Traceable Black-box Watermarks for Federated Learning

Jiahao Xu[1] Rui Hu[1] Olivera Kotevska[2] Zikai Zhang[1]
[1]University of Nevada, Reno [2]Oak Ridge National Laboratory
{jiahaox,ruihu,zikaiz}@unr.edu kotevskao@ornl.gov

Abstract

Due to the distributed nature of Federated Learning (FL) systems, each local client has access to the global model, posing a critical risk of model leakage. Existing works have explored injecting watermarks into local models to enable intellectual property protection. However, these methods either focus on non-traceable watermarks or traceable but white-box watermarks. We identify a gap in the literature regarding the formal definition of traceable black-box watermarking and the formulation of the problem of injecting such watermarks into FL systems. In this work, we first formalize the problem of injecting traceable black-box watermarks into FL. Based on the problem, we propose a novel server-side watermarking method, **TraMark***, which creates a traceable watermarked model for each client, enabling verification of model leakage in black-box settings. To achieve this,* **TraMark** *partitions the model parameter space into two distinct regions: the main task region and the watermarking region. Subsequently, a personalized global model is constructed for each client by aggregating only the main task region while preserving the watermarking region. Each model then learns a unique watermark exclusively within the watermarking region using a distinct watermark dataset before being sent back to the local client. Extensive results across various FL systems demonstrate that* **TraMark** *ensures the traceability of all watermarked models while preserving their main task performance.*

1. Introduction

Federated Learning (FL) is a promising training paradigm that enables collaborative model training across distributed local clients while ensuring that private data remains on local devices [18]. Instead of sharing raw data, clients train local models independently and periodically send updates to a central server, which aggregates them into a global model. This privacy-preserving benefit has led to FL's widespread adoption in various fields, including healthcare [21], finance [16], and remote sensing [15], where local data privacy is a critical concern. However, sharing the global model with all participants introduces risks of model leakage. Specifically, malicious clients may exploit their access by duplicating and illegally distributing the model [10]. Such misconduct undermines the integrity of the FL system and compromises the collective interests of all participants. Consequently, protecting intellectual property (IP) [37] rights of FL-trained model and detecting copyright infringement have become critical challenges in FL [25].

Protecting the IP of FL-trained models requires mechanisms to verify rightful ownership if a model is unlawfully distributed [25] (i.e., proving that the model originated from the FL system). To address this, researchers have proposed embedding watermarks into the global model to enable ownership verification. Existing approaches primarily fall into two categories: parameter-based [30] and backdoor-based [1, 29] watermarking techniques. Parameter-based methods embed signatures (*e.g.*, bit strings) within the model's parameters as a secret key. During verification, the verifier extracts this key and applies a cryptographic function with the corresponding public key to validate the model's ownership. However, this process requires white-box access to the model parameters, which is often impractical, particularly when the suspect model is only accessible in a black-box setting (*e.g.*, via an API). To overcome this limitation, backdoor-based watermarking leverages backdoor injection techniques to ensure that the model learns a specific trigger. A watermarked model outputs predefined responses when presented with inputs containing the trigger. Unlike parameter-based methods, this verification process does not require access to the model parameters, making backdoor-based watermarking a more practical solution.

Beyond ownership verification, the verifier also needs to trace the source of model leakage, identifying which client was responsible for the unauthorized distribution. Recent studies have explored methods for ensuring the traceability of watermarked models in FL [22, 25, 36, 38]. For instance, FedTracker [25] embeds unique bit strings into each client's model and identifies the leaker by measuring bit string similarities. However, this approach requires white-box access to the suspect model, which limits its practicality. Another approach, FedCRMW [22], introduces a

black-box watermarking mechanism that injects unique watermarks into each model shared with the client by mixing clients' local datasets with multiple types of triggers. The model leaker is then identified based on the models' predictions on these watermarked datasets. However, this method modifies the local training protocol and requires access to clients' local data, making it vulnerable to tampering by malicious clients. Moreover, such an approach contradicts FL's core principle of data privacy preservation. Despite these advancements, the literature lacks a formal definition and formulation of the problem of black-box watermarking for model ownership verification and traceability in FL.

In this work, we first formalize the traceable black-box watermarking injection problem in an FL system. Building on this, we propose **TraMark** , which creates a personalized, traceable watermarked model for each client, enabling verification of model leakage in black-box settings. Specifically, to ensure traceability, the server partitions the model's parameter space into two regions: the main task region, responsible for learning the primary FL task, and the watermarking region, designated for embedding a watermark. The server then generates personalized global models for each client via masked aggregation. Each model is subsequently injected with a distinct watermark exclusively in the watermarking region using a dedicated watermark dataset. This process ensures that every client receives a personalized global model that integrates aggregated knowledge from other clients while embedding a unique watermark for model leakage verification. We summarize the contribution of our paper as follows.

- To the best of our knowledge, this is the first work to formally formulate the problem of traceable black-box watermark injection in FL systems. Based on this, we propose **TraMark**, a novel watermarking method that seamlessly integrates into existing FL systems.
- **TraMark** is designed to inject unique watermarks into models shared with clients while preventing watermark collisions in FL, enabling the identification of model leakers in black-box settings.
- We demonstrate the effectiveness of **TraMark** through extensive experiments across various FL settings. Results show that **TraMark** ensures clients receive traceable global models while maintaining main task performance, with only a slight average drop of 0.54%. Additionally, we conduct a detailed hyperparameter analysis of **TraMark** to evaluate the impact of each configuration on both main task performance and leakage verification.

2. Background and System Settings

Federated Learning. A typical FL system consists of a central server and a set of n local clients, which collaboratively train a shared model $\theta \in \mathbb{R}^d$. The FL problem is generally formulated as: $\min_\theta (1/n) \sum_{i=1}^{n} F_i(\theta; \mathcal{D}_i^l)$, where $F_i(\cdot)$ represents the local learning objective of client i, and $\mathcal{D}_i^l$ is its local dataset. For instance, for a classification task, client i's local objective can be expressed as: $F_i(\theta; \mathcal{D}_i^l) := \mathbb{E}_{(z,y)\in\mathcal{D}_i^l} \mathcal{L}(\theta; z, y)$, where $\mathcal{L}(\cdot)$ is the loss function, and (z, y) represents a datapoint sampled from $\mathcal{D}_i^l$. A classic method to solve the FL problem is Federated Averaging (FedAvg) [18]. Specifically, in each training round t, the server broadcasts the current global model θ^t to each client $i \in [n]$. Upon receiving θ^t, client i performs τ_l iterations of local training on it using $\mathcal{D}_i^l$, resulting in an updated local model $\theta_i^{t,\tau}$. The client then computes and sends the model update $\Delta_i^t = \theta_i^{t,\tau} - \theta^t$ back to the server. The server aggregates these updates from all clients and refines the global model as: $\theta^{t+1} = \theta^t + (1/n) \sum_{i=1}^{n} \Delta_i^t$. This process repeats until the global model converges.

Attack Model. In this work, we consider a malicious scenario where all clients in an FL system are potential model leakers. Specifically, we assume malicious clients follow a predefined local training protocol to complete the FL task but may illegally distribute their local models for personal profit. Importantly, they are unaware of the watermarking process and do not collude with others.

Defense Model. We assume that the server acts as the defender, responsible for injecting traceable watermarks to the FL system. The server is considered always reliable and equipped with sufficient computational resources. It is also assumed to have full access to all local models but no access to local data. Moreover, the server aims to keep the watermark injection process confidential from all local clients. Additionally, the server also acts as the verifier: if a model is deemed suspicious, it initiates a verification process to determine whether the model originates from the FL system and to identify the responsible model leaker.

