PDLRecover: Privacy-preserving Decentralized Model Recovery with Machine Unlearning

Xiangman Li, Xiaodong Wu, Jianbing Ni, Mohamed Mahmoud, and Maazen Alsabaan

***Abstract*—Decentralized learning is highly susceptible to poison attacks, in which malicious participants can manipulate local updates to degradate of model performance. Existing defense methods primarily focus on detecting and filtering malicious models to prevent a certain number of malicious clients from altering local models and poison the global model. However, correcting and recovering an already compromised global model remains a significant challenge. One of direct methods is to remove the malicious clients and retrain the model with the remaining clients to restore model performance. However, retraining needs substantial computational and temporal costs and cannot guarantee consistency and privacy of the original model.**

In this paper, we propose a novel method called PDLRecover that can effectively recover a poisoned global model by utilizing historical information and prevent potential local model leakage. The key challenge is to protect the shared historical models, while enabling to estimate the parameters of the recovered model for model reconstruction and recovery by the remaining clients. By leveraging the linear property of approximate Hessian matrix computation, we achieve privacy protection of the historical information based on secret sharing, preventing local model leakage during transmission and reconstruction. PDLRecover involves clients performing preparation steps, periodic steps, and final exact training to guarantee the accuracy and robustness of the recovered model. Our recovery process performs periodic exact updates to maintain accurate local curvature information, followed by a final precise update to ensure convergence quality. We demonstrate that the recovered global model is comparable to the retrained model, but with significantly reduced computational time and cost, while enabling the protection of local model parameters against privacy leakage. Our experimental results validate the accuracy of the recovered global model and the efficiency in global model recovery.

***Index Terms*—Decentralized Learning, Machine Unlearning, Privacy Preservation, Poison Attack.**

I. INTRODUCTION

Decentralized learning is an emerging distributed machine learning paradigm that enables multiple clients, such as smartphones and IoT devices, to collaboratively train a global model without sharing their raw local data [1]. It allows clients to perform local model training and exchange only encrypted model parameters, thereby preserving data privacy and reducing the risk of data breaches. Meanwhile, decentralized learning significantly lowers communication overhead by eliminating the need to upload large volumes of raw data, making it well-suited for distributed and bandwidth-constrained environments. Decentralized learning has demonstrated practical advantages in real-world applications. For example, Google's Gboard uses federated learning for privacy-preserving text prediction, while autonomous driving systems leverage decentralized models trained on rich, on-board sensor data to recognize roads, pedestrians, and traffic signs [2].

However, decentralized learning still faces significant privacy and security challenges. Although it avoids centralized data collection, this paradigm relies on the exchange of intermediate model parameters or updates, which can unintentionally leak sensitive information. Two major privacy attack vectors in this context are model inversion attacks and reconstruction attacks. Model inversion attacks attempt to recover original training data by reversing the outputs of a model, while reconstruction attacks exploit intermediate parameters to reconstruct a client's local data [3]. For example, the structural patterns embedded in shared parameters may enable partial data recovery by adversarial clients. In addition, privacy risks persist even when raw data is not explicitly exchanged. Membership inference attacks [4] can determine whether specific data records were part of a client's training dataset, simply by analyzing the exposed model behavior. As a result, unprotected decentralized learning systems remain vulnerable to various forms of information leakage, compromising user privacy during collaborative model training.

Moreover, malicious clients in decentralized learning can launch poison attacks by tampering with their local training data or submitting manipulated model updates, thereby degrading the performance of the global model. These attacks fall into two main categories: data poisoning, where adversaries inject harmful or misleading data into the training set; and model poisoning, where clients send crafted updates to corrupt the aggregation process or mislead the global model's behavior [5]. While existing defenses [references] primarily aim to limit the influence of a small number of attackers and detect malicious local updates, the problem of recovering a compromised model has received comparatively little attention. A conventional solution [6] is to retrain the entire model from scratch, which is computationally expensive and time-consuming. Furthermore, in a decentralized setting, it is often impractical to ensure that all clients remain online and available for full model retraining. This underscores the need for efficient and collaborative model recovery mechanisms that do not rely on complete retraining. However, enabling clients

X. Li, X. Wu, and J. Ni are with the Department of Electrical and Computer Engineering and Ingenuity Labs Research Institute, Queen's University, Kingston, Ontario, Canada K7L 3N6. Email: {xiangman.li, xiaodong.wu, jianbing.ni}@queensu.ca.

M. Mahmoud is with the Department of Electrical and Computer Engineering, Tennessee Tech. University, Cookeville, TN 38505, USA. E-mail:mmahmoud@tntech.edu.

