusively by $\mathcal{C}$, and $W = \{W_{\mathcal{C}}, W_{\mathcal{T}_1}, \ldots, W_{\mathcal{T}_n}\}$ denote the parameters contributed by $\mathcal{C}$ and $\mathcal{T}_1, \ldots, \mathcal{T}_n$. The end-to-end system workflow respects the relay structure as follows [52]:

$$\mathbb{M}(W, \mathbb{D}) = \mathbb{M}_{\mathcal{T}_n}\big(W_{\mathcal{T}_n},\ \mathbb{M}_{\mathcal{T}_{n-1}}(\ldots \mathbb{M}_{\mathcal{T}_1}(W_{\mathcal{T}_1},\ \mathbb{M}_{\mathcal{C}}^{\mathrm{DP}}(W_{\mathcal{C}}, \mathbb{D}^*)))\big). \tag{5.4}$$

Here, only activations $\mathbb{M}(\cdot)$ are relayed downstream, while model parameters remain local and opaque to peers. Backpropagation proceeds locally at each trainer over its segment using the received upstream gradients. No true labels or raw inputs are disclosed to trainers, as $\mathbb{M}_{\mathcal{C}}^{\mathrm{DP}}(\cdot)$ is DP-protected activations and $\mathbb{D}^*$ indicates $\mathbb{D}$'variant after an expansion process (§5.4.2).

To enable market settlement and dispute resolution, trainers have agreed to embed a chained watermark lineage during training (§5.4.3), thus segments lacking verifiable marks are deemed ineligible.

5.4.2 Secret-mapping Expansion and DP-based Protection

CliCooper answers **RQ5** (data privacy) and **RQ7** (unauthorized-use defense) by coupling secret-mapping label expansion with DP-guarded activations. The data client $\mathcal{C}$ never discloses true labels; instead, it obfuscates task semantics and label cardinality while protecting raw inputs against inversion attack (cf. **Algorithm** 5).

Secret-mapping label expansion

To hide both label semantics and quantity while preserving trainability, we employ a **secret-mapping** expansion controlled by the data client $\mathcal{C}$. Let $Y = \{Y_1, \ldots, Y_q\}$ be true labels. Data client $\mathcal{C}$ samples a secret one-to-many map $\mathcal{G}_Y$ and expands each true class Y_i to g_i ($g_i \geq 1$) pseudo-classes, $Y_i \mapsto \{Y_{i_1}^*, \ldots, Y_{i_{g_i}}^*\}$, yielding Y^* with $\sum_{i=1}^{q} g_i$ categories (Lines 1–3):

$$Y \xrightarrow{\mathcal{G}_Y} Y^*,\ Y^* = \{Y_1^*, \ldots, Y_{\sum_{i=1}^{q} g_i}^*\}, \tag{5.5}$$

where Y^* is the union of all per-class pseudo-labels, i.e., $Y^* = \bigcup_{i=1}^{q} \{Y^*_{i_1}, \ldots, Y^*_{i_{g_i}}\}$, and the ordering is randomly shuffled but recorded.

Data-and-label pairs are remapped accordingly (Line 4):

$$\{(\mathbb{D}_i, Y_i)\} \;\mapsto\; \{(\mathbb{D}^*_{i_1}, Y^*_{i_1}) \ldots (\mathbb{D}^*_{i_{g_i}}, Y^*_{i_{g_i}})\}, \tag{5.6}$$

where $\{\mathbb{D}^*_{i_1}, \ldots, \mathbb{D}^*_{i_{g_i}}\}$ are obtained by partitioning and noise-aware augmentation of $\mathbb{D}_i$ [121], [122], so that class priors and balance remain stable under expansion. Collecting the above, the dataset–label pair transforms as

$$\{(\mathbb{D}, Y)\} \;\mapsto\; \{(\mathbb{D}^*, Y^*)\}, \tag{5.7}$$

where $\mathbb{D}^*$ and Y^* denote the expanded, anonymized, and permuted versions of the original dataset and labels, with $\mathbb{D}^*$ transformed accordingly by $\mathcal{G}_Y$. For notational convenience, we define the average expansion factor $\gamma = \frac{\sum_{i=1}^{q} g_i}{q} > 1$.

This expansion does not harm utility for authorized users because training is performed consistently on the expanded label space, so the loss is invariant to a fixed mapping [85]. Authorized parties who legally possess $\mathcal{G}_Y$ perform **demasking** at the inference with the inverse mapping $\mathcal{G}_Y^{-1}$:

$$\{Y^*_1, \ldots, Y^*_{\sum g_i}\} \xrightarrow{\mathcal{G}_Y^{-1}} \{Y_1, \ldots, Y_q\}. \tag{5.8}$$

This binds model utility to authorization and obfuscates both semantics and quantities. Without $\mathcal{G}_Y$ or $\mathcal{G}_Y^{-1}$, outputs become operationally degraded for unauthorized users, impeding functional repurposing and correct re-labeling of downstream data.

DP-protected activations

To curb activation inversion and sensitive-attribute inference, $\mathcal{C}$ perturbs smashed activations with calibrated DP noise before any upload. Let $\tilde{A} = \mathbb{M}_{\mathcal{C}}(W_{\mathcal{C}}, \mathbb{D}^*)$ be the client-side activations after forward pass with the expanded dataset $\mathbb{D}^*$. We enforce an ℓ_1 sensitivity bound via *per-layer ℓ_1-clipping* to radius S, which yields $\Delta_1 \leq 2S$ for neighboring inputs. Then we add zero-mean Laplace noise calibrated to Δ_1 and the target privacy budget ε (Lines 5–7):

$$\tilde{A} \xrightarrow{\mathsf{Clip}_S} \bar{A}, \text{ and } \bar{A} \xrightarrow{\mathbb{G}(\varepsilon)} \mathbb{M}^{\mathrm{DP}}_{\mathcal{C}}(W_{\mathcal{C}}, \mathbb{D}^*) = \bar{A} + \mathrm{Laplace}\big(0,\, \tfrac{\Delta_1}{\varepsilon}\big)\,, \tag{5.9}$$

where $\mathbb{G}(\varepsilon)$ denotes a randomized Laplace mechanism [130] with privacy budget ε. Combined with the secret-mapping label expansion, this Laplace-based DP mechanism injects calibrated randomness that yields a provable ε-DP guarantee at the activation interface.

Theorem 1 (DP-protected activations in CliCooper)**.** *Let $\bar{A}(x) = \mathit{Clip}^{(1)}_S\big(\mathbb{M}_{\mathcal{C}}(W_{\mathcal{C}}, x)\big) \in \mathbb{R}^d$ be the per-sample activation after per-layer ℓ_1-clipping with radius S, so that the ℓ_1-sensitivity satisfies $\Delta_1 = \sup_{x \sim x'} \|\bar{A}(x) - \bar{A}(x')\|_1 \leq 2S$. Consider the Laplace mechanism $\mathcal{M}_\varepsilon(x) = \bar{A}(x) + Z, Z \overset{i.i.d.}{\sim} \mathrm{Laplace}\big(0, \frac{\Delta_1}{\varepsilon}\big)$, which guarantees pure ε-DP. For any fixed DP activation $y_a = \mathcal{M}_\varepsilon(x_a)$, any two DP activations y_b, y_c, and any candidate class t,*

$$\frac{\Pr\left[t \;\mathit{gen}(y_a \wedge y_b)\right]}{\Pr\left[t \;\mathit{gen}(y_a \wedge y_c)\right]} \;\leq\; e^{2\varepsilon}, \tag{5.10}$$

where t generated $(y_a \wedge y_b)$ denotes the event that both DP activations y_a and y_b were produced from inputs whose (hidden) true class is t.

Proof 1. The per-coordinate Laplace noise with scale Δ_1/ε ensures that for any input x and any observation y, the likelihood ratio is bounded

$$e^{-\varepsilon} \leq \frac{\Pr[y \mid x]}{\Pr[y \mid x']} \leq e^{\varepsilon}, \tag{5.11}$$

for all neighboring $x \sim x'$, according to the principle of the standard Laplace ε-DP [130].

Fix a candidate class t and write the joint "same-class" event under a product model:

$$\Pr[t \text{ gen } (y_a \wedge y_b)] = \sum_{x \in t} \Pr[y_a|x] \Pr[y_b|x] \Pr[x], \tag{5.12}$$

and similarly for $(y_a \wedge y_c)$:

$$\Pr[t \text{ gen } (y_a \wedge y_c)] = \sum_{x \in t} \Pr[y_a|x] \Pr[y_c|x] \Pr[x]. \tag{5.13}$$

Then, compared (5.12) and (5.13), we have

$$\begin{aligned} \frac{\Pr[t \text{ gen } (y_a \wedge y_b)]}{\Pr[t \text{ gen } (y_a \wedge y_c)]} &= \frac{\sum_{x \in t} \Pr[y_a|x] \Pr[y_b|x] \Pr[x]}{\sum_{x \in t} \Pr[y_a|x] \Pr[y_c|x] \Pr[x]} \\ &\leq \frac{\sum_{x \in t} \Pr[y_a|x] \left(e^{\varepsilon} \Pr[y_a|x]\right) \Pr[x]}{\sum_{x \in t} \Pr[y_a|x] \left(e^{-\varepsilon} \Pr[y_a|x]\right) \Pr[x]} \\ &= e^{2\varepsilon}, \end{aligned} \tag{5.14}$$

where we applied the ε-DP likelihood bound as shown in (5.11) to $\Pr[y_b|x]/\Pr[y_a|x]$ in the numerator and $\Pr[y_c|x]/\Pr[y_a|x]$ in the denominator, yielding factors e^{ε} and $e^{-\varepsilon}$ respectively. The inequality then follows by cancellation inside the sums. Hence (5.10) holds.

As demonstrated in Theorem 1, under ℓ_1-clipping with Laplace noise, no observer can improve the odds that two DP activations (y_a, y_b) came from the same candidate class t relative to (y_a, y_c) by more than $e^{2\varepsilon}$. This caps pairwise linkage attacks and underpins the secrecy of class membership.

Given $\mathcal{C}$'s limited compute and to avoid cumulative privacy loss from repeated composition, both label-space expansion and DP perturbation are performed once before collaborative training. That is, $\mathcal{C}$ expands labels, computes a single forward pass over $\mathbb{D}^*$ to produce $\tilde{A}$, applies clipping and the Laplace mechanism to obtain $\mathbb{M}_{\mathcal{C}}^{\text{DP}}$, and caches these DP-guarded activations for the further training rounds. In the subsequent pipeline, $\mathcal{C}$ *sends only* $\mathbb{M}_{\mathcal{C}}^{\text{DP}}$ to the first trainer $\mathcal{T}_1$ and the pseudo-labels Y^* to the last trainer $\mathcal{T}_n$, keeping raw inputs and true labels remain undisclosed.

5.4.3 Training with Chained Watermarking

Since trainers $\mathcal{T}$ are only partially trusted, they must demonstrate that they actually performed training and that they own the layers they optimized—rather than submitting a "free-lunch" or open-source pretrained model to claim rewards. CliCooper answers **RQ2** (layer ownership) by enforcing pipeline training with chained watermarks embedded into the $\mathcal{T}$-side submodels (cf. **Algorithm** 6). These watermarks create cryptographic links between consecutive trainers' output activations. In practice, once a trainer $\mathcal{T}_i$ brings its submodel to convergence, it must

Algorithm 6 Training & Chained Watermarking

Require: $\mathbb{M}_{\mathcal{C}}^{\mathrm{DP}}(W_{\mathcal{C}}, \mathbb{D}^*)$ (DP-protected activation), $W_{\mathcal{C}}$ (the pretrained model of $\mathcal{C}$), $ID_{\mathcal{T}_i}$ ($\mathcal{T}_i$ node identity), μ_i (secret nonce), η_G (threshold for successful watermark embedding), l_w (main task loss function), l_Λ (watermark embedding regularizers).

Ensure: $W = \{W_{\mathcal{C}}, W_{\mathcal{T}_1}, \ldots, W_{\mathcal{T}_n}\}$ (SL model with watermark).

1: **while** ($\neg$ converging) **do** ▷ until convergence
2: $W_{\mathcal{C}} \overset{\mathbb{M}_{\mathcal{C}}^{\mathrm{DP}}}{\rightarrow} W_{\mathcal{T}_1} \overset{\mathbb{M}_{\mathcal{T}_1}}{\rightarrow} \ldots \overset{\mathbb{M}_{\mathcal{T}_{n-1}}}{\rightarrow} W_{\mathcal{T}_n}$ ▷ pipeline training.
3: **end while**
4: $\mathbb{M}_{\mathcal{C}}^{\mathrm{DP}}$, μ_i, i ▷ initial activation, and trainer index.
5: **for** $i = 1, 2 \ldots n$ **do** ▷ trainer's watermark embedding loop
6: $Z_{\mathcal{T}_i} = \mathsf{WMPosition}(\mu_i)$ ▷ selection matrix for weights used for $\Lambda_{\mathcal{T}_1}$
7: $\Lambda_{\mathcal{T}_i}, k_{\mathcal{T}_i} = \mathsf{WMGen}(\mathcal{H}_i), \mathsf{KeyGen}(\mathcal{H}_i)$ where
8: **if** $i = 1$ **then**
9: $\mathcal{H}_1 = \mathbb{H}(\mathbb{M}_{\mathcal{C}}^{\mathrm{DP}}, 1, \mu_1, ID_{\mathcal{T}_1})$
10: **else**
11: $\mathcal{H}_i = \mathbb{H}(\mathbb{M}_{\mathcal{T}_{i-1}},\ i,\ \mu_i,\ ID_{\mathcal{T}_i})$
12: **end if** ▷ calculate trainer's watermark
13: **while** $\eta < \eta_G$ **do** ▷ until watermark convergence
14: $W_{\mathcal{T}_i} = \arg\min_W \left(l_w(W) + \lambda l_\Lambda(W)\right)$ ▷ watermark embedding
15: $\eta = 1 - \frac{\sum_b^B (\mathbb{O}(W_{\mathcal{T}_i}, Z_{\mathcal{T}_i}, k_{\mathcal{T}_i})[j] \neq \Lambda_{\mathcal{T}_i}[j])}{B}$, ▷ update watermark detection rate
16: **if** $\eta \geq \eta_G$ **then**
17: Save $W_{\mathcal{T}_i}$
18: **end if**
19: **end while**
20: $i \mathrel{+}= 1$
21: **end for**
22: **RETURN** $W = \{W_{\mathcal{C}}, W_{\mathcal{T}_1}, \ldots, W_{\mathcal{T}_n}\}$

embed a watermark $\Lambda_{\mathcal{T}_i}$ to attest ownership; crucially, $\Lambda_{\mathcal{T}_i}$ is not chosen by $\mathcal{T}_i$ but is deterministically derived from the predecessor's output activation via a collision-resistant hash. By seeding each watermark with $\Lambda_{\mathcal{T}_i} \leftarrow \mathbb{H}(\cdot)$ ($\mathbb{H}(\cdot)$ is a hash function to generate a watermark), the scheme prevents precomputation or substitution attacks, ensuring that a trainer cannot cheat without executing the prescribed protocol.

Pipeline training

Before the cooperative training, the data client $\mathcal{C}$ processes the raw dataset with a pre-trained split encoder and safeguards the resulting activations via a DP mechanism, producing DP-protected activations $\mathbb{M}_{\mathcal{C}}^{\mathrm{DP}}$. Once $\mathcal{C}$ uploads the activation $\mathbb{M}_{\mathcal{C}}^{\mathrm{DP}}$ to $\mathcal{T}_1$ and the pseudo-labels Y^* to $\mathcal{T}_n$, it ceases participation in the subsequent pipeline training. The trainers $\mathcal{T} = \{\mathcal{T}_1, \mathcal{T}_2 ... \mathcal{T}_n\}$ then execute a predefined SL protocol, proceeding sequentially in a pipeline (Lines 1–3).

During each training epoch, $\mathcal{T}_1$ consistently consumes the $\mathcal{C}$-provided activation $\mathbb{M}_{\mathcal{C}}^{\mathrm{DP}}$ as input and forwards its output to $\mathcal{T}_2$. Each intermediate trainer $\mathcal{T}_i$ for $i = 2, 3..n-1$ takes as input the activation produced by $\mathcal{T}_{i-1}$, performs its designated computation, and passes the resulting activation to $\mathcal{T}_{i+1}$. Upon receiving the activation from $\mathcal{T}_{n-1}$, $\mathcal{T}_n$ combines it with the pseudo-labels Y^* supplied by the client $\mathcal{C}$ to compute the loss, perform gradient descent, and backpropagate gradients upstream through $\mathcal{T}_{n-1}, \mathcal{T}_{n-2} ... \mathcal{T}_1$. The trainers repeat this process for N rounds until the SL model converges.