3. Problem Formulation

Black-box Watermarking. A black-box watermark is a practical watermarking solution that the verification process that does not require access to model parameters, making it more suitable for real-world deployment than white-box watermarking. A common black-box watermarking approach leverages backdoor injection [1], where a watermark dataset $\mathcal{D}^w$ (as defined in Definition 1) containing triggers is used to train the model to produce a predefined output when presented with the triggers.

Definition 1 (Watermark Dataset)**.** *A watermark dataset $\mathcal{D}^w$ is a designated set of trigger-output pairs used to embed a watermark into a model. Formally,*

$$\mathcal{D}^w = \{(x, \phi(x)) \mid x \in \mathcal{X}^w\},$$

where $\phi(x)$ is the unique predefined output distribution assigned to each trigger x in the trigger set $\mathcal{X}^w$.

With the watermark dataset, we define the black-box watermark as follows.

Definition 2 (Black-box Watermark)**.** *A valid black-box watermark δ is a carefully crafted perturbation learned from the watermark dataset $\mathcal{D}^w$. It is applied to a model θ to obtain the watermarked model $\theta' = \theta + \delta$, which produces outputs following the predefined distribution $\phi(x)$ when evaluated on $\mathcal{D}^w$. Formally, the model θ' is considered watermarked with δ if:*

$$\mathbf{y}(\theta'; x) \sim \phi(x), \quad \forall (x, \phi(x)) \in \mathcal{D}^w,$$

where $\mathbf{y}(\theta'; x)$ denotes the output probability distribution of the watermarked model given trigger x.

If a black-box watermark is successfully embedded, one can verify whether a suspicious model originates from the system by testing its outputs on triggers from $\mathcal{D}^w$. For example, in classification tasks, verification is typically performed by evaluating the *prediction accuracy* of the suspicious model θ' on $\mathcal{D}^w$. Specifically, if the model's prediction accuracy $\sum_{x \in \mathcal{D}^w} \mathbf{1}[\arg\max \mathbf{y}(\theta'; x) = \arg\max \phi(x)]/|\mathcal{D}^w|$ exceeds a predefined threshold ν, this indicates that the suspicious model contains the watermark δ, thereby verifying its ownership [14, 23, 25, 29].

Traceability. However, ownership verification alone is insufficient for detecting model leaking of an FL system. While ownership verification confirms whether a model originates from the system, it does not identify which client leaked it. The ability to pinpoint the source of leakage is known as *traceability*. To achieve traceability, each watermarked model should carry a distinct watermark, ensuring that every client receives a unique identifier embedded in their model. More formally, for any two watermarked models θ'_i and θ'_j, their outputs should be as different as possible when evaluated on the same watermark dataset. If their outputs are too similar, a watermark collision (as defined in Definition 3) occurs, which can compromise traceability.

Definition 3 (Watermark Collision)**.** *A watermark δ_i learned from a watermark dataset $\mathcal{D}_i^w$ is said to collide with another watermark δ_j learned from $\mathcal{D}_j^w$ if their corresponding watermarked models, $\theta'_i = \theta + \delta_i$ and $\theta'_j = \theta + \delta_j$, produce highly similar outputs on watermark dataset $\mathcal{D}_i^w$ or $\mathcal{D}_j^w$. Formally, a collision occurs if:*

$$\mathbb{E}_x \left[\mathrm{Div}\big(\mathbf{y}(\theta'_i; x), \mathbf{y}(\theta'_j; x)\big) \right] \leq \sigma,$$

*for $x \in \mathcal{D}_i^w$ or $x \in \mathcal{D}_j^w$, where $\mathrm{Div}(\cdot)$ is a divergence measurement function (*e.g.*, KL divergence), and σ is a predefined collision threshold.*

Remark 1. *Watermark collision poses a significant challenge in ensuring the traceability of watermarked models. Since δ_i and δ_j are learned from $\mathcal{D}_i^w$ and $\mathcal{D}_j^w$, respectively, the distinctiveness of these watermark datasets plays a crucial role in preventing collisions. Specifically, if $\mathcal{D}_i^w$ and $\mathcal{D}_j^w$ are too similar, the resulting δ_i and δ_j will also be similar, increasing the risk of collision. Therefore, it is essential to ensure an intrinsic difference between $\mathcal{D}_i^w$ and $\mathcal{D}_j^w$. We discuss strategies for constructing distinct watermark datasets to mitigate collisions in Section 4.3.*

With Definition 1–3, we formally define the traceability of watermarked models as follows.

Definition 4 (Traceability of Watermarked Models)**.** *Given n watermarked models $\{\theta'_1, \theta'_2, \ldots, \theta'_n\}$, where each model is derived as $\theta'_i = \theta + \delta_i$, with a successfully embedded watermark δ_i, such that $\mathbf{y}(\theta'_i; x) \sim \phi(x)$, $\forall (x, \phi(x)) \in \mathcal{D}_i^w$. The traceability property ensures that different watermarked models produce distinguishable outputs on their respective watermark datasets. Formally, if for any watermarked model θ'_i, $i \in [n]$, the following holds:*

$$\mathbb{E}_{x \in \mathcal{D}_i^w} \left[\mathrm{Div}\big(\mathbf{y}(\theta'_i; x), \mathbf{y}(\theta'_j; x)\big) \right] > \sigma, \quad \forall j \in [n], j \neq i.$$

then the traceability of these models is ensured.

Intuitively, if each watermark in the system remains distinct and does not collide with any other watermark, then all watermarked models in the system are considered traceable.

Problem Formulation. Now, we define the problem of injecting traceable black-box watermarks in FL. Consider an FL system with n clients collaboratively training a global model θ under the coordination of the server. For watermark injection, the server prepares n distinct watermark datasets $\{\mathcal{D}_i^w\}_{i=1}^n$ to be used to inject watermarks into the global models for every client. The overall goal is to optimize both the main task learning objective and the watermarking objective while ensuring that the watermarked models remain traceable. This is formulated as follows:

$$\begin{aligned} \min_{\theta, \{\delta_i\}_{i=1}^n} & \underbrace{\frac{1}{n} \sum_{i=1}^{n} F_i(\theta; \mathcal{D}_i^l)}_{\text{Main Task}} + \underbrace{\frac{1}{n} \sum_{i=1}^{n} L_i(\theta + \delta_i; \mathcal{D}_i^w)}_{\text{Watermarking Task}}, \\ s.t.\ & \mathbb{E}_{x \in \mathcal{D}_i^w} [\mathrm{Div}(\mathbf{y}(\theta + \delta_i; x), \mathbf{y}(\theta + \delta_j; x))] > \sigma, \\ & \forall i, j \in [n],\ i \neq j. \end{aligned} \tag{1}$$

Here, δ_i denotes the traceable black-box watermark for the model θ_i shared with client i. The function $L_i(\cdot)$ represents the watermarking objective for δ_i, defined as: $L_i(\theta + \delta_i; \mathcal{D}_i^w) := \mathbb{E}_{(x, \phi(x)) \in \mathcal{D}_i^w} \mathcal{L}(\theta + \delta_i; x, \phi(x))$.