M. Alsabaan is with the Department of Computer Engineering, College of Computer and Information Sciences, King Saud University, Riyadh 11451, Saudi Arabia. E-mail: malsabaan@ksu.edu.sa.

to cooperatively restore the global model introduces additional privacy risks, as the communication and computation required for recovery may inadvertently expose sensitive information. Therefore, it is crucial to design recovery approaches that are not only effective and scalable but also privacy-preserving.

In this paper, we propose PDLRecover, a machine unlearning based privacy-preserving decentralized model recovery framework that uses clients' historical updates to mitigate the impact of malicious clients on the global model without evading the privacy of clients. Firstly, PDLRecover leverages the Hessian-vector product (HVP) computation [7] to approximate model updates using historical gradients and parameter differences. To enhance the fidelity of the recovered global model, PDLRecover performs periodic exact updates to maintain accurate local curvature information, followed by a final precise update to ensure convergence quality. However, the exposure of historical updates leads to the potential model leakage in model recovery, which is a severe but has not been resolved yet. To prevent potential model leakage from the historical updates, PDLRecover employs shamir secret sharing [8] to allow each client to securely encode their local gradients for model protection. However, the integration of secret sharing and the HVP is not trivial. According to the linear property of approximate Hessian matrix computation [9], the secret shares are aggregated in an encrypted manner and used to reconstruct global statistics without revealing individual client updates during training. In the recovery phase, each client independently computes its local approximated update direction using only its own private shares. The global direction is then reconstructed collaboratively through Lagrange interpolation. As a result, PDLRecover enables decentralized model repair in the presence of partial client dropout, while preserving client privacy and avoiding directly accessing or revealing other clients' model parameters.

Contributions. The main contributions of this paper are summarized in threefold:

- We propose an efficient unlearning method for decentralized learning that leverages Hessian-vector product (HVP) computation and clients' historical updates to mitigate the impact of malicious or dropout clients. PDLRecover provides a technical solution for removing the influence of poisoned updates and enables the restoration of a contaminated global model without the need for full retraining.
- We extend the standard L-BFGS method with secret sharing to ensure privacy preservation for clients. It enables the computation of approximate Hessian matrix computation with the input of secret shares, enabling clients reconstructing local updates for unlearning during the recovery process.
- We provide theoretical proofs to demonstrate the feasibility of PDLRecover in recovering the global model and preserving client privacy. Additionally, experimental simulations show that PDLRecover can effectively restore model performance to a high level while maintaining overall stability.

The remaining of this paper is organized as follows. In Section II, we introduce the related work about poison attacks and machine unlearning. Section III proposes an overview of preliminary knowledge and problem formalization, followed by PDLRecover background in Section IV. In Section V, we describe our detailed PDLRecover and the corresponding algorithms. Section VI shows the experimental results, followed by the security analysis of PDLRecover in Section VII. Finally, we conclude this paper in Section VIII.

II. Related Work

In this section, we review some designs of machine unlearning and the novel poison attacks and detection methods.

A. Machine Unlearning

Machine unlearning was originally proposed to "forget" specific data samples, along with their influence, from a trained model [10], in order to comply with the "Right to be Forgotten" as mandated by the General Data Protection Regulation (GDPR) [11]. It offers effective and reliable solutions for detecting and eliminating training data samples in response to privacy and regulatory requirements. Machine unlearning enables the generation of a model that behaves as if it has never learned the data points or samples designated for deletion at a client's request. Machine unlearning techniques can be broadly categorized into two types: exacting unlearning and approximate unlearning [12]. Exact unlearning is, for any data point $x_i \notin D$, the prediction of $M(x) = M'(x)$ [13]. Retraining is one of the methods of exact unlearning. This method retrains the model after removing the required data. However, this method is expensive and time-consuming, particularly for large datasets, decentralized architecture, and complex algorithms. Approximate unlearning aims to balance the cost and the accuracy of the model in training. Given an acceptable error threshold ϵ, if for all $x \in D$ the difference $|M(x) - M'(x)| < \epsilon$, the unlearning is considered accurate [13]. The approximate unlearning methods include influence function method [14], gradient modification [15], and re-optimization method [16]. These methods strike a balance between operational costs and model performance by ensuring that the precision of the unlearning model remains within an acceptable range, while substantially reducing resource consumption and processing time.