Watermark generation

Watermark embedding follows pipeline training and proceeds first with the generation of the watermark. Watermark generation begins with the task publisher (which may be either $\mathcal{C}$ or the $\mathcal{V}$) assigning each $\mathcal{T}_i$ node a fixed nonce μ_i, which $\mathcal{T}_i$ as the basis for generating the watermark $\Lambda_{\mathcal{T}_i}$, the embedding key $k_{\mathcal{T}_i}$, and the position matrix $Z_{\mathcal{T}_i}$ that specifies where $\Lambda_{\mathcal{T}_i}$ is embedded. $Z_{\mathcal{T}_i}$ is generated solely from the fixed seed μ_i, whereas $\Lambda_{\mathcal{T}_i}$ and are derived from the predecessor's output jointly with $k_{\mathcal{T}_i}$ and the trainer identity $ID_{\mathcal{T}_i}$ (Lines 4–12):

$$\begin{aligned} Z_{\mathcal{T}_i} &= \mathsf{WMPosition}(\mu_i), i = 1, 2...n \\ \Lambda_{\mathcal{T}_i},\ k_{\mathcal{T}_i} &= \mathsf{WMGen}(\mathcal{H}_i),\ \mathsf{KeyGen}(\mathcal{H}_i) \text{ with} \\ \mathcal{H}_i &= \mathbb{H}(\mathbb{M}_{\mathcal{T}_{i-1}},\ i,\ \mu_i,\ ID_{\mathcal{T}_i}), i = 2, 3..n, \\ \text{while } \mathcal{H}_1 &= \mathbb{H}(\mathbb{M}_{\mathcal{C}}^{\mathrm{DP}},\ 1,\ \mu_1,\ ID_{\mathcal{T}_1}), i = 1, \end{aligned} \tag{5.15}$$

where $\mathsf{WMPosition}(\cdot)$, $\mathsf{WMGen}(\cdot)$, and $\mathsf{KeyGen}(\cdot)$ are deterministic functions, with all randomness derived from the publisher-fixed nonce μ_i and $\mathcal{T}_i$ identity $ID_{\mathcal{T}_i}$. This ensures only the legitimate $\mathcal{T}_i$ can generate valid $\Lambda_{\mathcal{T}_i}$ and $k_{\mathcal{T}_i}$, while the $\mathcal{T}$'s index i enforces sequential linkage. This chained derivation of the watermark and embedding keys thwarts precomputation: a trainer cannot anticipate ($\Lambda_{\mathcal{T}_i}$, $k_{\mathcal{T}_i}$) before observing the predecessor's activation, nor embed a valid watermark into an unauthorized model in advance.

Watermark embedding

Since $\mathcal{T}$ does not possess any local data, it is unable to embed backdoor-based watermarks [80]. Instead, watermarking approaches that are integrated into the training process by modifying model features (e.g., EWDNN [10], RIGA [45], and FedIPR [31]) represent the most suitable schemes for watermark embedding in this setting.

After N epoch training, the model has converged. In the $(N+1)$th epoch, the unique watermark $\Lambda_{\mathcal{T}_i}$ is then embedded into the weights $W_{\mathcal{T}_i}$ according to the matrix $Z_{\mathcal{T}_i}$, gradually forming the weights of the final epoch, where Λ_i has been successfully embedded. This embedding process, which spans multiple rounds in the $(N+1)$th epoch, can be formally expressed as follows (Lines 13–14):

$$W_{\mathcal{T}_i} = \arg\min_{W} \left(l_w(W) + \lambda l_\Lambda(W) \right), \tag{5.16}$$

where l_w is the main task loss function, l_Λ is the watermark embedding regularizer, and λ is a hyper-parameter that balances the main task training and watermark embedding.

While embedding the watermark $\Lambda_{\mathcal{T}_i}$, the process is actively monitored in each round of the $(N+1)$th epoch. Simultaneously, the watermark is extracted from $W_{\mathcal{T}_i}$ and compared to the expected watermark. The watermark detection rate η is calculated using the Hamming distance (Line 15):

$$\eta = 1 - \frac{\sum_b^B (\mathbb{O}(W_{\mathcal{T}_i}, Z_{\mathcal{T}_i}, k_{\mathcal{T}_i})[j] \neq \Lambda_{\mathcal{T}_i}[j])}{B}, \tag{5.17}$$

where $\mathbb{O}(\cdot)$ is the watermark extraction function, B is the size of $\Lambda_{\mathcal{T}_i}$. Once η reaches or exceeds the predefined threshold η_G (i.e., $\eta \geq \eta_G$), the embedding process for the current trainer $\mathcal{T}_i$ is considered complete. Then $\mathcal{T}_i$ freezes and save its checkpoint (Lines 16–17) and outputs activation $\mathbb{M}_{\mathcal{T}_i}$ to $\mathcal{T}_{i+1}$, who will follow the same watermark generation and embedding process as for the $\mathcal{T}_i$-th trainer.

Algorithm 7 Watermark Verification

Require: $W = \{W_{\mathcal{C}}, W_{\mathcal{T}_1}, \ldots, W_{\mathcal{T}_n}\}$ (SL model with watermark), $\mathbb{M}_{\mathcal{C}}^{\text{DP}}(W_{\mathcal{C}}, \mathbb{D}^*)$ (DP-protected activation), $ID_{\mathcal{T}_i}$ ($\mathcal{T}_i$ node identity), μ_i (secret nonce), $\mathbb{D}_t$ (test dataset), η_G(threshold of similarity between the hashed and extracted watermarks).

Ensure: "**Fail**" or "**Success**".

1: $\mathcal{V}$ receives $W = \{W_{\mathcal{C}}, W_{\mathcal{T}_1}, \ldots, W_{\mathcal{T}_n}\}$, $ID_{\mathcal{T}_i}$ from $\mathcal{T}$ and $\mathcal{C}$.
2: $\hat{Acc}_{main} = \mathsf{Test}(W, \mathbb{D}_t)$ ▷ test the model main task accuracy
3: **if** $\hat{Acc}_{main}$ not meet expectation **then**
4: **RETURN** "**Fail**", **break**
5: **for** $i = 1, 2 \ldots n$ **do** ▷ verification loop
6: $Z_{\mathcal{T}_i} = \mathsf{WMPosition}(\mu_i)$. ▷ recover the selection matrix for watermark
7: $\Lambda_{\mathcal{T}_i}, k_{\mathcal{T}_i} = \mathsf{WMGen}(\mathcal{H}_i), \mathsf{KeyGen}(\mathcal{H}_i)$ where
8: **if** $i = 1$ **then**
9: $\mathcal{H}_1 = \mathbb{H}(\mathbb{M}_{\mathcal{C}}^{\text{DP}}, 1, \mu_1, ID_{\mathcal{T}_1})$
10: **else**
11: $\mathcal{H}_i = \mathbb{H}(\mathbb{M}_{\mathcal{T}_{i-1}},\ i,\ \mu_i,\ ID_{\mathcal{T}_i})$
12: **end if** ▷ calculate the watermark from the previous model by $\mathbb{H}(\cdot)$
13: $\hat{\Lambda}_{\mathcal{T}_i} = \mathbb{O}(W_{\mathcal{T}_{\mathcal{T}_i}}, Z_{\mathcal{T}_i}, k_{\mathcal{T}_i})$ ▷ extract the current watermark
14: $\eta_{\mathcal{T}_i} = 1 - \frac{\sum_b^B (\hat{\Lambda}_{\mathcal{T}_i}[j] \neq \Lambda_{\mathcal{T}_i}[j])}{B}$ ▷ calculate the watermarks similarity
15: **if** $\eta_{\mathcal{T}_i} < \eta_G$ **then**
16: **RETURN** "**Fail**", **break**
17: **end for**
18: **RETURN** "**Success**"

This iterative process continues until all the trainers finish watermark embedding (Lines 5–21). The proposed chained watermarking mechanism ensures that each watermark is uniquely bound to its predecessor trainer's output, thereby preventing $\mathcal{T}$ from fabricating a valid contribution beforehand to access compensation.

5.4.4 Chained Watermark Verification

To confirm that $\mathcal{T}$ completed optimal model training, instead of submitting the "free-lunch" model to claim their contributions to access compensation. Ensuring the ownership verification, validating the accuracy and reliability of the training results are necessary. Prior to verification, the client $\mathcal{C}$ uploads the pretrained client-side model $W_{\mathcal{C}}$ together with $\mathbb{M}_{\mathcal{C}}^{\text{DP}}$ to the verifier, while each trainer $\mathcal{T}_i$ uploads its checkpoints $W_{\mathcal{T}_i}$. The verifier $\mathcal{T}$ assembles the complete SL model W by merging $W_{\mathcal{T}_i}$ and, using $W_{\mathcal{T}_i}$ to calculate $\mathbb{M}_{\mathcal{T}_i}$ for subsequent watermark generation. The verification procedure consists of two stages: model performance verification and chained watermark verification.

The process begins with the verifier $\mathcal{V}$ evaluating the performance of the checkpoint W (Lines 1–4). This step ensures that the SL model achieves an accuracy level above a predefined threshold, which is determined based on empirical values commonly adopted in ML marketplaces [131]. If the accuracy of W falls below this threshold, $\mathcal{V}$ rejects $\mathcal{T}$'s training as invalid. Once the SL model W passes the performance evaluation, $\mathcal{V}$ proceeds to chained watermark verification (Lines 5–17). Specifically, using (5.15), $\mathcal{V}$ employs the activation $\mathbb{M}_{\mathcal{T}_i}$ of the model $W_{\mathcal{T}_i}$ to regenerate the ground-truth watermark Λ_i, the embedding key k_i, and the position matrix $Z_{\mathcal{T}_i}$ required for verifying the checkpoint $W_{\mathcal{T}_i}$ (Lines 6–12). Specifically, the ground-truth watermark for $\mathcal{T}_1$ is computed by feeding $\mathbb{M}_{\mathcal{C}}^{\text{DP}}$ into (5.15). This step ensures $\Lambda_{\mathcal{T}_i}$ is legitimately derived from $\mathbb{M}_{\mathcal{T}_i}$ of the model $W_{\mathcal{T}_i}$, not fabricated. Then, $\mathcal{V}$ uses $Z_{\mathcal{T}_i}$ and $k_{\mathcal{T}_i}$ on the checkpoint model

$W_{\mathcal{T}_i}$ to extract the embedded watermark $\hat{\Lambda}_{\mathcal{T}_i}$ (Line 13):

$$\hat{\Lambda}_{\mathcal{T}_i} = \mathbb{O}(W_{\mathcal{T}_{\mathcal{T}_i}}, Z_{\mathcal{T}_i}, k_{\mathcal{T}_i}). \tag{5.18}$$

Given a predefined watermark detection threshold η_G, the verification for the checkpoint $W_{\mathcal{T}_i}$ is considered successful if the similarity between the extracted watermark $\hat{\Lambda}_i$ and the ground-truth watermark Λ_i exceeds η_G. The watermark detection rate of $\hat{\Lambda}_i$ can be formally expressed as (Line 14):

$$\eta_{\mathcal{T}_i} = 1 - \frac{\sum_b^B (\hat{\Lambda}_{\mathcal{T}_i}[j] \neq \Lambda_{\mathcal{T}_i}[j])}{B}. \tag{5.19}$$

$\mathcal{V}$ continues the chained watermark validation process $W_{\mathcal{T}_i}$ by $W_{\mathcal{T}_i}$, checking the extracted watermark detection rate. Once all of $W_{\mathcal{T}_i}$ have been successfully verified, the SL model W is deemed to have correctly completed the training process.

5.4.5 Security Analysis

We analyze how CliCooper addresses the identified threats under our serverless, partially trusted, multi-client setting. Internally, rational honest-but-curious trainers seek to infer information from activations or to claim undue credit; externally, adversaries interact only via black-box APIs and attempt unauthorized misuse, e.g., extraction/distillation.

Threats from internal users (trainers)

Each trainer $\mathcal{T}_i$ can inspect its own parameters and the activations it receives, but never observes $\mathcal{C}$'s raw inputs or true labels, peers' parameters, or the secret label mapping. The principal risks are curiosity-driven inference on activations, including clustering, inversion, membership/attribute/label inference, and credit/ownership disputes, rather than protocol sabotage.

- **Privacy-gated label space & DP activation boundary.** Secret-mapping label expansion with DP-guarded activations and intra-pipeline opacity. Trainers operate solely in the pseudo-label space Y^*, which removes direct semantic interpretability and decouples observable class frequencies from the true task. At the client boundary, Laplace(Δ_1/ε) noise yields a ε-DP interface for smashed activations, limiting linkage between any pseudo-label pairs and restricting reconstruction power.
- **Segment opacity & cross-segment inference hardness.** Each trainer retains a private model segment and relays only upstream activations, so the internal white-box view is strictly local. Without access to peers' parameters, end-to-end Jacobians, or backward signals beyond the split, standard cross-segment attacks, e.g., reconstructing upstream features or attributing peer capacity/architecture, become ill-posed: the mapping from a single-segment view to others' weights is non-identifiable, gradient information is locally confined, and multiple, diverse vantage points required for triangulation are unavailable by design. This opacity complements the DP boundary, further suppressing clustering and inversion reliability.
- **Chained watermark lineage for fair attribution and ownership.** A trainer that reuses a pre-trained/open-source fragment across different tasks could otherwise claim rewards without genuine participation. CliCooper prevents such "free-lunch segment replay" by chaining watermarks across trainer $\mathcal{T}_i$, where each mark $\Lambda_{\mathcal{T}_i}$ is derived from $(\mathbb{M}_{\mathcal{T}_{i-1}},\ i,\ \mu_i,\ ID_{\mathcal{T}_i})$, binding the segment to the predecessor checkpoint, the run chronology, and the intended task. The fully trusted verifier $\mathcal{V}$ with omniscient audit access reconstructs expected marks and

validates detection and utility, establishing segment-level provenance, deterring free-riding, and resolving credit disputes without exposing peers' parameters.

Threats from external adversaries (black-box only)

Outsiders can only query the released model via APIs and have no access to architecture or weights, so the primary threat is unauthorized reuse through extraction or functional repurposing.

In CliCooper, outputs reside in a pseudo-label space and are operationally degraded without the secret inverse map $\mathcal{G}_Y^{-1}$, as a black-box extractor or distilled surrogate can at best replicate pseudo semantics, which do not translate to the true task without authorization. This breaks reliable relabeling and task remapping. Recovering $\mathcal{G}_Y^{-1}$ from queries amounts to solving a combinatorial preimage problem under label expansion, further obfuscated by activation-level DP noise that induces stochasticity in outputs and elevates the Bayes risk of any student trained on them.

The black-box release also blocks white-box watermark attacks, including removal, overwriting, pruning, and fine-tuning. Moreover, chained watermarks in CliCooper are derived from hidden checkpoints and identity-bound nonces, so an extracted surrogate cannot reproduce a valid lineage; any redistribution or task-mismatch reuse lacking a consistent chain is flagged by the trusted verifier $\mathcal{V}$ during provenance checks without revealing collaborators' parameters.

5.5 Experiments for CliCooper

We evaluate the proposed method to demonstrate its practical viability. Our experiments focus on three key aspects: *fidelity* (model performance impact), *robustness* (against clustering, inversion, and model extraction attack), and *accountability* (ownership verifiability and overhead).

5.5.1 Experimental Settings

To evaluate the fidelity, robustness, and accountability of CliCooper, we conduct experiments on diverse neural network architectures and heterogeneous datasets. Experiments are executed within a consistent micro-server environment to guarantee fair comparisons across all configurations.

Setup. The experiments are conducted on a high-performance server equipped with an AMD EPYC 9254 CPU operating at 2.9GHz, comprising 24 physical cores and 128MB of L3 cache (max turbo frequency up to 4.15GHz). The system is configured with 192GB of 4800MHz ECC DDR5 memory in a twelve-channel layout, ensuring sufficient bandwidth for large-scale parallel processing. For storage, it provides dual 1.92TB NVMe SSDs, one of which is allocated for scratch space. The training is accelerated using two NVIDIA L40 GPUs, each offering 18,176 CUDA cores, 568 Tensor Cores, and 48GB of dedicated memory. All experiments are run on a 64-bit Ubuntu 22.04 environment with CUDA 12.2 and cuDNN enabled for optimized deep learning performance.

Datasets. We evaluate our approach using four benchmark datasets spanning both image and text classification tasks. In the SL framework, all datasets (including both features and labels) are retained locally at $\mathcal{C}$ to preserve privacy.

- *MNIST* is a benchmark dataset for handwritten digit recognition, containing 70,000 grayscale images of digits 0-9, each of size 28×28. It is split into 60,000 training and 10,000 testing samples, and is widely used as a standard baseline for image classification.
- *CIFAR-10* comprises 60,000 RGB images of size 32×32 pixels, categorized into 10 distinct classes. We adopt the standard train-test partition with 50,000 images for training and 10,000 for testing.
- *CIFAR-100* contains 60,000 32×32 images, but spans 100 fine-grained categories. Each image includes both fine and coarse labels, with the same train-test split as CIFAR-10.
- *AG News* is a text classification corpus containing news headlines and descriptions from four categories. The dataset provides 120,000 training samples and 7,600 test instances, each labeled with its corresponding topic.