Remark 2. *From Problem (1), we derive the following key insights: 1) A straightforward solution for Problem (1) is to offload each watermark dataset $\mathcal{D}_i^w$ to client i, allowing clients to mix $\mathcal{D}_i^w$ with their main task dataset $\mathcal{D}_i^l$ during local training to solve both objectives simultaneously, as proposed in [10, 14, 22, 23, 32, 36]. However, this method*

is highly vulnerable to malicious clients who may simply discard $\mathcal{D}_i^w$, leading to the absence of watermarks in their models. Furthermore, if malicious clients are aware of the watermarking process, they could intentionally tamper with it, undermining its effectiveness. To mitigate these risks, it is preferable to decompose Problem (1), leaving the main task to local clients while performing watermark injection solely on the server. 2) A critical challenge in watermark injection is the risk of watermark collisions due to model averaging during aggregation. Specifically, even if the server successfully injects distinct watermarks into the models before sending them to clients for local training, these watermarks will be fused during model aggregation in the next training round if parameters from all clients are simply averaged, as in FedAvg. To address this issue, a specialized mechanism is required to prevent watermark entanglement during aggregation. 3) The constraint in Problem (1) suggests that to avoid collisions, the server should maximize $\|\delta_i - \delta_j\|_2^2$. However, if δ_i and δ_j become too divergent, this may impair the main task performance of the watermarked models. Additionally, since the server lacks access to clients' local datasets, directly solving the watermarking objective $L(\cdot)$ could also lead to significant degradation in the performance of the main task. Thus, a specialized learning strategy is required to ensure that watermark injection does not compromise the model's main task performance.

4. Injecting Traceable Black-box Watermarks

Based on the insights in Remark 2, we propose a novel method called **TraMark** detailed in Algorithm 1, which can be easily integrated into existing FedAvg frameworks to solve Problem (1). We give the complete process of FedAvg with **TraMark** in Algorithm 2 in Appendix Section 8. Specifically, **TraMark** operates entirely on the server side. Once the server receives local model updates from the clients, it uses **TraMark** to aggregate these updates and derive a personalized global model for each client. For each global model, the watermark is injected by learning from a distinct watermark dataset. The server then sends the watermarked global model back to each client for the next round of training or deployment.

4.1. Constraining Watermarking Region

Existing watermarking approaches either retrain the global model directly on the watermark dataset [25, 29] or require local clients to collaboratively inject watermarks [10, 14, 23]. However, these methods cause watermark-related perturbations to spread across the entire parameter space, leading to two key issues. First, the dispersed watermark perturbations may significantly degrade the main task performance. Second, even if each client's model embeds a unique watermark, model aggregation in the next round fuses these watermarks, causing collisions that compromise traceability. To mitigate the impact on main task performance and ensure traceability, **TraMark** restricts watermarking to a small subset of the model's parameter space. Only this designated watermarking region carries the watermark, and its parameters are excluded from model aggregation, preserving distinct watermarks for each client in the next training round. Specifically, in **TraMark**, given a model $\theta \in \mathbb{R}^d$, the server partitions the whole parameter space into *watermarking region* and *main task region* with a partition ratio $k \in [0, 1)$, resulting in two *complementary* binary masks:

- The **watermarking mask** $\mathbf{M}_w \in \{0,1\}^d$, where $[\mathbf{M}_w]_j = 1$ means that the j-th parameter in θ is used for watermarking task and $\text{sum}(\mathbf{M}_w) = k \times d$.
- The **main task mask** $\mathbf{M}_m \in \{0,1\}^d$, where $[\mathbf{M}_m]_k = 1$ means that the k-th parameter in θ is used for main task and $\text{sum}(\mathbf{M}_m) = (1-k) \times d$.

These two complementary masks ensure that all model parameters are fully partitioned into the watermarking and main task regions (*i.e.*, $\mathbf{M}_w + \mathbf{M}_m = \mathbf{1}^d$). Moreover, once determined, the masks remain unchanged throughout the entire watermarking process. We discuss how to partition the model in Section 4.4.

Algorithm 1: TraMark

Input : The global models $\{\theta_i\}_{i=1}^n$, a set of model updates $\{\Delta_i\}_{i=1}^n$, watermark datasets $\{\mathcal{D}_i^w\}_{i=1}^n$, main task mask $\mathbf{M}_m$, watermarking mask $\mathbf{M}_w$, watermarking learning rate η_w, and watermarking iteration τ_w.

Output: A set of watermarked models $\{\theta_i'\}_{i=1}^n$.

// **Watermark injection**

for $i \in [n]$ **do**

  // Masked aggregation

  $\tilde{\theta}_i \leftarrow \mathbf{M}_m \odot (1/n) \sum_{i=1}^n (\theta_i + \Delta_i) + \mathbf{M}_w \odot (\theta_i + \Delta_i)$

  // Watermarking

  $\tilde{\theta}_i^0 \leftarrow \tilde{\theta}_i$

  for $s = 0$ **to** $\tau_w - 1$ **do**

    $g_i^s \leftarrow \nabla_{\tilde{\theta}_i^s} \mathcal{L}(\tilde{\theta}_i^s; \mathcal{D}_i^w)$

    $\tilde{\theta}_i^{s+1} \leftarrow \tilde{\theta}_i^s - \eta_w g_i^s \odot \mathbf{M}_w$

  end

  $\theta_i' \leftarrow \tilde{\theta}_i^{\tau_w}$

end

Return $\{\theta_i'\}_{i=1}^n$

4.2. Masked Aggregation and Watermark Injection

Masked Aggregation. With the constrained watermarking region, **TraMark** leverages a novel *masked aggregation* method to avoid watermark collision. Specifically, instead of applying a naive aggregation approach (*e.g.*, FedAvg), **TraMark** aggregates the parameters in the main task region only and prevents the watermarking region from parameter fusion. In detail, in each training round, given the

model updates $\{\Delta_i\}_{i=1}^n$, the server aggregates them to generate the personalized global model for each client individually via $\tilde{\theta}_i = \mathbf{M}_m \odot (1/n) \sum_{i=1}^n (\theta_i + \Delta_i) + \mathbf{M}_w \odot (\theta_i + \Delta_i),\ \forall i \in [n]$ (line 2 in Algorithm 1). Here, the first term $\mathbf{M}_m \odot (1/n) \sum_{i=1}^n (\theta_i + \Delta_i)$ averages the model updates in the main task region, and the second term $\mathbf{M}_w \odot (\theta_i + \Delta_i)$ preserves the update of client i in the watermarking region. In this case, the server generates a personalized global model for each client, which always contains a *distinct watermark* for the client while still benefiting from the aggregated model updates for the main task.

Watermark Injection. For each personalized global model $\tilde{\theta}_i, \forall i \in [n]$, its watermark is injected by training $\tilde{\theta}_i$ on the corresponding distinct watermark dataset $\mathcal{D}_i^w$ for τ_w iterations. In this process, only the watermarking region is updated, ensuring that knowledge from the watermark dataset does not spread to the main task region (line 4–line 7). Technically, in each step of local training, the mini-batch gradient is multiplied by $\mathbf{M}_w$ (line 5–line 6), zeroing out the gradients for the main task region to avoid the impact of the watermark on the main task. After watermark injection, the server obtains the watermarked global models, which will be sent to the clients to perform their main tasks of the next training round or deployment (line 10). Since the clients' local training protocol for the main task remains unchanged, model updates continue across the entire parameter space. Notably, this case leads to a potential risk: over time, the embedded watermarks may gradually fade. To prevent watermark fading, **TraMark** can be applied at every training round to continuously enhance the watermark, as advised by prior works [10, 23, 25, 29].