Unlearning has also been explored in federated learning, where techniques originally designed for centralized settings are adapted. For example, Su et al. [17] followed the design of SISA [10], and Liu et al. [18] further extended these concepts to design federated unlearning architectures. Their methods resemble the algorithm in [16], which leverages Fisher information [19] for gradient modification. Other federated unlearning methods are based on knowledge distillation [20] and model calibration [21] to enable the removal of trained samples from the global model. However, unlearning in fully decentralized learning has not been investigated. Achieving efficient unlearning in a fully decentralized model is challenging for two main reasons: First, the recovery process is time-consuming, as each client must communicate with others to exchange model updates in the absence of a central server. Second, model

updates provided by malicious or dropout clients can lead to the failure of model recovery and reconstruction.

B. Poison Attacks and Defenses

Poison attacks compromise the training process of machine learning models by deliberately manipulating input data, labels, or model updates, with the goal of degrading the performance or altering the behavior of the final model. Assume there is a clean dataset $D = \{(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\}$ for training a prediction model, where x_i and y_i are the data sample and the corresponding label of the i^{th} data in the training dataset, and n is the size of this dataset. The trainer trains a model G based on the dataset D and optimizes it based on a loss function L. Recently, poison attacks have been extensively explored, and various poison attacks have been identified in both centralized learning and decentralized learning. Zhang et al. [22] highlighted the vulnerability of EEG signal-based risk assessment systems to data poison attacks, particularly label-flipping attacks during the training stages of machine learning models. Similarly, Ovi et al. [23] emphasized the susceptibility of models to data poison attacks when malicious clients use tainted training data. In the realm of cybersecurity, backdoors can be activated by attackers to manipulate poisoned models, as demonstrated in the study on deep source code processing models [24].

Additionally, poison attacks in federated learning systems have been explored, with a focus on generative adversarial networks (GANs) as a means of attack [25]. These cases serve as reminders of the potential dangers associated with poison attacks in various contexts. Moreover, a recent study [26] delves into more specific types of poison attacks, such as prompt-specific poison attacks that aim to make poisoned samples visually identical to benign images with matching text prompts. This level of detail underscores the evolving sophistication of poison attack techniques and the difficulty of their identification and prevention.

To counter these evolving poison attacks, defense methods typically focus on distinguishing between benign and malicious samples. This is generally achieved by using model properties or activation statistics to determine if the model, training data, or test samples have been compromised. Various statistical methods have been applied to spot anomalies in poisoned datasets [27], [28]. For example, Hayase et al. [27] introduced a defense mechanism against backdoor attacks in machine learning models by leveraging robust statistics to detect and mitigate the impact of poisoned training samples. Li et al. [29] proposed a defense against DNN backdoor attacks using neural attention distillation, which fine-tunes a backdoor-affected student network with clean data from a trusted teacher network. Liu et al. [30] assessed pruning and fine-tuning strategies for defense against backdoor attacks, concluding that a combined approach, termed "fine-pruning," effectively reduced attack success rates without harming the accuracy of clean inputs.

Erasure techniques have also proven effective, as shown in works like [31] and [32]. These techniques aim to remove the influence of poisoned data by directly identifying and erasing corrupted parameters or activations, thus restoring the model's integrity. Additionally, Fang et al. [33] investigated the vulnerability of Byzantine-robust federated learning systems to local model poison attacks, showing how attackers can significantly raise global model error rates by corrupting client devices and manipulating local parameters. They modeled these attacks as optimization problems and evaluated them against various Byzantine-robust methods, highlighting severe potential performance issues. Li et al. [34] integrated detection techniques with blockchain technology, using an anti-Byzantine consensus mechanism called Proof of Accuracy to validate models and ensure the learning process's integrity.

However, these methods primarily focus on identifying malicious samples or differentiating between poisoned and clean models, without considering the potential to recover compromised models. Cao et al. [35] introduced the first model recovery technique that enables the server in federated learning to recover the global model after a poison attack by leveraging stored global models and clients' model updates from each training round. The server estimates each client's model update using historical data and restores the global model based on these estimations during the recovery process. Subsequently, Jiang et al. [36] enhanced this approach by improving recovery speed and reducing memory usage with an efficient recovery method that employs selective information storage and adaptive model rollback. However, both methods present privacy concerns as the server must store clients' local models for recovery, thereby exposing all historical data to the curious server, including clients' model updates. Additionally, current designs such as [35] and [36] are tailored for federated learning, leaving fully decentralized learning unexplored.