Model architectures. We select a variety of widely used neural architectures, including *LeNet*, *AlexNet*, *ResNet18*, and *WideResNet* for image classification tasks, and *TextCNN* and *MiniBERT* for text understanding. For vision models, *LeNet* and *AlexNet* provide foundational CNN baselines, while *ResNet* introduces skip connections that facilitate deeper optimization. *WideResNet* enhances accuracy by increasing the network's width. In the NLP domain, *TextCNN* captures local semantics through convolutional operations, and *MiniBERT* delivers efficient contextual encoding through lightweight transformer-based pretraining.

In the SL framework, $\mathcal{C}$ is responsible for initial data embedding and feature projection, after which the activation is forwarded to $\mathcal{T}$ for subsequent training and gradient updates. We apply *LeNet* and *AlexNet* to MNIST, *AlexNet* and *ResNet18* to CIFAR-10, *ResNet18* and *WideResNet* to CIFAR-100, and use both *TextCNN* and *MiniBERT* for AG News.

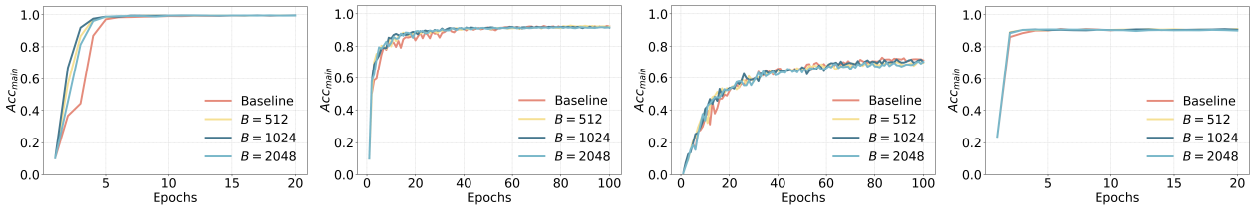
All models are optimized using SGD with momentum set to 0.9. Learning rates are chosen based on the dataset characteristics: 0.01 for MNIST, 0.1 for CIFAR-10 and CIFAR-100, and 0.005 for AG News. A consistent batch size of 256 is used across all training runs to maintain comparability.

5.5.2 Model Performance

To safeguard data and label privacy while enabling training traceability and copyright, CliCooper integrates label expansion, DP noise perturbation, and chained watermarking. Since these mechanisms inevitably interfere with the SL training, it is necessary to assess their impact on model performance. We therefore conducted three sets of experiments: (1) fixing γ=2.0 and ε=5.0, while varying watermark lengths $B \in$ 512, 1024, 2048; (2) fixing ε=5.0 and B=1024, while varying $\gamma \in$ 1.5, 2.0, 2.5; and (3) fixing B=1024 and γ=2.0, while varying $\varepsilon \in$ 2.0, 5.0, 10.0. The baseline is a standard SL model without any protection, achieving accuracies of 99.42% (DeepLeNet on MNIST), 99.45% (AlexNet on MNIST), 87.97% (AlexNet on CIFAR-10), 92.32% (ResNet18 on CIFAR-10), 71.97% (ResNet18 on CIFAR-100), 71.14% (WideResNet on CIFAR-100), 86.59% (TextCNN on AG News), and 90.68% (MiniBERT on AG News).

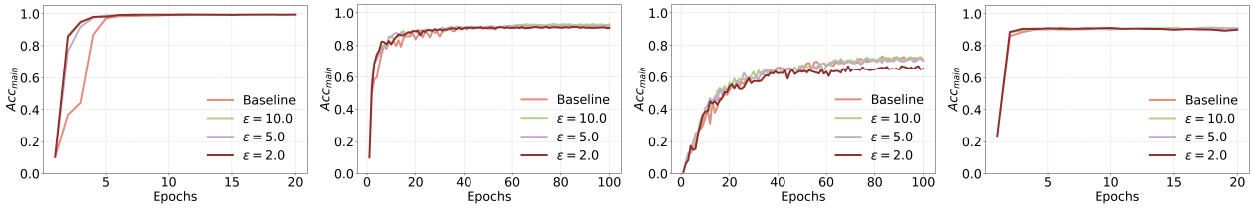
Figs.5.2–5.3 indicate that, in most cases, CliCooper has a negligible impact on accuracy, with performance remaining close to the baseline. This can be attributed to the redundancy of deep network parameters and the existence of multiple local optima; moderate noise injection guides the model toward different local optima without degrading accuracy.

In some extreme settings, $\varepsilon = 2.0$, accuracy drops by up to 3% due to excessive noise interference. Interestingly, in certain cases, noise injection even improves generalization: with $\varepsilon = 10.0$,



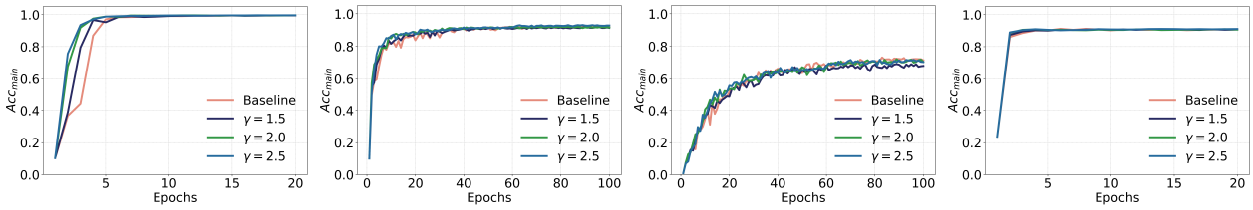
(a) AlexNet_MNIST (b) ResNet18_CIFAR10 (c) WideResNet_CIFAR100 (d) MiniBert_AG News

Figure 5.2: Comparison of the main accuracy with baseline across different architectures and datasets, where ε=5.0, γ=2.0, B=512/1024/2048.



(a) AlexNet_MNIST (b) ResNet18_CIFAR10 (c) WideResNet_CIFAR100 (d) MiniBert_AG News

Figure 5.3: Comparison of the main accuracy with baseline across different architectures and datasets, where B=1024, γ=2.0, ε=2.0/5.0/10.0.



(a) AlexNet_MNIST (b) ResNet18_CIFAR10 (c) WideResNet_CIFAR100 (d) MiniBert_AG News

Figure 5.4: Comparison of the main accuracy with baseline across different architectures and datasets, where ε=5.0, B=1024, γ=1.5/2.0/2.5.

AlexNet on CIFAR-10 exhibits a 1% accuracy gain, while ResNet18 on CIFAR-100 improves by 1.2%, which suggests that an appropriate noise can act as a regularizer, alleviating overfitting and slightly enhancing performance.

These results demonstrate that CliCooper achieves high fidelity, effectively preserving data privacy and model ownership while maintaining the original model's performance, and in some cases slightly improving it.

5.5.3 Resistance to Data Clustering Attacks

We evaluate a clustering-based label matching attacker in the split-learning setting where an internal trainer $\mathcal{T}_1$ cannot access the client's raw inputs or true labels, but can observe DP-protected activations and the corresponding pseudo-labels produced by label expansion. The attacker applies **K-means** [126], **Birch** [132], and **DBSCAN** [133] to the observed activations (all the clustering methods infer the number of clusters automatically) and tries to recover the hidden grouping structure induced by label expansion. We report two metrics averaged over three independent runs:

Table 5.2: Perfect clustering accuracy (%) under different cluster methods and protection settings.

Cluster Methods		MNIST DeepLeNet	MNIST AlexNet	CIFAR10 AlexNet	CIFAR10 ResNet18	CIFAR100 ResNet18	CIFAR100 WideResNet	AG News TextCNN	AG News MiniBert
Baseline		100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00
K-means	ε=2.0	23.33	16.66	0.00	0.00	0.00	0.00	41.66	83.33
	ε=5.0	40.00	30.00	0.00	0.00	0.00	0.00	50.00	91.66
	ε=10.0	46.66	50.00	0.00	0.00	0.00	0.00	66.66	91.66
	γ=1.5	23.33	20.00	0.00	0.00	0.00	0.00	75.00	91.66
	γ=2.0	20.00	13.33	0.00	0.00	0.00	0.00	50.00	91.66
	γ=2.5	13.33	13.33	0.00	0.00	0.00	0.00	33.33	83.33
Birch	ε=2.0	13.33	13.33	0.00	0.00	0.00	0.00	16.66	91.66
	ε=5.0	30.00	16.66	0.00	0.00	0.00	0.00	33.33	91.66
	ε=10.0	43.33	60.00	0.00	0.00	0.00	0.00	75.00	91.66
	γ=1.5	56.66	50.00	0.00	0.00	0.00	0.00	66.66	91.66
	γ=2.0	53.33	46.66	0.00	0.00	0.00	0.00	50.00	91.66
	γ=2.5	26.66	26.66	0.00	0.00	0.00	0.00	26.66	83.33
DBSCAN	ε=2.0	0.00	3.33	0.00	0.00	0.00	0.00	0.00	0.00
	ε=5.0	10.00	10.00	0.00	0.00	0.00	0.00	0.00	8.33
	ε=10.0	13.33	13.33	0.00	0.00	0.00	0.00	0.00	8.33
	γ=1.5	10.00	16.66	0.00	0.00	0.00	0.00	25.00	16.67
	γ=2.0	6.66	13.33	0.00	0.00	0.00	0.00	16.67	0.00
	γ=2.5	6.66	13.33	0.00	0.00	0.00	0.00	0.00	0.00

Table 5.3: Subset clustering accuracy (%) under different cluster methods and protection settings.

Cluster Methods		MNIST DeepLeNet	MNIST AlexNet	CIFAR10 AlexNet	CIFAR10 ResNet18	CIFAR100 ResNet18	CIFAR100 WideResNet	AG News TextCNN	AG News MiniBert
Baseline		100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00
K-means	ε=2.0	26.66	16.66	0.00	3.33	0.00	3.33	41.66	91.66
	ε=5.0	43.33	30.00	3.33	6.66	0.00	1.00	58.33	91.66
	ε=10.0	53.33	56.66	3.33	6.66	1.33	1.33	66.66	91.66
	γ=1.5	30.00	20.00	3.33	3.33	1.33	0.60	83.33	100.00
	γ=2.0	26.66	13.33	3.33	3.33	0.30	0.60	58.33	91.66
	γ=2.5	20.00	13.33	3.33	0.00	0.00	0.30	50.00	91.66
Birch	ε=2.0	16.66	13.33	0.00	0.00	1.66	4.66	25.00	91.66
	ε=5.0	33.33	16.66	0.00	0.00	2.66	6.66	33.33	91.66
	ε=10.0	50.00	70.00	3.33	0.00	2.66	6.66	75.00	91.66
	γ=1.5	30.00	50.00	6.33	3.33	3.33	3.66	75.00	100.00
	γ=2.0	5.66	40.00	3.33	3.33	1.00	4.00	50.00	100.00
	γ=2.5	20.00	40.00	3.33	3.33	1.00	2.33	46.66	83.33
DBSCAN	ε=2.0	0.00	3.33	0.00	0.00	0.00	0.00	8.33	0.00
	ε=5.0	10.00	10.00	0.00	0.00	0.00	0.00	16.66	8.33
	ε=10.0	13.33	13.33	3.33	0.00	0.00	0.00	16.66	8.33
	γ=1.5	10.00	16.66	3.33	0.00	0.00	0.00	25.00	16.66
	γ=2.0	6.66	10.00	0.00	0.00	0.00	0.00	16.66	0.00
	γ=2.5	10.00	13.33	0.00	0.00	0.00	0.00	0.00	10.00

- **Perfect clustering accuracy**, the fraction of true-label groups that are recovered exactly (i.e., the predicted cluster matches the complete true group without extra/missing samples).

- **Subset clustering accuracy**, the fraction of true-label groups that are recovered partially or fully (i.e., a group's samples are contained within a cluster even if the cluster contains partial samples—analogous to receiving partial credit in a multi-select question).

Tables 5.2-5.3 show that without protection (Baseline), clustering accuracy is 100% because activations and true labels are one-to-one, making grouping trivial. Under the protection, on CIFAR10/CIFAR100, clustering essentially fails: perfect accuracy is 0% across settings, and subset accuracy stays below 10%, indicating that DP perturbation plus label expansion largely destroys cluster separability in harder vision tasks. On MNIST, the attacker achieves moderate success (often $\leq$50%) because the underlying representations are more separable, making residual structure easier to exploit even after perturbation. The easiest case is AG News, where clustering reaches 50%–90%, suggesting that text representations can retain strong class separability and thus remain vulnerable to unsupervised grouping in some configurations.

As expected, decreasing ε (stronger DP noise) reduces both accuracies by obscuring activation geometry, while increasing γ (more pseudo-labels per true class) further lowers accuracy by increasing intra-class dispersion and inter-group mixing, making correct grouping harder for unsupervised clustering. Overall, label expansion plus DP noise renders $\mathbb{MC}^{\text{DP}}$ non-clusterable to an honest-but-curious $\mathcal{T}_1$, addressing **RQ5** (data privacy).

5.5.4 Resistance to Data Inversion Attacks

An honest-but-curious $\mathcal{T}_1$, given only the activation uploaded by $\mathcal{C}$ and no access to $\mathcal{C}$'s parameters, may attempt to reconstruct $\mathcal{C}$'s training data. Following the UNSplit framework [52], $\mathcal{T}_1$ initializes a surrogate network matching $\mathcal{C}$'s architecture and poses reconstruction as joint optimization over inputs and surrogate weights. We evaluate robustness on two representative settings—ResNet18/CIFAR-10 and WideResNet/CIFAR-100—where $\mathcal{C}$ first performs training-data embedding and then applies differential privacy (DP) with varying noise to the activation.

We quantify similarity using the Structural Similarity Index (SSIM) and summarise visuals in Table 5.4. The first row contains target images; the second shows reconstructions from the unprotected baseline, which recover clear shapes and textures, indicating severe leakage. Introducing DP progressively degrades reconstructions: with $\varepsilon = 10.0$, outlines and colors blur; at $\varepsilon = 5.0$, structures largely vanish, and noise dominates; by $\varepsilon = 2.0$, outputs collapse into snowflake-like patterns devoid of semantic content. SSIM scores follow the same monotonic decline.

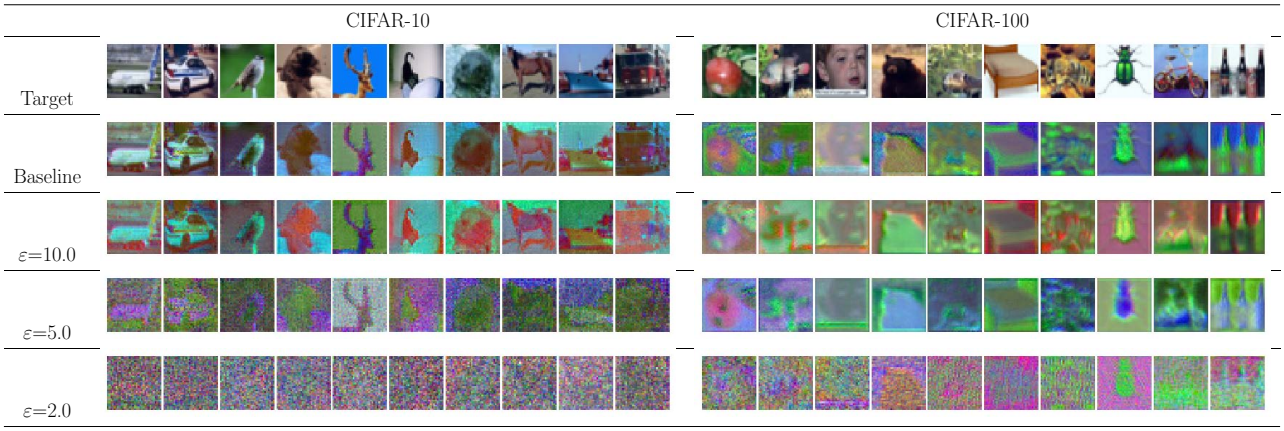
Stronger privacy budgets (smaller ε) thus render UNSplit ineffective in practice, while label expansion further prevents alignment between any residual visual cues and true class semantics, adding defense-in-depth. Overall, the combination of label expansion and DP noise on activations robustly thwarts inversion attempts, ensuring that CliCooper addresses **RQ5** (data privacy).