4.3. Distinct Watermark Dataset

As noted in Remark 1, ensuring sufficient differences between watermark datasets is crucial for learning distinct watermarks and preventing collisions. To achieve this, in **TraMark** , the server assigns each global model a unique watermark dataset. Specifically, the watermark datasets designed for different models should differ from each other in both its triggers and its output distribution. Let $\mathcal{D}_i^w = \{(x, \phi_i(x)) \mid x \in \mathcal{X}_i^w\}$ denote the watermark dataset for personalized global model $\tilde{\theta}_i$, where $\mathcal{X}_i^w \cap \mathcal{X}_j^w = \emptyset$ for any $i \neq j,\ \forall i, j \in [n]$. Furthermore, each client is assigned a unique output distribution $\phi_i(x)$, guaranteeing that $\phi_i(x) \neq \phi_j(x)$. For trigger selection, existing methods have explored various approaches, including randomly generated patterns [25, 29], adversarially perturbed samples [10], and samples embedded with backdoor triggers [14]. In our case, to ensure each client receives a maximally distinct trigger, we select out-of-distribution samples absent from the main task dataset. This guarantees that the learned watermark remains independent of the main task. For example, in a classifier trained for traffic sign recognition, per-label samples from the MNIST dataset serve as effective triggers for different clients. Assigning a distinct watermark dataset to each personalized global model ensures that a watermarked model responds only to triggers from its own dataset, mapping them to the predefined output. When exposed to triggers from other watermark datasets, it produces random guesses, effectively minimizing the risk of collisions.

4.4. Selection of Watermarking Region

A key question that remains is: how should the server select the watermarking region? A naive approach is to randomly assign a small portion of parameters for watermark injection before training begins. However, this risks degrading main task performance, as critical parameters for the main task may be allocated to the watermarking region, leading to main task performance loss. Recall from Remark 2 that maximizing $\|\delta_i - \delta_j\|_2^2$ is crucial for avoiding collisions. However, excessive divergence between δ_i and δ_j may negatively impact main task performance. Given that $\|\delta_i - \delta_j\|_2^2 = \|\mathbf{M}_m \odot (\delta_i - \delta_j)\|_2^2 + \|\mathbf{M}_w \odot (\delta_i - \delta_j)\|_2^2$, where $\mathbf{M}_m = \mathbf{1}^d - \mathbf{M}_w$ and the watermarking process is confined to the watermarking region, the objective simplifies to maximizing $\|\mathbf{M}_w \odot (\delta_i - \delta_j)\|_2^2$. This ensures that watermark injection and collision avoidance should not affect parameters in the main task region. Consequently, if $\mathbf{M}_m$ contains the most important parameters while $\mathbf{M}_w$ is assigned to unimportant ones, the primary accuracy remains largely unaffected. Typically, parameter importance is measured by magnitude (absolute value), with larger values indicating greater importance [5, 24, 34, 35]. However, since network parameters are randomly initialized at the start of training, their importance is not yet established. As a result, assigning parameters to regions too early may lead to suboptimal partitioning, potentially degrading main task performance. To this end, **TraMark** introduces a *warmup training phase*, where the global model undergoes standard federated training (*e.g.*, FedAvg) for $\alpha \times T$ rounds before watermarking. The warmup training ratio $\alpha \in [0, 1)$ determines the fraction of total training rounds allocated to this phase, ensuring the model is robust enough to the main task before starting the watermark injection. Once warmup training is complete, the server obtains the watermarking region by selecting $k \times d$ least important parameters (i.e., those that have the smallest absolute values), and the remaining parameters are assigned to the main task region.

5. Experiments

5.1. Experimental Settings

General Settings. Following previous works [22, 23, 25, 39], we evaluate our methods on FMNIST [33], CIFAR-10 [7], and CIFAR-100 [7], using a CNN, AlexNet [6],

and VGG-16 [26], respectively. In addition, we test **TraMark** on large-scale Tiny-ImageNet using ViT [4]. For all datasets, we consider both independent and identically distributed (IID) data and non-IID data scenarios. To simulate non-IID cases, we use the Dirichlet distribution [19] with a default degree $\gamma = 0.5$. Following previous works [10, 22, 23, 36], we set up a cross-silo FL system with 10 clients. We also test **TraMark** on large-scale FL with client sampling in Appendix Section 12. Each client performs local training with $\tau_l = 5$ iterations and a learning rate of $\eta_l = 0.01$. The training rounds for FMNIST, CIFAR-10, CIFAR-100, and Tiny-ImageNet datasets are $50, 100, 100$, and 50. *All experiments are repeated* 3 *times with different seeds and we report the averaged results.*

Baselines and TraMark Settings. In all experiments, we use the MNIST [9] dataset as the source for watermarking. Each global model is assigned a watermark dataset containing samples from a distinct MNIST label, ensuring both $\mathcal{X}_i^w \cap \mathcal{X}_j^w = \emptyset$ and $\phi_i(x) \neq \phi_j(x)$ for any two clients i and j. We present the results of **TraMark** using other watermark datasets in Appendix Section 11. Each watermark dataset consists of 100 samples. The watermarking learning rate is set to $\eta_w = 1e^{-4}$, and the number of watermarking iterations is $\tau_w = 5$. The partition ratio k is set to 1%. The warmup training ratio α is set to 0.5. To ensure a fair comparison, we evaluate **TraMark** against two *server-side* watermarking approaches: WAFFLE [29], a black-box watermarking method that does not ensure traceability of watermarked models, and FedTracker [25], a white-box watermarking method that ensures traceability.

Evaluation Metrics. We evaluate the performance of each method using two key metrics: main task accuracy (MA) and model leakage verification rate (VR). MA is measured using the main task test set. Since **TraMark** and FedTracker introduce slight variations in each local model due to watermark injection, we compute MA as the average accuracy across all local models, following [25]. VR quantifies the proportion of watermarked models that are successfully attributed to their respective owners. We evaluate each watermarked model on the full test set, compute the per-label accuracy, and identify the label with the highest accuracy. If this highest-accuracy label matches the pre-assigned label of the model's owner, the model is considered successfully verified (more details are given in Appendix Section 9). For FedTracker, we follow its original definition of VR, where traceability is determined based on fingerprint similarity in a white-box setting.

5.2. Empirical Results

Verification Interval. We first demonstrate the verifier's confidence in identifying the leaker of suspect models embedded with watermarks injected by **TraMark**. Specifically, we calculate two key metrics: *verification confidence*—the test accuracy of a watermarked model on its own watermarking dataset, and *verification leakage*—its average test accuracy on other clients' watermarking datasets. The difference between these two metrics termed the *verification interval*, reflects the verifier's confidence. Figure 1 illustrates the averaged verification confidence and leakage across training rounds on CIFAR-10 and CIFAR-100 datasets. We observe that **TraMark** maintains a consistently large verification interval throughout training. Moreover, as training progresses, the interval widens due to a steady increase in verification confidence, indicating that watermark injection in **TraMark** continuously enhances watermark effectiveness despite potential performance degradation from local training. Additionally, since **TraMark** injects watermarks only within the designated watermarking region and employs masked aggregation, the watermarked model consistently performs poorly on other clients' watermarking datasets. These factors collectively contribute to a clear verification interval, ensuring successful model leakage verification. Further results on the changes in the divergence of each watermarked model's output on its watermark dataset (the constraint in Problem (1)) are provided in Appendix Section 10.

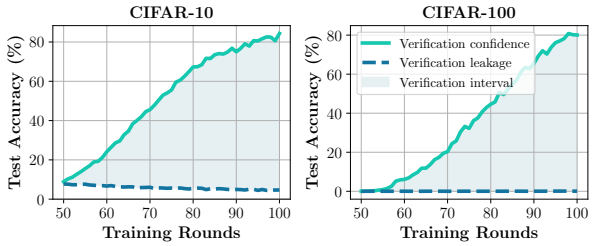

Figure 1. Verification confidence and verification leakage across training rounds on CIFAR-10 and CIFAR-100.

Table 1. Comprehensive comparison of MA and VR between **TraMark** and its counterparts under both IID and non-IID settings (highlighted with a gray background). MAs are shown in percentages (%). If a method achieves a satisfactory VR (exceeds 95%), we denote it with "✓"; otherwise, we use "✗".