III. Preliminary Knowledge

A. Decentralized Learning

In decentralized learning [37], suppose that n clients have their local dataset D_i, aiming to train a global model. Their target is to minimize the loss function $min\ L(D; \mathbf{w})$, where D is the joint dataset of n clients, i.e., $D = \sum D_i$, $\mathbf{w}$ is the parameter of the global model, and L is the loss function, such as mean square error and cross-entropy loss. Each client trains its local machine learning model using its own dataset and receives model updates from other clients to maintain the global model. The global model is updated iteratively, in each iteration consisting of three main steps:

- Model Initialization: Clients establish bidirectional communication with their neighboring peers by broadcasting their availability. All clients start with the same model parameters $\mathbf{w}_0$.
- Model Training: Each client trains its local model using its own dataset and computes the local update, g_t^i, based on an optimization algorithm, such as the steepest gradient method or stochastic gradient descent, where i denotes the client index. A client then updates its local model parameters, $\mathbf{w}_t^i$, using the learning rate γ, according to the formula $\mathbf{w}_{t+1}^i = \mathbf{w}_t^i - \gamma \cdot g_t^i$. Afterward, clients broadcast their local model parameters through secret

TABLE I: Notations

Notation	Description
n	Number of clients
m	Number of malicious clients
t	Iteration index
i	Client index
γ	Learning rate
T	Total number of rounds
T_p	Number of preparation step
T_r	Index of periodic step
T_f	Number of final exact training
s	Buffer size of the L-BFGS algorithm
$\bar{\mathbf{w}}_t$	Original global model at iteration t
$\hat{\mathbf{w}}_t$	Recovery global model at iteration t
$\nabla L_i(\bar{\mathbf{w}}_t)$	Original local update for client i at iteration t
$\nabla L_i(\hat{\mathbf{w}}_t)$	Recovery local update for client i at iteration t
$\nabla L(\bar{\mathbf{w}}_t)$	Sum of original local update for client i at iteration t
$\nabla L(\hat{\mathbf{w}}_t)$	Sum of recovery local update at iteration t
$\nabla L^{(x_i)}(\hat{w}_t)$	Sub-secret recovery local update for client i at iteration t
$\widetilde{\mathbf{H}_t}$	Estimated Hessian matrix at iteration t
$\widetilde{G}^{(x_i)}$	Local update difference buffer for client i
$\widetilde{W}$	Global model difference buffer

sharing to their neighbours and receive model parameters from them.
- Model Aggregation: Each client collects the model parameters from its neighbors and applies an aggregation rule A, to update the global model: $\mathbf{w}_{t+1} = A(\mathbf{w}_t^1, \mathbf{w}_t^2, \ldots, \mathbf{w}_t^n)$.

Clients conduct these three steps to continuously update their local models. During the model aggregation step, different aggregation rules offer distinct advantages. In PDLRecover, we employ one of the most popular methods, FedAvg [38], which efficiently combines local model updates from all clients to produce a robust global model.

FedAvg is a parameter aggregation process that uses weighted averaging to update the global model. After each client collects all the neighbors' local model parameters $\mathbf{w}_i$ and local dataset sizes n_i, it calculates the weighted average of the global model parameters, where the weights are the sizes of the local datasets, following the equation as:

$$\mathbf{w}_{\text{new}} = \frac{\sum_{i=1}^{K} n_i \mathbf{w}_i}{\sum_{i=1}^{K} n_i}, \tag{1}$$

where K is the number of participating devices, $\mathbf{w}_{\text{new}}$ is the new global model parameters, n_i is the size of the i-th device's local dataset, and $\mathbf{w}_i$ is i-th device's local model parameters.

B. Shamir's Secret Sharing

In PDLRecover, we employ a secret sharing algorithm to secure each client's local update g_t^i, during the collection of the aggregated model parameters $\mathbf{w}_t$, which are essential for model recovery. Specifically, we use the Shamir's secret sharing to facilitate secure exchange of local updates.

Shamir's secret sharing [8] is a cryptographic method that divides a secret into multiple shares, which are distributed among a group of clients. The secret can only be reconstructed when a required number of shares are combined, ensuring that individual client updates remain private. This technique enables secure aggregation during the model recovery process while protecting the confidentiality of each client's data.

The details of Shamir's secret sharing are as follows:

1) Secret Distribution: A client j shares a secret s among n clients, such that any t of n clients can reconstruct the secret s. To achieve this, the client j uses a $t-1$ degree polynomial to generate the shares.

- Select a random polynomial

$$f(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_{t-1} x^{t-1}, \tag{2}$$

  where $a_0 = s$ is the secret, and $a_1, a_2, \ldots, a_{t-1}$ are randomly chosen coefficients.
- For each client i, where $i = 1, 2, \ldots, n$, the client j selects a random x_i and computes $f(x_i)$.
- The share given to the client i is the point $(x_i, f(x_i))$.