5.5.5 Resistance to Model Extraction Attacks

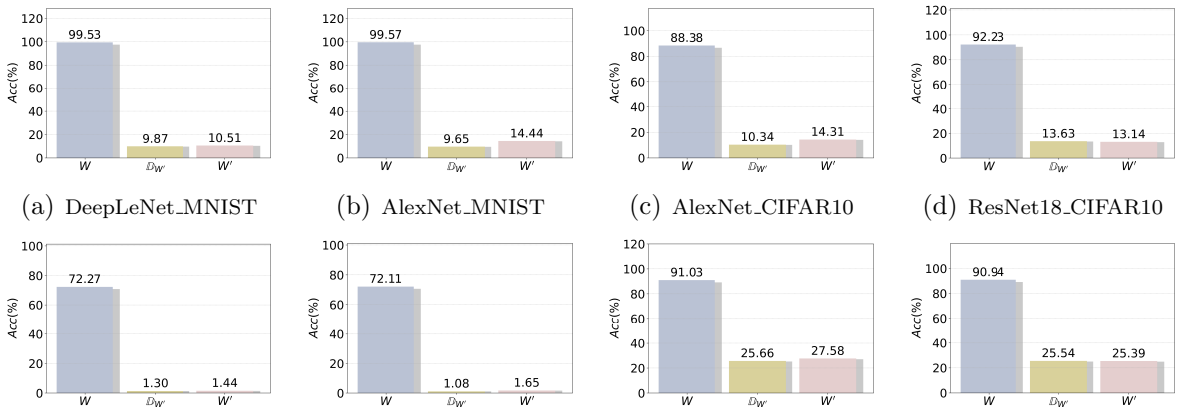
In our setting, the trained SL model is released in a black-box manner: adversaries cannot access the architecture or parameters and can only issue queries to the public API. Given an unlabeled dataset, an attacker may attempt model extraction [128], [134] by querying the API for pseudo-labels and training a surrogate. Our label expansion prevents this: the API returns only pseudo-labels, and without the one-to-many and inverse mappings ($\mathcal{G}_Y$,$\mathcal{G}_Y^{-1}$), the true labels are unrecoverable.

To test robustness, we construct unlabeled inputs by adding noise to MNIST, CIFAR10, CI-

Table 5.4: Visual comparison of inversion reconstructed samples under DP noise levels (ε) for CIFAR-10 and CIFAR-100.



•Reconstructing from unprotected activation yields relatively high similarity to the input sample—SSIM 0.50 on CIFAR-10 and 0.26 on CIFAR-100. Injecting DP noise at release markedly reduces fidelity: on CIFAR-10, SSIM drops to 0.38, 0.22, and 0.03; on CIFAR-100, it falls to 0.20, 0.15, and 0.07 for ε=10.0, 5.0, 2.0, respectively.



(a) DeepLeNet_MNIST (b) AlexNet_MNIST (c) AlexNet_CIFAR10 (d) ResNet18_CIFAR10

(e) ResNet18_CIFAR100 (f) WideResNet_CIFAR100 (g) TextCNN_AG News (h) MiniBert_AG News

Figure 5.5: Accuracy comparison for resisting model extraction attacks across different datasets and architectures.

FAR100 training data and by randomly shuffling tokens for AG News. We query the API for pseudo-labels with expansion factor $\gamma = 2.0$ (e.g., 10 true classes $\rightarrow$ 20 pseudo-labels). Lacking $(\mathcal{G}_Y, \mathcal{G}_Y^{-1})$, the attacker can only map each pseudo-label group to a true class at random, then trains a surrogate on the pseudo-labeled set and evaluates it on the true test set.

Fig.5.5 summarizes outcomes: gray bars show SL W's accuracy, yellow bars the pseudo-labeled dataset accuracy, and pink bars the attacker's surrogate $\mathcal{D}'$. Across datasets/architectures, attacker performance remains near random guess, $\approx$10% for MNIST and CIFAR10, 1% for CIFAR100, 25% for AG News, far below the SL model (e.g., 99.5% on MNIST, $\approx$90% on AG News). Thus, API outputs cannot be exploited to assemble a meaningful training set or a competitive surrogate.

Overall, these results show that label expansion offers strong protection against black-box model extraction, effectively blocking unauthorized reproduction or misuse of released SL models even under full query access—thereby demonstrating that CliCooper resolves **RQ7** (unauthorized-use defense).

Table 5.5: Total training time, watermark embed/verify time (ms), and extraction accuracy (%) across MNIST, CIFAR10/100, and AG News models. We vary ε, γ and watermark length B. Overhead remains small even at B=2048; accuracy >99%. Larger γ increases total training time; "Baseline" has no watermarking.

	Time/Acc	MNIST DeepLeNet	MNIST AlexNet	CIFAR10 AlexNet	CIFAR10 ResNet18	CIFAR100 ResNet18	CIFAR100 WideResNet	AG News TextCNN	AG News MiniBert
Baseline	Training time (s)	629.98	509.71	473.07	1013.43	1013.63	6267.04	288.93	1488.34
	Watermark Acc (%)	0	0	0	0	0	0	0	0
ε=2.0	Training time (s)	778.87	623.61	611.44	1200.14	1214.14	7188.71	389.74	1614.01
	Embed time (ms)	8.68	8.94	8.83	13.17	13.85	9.56	8.31	8.17
	Verify time (ms)	8.65	8.95	8.67	12.99	12.98	9.45	8.29	8.17
	Watermark Acc (%)	99.67	99.81	99.78	99.56	99.67	99.21	99.17	99.78
ε=5.0	Training time (s)	784.98	633.46	628.14	1192.48	1215.37	7463.68	393.29	1650.14
	Embed time (ms)	8.12	8.72	8.66	13.31	13.69	9.19	8.53	7.97
	Verify time (ms)	8.22	8.65	8.44	13.05	13.65	9.11	8.47	7.67
	Watermark Acc (%)	99.56	98.89	99.75	99.35	99.32	99.23	99.27	98.89
ε=10.0	Training time (s)	786.43	640.82	639.48	1199.61	1206.17	7687.28	399.01	1576.13
	Embed time (ms)	8.61	8.77	8.81	13.41	13.64	9.27	7.51	7.47
	Verify time (ms)	8.62	8.57	8.81	13.34	13.56	9.21	7.47	7.35
	Watermark Acc (%)	98.89	99.21	99.37	98.99	99.24	99.24	99.18	99.03
γ=1.5	Training time (s)	786.43	640.82	639.48	1199.61	1206.17	7687.28	399.01	1576.13
	Embed time (ms)	8.61	8.77	8.81	13.41	13.64	9.27	7.51	7.47
	Verify time (ms)	8.62	8.57	8.81	13.34	13.56	9.21	7.47	7.35
	Watermark Acc (%)	99.32	99.87	99.37	99.21	99.18	99.23	99.28	99.38
γ=2.0	Training time (s)	988.69	809.33	822.27	1564.91	1517.11	9627.41	511.65	2088.91
	Embed time (ms)	8.61	8.52	9.21	13.37	13.62	8.98	7.99	7.32
	Verify time (ms)	8.61	8.67	9.23	13.28	13.34	9.01	7.89	7.25
	Watermark Acc (%)	99.45	99.78	99.38	99.25	99.52	99.21	99.28	99.26
γ=2.5	Training time (s)	989.13	817.74	819.56	1538.17	1548.11	9691.42	511.15	2080.57
	Embed time (ms)	8.53	8.68	9.34	13.99	13.53	9.01	7.42	7.09
	Verify time (ms)	8.51	8.63	9.12	12.98	13.22	8.99	7.56	7.21
	Watermark Acc (%)	98.99	99.43	98.87	99.26	99.63	99.23	99.18	99.24
B=512	Training time (s)	786.43	640.82	639.48	1199.61	1206.17	7687.28	399.01	1576.13
	Embed time (ms)	8.61	8.77	8.81	13.41	13.64	9.27	7.51	7.47
	Verify time (ms)	8.62	8.57	8.81	13.34	13.56	9.21	7.47	7.35
	Watermark Acc (%)	99.21	99.48	99.89	99.26	99.02	98.89	99.25	99.34
B=1024	Training time (s)	803.34	659.94	661.51	1185.81	1206.68	7654.09	416.78	1637.21
	Embed time (ms)	12.89	12.88	14.23	22.64	21.53	13.42	12.45	13.13
	Verify time (ms)	12.83	12.65	14.32	22.46	21.78	13.21	13.01	13.27
	Watermark Acc (%)	99.36	99.21	99.21	99.37	99.71	99.24	99.39	99.28
B=2048	Training time (s)	906.76	680.91	680.39	1225.16	1247.42	8013.85	433.59	1773.24
	Embed time (ms)	21.41	25.57	25.98	43.79	40.65	24.48	24.89	24.21
	Verify time (ms)	21.22	25.22	24.99	43.78	40.22	24.68	24.62	24.18
	Watermark Acc (%)	99.68	99.23	99.79	99.29	98.87	99.23	99.09	99.13

5.5.6 Ownership Verifiability and Overhead

Our experiments (Table 5.5) show that trainer clients can furnish cryptographically anchored ownership claims with minimal overhead. Across MNIST (DeepLeNet/AlexNet), CIFAR-10 (AlexNet/ResNet18), CIFAR-100 (ResNet18/WideResNet), and AG News (TextCNN/MiniBERT), increasing watermark length B predictably raises embedding and verification time (linear encoder/decoder), yet the absolute costs remain small relative to training: even at $B = 2048$, embed and verify each stay in the single– to low–tens–of–milliseconds range, so the end-to-end impact is marginal and does not affect scheduling or throughput for compensated training.

Watermark extraction achieves over 99% accuracy across all models and settings, providing reliable and automated proof of contribution. This detection rate confirms the effectiveness of

Table 5.6: Per-link communication latency (s) between adjacent parties.

Transfer	$\mathcal{C} \longrightarrow \mathcal{T}_1$	$\mathcal{T}_1 \longrightarrow \mathcal{T}_2$	$\mathcal{T}_2 \longrightarrow \mathcal{T}_3$	T_{total}
MNIST DeepLeNet	0.24	0.12	0.06	0.42
MNIST AlexNet	0.06	0.09	0.01	0.16
CIFAR10 AlexNet	0.08	0.12	0.02	0.22
CIFAR10 ResNet18	0.32	0.16	0.08	0.56
CIFAR100 ResNet18	0.32	0.16	0.08	0.56
CIFAR100 WideResNet	0.80	0.40	0.20	1.40
AG News TextCNN	0.08	0.01	0.01	0.10
AG News MiniBert	0.08	0.06	0.06	0.20

the chained design, where each trainer's seed derives from its predecessor's activation, producing stable model-dependent signals that cannot be precomputed or substituted without following the designated training process. Although total training time increases with the label-expansion factor γ due to larger pseudo-labeled datasets, this effect is independent of watermarking and remains within acceptable bounds.

Table 5.6 reports the per-link communication latency in CliCooper with batch size 256 and an assumed bandwidth of 200 MB/s. Since only intermediate activations are relayed along $\mathcal{C} \longrightarrow \mathcal{T}_1 \longrightarrow \mathcal{T}_2 \longrightarrow \mathcal{T}_3$, the latency mainly depends on the interface tensor size and thus increases with model/dataset complexity. For n trainer clients in a relay chain, the per-iteration communication time scales linearly with the number of segment boundaries and the interface size, i.e., $T_{comm} = O(n(\frac{\bar{s}}{R} + \theta))$, where $\bar{s}$ is the average interface tensor size, R is the link bandwidth, and θ is the per-link propagation/handshake latency. The worst case is CIFAR100 with WideResNet, where the bottleneck link takes 0.80 s, and the total end-to-end latency is 1.40 s. In contrast, AG News with TextCNN yields a bottleneck of 0.06 s and a total of 0.10 s. Even the maximum latency is negligible compared to the training time in Table 5.5; multi-trainer coordination incurs only modest overhead under typical bandwidth.

Overall, the chained watermark scheme is lightweight, robust, and scalable, providing high-confidence evidence of trainer ownership suitable for compensation in partially trusted collaborative training—demonstrating that CliCooper resolves **RQ6** (layer ownership).

5.6 Introduction of Watermarking for FL

FL [12] is designed to enable powerful machine learning across multiple participants while ensuring data privacy by keeping data local [135], which is a critical concern in today's landscape of rapidly advancing deep neural networks (DNNs). However, training a sophisticated FL model is a collaborative effort that is both time-consuming and resource-intensive, highlighting the need for robust solutions to protect the model's intellectual property (IP) [136], [137]. Such solutions should not only safeguard the model's utility and ownership but also reflect each participant's contribution without causing interference between them. This means that these solutions need to be universal in terms of performance and security requirements [138], [139], [140].

Various watermarking techniques have been proposed to protect the models' IP, demonstrating their powerful capabilities across multiple domains [10], [45], [47], [48], [80], [141], [142], [143].

These existing watermarking techniques can be categorized into parameter-based watermarking schemes [10], [45], [47], [48] and backdoor-based watermarking schemes [80], [141], [142], [143]. FL model training differs from non-FL model training. FL involves a collaborative training process with multiple clients, where the final model is co-owned by all participating clients. As a result, the IP protection for FL models differs from that of non-FL models. The FL model watermarking scheme must allow each client to embed their own watermark to verify their model ownership while avoiding conflicts between the embedded watermarks from different clients.

WAFFLE [79] is the first scheme to embed backdoor-based watermarks into FL models. Subsequently, Methods [83], [84] and [31] propose hybrid federated model watermarking verification schemes that combine parameter-based and backdoor-based watermark. However, these FL model watermarking schemes are still not fully optimized. They present some drawbacks, for example, the watermarks embedded by different client nodes can interfere with each other, reducing the watermark detection rate in the later stages [31]. Watermark conflicts are likely to occur because clients may embed their watermarks in the same region of the model. When this situation arises, during the model aggregation process, the watermarks from different clients can overlap, resulting in conflicts among the embedded watermarks.

To address the issue of conflicts between client watermarks while ensuring that watermark embedding does not degrade the performance of the main task in FL models, we propose a scheme named Non-Interfering Fragmented Watermarking (FedNIFW). The key aspect of FedNIFW lies in the pre-agreement among clients regarding the specific regions to embed their watermarks. By independently embedding watermarks in different regions, their watermarks will not interfere with one another. During the watermark embedding process, each client has the flexibility to define their own watermark, embedding key, and watermarking algorithm. Another innovation is the exclusivity training for the watermark embedding regions. It means that the regions assigned to a client for watermark embedding are exclusively accessible to that client for parameter training and watermark embedding, while other clients must freeze these regions during their model training. When the server aggregates the model, the parameter values in the watermark embedding regions from each client local model are directly assigned to the corresponding regions of the global model without considering the parameter values from other client local models in those regions.

The contributions can be summarized as follows:

- We propose a novel, non-interfering fragmented watermark scheme for federated models. By embedding watermarks in distinct regions, our scheme avoids the problem of watermark interference. We also introduce an exclusive region-based aggregation method for FL. Specifically, when aggregating the model, the parameter values in the watermark embedding regions are exclusively derived from the client responsible for that region, ensuring that the global model adopts the parameters corresponding to the watermark-embedding client.
- We conduct rigorous experiments to demonstrate the effectiveness of FedNIFW. The experimental analysis shows that the proposed watermarking scheme achieves high fidelity, as evidenced by only a 2% reduction in model accuracy. FedNIFW also has excellent watermark significance, with no decline observed as the watermark length increased or as the number of clients grew. In terms of robustness, our scheme proved resilient, with the watermark detection rate decreasing less than 10% under model fine-tuning and model pruning attacks.

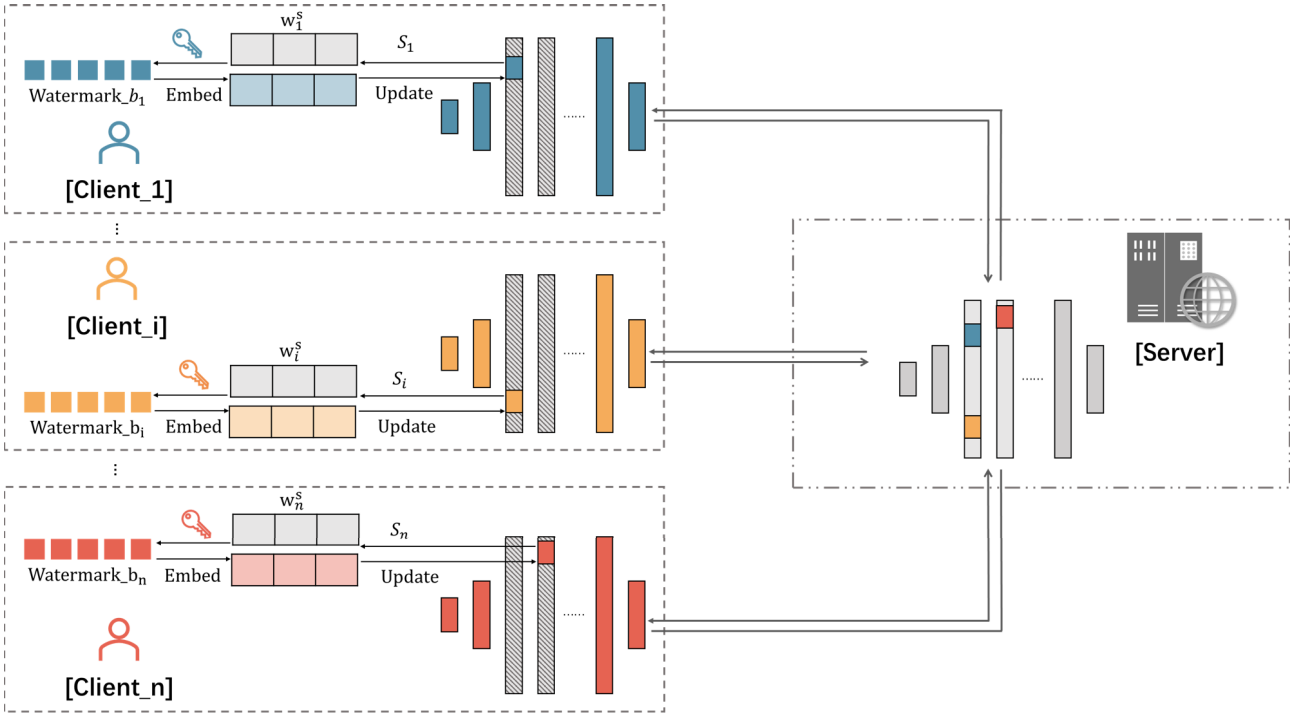


Figure 5.6: The FL watermarking framework of the proposed scheme. Before the commencement of federated learning, certain layers of the neural network are designated as watermark layers specifically for embedding watermarks. Each client is allocated a specific segment within the watermark layers for watermark embedding, while other segments of the watermark layers are frozen to prevent watermark interference.