Datasets	FedAvg		WAFFLE		FedTracker		TraMark	
	MA	VR	MA	VR	MA	VR	MA	VR
FM	92.60	-	92.21	✗	89.95	✓	91.20	✓
FM	91.52	-	91.41	✗	67.50	✓	91.31	✓
C-10	89.15	-	89.16	✗	87.56	✗	88.58	✓
C-10	87.01	-	86.75	✗	83.42	✗	86.26	✓
C-100	61.91	-	61.68	✗	61.05	✓	61.13	✓
C-100	60.19	-	60.04	✗	60.12	✓	58.95	✓
Tiny	21.05	-	21.24	✗	20.40	✓	20.91	✓
Tiny	20.09	-	19.97	✗	20.00	✓	20.06	✓
Average	65.44	-	65.31	✗	61.25	✗	64.90	✓

Main Results. We report the MA and VR of each

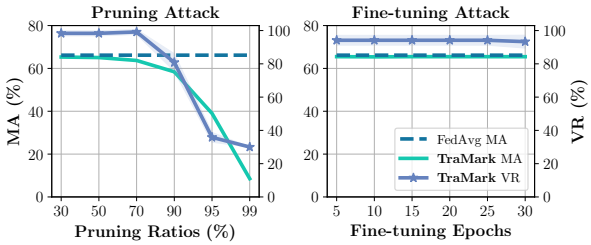

Figure 2. Averaged MA and VR results of **TraMark** under pruning and fine-tuning attacks across four datasets.

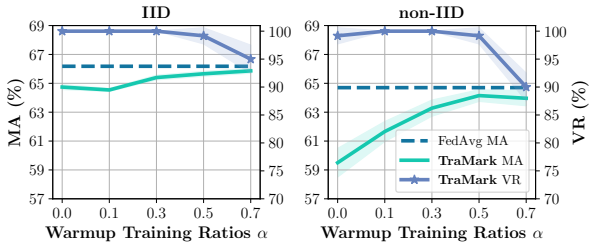

Figure 3. Impact of warmup training ratio α on the MA and VR of **TraMark** under IID and non-IID settings.

method on FMNIST (FM), CIFAR-10 (C-10), CIFAR-100 (C-100), and Tiny-ImageNet (Tiny) under both IID and non-IID settings in Table 1. Overall, **TraMark** effectively injects traceable watermarks into personalized global models while preserving high model performance. It consistently achieves a high VR across all datasets, maintaining an average of 99.17%. In contrast, FedTracker fails to ensure satisfactory traceability on CIFAR-10, resulting in an average VR of only 87.50%. This instability is due to the injection of key matrices into model parameters and the absence of explicit mechanisms to prevent watermark collisions after aggregation. Regarding MA, **TraMark** exhibits strong model performance, with only a 0.54% drop compared to FedAvg. While WAFFLE achieves a slightly higher average MA (0.41% above **TraMark**), it does not guarantee the traceability of watermarked models. FedTracker, on the other hand, suffers a significant 4.19% decline in MA due to the unconstrained watermarking region, which compromises model utility. In conclusion, **TraMark** successfully embeds traceable black-box watermarks while incurring minimal performance loss, making it a robust and practical watermarking solution for FL.

Robustness to Attacks. We evaluate the robustness of watermarked models trained by **TraMark** against pruning and fine-tuning attacks. Specifically, malicious clients may prune or fine-tune their local models to remove or reduce the effectiveness of watermarks. For the pruning attack, we test pruning ratios ranging from 30% to an extreme 99%. For fine-tuning attacks, we assume that malicious clients fine-tune their models for 30 epochs on their own datasets. The averaged MA and VR results of **TraMark** across four datasets are summarized in Figure 2. With moderate pruning ratios (30% to 70%), MA remains largely unaffected, while VR is also preserved. As the pruning ratio increases, MA declines rapidly, accompanied by a decrease in VR. These results demonstrate that the parameters in the watermarking region are coupled with the main task parameters, making simple magnitude-based pruning ineffective in removing the watermarks. This coupling also contributes to stable VR across various fine-tuning epochs during the fine-tuning attack. Additionally, we evaluate **TraMark** against the quantization attack and stronger adaptive attack, with results provided in Appendix Section 13.

In the following, we study how each hyperparameter functions of* TraMark *in detail.

Warmup Training Ratio α. **TraMark** leverages warmup training to achieve better partitioning between the main task region and the watermarking region. Figure 3 presents the averaged MA and VR of **TraMark** on four datasets under both IID and non-IID settings with varying warmup training ratios α. Notably, **TraMark** consistently achieves satisfactory VR with $\alpha \leq 0.5$. When $\alpha = 0.7$, **TraMark** fails to inject effective watermarks into each personalized global model *on time*, leading to degraded VR. For both settings, a larger α generally results in a higher MA. Specifically, under the non-IID setting, **TraMark** with the default $\alpha = 0.5$ achieves an average MA of 64.15%, which is only 0.55% lower than FedAvg but 4.65% higher than **TraMark** without warmup training. These results highlight the importance of warmup training, as it enables **TraMark** to accurately assign unimportant parameters to the watermarking region, thereby minimizing the negative impact of watermarking on main task performance in watermarked models.

Partition Ratio k. The partition ratio k controls the size of the watermarking region. Intuitively, a small k may hinder the watermark injection process as the watermarking region is unable to learn watermark-related information completely. We vary the partition ratio k from 0.1% to 5% to examine its impact on the performance of **TraMark**. The averaged MA and VR results across all datasets are shown in the left sub-figure of Figure 4. As expected, a smaller k leads to a significant drop in VR, while MA remains nearly unchanged. For example, compared to **TraMark** with the default setting ($k = 1.0\%$), reducing k to 0.5% causes VR to drop from 99.17% to 84.17%, whereas MA shows only a slight increase from 65.66% to 65.70%. Moreover, with an extreme value of $k = 5.0\%$, MA only drops to 65.16%, resulting in a gap of less than 1%, while achieving full VR. Therefore, selecting an appropriate k requires balancing MA and VR, with $k = 1.0\%$ serving as a practical choice that ensures both reliable watermark injection and minimal performance degradation.

Size of Watermark Dataset. A larger watermark

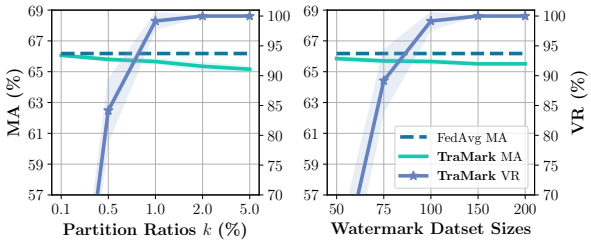

Figure 4. Impact of partition ratio k and watermark dataset size on the performance of **TraMark**.

dataset may improve the effectiveness of the watermark injection process by increasing the diversity and representativeness of watermark triggers, allowing the model to learn a more robust mapping between the watermark trigger and its intended response. To validate this, we vary the size of the watermark dataset from 50 to 200 samples. The averaged MA and VR results across all datasets are summarized in the right sub-figure of Figure 4. Similar to the effect of the partition ratio k, the size of the watermark dataset significantly impacts the traceability of watermarked models, while having minimal effect on MA. Specifically, when the watermark dataset contains only 50 triggers, **TraMark** achieves a suboptimal VR of 54.17%, despite obtaining the highest MA of 65.85%. However, with 100 or more samples, **TraMark** ensures successful watermark injection, with only a limited MA drop (at most 0.34%), striking a better balance between MA and VR. Therefore, choosing a sufficiently large watermark dataset, such as 100 samples or more, is essential to ensure the effectiveness of **TraMark**.