2) Secret Reconstruction: Any group of at least t clients can reconstruct the secret using their shares. This is done using the Lagrange interpolation.

- Collect at least t shares $(x_1, y_1), (x_2, y_2), \ldots, (x_t, y_t)$.
- Use Lagrange interpolation to reconstruct polynomial as

$$f(x) = \sum_{i=1}^{t} y_i \prod_{1 \le j \le t, j \ne i} \frac{x - x_j}{x_i - x_j}. \tag{3}$$

- Evaluate this polynomial at $x = 0$ to find the secret $s = f(0) = \sum_{i=1}^{t} y_i \prod_{1 \le j \le t, j \ne i} \frac{-x_j}{x_i - x_j}$.

C. L-BFGS Method

The L-BFGS algorithm [39] plays a crucial role in efficiently computing the approximate Hessian-vector product (HVP), significantly reducing the computational complexity of large-scale optimization problems by avoiding direct Hessian matrix calculations. It achieves this by leveraging two key buffers: a global-model difference buffer, $\widetilde{W} = [\Delta \mathbf{W}_{b_1}, \Delta \mathbf{W}_{b_2}, \ldots, \Delta \mathbf{W}_{b_s}]$, which tracks changes in global model parameters across iterations, and a local update difference buffer, $\widetilde{G}^i = [\Delta G^i_{b_1}, \Delta G^i_{b_2}, \ldots, \Delta G^i_{b_s}]$, which stores gradient updates in each iteration, s is the buffer size. An input vector, $\mathbf{v}$, is also used in the computation process. The L-BFGS algorithm then uses $\widetilde{W}_t$ and $\widetilde{G}^i_t$ as inputs to generate an approximate Hessian matrix $\tilde{\mathbf{H}}^i_t$ for the i-th client in the t-th iteration. This process is described as $\mathbf{B}^i_t = \text{L-BFGS}(\widetilde{W}_t, \widetilde{G}^i_t)$.

IV. BACKGROUND

A. Notation

We first define the notations (shown in Table I) that are used to construct PDLRecover.