5.7 Non-Interfering Fragmented Watermarking Design

Traditional client-side-based federated watermarking schemes are prone to watermark conflicts, as clients may embed watermarks in the same segments, and the watermarks may overlap during the aggregation. This issue becomes more pronounced when there are a large number of clients involved in federated learning or when the watermark length is oversized [31], leading to a reduced watermark detection rate.

To address this problem, we propose a novel Non-Interfering Fragmented Watermarking scheme for Federated Learning (FedNIFW). Assume that the model used in FL, M, is composed of m layers, denoted as $\mathbb{L} = L_1, L_2, ..., L_m$, and we have:

$$M = Concat(L_1, L_2, ..., L_m) \tag{5.20}$$

Accordingly, the global model in FL is denoted as $M^G = Concat(L_1^G, L_2^G, ..., L_m^G)$ and the local model of client i is denoted as $M^i = Concat(L_1^i, L_2^i, ..., L_m^i)$.

FedNIFW consists of two stages: In the first stage, certain layers of the neural network are designated as watermark layers $\mathbb{L}_v = L_{v_1}, L_{v_2}, ..., L_{v_p}$ specifically for embedding watermarks. Before the onset of FL, each client is assigned a specific segment of the $\mathbb{L}_v$ for embedding their watermark, ensuring that the watermarks do not interfere with one another. In the second stage, to further prevent watermark interference, clients are instructed to train only their designated watermark segments while freezing the segments allocated for watermark embedding by other clients. Each client trains the non-watermark layers L_o in $\mathbb{L}_{oth} = \mathbb{L} \setminus \mathbb{L}_v$ to ensure the local model can adapt to the client's main task. During model aggregation, the watermark layers in

Algorithm 8 Watermark Embedding and FL

Require: b_i (Watermark), S_i (Selection key), X_i (Embedding key).
Ensure: Global model M^G

for $r \in R$ rounds **do** ▷ training loop
  The server initializes the global model M^G and distributes the global model M^G to each client.
  $w_1, w_2, ..., w_n \overset{splited}{\leftarrow} \mathbb{L}_v$. ▷ The participants of FL mutually agree on the specific layer $\mathbb{L}_v$ to be used for embedding watermarks. $\mathbb{L}_v$ is then evenly partitioned among the n clients.
  $w_1^s, w_2^s, ..., w_n^s \overset{S_i}{\leftarrow} w_1, w_2, ..., w_n$.. ▷ Each client customizes its own watermark b_i, embedding key X_i, and selection key S_i, and selects the weights w_i^s for watermark embedding from the designated watermark segment w_i.
  Each client trains its local model and embeds the watermark within w_i^s while freezing the segments of the watermark layer allocated to other clients.
  for $i \in n$ *clients* **do**
    for $l \in \mathbb{L}$ **do**
      if $l \in \mathbb{L}_{oth}$ or $l \in w_i$ **then**
        $Loss(\theta) = Loss_o(\theta) + \lambda Loss_w(\theta)$. ▷ Train and embed the watermark b_i.
      else
        freeze l
      end if
    end for
  end for
  Upload M^i to the server.
  for $L_q^G \in \mathbb{L}$ **do**
    if $L_q^G \in \mathbb{L}_v$ **then**
      $L_q^G = \mathcal{H}(w_1, w_2, ..., w_n)$.
    else
      $L_q^G = \frac{\sum |D_i| L_q^i}{\sum |D_i|}$.
    end if
  end for
end for
$M_G = Concat(L_q^G, \forall L_q^G \in \mathbb{L})$.
RETURN Global model M^G

$\mathbb{L}_v$ are aggregated exclusively. That is, the parameter values of each segment in the $\mathbb{L}_v$ of the global model are determined solely by the local model of the client responsible for embedding the watermark in that segment, without considering the parameter values from other local models for that segment. All non-watermark layers in $\mathbb{L}_{oth}$ use the FedAvg algorithm [12] to aggregate. Fig. 5.6 illustrates the framework of the proposed scheme.

5.7.1 Watermark Embedding

Algorithm 8 illustrates the process of FL and watermarking. FL begins with initiating and distributing the global model to clients. Before the local training, specific layers of the neural network are designated as watermark layers, i.e., $\mathbb{L}_v = L_{v_1}, L_{v_2}, ..., L_{v_p}$. These watermark layers are then evenly distributed to n segments w_i, where n is the number of clients.

$$w_1, w_2, ..., w_n \overset{splited}{\leftarrow} \mathbb{L}_v. \tag{5.21}$$

where w_i is the watermark layer segment that is distributed to the i-th client. Each client is allowed to define its own watermark $b_i \in \{0,1\}^T$, along with the watermark embedding algorithm and the watermark embedding key X_i. To facilitate the demonstration of the proposed

method, we assume that each client employs the watermarking approach introduced by Uchida et al. [10]. For security purposes, each client will select a subset of weights from the allocated watermark segment for embedding the watermark. The key used for selecting the weights is denoted as S_i.

$$w_1^s, w_2^s, ..., w_n^s \overset{S_i}{\leftarrow} w_1, w_2, ..., w_n, \tag{5.22}$$

where w_i^s is the selected weights from the watermark layer segment by the i-th client. This fragmented watermarking method ensures that the watermarks embedded by different clients do not conflict with one another.

Using Uchida method [10], each client integrates Eq. 5.23 to perform local model training while embedding their respective watermark b_i into the selected weights w_i^s.

$$Loss(\theta) = Loss_o(\theta) + \lambda Loss_w(\theta), \tag{5.23}$$

where λ is the adjustable parameter, $Loss_o(\theta)$ is the main task loss function, and $Loss_w(\theta)$ is the watermark embedding regularizer.

$$Loss_w(\theta) = -\sum_{j=1}^{T}(b_j \log(b_j^{'}) + (1 - b_j)\log(1 - b_j^{'})), \tag{5.24}$$

where $b_j^{'} = \sigma(\sum_k X_{jk} w_k^s)$ and $\sigma(z)$ is the sigmoid function.

$$\sigma(z) = \frac{1}{1 + \exp(-z)}. \tag{5.25}$$

During the training of the watermark layers $\mathbb{L}_v$, each client is restricted to training and watermarking only within their allocated watermark segment w_i. To prevent watermark interference, it is necessary to freeze other watermark segments that are allocated to other clients.

5.7.2 Global Model Aggregation

The server collects the local models from all participating clients to perform model aggregation. This process is primarily divided into two cases: aggregation of non-watermark layers $\mathbb{L}_{oth}$ and aggregation of watermark layers $\mathbb{L}_v$. For the aggregation of non-watermark layers $L_o \in \mathbb{L}_{oth}$, we utilize the FedAvg algorithm [12], which involves weighted averaging of the parameters from all local models. In contrast, the aggregation of watermark layers $\mathbb{L}_v$ follows an exclusive direct concatenation approach. Specifically, the parameter values from the segments w_i where clients have embedded their watermarks are directly assigned to the corresponding segments in the global model M^G while disregarding the parameters from other client local models M^i in those segments.

$$L_q^G = \begin{cases} \mathcal{H}(w_1, w_2, ..., w_n), & \text{if } L_q^G \in \mathbb{L}_v; \\ \frac{\sum |D_i| L_q^i}{\sum |D_i|}, & \text{if } L_q^G \in \mathbb{L}_{oth}, \end{cases} \tag{5.26}$$

where $\mathcal{H}(\cdot)$ represents the concatenation of the segments w_i, connecting only the segments relevant to L_q^G while ignoring those that do not belong to this layer. And then, we have:

$$M^G = Concat(L_q^G, \forall L_q^G \in \mathbb{L}). \tag{5.27}$$

Algorithm 9 Watermark Extraction

Require: M^G (Global model), S_i (Selection key), X_i (Embedding key).
Ensure: Watermark detection rate η.

$w_1^s, w_2^s, ..., w_n^s \xleftarrow{S_i} w_1, w_2, ..., w_n$. ▷ Each client selects their designated watermark-embedded segment w_i from the global model M^G. Utilizing the selection key S_i, the client extracts the selected weights w_i^s from w_i.
Each client employs their embedding key X_i to extract the watermark from the selected weights w_i^s.
$b_i' = []$
for $t \in T$ **do**
    if $X_{i_t} \times w_{i_t}^s \geq 0$ **then**
        b_i'.append$'1'$.
    else
        b_i'.append$'0'$.
    end if
end for
Calculate the watermark detection rate(η).
for $t \in T$ **do**
    if $|b_i'[t] - b'[t]| \geq 0.5$ **then**
        $c += 1$.
    end if
end for
Watermark detection rate η.
RETURN "**Success**"

5.7.3 Watermark Extraction

In the event of model plagiarism, the federated model owner can claim ownership through watermark extraction and verification. Due to external interference or adversarial attacks, the extracted watermark b' may exhibit discrepancies compared to the original watermark b. We utilize the watermark detection rate(η) to assess the fidelity of the watermark. A higher η indicates a stronger claim to the authenticity of the model IP.

$$\eta = 1 - \frac{\sum_{i=1}^{T} f(t)}{T}, \begin{cases} f(t) = 0 & \text{if } |b'[i] - b[i]| > 0.5; \\ f(t) = 1 & \text{else} \end{cases}, \tag{5.28}$$

The process of watermark extraction and verification can be represented by **Algorithm** 9. Initially, each client needs to identify their designated watermark segment w_i within the watermarked global model $\hat{M}$. Using the selection key S_i, the client extracts the weights w_i^s where the watermark has been embedded. The watermark extraction is performed by multiplying the embedding key X_i with the embedded weights w_i^s. Each bit of the extracted watermark b' is determined based on the dot result—if the product is greater than zero, the bit is set to 1; otherwise, it is set to 0.

The watermark extraction is performed according to the following process:

$$b_t' = s(X_{i_t} w_{i_t}), \tag{5.29}$$

where $s(z)$ is the step function,

$$s(z) = \begin{cases} 1 & z \geq 0 \\ 0 & else \end{cases}. \tag{5.30}$$

The final step of the watermark extraction algorithm is to calculate the watermark detection rate η, which serves as the criterion for determining the ownership of the model.

5.8 Experiments and Analysis for FL Watermarking

In this session, we will demonstrate the effectiveness of our proposed method through a series of experiments. The proposed method employs a region-based, non-interfering watermark embedding scheme. Consequently, it avoids watermark conflicts that could arise from an increase in the number of clients or the length of the watermark, thereby preventing a decline in the watermark detection rate. By controlling the size of the watermark embedding regions, our method ensures minimal impact on the performance of the federated model. We have also evaluated the robustness of our method against common watermark attacks, confirming its strong resilience.

5.8.1 Experimental Setup

Dataset: In this experiment, we utilized two widely recognized datasets for image classification tasks: **CIFAR10** and **CIFAR100**. Both datasets consist of 60,000 image samples, each being a 32×32 three-channel color image. The datasets are partitioned into 50,000 training images and 10,000 test images. The content of the images in both datasets is relatively similar; however, CIFAR10 comprises 10 labels, while CIFAR100 includes 100 labels. These datasets are commonly used to evaluate deep learning and machine learning models, providing a standard benchmark for assessing the effectiveness of federated learning methods and watermarking techniques for model IP protection.

Model Neural Networks: In this experiment, two classic deep learning models, **AlexNet** and **ResNet18**, are employed. Specifically, AlexNet is utilized for training on the CIFAR10 dataset, while ResNet18 is employed for training on the CIFAR100 dataset. These neural network architectures are well-established in the field of deep learning and are commonly used for benchmarking in image classification tasks.

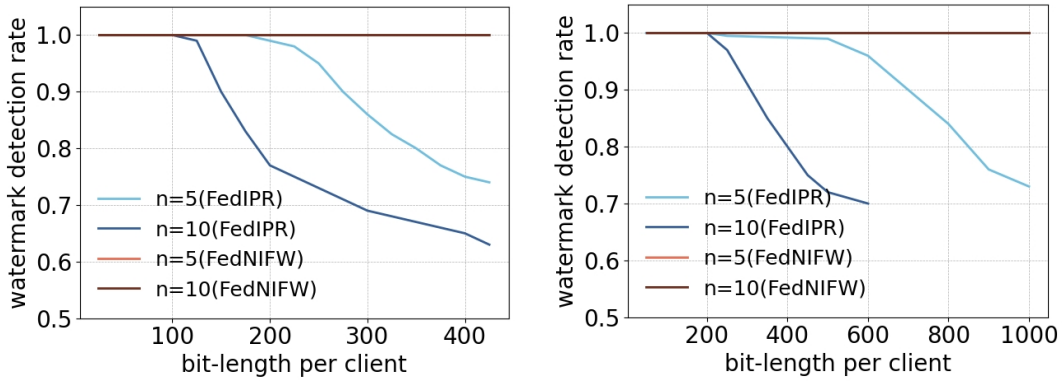
Federated Learning Setting: In the FL setup, the number of participating clients is set to 5 and 10, respectively. In distributing data to each client, we employed an Independent and Identically Distributed (IID) allocation strategy. During each iteration, each client first performs local model training for 1 epoch and then submits the model to the server for aggregation. During global model aggregation, the non-watermarked layers are aggregated using the FedAvg algorithm [12], while the watermarked layers are concatenated using segments from the client local watermark layers. The learning rate for all client local models is set to 0.1, utilizing SGD as the optimizer, with the main task loss function being binary cross-entropy. The communication rounds between clients and the server are set to 300.

5.8.2 Fidelity

We compared the accuracy of watermarked and non-watermarked FL models. In the fidelity comparison experiments, the number of clients participating in FL is set to $n = 5$ and $n = 10$, respectively. The watermark length in the watermarked FL models is set to $T = 128$ and $T = 256$. Table 5.7 presents the decrease in the main task accuracy for watermarked FL models. The comparative results indicate that, under different watermark settings, the main task accuracy of the watermarked models experienced only a slight decline, with the maximum decrease not exceeding 2%. The fidelity results of FedIPR [31] are comparable to those of the proposed scheme, suggesting that embedding a watermark into the FL model does not significantly impact its accuracy.

Table 5.7: Presents a comparison between the main task accuracy of watermarked federated models and the non-watermarked federated models (Baseline). The comparison is conducted under different configurations: with client numbers set to $n = 5$ and $n = 10$, and watermark lengths set to $T = 128$ and $T = 256$. The models are trained on the CIFAR10 dataset using the AlexNet neural network and on the CIFAR100 dataset using the ResNet18 neural network.

	Dataset	Watermarked Model				Non-watermarked Model(Baseline)	
		n=5		n=10		n=5	n=10
		T=128	T=256	T=128	T=256		
FedNIFW	CIFAR10	88.93%±0.32%	88.88%±0.43%	88.91%±0.27%	88.83%±0.42%	90.65%±0.22%	90.48%±0.20%
	CIFAR100	73.02%±0.05%	73.00%±0.03%	73.02%±0.03%	73.96%±0.04%	73.57%±0.07%	73.53%±0.08%
FedIPR [31]	CIFAR10	88.96%±0.25%	88.92%±0.36%	88.87%±0.18%	88.78%±0.32%	90.43%±0.23%	90.27%±0.17%
	CIFAR100	73.43%±0.24%	73.12%±0.42%	72.96%±0.21%	72.85%±0.15%	74.03%±0.13%	73.91%±0.24%
FedTracker [81]	CIFAR10	86.66%±0.15%	86.52%±0.16%	86.43%±0.13%	86.32%±0.12%	87.10%±0.23%	86.79%±0.15%
	CIFAR100	74.64%±0.14%	74.62%±0.32%	74.36%±0.11%	74.25%±0.18%	74.88%±0.23%	74.81%±0.21%



(a) AlexNet with CIFAR10

(b) ResNet18 with CIFAR100

Figure 5.7: A comparison of the watermark extraction success rates between our proposed FedNIFW scheme and the FedIPR scheme [31] under two different model configurations. The comparison was carried out across varying numbers of clients ($n = 5$ and $n = 10$) and different watermark lengths (ranging from 25 to 1000 bits).