6. Related Work

Protecting the IP of FL models has been extensively studied recently, with most existing approaches leveraging either parameter-based [2, 3, 11, 13, 30, 36, 38, 39] or backdoor-based watermarking [10, 14, 17, 22, 23, 25, 29, 32]. While both approaches aim to verify model ownership, backdoor-based methods are more practical as they do not require access to model parameters. However, ensuring traceability, i.e., identifying the specific source of a leaked model remains an open challenge for backdoor-based watermarks.

Parameter-based Watermarking. Parameter-based watermarking methods typically embed cryptographic information directly into the parameter space of the global model. For example, Uchida et al. [30] proposed the first watermarking method for DNNs by incorporating a regularization loss term to embed a watermark into the model weights. Similarly, FedIPR [10] embeds messages in the Batch Normalization layers by assigning each client a random secret matrix and a designated embedding location. However, during verification, the verifier must access the model parameters to extract the embedded information. Consequently, these approaches assume that the verifier has full access to the suspect model, which is often unrealistic in real-world scenarios where leaked models may be only partially accessible (*e.g.*, via API queries) [8].

Backdoor-based Watermarking. Backdoor-based watermarking has been explored as a more practical alternative, as it does not require direct access to the model's internal parameters. These methods leverage backdoor injection techniques to ensure that the model learns a specific trigger. A watermarked model outputs predefined responses when presented with inputs containing the trigger [1]. For instance, WAFFLE [29] generates a global trigger dataset and fine-tunes the global model on it in each training round, thereby embedding the trigger into the model. Similarly, Liu et al. [14] assume the presence of an honest client in the system and injects a trigger set (constructed by sampling Gaussian noise) through local training. While these methods enable black-box verification, their watermarked model lacks traceability. Moreover, some approaches rely on client-side trigger injection, which poses a high risk of exposure if a malicious client becomes aware of the process.

Traceability of Watermarked Models. To ensure the traceability of watermarks, Yu et al. [38] propose replacing the linear layer of a suspect model with a verification encoder that produces distinct responses if the model originates from the FL system. FedTracker [25] extends WAFFLE by injecting a trigger into the global model while embedding local fingerprints (key matrices and bit strings) for individual clients. A recent work, RobWe [36], follows a similar workflow to ours, splitting the network into two parts: one for model utility and another for embedding watermarks (key matrices). However, since watermark injection occurs on the client side, this approach is less practical. Another client-side method FedCRMW [22], proposes a collaborative ownership verification method that indicates the leaker by the consensus of results of multiple watermark datasets. However, the watermark dataset used by FedCRMW is constructed based on the main task dataset, which incurs data privacy risks.

7. Conclusion

We formalize the problem of injecting traceable black-box watermarks in FL. We propose **TraMark**, which creates a personalized, traceable watermarked model for each client. **TraMark** first constructs a personalized global model for each client via masked aggregation. Subsequently, the watermarking process is exclusively performed in the watermarking region of each model using a distinct watermark dataset. The personalized watermarked models are then sent back to each client for local training or deployment. Extensive experiments demonstrate the effectiveness of **TraMark** in various FL settings. Additionally, we conduct a comprehensive hyperparameter study of **TraMark**.

References

[1] Yossi Adi, Carsten Baum, Moustapha Cisse, Benny Pinkas, and Joseph Keshet. Turning your weakness into a strength: Watermarking deep neural networks by backdooring. In *27th USENIX security symposium (USENIX Security 18)*, pages 1615–1631, 2018. 1, 2, 8

[2] Jinyin Chen, Mingjun Li, Yao Cheng, and Haibin Zheng. Fedright: An effective model copyright protection for federated learning. *Computers & Security*, 135:103504, 2023. 8

[3] Bita Darvish Rouhani, Huili Chen, and Farinaz Koushanfar. Deepsigns: An end-to-end watermarking framework for ownership protection of deep neural networks. In *Proceedings of the twenty-fourth international conference on architectural support for programming languages and operating systems*, pages 485–497, 2019. 8

[4] Alexey Dosovitskiy. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020. 6

[5] Rui Hu, Yuanxiong Guo, and Yanmin Gong. Federated learning with sparsified model perturbation: Improving accuracy under client-level differential privacy. *IEEE Transactions on Mobile Computing*, 2023. 5

[6] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25, 2012. 5

[7] Alex Krizhevsky et al. Learning multiple layers of features from tiny images. *University of Toronto*, 2009. 5

[8] Mohammed Lansari, Reda Bellafqira, Katarzyna Kapusta, Vincent Thouvenot, Olivier Bettan, and Gouenou Coatrieux. When federated learning meets watermarking: A comprehensive overview of techniques for intellectual property protection. *Machine Learning and Knowledge Extraction*, 5(4): 1382–1406, 2023. 8

[9] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998. 6

[10] Bowen Li, Lixin Fan, Hanlin Gu, Jie Li, and Qiang Yang. Fedipr: Ownership verification for federated deep neural network models. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 45(4):4521–4536, 2022. 1, 3, 4, 5, 6, 8

[11] Fang-Qi Li, Shi-Lin Wang, and Alan Wee-Chung Liew. Watermarking protocol for deep neural network ownership regulation in federated learning. In *2022 IEEE International Conference on Multimedia and Expo Workshops (ICMEW)*, pages 1–4. IEEE, 2022. 8

[12] Yige Li, Xixiang Lyu, Xingjun Ma, Nodens Koren, Lingjuan Lyu, Bo Li, and Yu-Gang Jiang. Reconstructive neuron pruning for backdoor defense. In *International Conference on Machine Learning*, pages 19837–19854. PMLR, 2023. 2

[13] Junchuan Liang and Rong Wang. Fedcip: Federated client intellectual property protection with traitor tracking. *arXiv preprint arXiv:2306.01356*, 2023. 8

[14] Xiyao Liu, Shuo Shao, Yue Yang, Kangming Wu, Wenyuan Yang, and Hui Fang. Secure federated learning model verification: A client-side backdoor triggered watermarking scheme. In *2021 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, pages 2414–2419. IEEE, 2021. 3, 4, 5, 8

[15] Yi Liu, Jiangtian Nie, Xuandi Li, Syed Hassan Ahmed, Wei Yang Bryan Lim, and Chunyan Miao. Federated learning in the sky: Aerial-ground air quality sensing framework with uav swarms. *IEEE Internet of Things Journal*, 8(12):9827–9837, 2020. 1

[16] Guodong Long, Yue Tan, Jing Jiang, and Chengqi Zhang. Federated learning for open banking. In *Federated Learning: Privacy and Incentive*, pages 240–254. Springer, 2020. 1

[17] Kaijing Luo and Ka-Ho Chow. Unharmful backdoor-based client-side watermarking in federated learning. *arXiv preprint arXiv:2410.21179*, 2024. 8

[18] Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Aguera y Arcas. Communication-efficient learning of deep networks from decentralized data. In *Artificial intelligence and statistics*, pages 1273–1282. PMLR, 2017. 1, 2

[19] Thomas Minka. Estimating a dirichlet distribution, 2000. 6

[20] Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Baolin Wu, Andrew Y Ng, et al. Reading digits in natural images with unsupervised feature learning. In *NIPS workshop on deep learning and unsupervised feature learning*, page 4. Granada, 2011. 2

[21] Dinh C Nguyen, Quoc-Viet Pham, Pubudu N Pathirana, Ming Ding, Aruna Seneviratne, Zihuai Lin, Octavia Dobre, and Won-Joo Hwang. Federated learning for smart healthcare: A survey. *ACM Computing Surveys (CSUR)*, 55(3): 1–37, 2022. 1

[22] Hewang Nie and Songfeng Lu. Fedcrmw: Federated model ownership verification with compression-resistant model watermarking. *Expert Systems with Applications*, 249:123776, 2024. 1, 3, 5, 6, 8