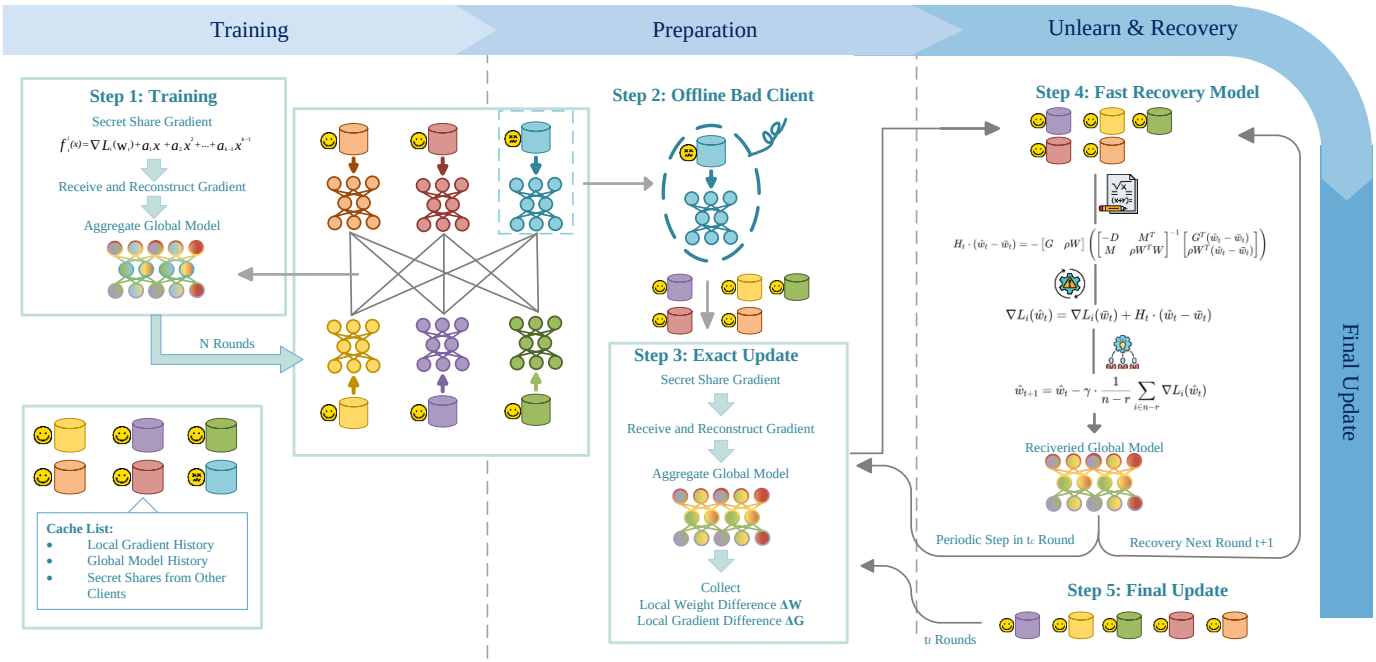

Fig. 1: An overview of PDLRecover

B. PDLRecover Background

In decentralized learning, suppose each client has a local dataset D_i with n samples to train a global machine learning model. The loss function for a client i is defined as

$$L_i(\mathbf{w}) = \frac{1}{n}\sum_{j=1}^{n} L_{ij}(\mathbf{w}), \tag{4}$$

where $\mathbf{w}$ is the local model parameter, and $L_{ij}(\mathbf{w})$ is the j-th sample's loss function for the client i. The gradient of $L(\mathbf{w})$ for the client i is

$$\nabla L_i(\mathbf{w}) = \frac{1}{n}\sum_{j=1}^{n} \nabla L_{ij}(\mathbf{w}). \tag{5}$$

Model parameters are aggregated through FedAvg in each iteration $t = 1, \ldots, T$ as

$$\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t - \frac{\gamma_t}{n}\sum_{i\in n} \nabla L_i(\mathbf{w}_t), \tag{6}$$

where n is the number of clients and γ_t is the learning rate at iteration t.

If a subset $P = \{i_1, i_2, \cdots, i_p\}$ of clients drop out or are identified as dishonest, submitting malicious updates, the updates from dropped clients can be removed as:

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \frac{\gamma_t}{n-p}\left[\sum_{i\in n} \nabla L_i(\mathbf{w}_t) - \sum_{i\in P} \nabla L_i(\mathbf{w}_t)\right], \tag{7}$$

where p is the size of P.

However, computing $\sum_{i\in P} \nabla L_i(\mathbf{w}_t)$ is impossible since updates from dropped clients are unavailable. The model parameters can be updated using only the remaining clients' updates

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \frac{\gamma_t}{n-p}\sum_{i\notin P} \nabla L_i(\mathbf{w}_t). \tag{8}$$

Suppose the client i caches model parameters $\{\bar{\mathbf{w}}_0, \bar{\mathbf{w}}_1, \cdots, \bar{\mathbf{w}}_t\}$ and gradients $\{\nabla L_i(\mathbf{w}_0), \nabla L_i(\mathbf{w}_1), \cdots, \nabla L_i(\mathbf{w}_t)\}$ for each iteration. Using Cauchy's Mean Value Theorem, the unlearned local updated $\nabla L_i(\hat{\mathbf{w}}_t)$ for each iteration can be estimated as

$$\nabla L_i(\hat{\mathbf{w}}_t) = \nabla L_i(\bar{\mathbf{w}}_t) + \mathbf{H}_t \cdot (\hat{\mathbf{w}}_t - \bar{\mathbf{w}}_t), \tag{9}$$

where $\mathbf{H}_t$ is an integrated Hessian

$$\mathbf{H}_t = \int_0^1 \mathbf{H}(\bar{\mathbf{w}}_t + x(\hat{\mathbf{w}}_t - \bar{\mathbf{w}}_t))\, dx. \tag{10}$$

This approach enables accurate estimation of updates during the recovery process while maintaining efficiency and client privacy.

According to Equation 9, we propose PDLRecover, a novel method to address the challenges posed by detected malicious or disconnected clients during distributed training. PDLRecover effectively removes compromised and dropped clients, reinitialized the global model, and restores its performance using stored historical data, thereby eliminating the need to retrain from scratch.

To facilitate efficient model recovery, each client collects valuable historical data during the training process, including the global model parameters and client model updates. PDLRecover enables each client to gather this information from neighboring clients during the training process, supporting the estimation of clients' model updates during recovery while ensuring privacy and model performance. The main framework of PDLRecover is shown in Fig. 1.

In terms of privacy, we employ a secret-sharing mechanism to protect client data. During training, clients share their local model updates in a secret-shared manner. During the recovery phase, each client broadcasts reconstructed model updates to facilitate global model restoration. While this approach

imposes certain constraints on our aggregation technique and introduces some computational overhead, the benefits of maintaining client privacy outweigh these costs.