5.8.3 Watermark Significance

The watermark significance experiment reflects the occurrence of watermark conflicts among different clients. If watermark conflicts occur, the watermark detection rate will decrease. Fig. 5.7 illustrates the watermark detection rate η from watermarked FL models under various watermark settings (watermark length ranging from 25 to 1000 bits) and different numbers of clients ($n = 5$ and $n = 10$). Here, the value of η refers to the average watermark detection rate across all clients. In the following experiments, unless specified, η represents the average watermark detection rate across all clients. As shown in Fig. 5.7, regardless of the watermark length or the number of clients, the proposed method (FedNIFW) consistently achieves a 100% watermark detection rate. In contrast, the watermark detection rate of the FedIPR [31] scheme significantly decreases as the number of clients increases or the watermark length grows. This phenomenon demonstrates that the FedNIFW scheme effectively prevents watermark conflicts among clients.

5.8.4 Fine-tuning Attack

Many large models are fine-tuned on specific domain datasets based on existing pre-trained models. In the scenario of our research, model fine-tuning can be regarded as a potential attack method against model watermarking. To evaluate the robustness of the proposed scheme against model fine-tuning attacks, we first pre-trained an FL model, AlexNet on the CIFAR10 dataset. During the training of the pre-trained model, each client embeds a watermark of 256 bits in length. The pre-trained model is then subjected to a fine-tuning attack. During the fine-tuning process, the watermark regularizer term is removed, and only the main task loss function is employed. The learning rates for both the non-output and output layers are set to one-tenth of the original pre-trained model's learning rate, and the fine-tuning process is conducted over 50 epochs.

Fig. 5.8 demonstrates the effectiveness of the proposed method in resisting model fine-tuning attacks, as the watermark detection rate consistently remains at 100%, even after 50 epochs of fine-tuning. FedIPR [31] also shows strong resistance to model fine-tuning attacks. Although there is a slight fluctuation in the watermark detection success rate at the initial stages of fine-tuning, it quickly stabilizes around 100%. In contrast, FedTracker [81] exhibits a different pattern, with its watermark detection rate gradually decreasing as the number of fine-tuning rounds increases, dropping from an initial 100% to 94%. These findings indicate that the proposed method offers superior robustness against model fine-tuning attacks.

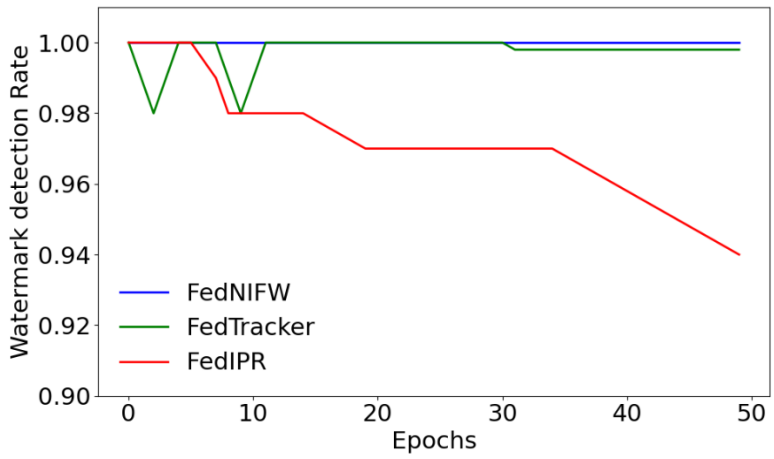


Figure 5.8: The figure illustrates the variation in watermark detection success rates of different schemes under model fine-tuning attacks. In the federated learning and watermark parameter settings, 10 clients utilized the AlexNet network to train on the CIFAR10 dataset, embedding a watermark of 256 bits in length. Both FedNIFW and FedIPR [31] exhibit strong robustness against model fine-tuning attacks, maintaining high watermark detection rates. In contrast, the watermark detection rate for FedTracker [81] decreases as the epochs of fine-tuning increase, indicating less resilience against such attacks.

5.8.5 Pruning Attack

Model pruning is a crucial technique for reducing the size of model files, enabling them to be deployed on low-power devices. However, it is also a common method used by attackers to remove model watermarks. To assess the robustness of the proposed scheme against model pruning, we first pre-train AlexNet on the CIFAR10 dataset. We then perform pruning opera-

tions on the pre-trained federated model. In the pruning experiments, the pruning rates vary from 0 to 0.9.

As shown in the experimental results in Fig. 5.9 The experimental results depicted in Fig. 5.9 indicate that both FedNIFW and FedIPR [31] demonstrate strong robustness against model pruning attacks. Their watermark detection rates remain at 100% as the pruning rate increases, although the detection rate for FedNIFW starts to decline when the pruning rate reaches 0.8, ultimately dropping to 93%. In contrast, the watermark detection rate for FedTracker [81] decreases more significantly as the pruning rate increases, falling to 50% when the pruning rate reaches 0.5, followed by minor fluctuations. These findings suggest that the proposed scheme is effective in defending against model pruning attacks.

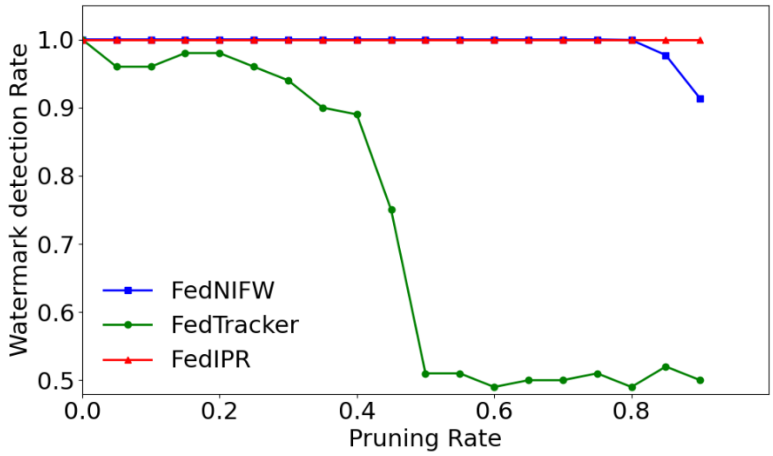


Figure 5.9: The figure illustrates the variations in watermark detection rates across different schemes when subjected to model pruning attacks. Within the federated learning framework and watermark parameter settings, the number of clients is set to 10, the deep network utilized is AlexNet, the dataset is CIFAR10, and the watermark length is 256 bits. Both FedNIFW and FedIPR [31] maintain a watermark detection rate around 100%, whereas the detection success rate of FedTracker [81] declines sharply as the pruning rate increases.

5.9 Conclusion

This chapter presented two complementary designs that bring privacy preservation, verifiable training, and ownership protection to collaborative learning. For multi-client Split Learning, CliCooper rethinks the client–trainer workflow to ensure that neither data nor labels are exposed while the training process remains auditable. The framework employs label expansion to conceal true label semantics and cardinality, thereby preventing unauthorized model use and reducing label leakage. It further applies a Gaussian noise mechanism to the shared smashed activations, mitigating clustering and inversion risks without revealing raw examples or gradients. On top of these privacy controls, dynamic chained watermarks bind consecutive training stages, creating a cryptographically linked trace that supports unified Proof-of-Learning and Proof-of-Ownership. Empirical evaluations indicate that CliCooper retains baseline accuracy, with several models exhibiting modest gains of up to two percentage points, and it significantly strengthens defenses: internal clustering success drops to zero, inversion similarity falls from forty-five percent to approximately 0.03, and surrogate extraction accuracy is reduced to about one percent, which is effectively random guessing.

For Federated Learning, the chapter introduced FedNIFW, a non-interfering fragmented watermarking scheme that overcomes watermark collisions during aggregation. The key idea is to pre-assign client-exclusive embedding regions within designated watermark layers. Each client embeds only in its own region and freezes others during local training. At the server, aggregation proceeds exclusively for watermark regions by directly adopting the corresponding client parameters, while standard FedAvg is applied to non-watermark regions. This segmented design prevents overlapping marks, preserves clear attribution, and scales with both the number of clients and watermark length. Experiments show that FedNIFW resolves cross-client conflicts without degrading global model performance and that it remains robust under common post-processing, including fine-tuning and pruning.

Chapter 6

Conclusions and Future Work

6.1 Conclusion

This dissertation develops a principled foundation for neural models whose provenance is auditable and whose ownership is enforceable under realistic constraints of cost, privacy, and robustness. The central premise is that verification should not be an afterthought applied to a terminal artifact, but a property engineered into both the learning trajectory and the post-training artifact.

To that end, PoLO recasts verification as cryptographic evidence production during training. Watermarks evolve across shards and are bound by hash chains, so challenges issued at arbitrary points certify both training integrity and ownership without replaying optimization or exposing data. The construction aligns security with economic rationality: attempts to forge a plausible history approach the effort of honest training or yield models that fail challenge, thereby making routine audits practical for large-scale systems. This directly addresses **RQ1**–**RQ2** by turning PoL into a chained, tamper-evident challenge–response protocol: each shard's watermark certifies both training effort and provenance, decoupling verifier cost from full retraining and making low-cost forgery economically irrational.

Complementing trajectory-level evidence, NNFMAC provides a non-intrusive post-training attestation mechanism. By extracting critical weights, forming a stable binary fingerprint via median-based binarization, and encoding authorship with an index-based expansion, it verifies both uniqueness and ownership while preserving accuracy and avoiding retraining. The design remains resilient to common post-processing, including fine-tuning, pruning, weight shifting, and weight perturbation, and thus supports procurement, compliance, and marketplace settings where only the final model is available. This answers **RQ3**–**RQ4** by binding ownership without intrusive weight modifications or extra training and by authenticating model uniqueness from critical-weight fingerprints, thereby preserving accuracy while remaining robust to standard post-training edits.

The framework is further generalized to collaborative regimes. In multi-client Split Learning, CliCooper combines label expansion and DP mechanisms to suppress label leakage, clustering, and inversion on shared representations, while chained watermarks provide privacy-preserving proofs of training and ownership. In Federated Learning, FedNIFW resolves watermark collisions by allocating client-exclusive embedding regions and applying exclusive aggregation over watermark layers, thereby eliminating cross-client interference without sacrificing

utility. This resolves **RQ5**–**RQ7**: label expansion plus DP-guarded activations protect raw inputs and labels in multi-client SL (privacy), chained watermarks bind segment provenance for auditable compensation (ownership), and the protocol constrains unauthorized reuse; in FL, FedNIFW further prevents cross-client watermark collisions via client-exclusive layers and exclusive aggregation.

Taken together, these results yield a coherent methodology that integrates trajectory-embedded proofs with performance-neutral post-training attestation and extends both to partially trusted, distributed settings. Remaining challenges include automating shard sizing and challenge scheduling for very large models, tightening formal privacy accounting for split representations, and co-optimizing watermark placement with adaptive aggregation in federated systems. Addressing these directions would further strengthen the reliability of training verification and the enforceability of ownership in next-generation AI deployments.

6.2 Future Works

This dissertation establishes practical mechanisms for verifying training effort and enforcing ownership across centralized and collaborative learning. Looking forward, we aim to move from "working proofs" to deployment-grade verification under real operational constraints—federation, scale, privacy, and economic incentives. To make the roadmap actionable, we outline concrete next steps grouped by PoO and PoL, followed by a unifying research thread that connects them in multi-party pipelines.

PoO — Aggregation-robust ownership & non-intrusive fingerprinting.

- *Federated Learning ownership that survives aggregation and enables attribution.* Current watermarks are fragile under FedAvg/FedOpt and cannot identify which client leaked a global model. We will design aggregation-invariant, client-attributable watermarks: signals whose statistics are preserved by convex combinations and that embed client-exclusive components to support post-hoc leakage attribution. This includes (i) allocation of ownership carriers to layers that are stable under aggregation, (ii) per-client pseudo-orthogonal codes with bounded cross-correlation so detection remains reliable after averaging, and (iii) compatibility with secure aggregation and partial participation. Benchmarks will measure detectability across participation rates, aggregation rules, and post-training edits.
- *Task-agnostic, lightweight model fingerprinting without weight edits.* To avoid any training interference, we will pursue non-intrusive fingerprints derived from intrinsic model traits—e.g., weight-space spectra, inter-layer correlation graphs, activation geometry, or lottery-ticket-style saliency patterns—combined with redundancy/error-correcting layouts for robust decoding. The targets are: (i) zero retraining, (ii) minutes-level verification, (iii) generalization across tasks/architectures, and (iv) calibrated false-accept/false-reject bounds. We will compare against embed-based watermarks on accuracy impact, verification latency, and resilience to pruning, quantization, LoRA fine-tuning, and model merging.

PoL — Lightweight verification for large models & compatibility with unlearning.

- *Large Language Model (LLM)-scale, retrain-free* PoL. Retrain-based PoL is impractical for modern LLMs. We will extend the chained-evidence view to parameter-efficient regimes: derive shard-level commitments over adapter/LoRA states, optimizer moments,

or sparse update masks; use per-stage hash links and sampling-based challenges to verify training effort without full replay. Protocol design will target sub-5% of the original training cost, support mixed-precision and distributed training, and include failure modes for non-deterministic hardware.

- PoL *under unlearning and privacy constraints.* Practical systems must retract the influence of specific data while maintaining auditability. We will study PoL-compatible unlearning, where proofs are revocable or updatable at fine granularity: proofs carry data-segment scopes; unlearning triggers proof re-issuance with verifiable deltas. We will jointly model privacy budgets (DP), unlearning guarantees, and verifier soundness, and explore zero-knowledge variants that reveal correctness without exposing raw trajectories.

Unifying PoL + PoO – Composable provenance and ownership in multi-stage, multi-party pipelines. Real workflows span pretraining, multi-round fine-tuning, cross-organization collaboration, and potential ownership transfers. We will develop a composable ledger of training and ownership: per-stage PoL commitments and PoO attestations that can be concatenated, subsetted, or transferred while preserving end-to-end soundness. Key tasks include: (i) formal composition rules and security proofs under model hand-offs; (ii) mechanisms for ownership transfer that keep prior contributions auditable; (iii) interfaces for marketplaces and contract settlement; and (iv) implementation that coexists with federated aggregation and split-learning activation relays. This unified direction aims to make provenance and ownership portable artifacts—verifiable across stages, parties, and deployment settings.

Chapter 7

Publication List

7.1 First-author Paper

1. **Haiyu Deng**, et al., "NNFMAC: A Neural Network Fingerprinting-based Model Authentication Code Scheme," ACM Transactions on Multimedia Computing, Communications, and Applications, 2025.

2. **Haiyu Deng**, et al., "FedNIFW: Non-Interfering Fragmented Watermarking for Federated Deep Neural Network," IEEE 23rd International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom), 2024.

3. **Haiyu Deng**, et al., "A novel weights-less watermark embedding method for neural network models," IEEE 22nd International Symposium on Communications and Information Technologies (ISCIT), 2023.

7.2 Co-author Papers

1. Yanna Jiang*, and **Haiyu Deng***, et al., "SoK: Credential-Based Trust Management in Decentralized Ledger Systems," IEEE 24th International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom), 2025.

2. Xiaocui, Dang, and **Haiyu Deng**, et al., "A Dual Defense Design Against Data Poisoning Attacks in Deep Learning-Based Recommendation Systems," IEEE 23rd International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom), 2024.

3. Xiaocui, Dang, and **Haiyu Deng**, et al., "A Novel Scheme for Recommendation Unlearning Verification (RUV) Using Non-influential Trigger Data," IEEE Consumer Communications & Networking Conference, 2025.

4. Xiaocui, Dang, and **Haiyu Deng**, et al., "Recommendation System Model Ownership Verification via Non-influential Watermarking,", IEEE 17th International Conference on Security of Information and Networks, 2024.

7.3 Under Review

7.3.1 First-author Paper

1. **Haiyu Deng**, et al., "PoLO: Proof-of-Learning and Proof-of-Ownership at Once with Chained Watermarking," submitted to ACM Conference on Computer and Communications Security (CCS), under review.

2. **Haiyu Deng**, et al., "Client-Cooperative Split Learning," IEEE Transactions on Service Computing (TSC), under review.

7.3.2 Co-author Paper

1. Xiaocui, Dang, and **Haiyu Deng**, et al., "Robust and Adaptive Dual-Defense Against Data Poisoning Attacks in Recommendation Systems,", Future Generation Computer Systems, under review.