[23] Hewang Nie and Songfeng Lu. Persistverify: Federated model ownership verification with spatial attention and boundary sampling. *Knowledge-Based Systems*, 293: 111675, 2024. 3, 4, 5, 6, 8

[24] Ashwinee Panda, Saeed Mahloujifar, Arjun Nitin Bhagoji, Supriyo Chakraborty, and Prateek Mittal. Sparsefed: Mitigating model poisoning attacks in federated learning with sparsification. In *International Conference on Artificial Intelligence and Statistics*, pages 7587–7624. PMLR, 2022. 5

[25] Shuo Shao, Wenyuan Yang, Hanlin Gu, Zhan Qin, Lixin Fan, and Qiang Yang. Fedtracker: Furnishing ownership verification and traceability for federated learning model. *IEEE Transactions on Dependable and Secure Computing*, 2024. 1, 3, 4, 5, 6, 8, 2

[26] Karen Simonyan. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014. 6

[27] Canh T Dinh, Nguyen Tran, and Josh Nguyen. Personalized federated learning with moreau envelopes. *Advances in neural information processing systems*, 33:21394–21405, 2020. 3

[28] Alysa Ziying Tan, Han Yu, Lizhen Cui, and Qiang Yang. Towards personalized federated learning. *IEEE transactions on neural networks and learning systems*, 34(12):9587–9603, 2022. 3

[29] Buse GA Tekgul, Yuxi Xia, Samuel Marchal, and N Asokan. Waffle: Watermarking in federated learning. In *2021 40th International Symposium on Reliable Distributed Systems (SRDS)*, pages 310–320. IEEE, 2021. 1, 3, 4, 5, 6, 8, 2

[30] Yusuke Uchida, Yuki Nagai, Shigeyuki Sakazawa, and Shin'ichi Satoh. Embedding watermarks into deep neural networks. In *Proceedings of the 2017 ACM on international conference on multimedia retrieval*, pages 269–277, 2017. 1, 8

[31] Qiong Wu, Kaiwen He, and Xu Chen. Personalized federated learning for intelligent iot applications: A cloud-edge based framework. *IEEE Open Journal of the Computer Society*, 1: 35–44, 2020. 3

[32] Tong Wu, Xinghua Li, Yinbin Miao, Mengfan Xu, Haiyan Zhang, Ximeng Liu, and Kim-Kwang Raymond Choo. Citsmew: Multi-party entangled watermark in cooperative intelligent transportation system. *IEEE Transactions on Intelligent Transportation Systems*, 24(3):3528–3540, 2022. 3, 8

[33] Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *arXiv preprint arXiv:1708.07747*, 2017. 5

[34] Jiahao Xu, Zikai Zhang, and Rui Hu. Achieving byzantine-resilient federated learning via layer-adaptive sparsified model aggregation. In *Proceedings of the Winter Conference on Applications of Computer Vision (WACV)*, pages 1508–1517, 2025. 5

[35] Jiahao Xu, Zikai Zhang, and Rui Hu. Detecting backdoor attacks in federated learning via direction alignment inspection. *arXiv preprint arXiv:2503.07978*, 2025. 5

[36] Yang Xu, Yunlin Tan, Cheng Zhang, Kai Chi, Peng Sun, Wenyuan Yang, Ju Ren, Hongbo Jiang, and Yaoxue Zhang. Robwe: Robust watermark embedding for personalized federated learning model ownership protection. *arXiv preprint arXiv:2402.19054*, 2024. 1, 3, 6, 8

[37] Mingfu Xue, Yushu Zhang, Jian Wang, and Weiqiang Liu. Intellectual property protection for deep learning models: Taxonomy, methods, attacks, and evaluations. *IEEE Transactions on Artificial Intelligence*, 3(6):908–923, 2022. 1

[38] Shuyang Yu, Junyuan Hong, Yi Zeng, Fei Wang, Ruoxi Jia, and Jiayu Zhou. Who leaked the model? tracking ip infringers in accountable federated learning. In *NeurIPS 2023 Workshop on Regulatable ML*, 2023. 1, 8

[39] Lan Zhang, Chen Tang, Huiqi Liu, Haikuo Yu, Xirong Zhuang, Qi Zhao, Lei Wang, Wenjing Fang, and Xiang-Yang Li. Fedmark: Large-capacity and robust watermarking in federated learning. In *2024 IEEE 44th International Conference on Distributed Computing Systems (ICDCS)*, pages 821–832. IEEE, 2024. 5, 8

Traceable Black-box Watermarks for Federated Learning

Supplementary Material

Algorithm 2: FedAvg with **TraMark**

Input : number of clients n, local learning rate η_l, local iterations τ_l, warmup rounds t', total training rounds T, watermark dataset $\{\mathcal{D}_i^w\}_{i=1}^n$, main task region mask $\mathbf{M}_m$, watermarking region mask $\mathbf{M}_w$, watermarking learning rate η_w, watermarking iteration τ_w.

Output: $\{\theta_i^T\}_{i=1}^n$.

Initialization: Initialized model $\theta^0 \in \mathbb{R}^d$
Function `LocalTraining`(θ)**:**
  for $s = 0$ **to** $\tau_l - 1$ **do**
    $g_i^s \leftarrow \nabla_\theta \mathcal{L}(\theta^s; \mathcal{D}_i^l)$
    $\theta_i^{s+1} \leftarrow \theta_i^s - \eta_l g_i^s$
  end
  return $\theta_i^{\tau_l} - \theta$
$\theta_i^0 \leftarrow \theta^0,\ \forall i \in [n]$
for $t = 0$ **to** $T - 1$ **do**
  Broadcast θ_i^t to each client i
  for each $i \in [n]$ **in parallel do**
    $\Delta_i^t \leftarrow$ `LocalTraining`(θ_i^t)
  end
  if $t < t' - 1$ **then**
    // **FedAvg (warmup training)**
    $\theta_i^{t+1} \leftarrow (1/n) \sum_{i=1}^n (\theta_i^t + \Delta_i^t),\ \forall i \in [n]$
  else
    // **TraMark process**
    $\{\theta_i^{t+1}\}_{i=1}^n \leftarrow$ **TraMark** $(\{\theta_i^t, \Delta_i^t\}_{i=1}^n, \{\mathcal{D}_i^w\}_{i=1}^n, \mathbf{M}_m, \mathbf{M}_w, \eta_w, \tau_w)$
  end
end
Return $\{\theta_i^T\}_{i=1}^n$

8. FedAvg with TraMark

The complete process of FedAvg with **TraMark** is given in Algorithm 2. Specifically, during the warmup training process, the server follows the FedAvg training paradigm. Once warmup training is complete, the server transitions to **TraMark** training, as outlined in Algorithm 1.

9. Model Leaker Verification

We present the algorithm for the verifier to verify a leaked model, θ', in Algorithm 3. Specifically, given a leaked model θ' associated with a pre-assigned label i, where client i is the suspected leaker, the verifier evaluates θ' on the full test set and computes the per-label accuracy (line 1 in Algorithm 3). The verifier then selects the label with the highest accuracy. If this label matches the assigned label i (line 2), the verification is considered successful (line 3); otherwise, it is considered a failure (line 5). Since our attack model assumes any local client could be a potential leaker, we apply the verification process to each watermarked model. VR is then defined as the percentage of watermarked models successfully verified.

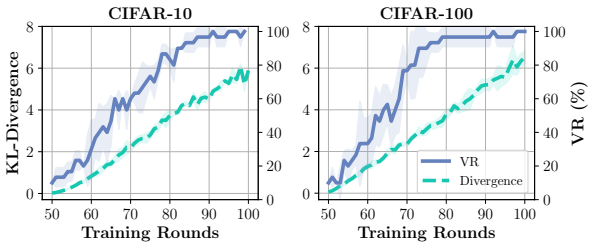

Figure 5. The averaged KL divergence and VR over training rounds on CIFAR-10 and CIFAR-100 datasets.