The core of PDLRecover lies in enabling clients to communicate and collaborate in estimating global model updates during recovery in decentralized learning. Clients store not only their own training history but also global model updates, including corrupted updates introduced by malicious clients. During recovery, we estimate the model updates for each client in each iteration using Cauchy's Mean Value Theorem. Although calculating the integrals required by this theorem is complex, we overcome this challenge by estimating the Hessian matrix using the extended L-BFGS technique. While this approach necessitates certain computational and storage capabilities from clients, the benefits in terms of recovery accuracy outweigh the associated costs.

C. Threat Model

Attacker's Goals. The primary objective of an untargeted poison attack is to indiscriminately increase the global model's test error rate across a significant number of test inputs. In contrast, a targeted poison attack aims to manipulate the global model to produce inaccurate predictions specifically for certain target labels chosen by the attacker, while still maintaining accuracy for other test inputs. Backdoor attacks are a form of targeted attack that involve inserting a unique trigger, such as a specific feature pattern, into the target test inputs.

Attacker's Capabilities. An attacker has controled over specific malicious clients while maintaining the integrity of honest clients. Malicious clients can either be fabricated entities introduced by the attacker or compromised legitimate clients within the decentralized learning system. They possess the ability to send arbitrary model updates to the server, thereby impacting the overall learning process.

Attacker's Background Knowledge. The attacker's background knowledge can be classified into two distinct settings: partial-knowledge and full-knowledge settings [33].

- Partial-Knowledge Setting: An attacker possesses knowledge of the global model, the loss function, and has access to local training data and model updates on the malicious clients only.
- Full-Knowledge Setting: An attacker possesses a thorough understanding of the local training data and model updates from all clients, as well as knowledge of the aggregation rules. This level of insight significantly enhances the effectiveness of poison attacks compared to a partial-knowledge scenario, enabling the attacker to devise more potent strategies for corrupting the model.

In summary, the effectiveness of poison attacks depends on the attacker's objectives, the degree of control over malicious clients, and the depth of knowledge about the decentralized learning system. Understanding these factors is essential for developing robust defense mechanisms.

D. Design Goals

Our goal is for PDLRecover to perform comparably to the drop client retraining method while significantly outperforming the historical information retraining method. Additionally, PDLRecover should minimize computational and communication overhead for clients and protect the privacy of honest participants. The key design goals include:

Accuracy: PDLRecover must recover an accurate global model from poison attacks, maintaining performance similar to the drop client retraining method. The accuracy should remain high, with minimal impact from the number of malicious clients up to a certain threshold.

Efficiency: PDLRecover should lower the computational and communication load on clients, requiring minimal rounds for clients to compute local updates while effectively adjusting their models. The server's processing and storage overhead should also be kept within practical limits.

Privacy: The privacy of clients must be safeguarded during the recovery process, even when local updates are shared for aggregation. We aim to design a privacy-preserving scheme that can be integrated with the recovery method to ensure that clients' privacy is maintained during the recovery process.

Independence from detection methods: PDLRecover should be a versatile recovery method compatible with various malicious client detection techniques. It should leverage existing models that identify client behavior to facilitate recovery and remain resilient, ensuring accuracy and stability even if some malicious clients go undetected or honest clients are mistakenly flagged.

V. PDLRecover

In this section, we present our PDLRecover, as shown in Algorithm 1, including prepareation, recovery, periodic step, and final exact update. The proposed design integrates exact training, gradient recovery through the extended L-BFGS algorithm, and secure reconstruction of model updates through secret sharing. This hybrid strategy ensures resilience in the face of partial client dropout while preserving privacy and maintaining high recovery fidelity.

Preparation. In the preparation phase, all remaining clients actively participate in federated training. At each iteration t, client i computes the local gradient $\nabla L_i(\bar{w}_t)$ and engages in a secret sharing protocol to ensure privacy of its update. Specifically, client i generates a random polynomial of degree at most n:

$$f_i^t(x) = a_{i,0}^t + a_{i,1}^t x + a_{i,2}^t x^2 + \cdots + a_{i,n}^t x^n, \tag{11}$$

where the constant term encodes the private local gradient, i.e., $a_{i,0}^t = \nabla L_i(\bar{w}_t)$, and the remaining coefficients $a_{i,1}^t, a_{i,2}^t, \cdots, a_{i,n}^t$ are selected uniformly at random from a finite field $\mathbb{Z}_q$.