Bibliography

[1] Z. Wang, Q. She, and T. E. Ward, "Generative adversarial networks in computer vision: A survey and taxonomy," *ACM Comput. Surv.*, vol. 54, no. 2, Feb. 2021, ISSN: 0360-0300. DOI: 10.1145/3439723. [Online]. Available: https://doi.org/10.1145/3439723.

[2] S. Chen, Y. Zhang, and Q. Yang, "Multi-task learning in natural language processing: An overview," *ACM Comput. Surv.*, vol. 56, no. 12, Jul. 2024, ISSN: 0360-0300. DOI: 10.1145/3663363. [Online]. Available: https://doi.org/10.1145/3663363.

[3] C. Wang et al., "Arobert: An asr robust pre-trained language model for spoken language understanding," *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, vol. 30, pp. 1207–1218, 2022.

[4] M. Zhang and J. Li, "A commentary of gpt-3 in mit technology review 2021," *Fundamental Research*, vol. 1, no. 6, pp. 831–833, 2021.

[5] S. Markidis, S. W. Der Chien, E. Laure, I. B. Peng, and J. S. Vetter, "Nvidia tensor core programmability, performance & precision," in *IEEE International Parallel and Distributed Processing Symposium (IPDPS) Workshops*, IEEE.

[6] N. P. Jouppi et al., "In-datacenter performance analysis of a tensor processing unit," in *Proceedings of the 44th annual international symposium on computer architecture*, 2017, pp. 1–12.

[7] A. Putnam et al., "A reconfigurable fabric for accelerating large-scale datacenter services," *ACM SIGARCH Computer Architecture News*, vol. 42, no. 3, pp. 13–24, 2014.

[8] P. Blanchard, E. M. El Mhamdi, R. Guerraoui, and J. Stainer, "Machine learning with adversaries: Byzantine tolerant gradient descent," *Advances in neural information processing systems*, vol. 30, 2017.

[9] Q. Wang, G. Yu, Y. Sai, H. D. Bandara, and S. Chen, "Is your ai truly yours? leveraging blockchain for copyrights, provenance, and lineage," *IEEE Transactions on Services Computing (TSC)*, pp. 1–19, 2025.

[10] Y. Uchida, Y. Nagai, S. Sakazawa, and S. Satoh, "Embedding watermarks into deep neural networks," in *International Conference on Multimedia Retrieval (ICML)*, 2017, pp. 269–277.

[11] X. Cao, J. Jia, and N. Z. Gong, "Ipguard: Protecting intellectual property of deep neural networks via fingerprinting the classification boundary," in *Proceedings of the 2021 ACM Asia Conference on Computer and Communications Security*, ser. ASIA CCS '21, Virtual Event, Hong Kong: Association for Computing Machinery, 2021, pp. 14–25, ISBN: 9781450382878. DOI: 10.1145/3433210.3437526.

[12] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas, "Communication-efficient learning of deep networks from decentralized data," in *Artificial intelligence and statistics*, PMLR, 2017, pp. 1273–1282.

[13] P. Vepakomma, O. Gupta, T. Swedish, and R. Raskar, "Split learning for health: Distributed deep learning without sharing raw patient data," *arXiv preprint arXiv:1812.00564*, 2018.

[14] J. Weng, J. Weng, C. Cai, H. Huang, and C. Wang, "Golden grain: Building a secure and decentralized model marketplace for mlaas," *IEEE Transactions on Dependable and Secure Computing*, vol. 19, no. 5, pp. 3149–3167, 2022. DOI: 10.1109/TDSC.2021.3085988.

[15] Z. Ye et al., "Deep learning workload scheduling in gpu datacenters: A survey," *ACM Computing Surveys (CSUR)*, vol. 56, no. 6, pp. 1–38, 2024.

[16] L. Zhao, Q. Wang, C. Wang, Q. Li, C. Shen, and B. Feng, "Veriml: Enabling integrity assurances and fair payments for machine learning as a service," *IEEE Transactions on Parallel and Distributed Systems (TPDS)*, 2021.

[17] H. Jia et al., "Proof-of-learning: Definitions and practice," in *IEEE Symposium on Security and Privacy (SP)*, IEEE, 2021, pp. 1039–1056.

[18] K. Abbaszadeh, C. Pappas, J. Katz, and D. Papadopoulos, "Zero-knowledge proofs of training for deep neural networks," in *ACM SIGSAC Conference on Computer and Communications Security (CCS)*, 2024, pp. 4316–4330.

[19] Z. Zhao, Z. Fang, X. Wang, X. Chen, and Y. Zhou, "Proof-of-learning with incentive security," *arXiv preprint arXiv:2404.09005*, 2024.

[20] J. Weng, J. Weng, C. Cai, H. Huang, and C. Wang, "Golden grain: Building a secure and decentralized model marketplace for mlaas," *IEEE Transactions on Dependable and Secure Computing (TDSC)*, vol. 19, no. 5, pp. 3149–3167, 2022.

[21] P. Sun, X. Chen, G. Liao, and J. Huang, "A profit-maximizing model marketplace with differentially private federated learning," in *IEEE Conference on Computer Communications (INFOCOM)*, 2022, pp. 1439–1448.

[22] U. C. Office. "Section 512 of title 17: Limitations on liability relating to material online (dmca safe harbor)." Official DMCA §512 guidance, Accessed: Sep. 14, 2025. [Online]. Available: https://www.copyright.gov/512/.

[23] J. Liu, R. Zhang, S. Szyller, K. Ren, and N. Asokan, "False claims against model ownership resolution," in *33rd USENIX Security Symposium (USENIX Security 24)*, 2024, pp. 6885–6902.

[24] E. Commission. "Eu rules for general-purpose ai models start to apply, bringing more transparency, safety and accountability," Accessed: Sep. 12, 2025. [Online]. Available: https://digital-strategy.ec.europa.eu/en/news/eu-rules-general-purpose-ai-models-start-apply-bringing-more-transparency-safety-and-accountability.

[25] M. Spoczynski, M. S. Melara, and S. Szyller, "Atlas: A framework for ml lifecycle provenance & transparency," *arXiv preprint arXiv:2502.19567*, 2025.

[26] Y. Xie et al., "Dataset ownership verification in contrastive pre-trained models," *International Conference on Learning Representations (ICLR)*, 2025.

[27] A. Dziedzic, M. A. Kaleem, Y. S. Lu, and N. Papernot, "Increasing the cost of model extraction with calibrated proof of work," *arXiv preprint arXiv:2201.09243*, 2022.

[28] M. G. Poirot, P. Vepakomma, K. Chang, J. Kalpathy-Cramer, R. Gupta, and R. Raskar, "Split learning for collaborative deep learning in healthcare," *arXiv preprint arXiv:1912.12115*, 2019.

[29] Y. Matsubara, M. Levorato, and F. Restuccia, "Split computing and early exiting for deep learning applications: Survey and research challenges," *ACM Computing Surveys (CSUR)*, vol. 55, no. 5, pp. 1–30, 2022.

[30] Q. Yang, Y. Liu, T. Chen, and Y. Tong, "Federated machine learning: Concept and applications," *ACM Trans. Intell. Syst. Technol.*, vol. 10, no. 2, Jan. 2019, ISSN: 2157-6904. DOI: 10.1145/3298981. [Online]. Available: https://doi.org/10.1145/3298981.

[31] B. Li, L. Fan, H. Gu, J. Li, and Q. Yang, "Fedipr: Ownership verification for federated deep neural network models," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 45, no. 4, pp. 4521–4536, 2023. DOI: 10.1109/TPAMI.2022.3195956.

[32] G. Yu et al., "Ironforge: An open, secure, fair, decentralized federated learning," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 36, no. 1, pp. 354–368, 2025. DOI: 10.1109/TNNLS.2023.3329249.

[33] S. Idowu, D. Strüber, and T. Berger, "Asset management in machine learning: State-of-research and state-of-practice," *ACM Computing Surveys*, vol. 55, no. 7, pp. 1–35, 2022.

[34] J. Verbraeken, M. Wolting, J. Katzy, J. Kloppenburg, T. Verbelen, and J. S. Rellermeyer, "A survey on distributed machine learning," *ACM Computing Surveys*, vol. 53, no. 2, pp. 1–33, 2020.

[35] R. Zhang, J. Liu, Y. Ding, Z. Wang, Q. Wu, and K. Ren, ""adversarial examples" for proof-of-learning," in *IEEE Symposium on Security and Privacy (SP)*, 2022.

[36] A. Thudi, H. Jia, I. Shumailov, and N. Papernot, "On the necessity of auditable algorithmic definitions for machine unlearning," in *31st USENIX Security Symposium (USENIX Security 22)*, Boston, MA: USENIX Association, Aug. 2022, pp. 4007–4022, ISBN: 978-1-939133-31-1.

[37] C. Fang et al., "Proof-of-learning is currently more broken than you think," in *IEEE European Symposium on Security and Privacy (EuroSP)*, 2023.

[38] T. Wang and F. Kerschbaum, "Attacks on digital watermarks for deep neural networks," in *ICASSP 2019*, 2019, pp. 2622–2626. DOI: 10.1109/ICASSP.2019.8682202.

[39] O. Li et al., "Label leakage and protection in two-party split learning," in *International Conference on Learning Representations (ICLR)*, 2022.

[40] F. Yu, L. Wang, B. Zeng, K. Zhao, T. Wu, and Z. Pang, "Sia: A sustainable inference attack framework in split learning," *Neural Networks*, vol. 171, pp. 396–409, 2024.

[41] J. Liu, X. Lyu, Q. Cui, and X. Tao, "Similarity-based label inference attack against training and inference of split learning," *IEEE Transactions on Information Forensics and Security (TIFS)*, vol. 19, pp. 2881–2895, 2024.

[42] K. Zhao, X. Chuo, F. Yu, B. Zeng, Z. Pang, and L. Wang, "Splitaum: Auxiliary model-based label inference attack against split learning," *IEEE Transactions on Network and Service Management (TNSE)*, 2024.

[43] Z. Lin et al., "Adversarial example based fingerprinting for robust copyright protection in split learning," *arXiv preprint arXiv:2503.04825*, 2025.

[44] J. Wang, H. Wu, X. Zhang, and Y. Yao, "Watermarking in deep neural networks via error back-propagation," *Electronic Imaging*, vol. 32, pp. 1–9, 2020.

[45] T. Wang and F. Kerschbaum, "Riga: Covert and robust white-box watermarking of deep neural networks," in *Proceedings of the Web Conference (WWW)*, 2021.

[46] M. Kuribayashi, T. Tanaka, S. Suzuki, T. Yasui, and N. Funabiki, "White-box watermarking scheme for fully-connected layers in fine-tuning model," in *Proceedings of the 2021 ACM Workshop on Information Hiding and Multimedia Security*, 2021, pp. 165–170. DOI: 10.1145/3437880.3460402.

[47] P. Lv et al., "A robustness-assured white-box watermark in neural networks," *IEEE Transactions on Dependable and Secure Computing (TDSC)*, 2023.

[48] H. Chen, B. D. Rouhani, C. Fu, J. Zhao, and F. Koushanfar, "Deepmarks: A secure fingerprinting framework for digital rights management of deep learning models," in *International Conference on Multimedia Retrieval (ICML)*, 2019, pp. 105–113.

[49] X. Pan, M. Zhang, Y. Lu, and M. Yang, "Tafa: A task-agnostic fingerprinting algorithm for neural networks," in *Computer Security–ESORICS 2021: 26th European Symposium on Research in Computer Security, Darmstadt, Germany, October 4–8, 2021, Proceedings, Part I 26*, Springer, 2021, pp. 542–562. DOI: 10.1007/978-3-030-88418-5_26.

[50] M. Xue et al., "An explainable intellectual property protection method for deep neural networks based on intrinsic features," *IEEE Transactions on Artificial Intelligence*, vol. 5, no. 9, pp. 4649–4659, 2024. DOI: 10.1109/TAI.2024.3388389.

[51] X. Pan, Y. Yan, M. Zhang, and M. Yang, "Metav: A meta-verifier approach to task-agnostic model fingerprinting," in *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, ser. KDD '22, Washington DC, USA: Association for Computing Machinery, 2022, pp. 1327–1336, ISBN: 9781450393850. DOI: 10.1145/3534678.3539257.

[52] E. Erdoğan, A. Küpçü, and A. E. Çiçek, "Unsplit: Data-oblivious model inversion, model stealing, and label inference attacks against split learning," in *Proceedings of the Workshop on Privacy in the Electronic Society*, 2022, pp. 115–124.

[53] X. Xu et al., "A stealthy wrongdoer: Feature-oriented reconstruction attack against split learning," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2024, pp. 12 130–12 139.

[54] D. K. Mahto, A. K. Singh, K. N. Singh, O. P. Singh, and A. K. Agrawal, "Robust copyright protection technique with high-embedding capacity for color images," *ACM Trans. Multimedia Comput. Commun. Appl.*, vol. 20, no. 11, Sep. 2024, ISSN: 1551-6857. DOI: 10.1145/3580502. [Online]. Available: https://doi.org/10.1145/3580502.

[55] Z. Ji, Q. Hu, Y. Zheng, L. Xiang, and X. Wang, "A principled approach to natural language watermarking," in *Proceedings of the 32nd ACM International Conference on Multimedia*, ser. MM '24, Melbourne VIC, Australia: Association for Computing Machinery, 2024, pp. 2908–2916, ISBN: 9798400706868. DOI: 10.1145/3664647.3681544. [Online]. Available: https://doi.org/10.1145/3664647.3681544.

[56] Y. Liu, J. Liu, A. Argyriou, S. Ma, L. Wang, and Z. Xu, "360-degree vr video watermarking based on spherical wavelet transform," *ACM Trans. Multimedia Comput. Commun. Appl.*, vol. 17, no. 1, Apr. 2021, ISSN: 1551-6857. DOI: 10.1145/3425605. [Online]. Available: https://doi.org/10.1145/3425605.

[57] H. Chen et al., "Perceptual hashing of deep convolutional neural networks for model copy detection," *ACM Transactions on Multimedia Computing, Communications and Applications (TOMM)*, vol. 19, no. 3, pp. 1–20, 2023.

[58] X. You, Y. Jiang, J. Xu, M. Zhang, and M. Yang, "Gnnfingers: A fingerprinting framework for verifying ownerships of graph neural networks," in *Proceedings of the ACM Web Conference 2024*, ser. WWW '24, Singapore, Singapore: Association for Computing Machinery, 2024, pp. 652–663, ISBN: 9798400701719. DOI: 10.1145/3589334.3645489.

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

[60] P. Gaži, A. Kiayias, and D. Zindros, "Proof-of-stake sidechains," in *IEEE Symposium on Security and Privacy (SP)*, 2019, pp. 139–156. DOI: 10.1109/SP.2019.00040.

[61] H. Wang, G. Chen, Y. Zhang, and Z. Lin, "Multi-certificate attacks against proof-of-elapsed-time and their countermeasures.," in *Network and Distributed System Security (NDSS) Symposium*, 2022.

[62] T. Moran and I. Orlov, "Simple proofs of space-time and rational proofs of storage," in *Annual International Cryptology Conference (CRYPTO)*, 2019, pp. 381–409.

[63] Q. Wang, R. Li, Q. Wang, S. Chen, and Y. Xiang, "Exploring unfairness on proof of authority: Order manipulation attacks and remedies," in *ACM on Asia Conference on Computer and Communications Security (AsiaCCS)*, 2022, pp. 123–137.

[64] G. Tan, A. Gascón, S. Meiklejohn, M. Raykova, X. Wang, and N. Luo, "Founding zero-knowledge proofs of training on optimum vicinity," *Cryptology ePrint Archive*, 2025.

[65] A. S. Shamsabadi et al., "Confidential-profitt: Confidential proof of fair training of trees," in *The Eleventh International Conference on Learning Representations*, 2022.

[66] S. Shao, Y. Li, H. Yao, Y. He, Z. Qin, and K. Ren, "Explanation as a watermark: Towards harmless and multi-bit model ownership verification via watermarking feature attribution," *Network and Distributed System Security (NDSS) Symposium*, 2025.

[67] H. Liu, Z. Weng, and Y. Zhu, "Watermarking deep neural networks with greedy residuals.," in *ICML*, 2021, pp. 6978–6988.

[68] N. Lukas, E. Jiang, X. Li, and F. Kerschbaum, "Sok: How robust is image classification deep neural network watermarking?" In *2022 IEEE Symposium on Security and Privacy (SP)*, 2022, pp. 787–804. DOI: 10.1109/SP46214.2022.9833693.

[69] L. Hu, T. Zhou, Z. Liu, F. Liu, and Z. Cai, "Split learning on segmented healthcare data," *IEEE Transactions on Big Data (TBD)*, 2025.

[70] O. Gupta and R. Raskar, "Distributed learning of deep neural network over multiple agents," *Journal of Network and Computer Applications*, vol. 116, pp. 1–8, 2018.