Algorithm 3: Model Leaker Verification

Input : A potentially leaked model θ' suspected to belong to client i, where i is the assigned label; watermarking test set $\mathcal{D}_{\text{test}}^w$.

Output: Verification result.

1 acc $\leftarrow$ calculate_per-label_accuracy$(\theta', \mathcal{D}_{\text{test}}^w)$
2 **if** $i = \arg\max_j \text{acc}[j]$ **then**
3 **return** Verification successful
4 **else**
5 **return** Verification failed
6 **end**

10. Output Divergence

We provide empirical evidence demonstrating how **TraMark** effectively prevents watermark collisions. Recall that in Problem (1), the constraint is designed to maximize the divergence between the outputs of different models when given the same inputs, thereby mitigating the risk of watermark collisions. To illustrate this, we compute the KL divergence between each watermarked model and all other watermarked models on the respective watermark test set. We plot the average KL divergence and VR for CIFAR-10 and CIFAR-100 datasets in Figure 5. The results clearly show a consistent increase in KL divergence as training progresses. This trend arises because, as the watermarking injection process continues, each watermarked model refines its unique watermark patterns, making it more distinguishable from others. Consequently, the VR also increases, further confirming the effectiveness of **TraMark** in preventing watermark collisions.

11. Other Sources of Watermark Dataset

Table 2. Generalization of **TraMark** across different watermarking datasets. MAs are shown in percentages (%). If a method achieves a full VR, we denote it with "✓"; otherwise, we use "✗".

Watermark Dataset	CIFAR-10		CIFAR-100		Tiny-ImageNet	
	MA	VR	MA	VR	MA	VR
MNIST	88.58	✓	61.13	✓	20.91	✓
SVHN	$88.52_{\geq 0.05}$	✓	$60.92_{\geq 0.05}$	✓	$20.22_{\geq 0.05}$	✓
WafflePattern	$88.84_{\geq 0.05}$	✓	$61.31_{\geq 0.05}$	✓	$20.91_{\geq 0.05}$	✓

We also use the SVHN [20] dataset and WafflePattern [29] as sources for watermark datasets. Notably, WafflePattern consists of images containing only noise with specific patterns. Since SVHN and WafflePattern contain colorful images while FMNIST is grayscale, we conduct experiments on CIFAR-10, CIFAR-100, and Tiny-ImageNet. The MA and VR results are summarized in Table 2. We observe that **TraMark** achieves highly similar results regardless of the watermark dataset used. Furthermore, we perform a T-test on each dataset to compare the results of **TraMark** using SVHN or WafflePattern against MNIST. The obtained p-values across all main task datasets exceed the commonly used significance threshold of 0.05, indicating that the differences are not statistically significant. These results demonstrate the generalization ability of **TraMark** in selecting different watermarking datasets.

12. Large-scale FL with Client Sampling

Table 3. Performance of **TraMark** under different client sampling settings in large-scale FL. MAs and VRs are shown in percentages (%).

Method	Tiny-ImageNet		Tiny-ImageNet (CS)	
	MA	VR	MA	VR
TraMark	17.32	100	17.07	100

Here, we evaluate the effectiveness of **TraMark** in a large-scale FL setting with 50 local clients. We primarily consider two scenarios: FL without client sampling and FL with *client sampling* (CS). In the client sampling scenario, the server randomly selects 20% of the clients in each training round to perform local training. For **TraMark**, we enforce the injection of watermarks for all clients in each round, regardless of whether they are sampled or not. We use WafflePattern as the source for the watermark dataset, as it allows for generating an arbitrary number of distinct classes. Our experiments are conducted on the Tiny-ImageNet dataset, and the results are summarized in Table 3. The results show that **TraMark** consistently ensures a complete VR in both scenarios. This demonstrates the strong generalization ability of **TraMark** across different client settings.

13. Quantization Attack and Adaptive Attack

Table 4. Impact of model quantization on MA and VR, comparing FP16 and INT8 against the FP32 baseline. MAs and VRs are shown in percentages (%).

Dataset	FP32 (Baseline)		FP16		INT8	
	MA	VR	MA	VR	MA	VR
FMNIST	91.20	96.67	91.94	96.67	91.92	96.67
CIFAR-10	88.58	100.00	88.35	100.00	88.35	100.00
CIFAR-100	61.13	100.00	60.99	96.67	60.99	96.67
Tiny-ImageNet	20.91	100.00	20.20	100.00	20.19	100.00
Average	67.46	99.17	65.37	98.34	65.36	98.34

Quantization Attack. We assume that malicious clients may quantize their local models to impact the effectiveness of watermarks. Following [25], we conduct experiments on watermarked models trained by **TraMark** that are quantized to FP16 and INT8. The MA and VR results are summarized in Table 4. Compared to the FP32 baseline, quantizing the model to FP16 and INT8 leads to a 2.09% and 2.10% drop in MA, respectively, and a 0.83% drop in VR. The negligible decrease in VR demonstrates the robustness of the watermarks injected by **TraMark** against quantization attacks.

Table 5. Performance of **TraMark** under RNP. MAs and VRs are shown in percentages (%).

Method	FMNIST		CIFAR-10		CIFAR-100	
	MA	VR	MA	VR	MA	VR
TraMark	91.20	96.67	88.58	100.00	61.13	100.00
RNP	73.46	70.00	85.08	90.00	60.57	100.00

Adaptive Attack. We consider a more challenging scenario where malicious clients are aware that the received global model has been embedded with a black-box watermark. As a result, they attempt to remove the watermark using a backdoor removal method. We adopt Reconstructive Neuron Pruning (RNP) [12], a state-of-the-art backdoor removal technique, for this purpose. Since RNP requires a Batch normalization layer while ViT employs Layer normalization, we conduct experiments on FMNIST, CIFAR-10, and CIFAR-100. The MA and VR results of RNP on the watermarked models are summarized in Table 5. We observe that RNP has limited effectiveness on the FMNIST and CIFAR-10 datasets, where the VR decreases from 90% and 100% to 70% and 90%, respectively. However, in these cases, the MA also drops significantly. For CIFAR-100, RNP has no impact on the VR but still leads to a degradation in MA. These results highlight the robustness of the

watermarks embedded in each global model, demonstrating the strong effectiveness of **TraMark**.

14. Discussions and Future Directions

Malicious Client Collusion. In our work, we adopt the benchmark FL paradigm, FedAvg, where each local client receives an identical model. However, since **TraMark** injects watermarks within the designated watermarking region, malicious clients may collude to identify its location by comparing their identical main task parameters. This limitation can be solved by leveraging personalized FL [27, 28, 31], where the server assigns unique model weights to each client, ensuring distinct watermarked models and enhancing security.

Computational Overhead. Recall that in our defense model, we assume the server has sufficient computational resources to perform the watermarking process. However, in practical scenarios, computational resources may be limited. Since **TraMark** applies watermarking in every training round, this could introduce a non-negligible computational overhead, which increases linearly with the number of participating local clients. Nevertheless, in this work, we focus on cross-silo FL settings, such as collaborations among several hospitals or institutions, where the number of clients is relatively small. In such cases, the server is more likely to have sufficient computational resources, making the additional overhead manageable. Addressing the broader challenge of reducing **TraMark**'s computational cost in cross-device FL systems remains an open problem for future work.

Theoretical Analysis. Although **TraMark** demonstrates strong empirical performance, there remains a gap in providing a theoretical guarantee for ensuring the traceability of watermarked models. We leave this theoretical analysis as future work.