[71] L. Wang, H. Chen, L. Zuo, and H. Liu, "U-shaped vertical split learning with local differential privacy for privacy preserving," in *International Conference on Intelligent Computing*, Springer, 2024, pp. 72–81.

[72] S. Lyu, Z. Lin, G. Qu, X. Chen, X. Huang, and P. Li, "Optimal resource allocation for u-shaped parallel split learning," in *IEEE Globecom Workshops (GC Wkshps)*, IEEE, 2023, pp. 197–202.

[73] S. Abuadbba et al., "Can we use split learning on 1d cnn models for privacy preserving training?" In *ACM Asia Conference on Computer and Communications Security (AsiaCCS)*, 2020, pp. 305–318.

[74] B. Balle, G. Cherubin, and J. Hayes, "Reconstructing training data with informed adversaries," in *2022 IEEE Symposium on Security and Privacy (SP)*, IEEE, 2022, pp. 1138–1156.

[75] D. Pasquini, G. Ateniese, and M. Bernaschi, "Unleashing the tiger: Inference attacks on split learning," in *ACM SIGSAC Conference on Computer and Communications Security (CCS)*, 2021, pp. 2113–2129.

[76] N. D. Pham, A. Abuadbba, Y. Gao, K. T. Phan, and N. Chilamkurti, "Binarizing split learning for data privacy enhancement and computation reduction," *IEEE Transactions on Information Forensics and Security (TIFS)*, vol. 18, pp. 3088–3100, 2023.

[77] Y. Mao, Z. Xin, Z. Li, J. Hong, Q. Yang, and S. Zhong, "Secure split learning against property inference, data reconstruction, and feature space hijacking attacks," in *European Symposium on Research in Computer Security (ESORICS)*, Springer, 2023, pp. 23–43.

[78] L. T. Phong, Y. Aono, T. Hayashi, L. Wang, and S. Moriai, "Privacy-preserving deep learning via additively homomorphic encryption," *IEEE Transactions on Information Forensics and Security*, vol. 13, no. 5, pp. 1333–1345, 2018. DOI: 10.1109/TIFS.2017.2787987.

[79] B. G. A. Tekgul, Y. Xia, S. Marchal, and N. Asokan, "Waffle: Watermarking in federated learning," in *2021 40th International Symposium on Reliable Distributed Systems (SRDS)*, 2021, pp. 310–320. DOI: 10.1109/SRDS53918.2021.00038.

[80] Y. Adi, C. Baum, M. Cisse, B. Pinkas, and J. Keshet, "Turning your weakness into a strength: Watermarking deep neural networks by backdooring," in *USENIX Security Symposium (USENIX Sec)*, 2018.

[81] S. Shao, W. Yang, H. Gu, Z. Qin, L. Fan, and Q. Yang, "Fedtracker: Furnishing ownership verification and traceability for federated learning model," *IEEE Transactions on Dependable and Secure Computing*, pp. 1–18, 2024. DOI: 10.1109/TDSC.2024.3390761.

[82] J. Kirkpatrick et al., "Overcoming catastrophic forgetting in neural networks," *Proceedings of the national academy of sciences*, vol. 114, no. 13, pp. 3521–3526, 2017.

[83] X. Liu, S. Shao, Y. Yang, K. Wu, W. Yang, and H. Fang, "Secure federated learning model verification: A client-side backdoor triggered watermarking scheme," in *2021 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, 2021, pp. 2414–2419. DOI: 10.1109/SMC52423.2021.9658998.

[84] W. Yang et al., "Watermarking in secure federated learning: A verification framework based on client-side backdooring," *ACM Trans. Intell. Syst. Technol.*, vol. 15, no. 1, Dec. 2023, ISSN: 2157-6904. DOI: 10.1145/3630636. [Online]. Available: https://doi.org/10.1145/3630636.

[85] G. Yu et al., "Split unlearning," *ACM SIGSAC Conference on Computer and Communications Security (CCS)*, 2025.

[86] C. Reed, "Information ownership in the cloud," 2021.

[87] S. Sahoo and R. Halder, "Traceability and ownership claim of data on big data marketplace using blockchain technology," *Journal of Information and Telecommunication*, vol. 5, no. 1, pp. 35–61, 2021.

[88] E. Commission. "Digital services act (dsa)." Policy overview page, Accessed: Sep. 14, 2025. [Online]. Available: https://commission.europa.eu/strategy-and-policy/priorities-2019-2024/europe-fit-digital-age/digital-services-act_en.

[89] E. Union. "Artificial intelligence act, article 53 — transparency obligations for general-purpose ai models." Consolidated text, Accessed: Sep. 12, 2025. [Online]. Available: https://artificialintelligenceact.eu/article/53/.

[90] Y. Yan, X. Pan, M. Zhang, and M. Yang, "Rethinking white-box watermarks on deep learning models under neural structural obfuscation," in *USENIX Security Symposium (USENIX Sec)*, 2023, pp. 2347–2364.

[91] S. Ranjbar Alvar, M. Akbari, D. (X. Yue, and Y. Zhang, "Nft-based data marketplace with digital watermarking," in *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, ser. KDD '23, Long Beach, CA, USA: Association for Computing Machinery, 2023, pp. 4756–4767, ISBN: 9798400701030. DOI: 10.1145/3580305.3599876. [Online]. Available: https://doi.org/10.1145/3580305.3599876.

[92] A. Pegoraro, C. Segna, K. Kumari, and A.-R. Sadeghi, "Deepeclipse: How to break white-box dnn-watermarking schemes," in *33rd USENIX Security Symposium (USENIX Security 24)*, 2024, pp. 5287–5304.

[93] W. Gu, "Watermark removal scheme based on neural network model pruning," in *International Conference on Machine Learning and Natural Language Processing (MLNLP)*, 2023, pp. 377–382.

[94] W. Zong, Y.-W. Chow, W. Susilo, J. Baek, J. Kim, and S. Camtepe, "Ipremover: A generative model inversion attack against deep neural network fingerprinting and wa-

termarking," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 38, 2024, pp. 7837–7845.

[95] A. Kassis and U. Hengartner, "Unmarker: A universal attack on defensive image watermarking," in *2025 IEEE Symposium on Security and Privacy (SP)*, vol. 2, 2025, p. 8.

[96] W. Wei and L. Liu, "Gradient leakage attack resilient deep learning," *IEEE Transactions on Information Forensics and Security (TIFS)*, vol. 17, pp. 303–316, 2022.

[97] L. Du et al., "Sok: Dataset copyright auditing in machine learning systems," in *2025 IEEE Symposium on Security and Privacy (SP)*, IEEE Computer Society, 2024, pp. 25–25.

[98] X. Chen et al., "Refit: A unified watermark removal framework for deep learning systems with limited data," in *ACM Asia Conference on Computer and Communications Security (AsiaCCS)*, 2021, pp. 321–335.

[99] N. C. Luong, P. Wang, D. Niyato, Y. Wen, and Z. Han, "Resource management in cloud networking using economic analysis and pricing models: A survey," *IEEE Communications Surveys & Tutorials*, vol. 19, no. 2, pp. 954–1001, 2017. DOI: 10.1109/COMST.2017.2647981.

[100] X. Wu, F. Huang, Z. Hu, and H. Huang, "Faster adaptive federated learning," in *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, vol. 37, 2023, pp. 10 379–10 387.

[101] Y. Jiang et al., "Blockchained federated learning for internet of things: A comprehensive survey," *ACM Computing Surveys*, vol. 56, no. 10, pp. 1–37, 2024.

[102] X. Liu et al., "Blockful: Enabling unlearning in blockchained federated learning," *IEEE Transactions on Information Forensics and Security*, vol. 20, pp. 6635–6650, 2025. DOI: 10.1109/TIFS.2025.3583109.

[103] H. Deng et al., "Polo: Proof-of-learning and proof-of-ownership at once with chained watermarking," *arXiv preprint arXiv:2505.12296*, 2025.

[104] Z. Li, C. Wang, S. Wang, and C. Gao, "Protecting intellectual property of large language model-based code generation apis via watermarks," in *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '23, Copenhagen, Denmark: Association for Computing Machinery, 2023, pp. 2336–2350, ISBN: 9798400700507. DOI: 10.1145/3576915.3623120. [Online]. Available: https://doi.org/10.1145/3576915.3623120.

[105] F.-Y. Chen and L.-H. Yen, "Incentive-aware resource allocation for multiple model owners in federated learning," *IEEE Transactions on Services Computing*, vol. 17, no. 2, pp. 549–562, 2024. DOI: 10.1109/TSC.2024.3376259.

[106] Z. Wang, Q. Wang, G. Yu, and S. Chen, "Tdml—a trustworthy distributed machine learning framework," *Future Generation Computer Systems*, vol. 174, p. 107 951, 2026.

[107] F. Boenisch, "A systematic review on model watermarking for neural networks," *Frontiers in big Data*, vol. 4, p. 729 663, 2021. DOI: 10.3389/fdata.2021.729663.

[108] S. Han, J. Pool, J. Tran, and W. Dally, "Learning both weights and connections for efficient neural network," *Advances in neural information processing systems*, vol. 28, 2015. DOI: 10.48550/arxiv.1506.02626.

[109] Z. Zhu et al., "Fvw: Finding valuable weight on deep neural network for model pruning," in *Proceedings of the 32nd ACM International Conference on Information and Knowledge Management*, ser. CIKM '23, Birmingham, United Kingdom: Association for Computing Machinery, 2023, pp. 3657–3666, ISBN: 9798400701245. DOI: 10.1145/3583780.3614889. [Online]. Available: https://doi.org/10.1145/3583780.3614889.

[110] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 770–778.

[111] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," in *3rd International Conference on Learning Representations (ICLR 2015)*, Computational and Biological Learning Society, 2015.

[112] A. Dosovitskiy, "An image is worth 16x16 words: Transformers for image recognition at scale," *arXiv preprint arXiv:2010.11929*, 2020.

[113] A. Choromanska, M. Henaff, M. Mathieu, G. B. Arous, and Y. LeCun, "The loss surfaces of multilayer networks," in *Artificial intelligence and statistics*, PMLR, 2015, pp. 192–204. DOI: 10.48550/arxiv.1412.0233.

[114] D. Jung, J. Lee, H. Jeong, D. Cha, H. Kim, and S. Pack, "Split computing for mobile devices: Energy and latency perspective," *IEEE Transactions on Services Computing (TSC)*, 2025.

[115] F. Zafari, K. K. Leung, D. Towsley, P. Basu, A. Swami, and J. Li, "Let's share: A game-theoretic framework for resource sharing in mobile edge clouds," *IEEE Transactions on Network and Service Management (TNSE)*, vol. 18, no. 2, pp. 2107–2122, 2020.

[116] Z. Lin, G. Qu, X. Chen, and K. Huang, "Split learning in 6g edge networks," *IEEE Wireless Communications*, vol. 31, no. 4, pp. 170–176, 2024.

[117] B. S. Rawal, L. Berman, and H. Ramcharan, "Multi-client/multi-server split architecture," in *The International Conference on Information Networking 2013 (ICOIN)*, IEEE, 2013, pp. 696–701.

[118] Z. Li et al., "Split learning for distributed collaborative training of deep learning models in health informatics," in *AMIA Annual Symposium Proceedings*, vol. 2023, 2024, p. 1047.

[119] G. S. Hukkeri, R. Goudar, G. Dhananjaya, V. N. Rathod, and S. Ankalaki, "Split-fed learning: A deep dive into methods, innovations and future prospects for data privacy and efficiency in decentralized machine learning," *IEEE Access*, 2025.

[120] A. Hammoud, H. Otrok, A. Mourad, and Z. Dziong, "On demand fog federations for horizontal federated learning in iov," *IEEE Transactions on Network and Service Management (TNSM)*, vol. 19, no. 3, pp. 3062–3075, 2022.

[121] Z. Wang et al., "A comprehensive survey on data augmentation," *IEEE Transactions on Knowledge and Data Engineering*, pp. 1–20, 2025. DOI: 10.1109/TKDE.2025.3622600.

[122] M. Bayer, M.-A. Kaufhold, and C. Reuter, "A survey on data augmentation for text classification," *ACM Computing Surveys*, vol. 55, no. 7, pp. 1–39, 2022.

[123] A. Kassis and U. Hengartner, "Unmarker: A universal attack on defensive image watermarking," in *IEEE Symposium on Security and Privacy (SP)*, IEEE, 2025, pp. 2602–2620.

[124] W. Zong, Y.-W. Chow, W. Susilo, J. Baek, J. Kim, and S. Camtepe, "Ipremover: A generative model inversion attack against deep neural network fingerprinting and watermarking," in *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, vol. 38, 2024, pp. 7837–7845.

[125] Y. Ren et al., "Deep clustering: A comprehensive survey," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 36, no. 4, pp. 5858–5878, 2025. DOI: 10.1109/TNNLS.2024.3403155.

[126] A. M. Ikotun, A. E. Ezugwu, L. Abualigah, B. Abuhaija, and J. Heming, "K-means clustering algorithms: A comprehensive review, variants analysis, and advances in the era of big data," *Information Sciences*, vol. 622, pp. 178–210, 2023.

[127] B. Cao et al., "Performance analysis and comparison of PoW, PoS and DAG based blockchains," *Digital Communications and Networks*, vol. 6, no. 4, pp. 480–485, 2020.

[128] J. Liang, R. Pang, C. Li, and T. Wang, "Model extraction attacks revisited," in *Proceedings of the ACM Asia Conference on Computer and Communications Security (AsiaCCS)*, 2024, pp. 1231–1245.

[129] W. Jiang, H. Li, G. Xu, T. Zhang, and R. Lu, "A comprehensive defense framework against model extraction attacks," *IEEE Transactions on Dependable and Secure Computing (TDSC)*, vol. 21, no. 2, pp. 685–700, 2024.

[130] N. Phan, X. Wu, H. Hu, and D. Dou, "Adaptive laplace mechanism: Differential privacy preservation in deep learning," in *IEEE International Conference on Data Mining (ICDM)*, IEEE, 2017, pp. 385–394.

[131] J. Weng, J. Weng, C. Cai, H. Huang, and C. Wang, "Golden grain: Building a secure and decentralized model marketplace for mlaas," *IEEE Transactions on Dependable and Secure Computing (TDSC)*, vol. 19, no. 5, pp. 3149–3167, 2021.

[132] A. Lang and E. Schubert, "Betula: Fast clustering of large data with improved birch cf-trees," *Information systems*, vol. 108, p. 101 918, 2022.

[133] H. Xu and N. Pham, "Scalable dbscan with random projections," *Advances in Neural Information Processing Systems*, vol. 37, pp. 27 978–28 008, 2024.

[134] P. Lv et al., "Mea-defender: A robust watermark against model extraction attack," in *IEEE Symposium on Security and Privacy (SP)*, IEEE, 2024, pp. 2515–2533.

[135] Y. Jiang et al., "Preventing harm to the rare in combating the malicious: A filtering-and-voting framework with adaptive aggregation in federated learning," *Neurocomputing*, vol. 604, p. 128 317, 2024.

[136] M. Xue, Y. Zhang, J. Wang, and W. Liu, "Intellectual property protection for deep learning models: Taxonomy, methods, attacks, and evaluations," *IEEE Transactions on Artificial Intelligence*, vol. 3, no. 6, pp. 908–923, 2022. DOI: 10.1109/TAI.2021.3133824.

[137] M. Cao, L. Zhang, and B. Cao, "Toward on-device federated learning: A direct acyclic graph-based blockchain approach," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 34, no. 4, pp. 2028–2042, 2023. DOI: 10.1109/TNNLS.2021.3105810.

[138] Y. Li et al., "Direct acyclic graph-based ledger for internet of things: Performance and security analysis," *IEEE/ACM Transactions on Networking*, vol. 28, no. 4, pp. 1643–1656, 2020. DOI: 10.1109/TNET.2020.2991994.

[139] W. Liu, B. Cao, M. Peng, and B. Li, "Distributed and parallel blockchain: Towards a multi-chain system with enhanced security," *IEEE Transactions on Dependable and Secure Computing*, pp. 1–16, 2024. DOI: 10.1109/TDSC.2024.3417531.

[140] B. Cao et al., "Blockchain systems, technologies, and applications: A methodology perspective," *IEEE Communications Surveys I& Tutorials*, vol. 25, no. 1, pp. 353–385, 2023. DOI: 10.1109/COMST.2022.3204702.

[141] Y. Lao, P. Yang, W. Zhao, and P. Li, "Identification for deep neural network: Simply adjusting few weights!" In *2022 IEEE 38th International Conference on Data Engineering (ICDE)*, 2022, pp. 1328–1341. DOI: 10.1109/ICDE53745.2022.00104.

[142] A. Bansal et al., "Certified neural network watermarks with randomized smoothing," in *International Conference on Machine Learning*, PMLR, 2022, pp. 1450–1465.

[143] J. Zhang et al., "Deep model intellectual property protection via deep watermarking," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 44, no. 8, pp. 4005–4020, 2022. DOI: 10.1109/TPAMI.2021.3064850.