To construct shares, each client j computes $f_i^t(x_j)$ and sends it securely to client i. After receiving shares from all other clients, client i aggregates the sub-secret gradient values:

$$\nabla L^{(x_i)}(\hat{w}_t) = \sum_{j=1}^{n} f_j^t(x_i). \tag{12}$$

Then, each client reconstructs the global gradient using Lagrange interpolation:

$$\nabla L(\bar{w}_t) = \sum_{i=1}^{n} \nabla L^{(x_i)}(\hat{w}_t) \cdot \prod_{\substack{1 \le j \le n \\ j \ne i}} \frac{0 - x_j}{x_i - x_j}. \tag{13}$$

Algorithm 1 PDLRecover

Require: Clients $C_r = C_i \mid m+1 \le i \le n$; initial model $\bar{w}_0$; learning rate γ; original global models $\hat{\mathbf{w}}_0, \hat{\mathbf{w}}_1, \ldots, \hat{\mathbf{w}}_T$; sub-secret of client j's local model $\nabla L^{(x_i)}(\bar{w}_0), \nabla L^{(x_i)}(\bar{w}_1), \ldots, \nabla L^{(x_i)}(\bar{w}_t)$; periodic step interval T_r; final exact steps T_f; SS-L-BFGS buffer size s; share points x_i; total rounds T.

Ensure: Final recovered model $\hat{w}_T$

$\hat{w}_0 \leftarrow \bar{w}_0$
for $t = 0$ to $T_p - 1$ **do**
  $\hat{w}_{t+1} \leftarrow \text{ExactUpdate}(C_r, \hat{w}_t, \gamma)$ ▷ preparation step
end for
for $t = T_p$ to $T - T_f - 1$ **do**
  if $(t - T_p + 1) \bmod T_r = 0$ **then** ▷ Periodic step
    $\nabla L^{(x_i)}(\hat{w}_t)$, $\hat{w}_{t+1} \leftarrow \text{ExactUpdate}(C_r, \hat{w}_t, \gamma)$
    for each client C_i **do**
      $\Delta G_t^{(x_i)} = \nabla L^{(x_i)}(\hat{w}_t) - \nabla L^{(x_i)}(\bar{w}_t)$
      $\Delta W_t = \hat{w}_t - \bar{w}_t$
      $\widetilde{G}_t^{(x_i)} \leftarrow \widetilde{G}_t^{(x_i)} \cup \{\Delta G_t^{(x_i)}\}$
      $\widetilde{W}_t \leftarrow \widetilde{W}_t \cup \{\Delta W_t\}$
    end for
  else
    for each client C_i **do**
      $\widetilde{H}_i^t \mathbf{v} \leftarrow \text{SS-L-BFGS}(\widetilde{W}_t, \widetilde{G}_t^{(x_i)}, \hat{w}_t - \bar{w}_t)$
      $\hat{g}_t^{(x_i)} = \nabla L^{(x_j)}(\bar{w}_t) + \widetilde{H}_i^t \mathbf{v}$
    end for
    $\hat{g}^t = \sum_{j=1}^{t} \hat{g}_t^{(x_i)} \cdot \prod_{\substack{1 \le i \le t \\ i \ne j}} \frac{0 - x_i}{x_j - x_i}$
    $\hat{w}_{t+1} \leftarrow \hat{w}_t - \frac{\gamma}{n} \cdot \hat{g}^t$
  end if
end for
for $t = T - T_f$ to $T - 1$ **do** ▷ Final exact recovery
  $\hat{w}_{t+1} \leftarrow \text{ExactUpdate}(C_r, \hat{w}_t, \gamma)$
end for
return $\hat{w}_T$

ExactUpdate:
Client i **computes and share local updates:**
  $g_i = \nabla L_i(\mathbf{w}) = \frac{1}{n} \sum_{i=1}^{n} \nabla L_{ij}(\mathbf{w})$
  Generate polynomial for local updates $\nabla L(\bar{\mathbf{w}}_t^i)$
  Compute secret shares $(x_i^t, f_j^t(x_i))$ for each client i
Client i **aggregates received shares:**
  Upon receiving shares $\{(x_i, f_j^t(x_i))\}$ from all clients,
  Compute polynomial sum $\nabla L^{(x_j)}(\bar{w}_t) = \sum_{j \in n} f_j^t(x_i)$
Client i **reconstructs local model update:**
  $\nabla L(\hat{\mathbf{w}}_t) = \sum_{i=1}^{t} l_i^t \prod_{1 \le j \le t, j \ne i} \frac{x - j}{i - j}$
Client i **updates model:**
  $\hat{\mathbf{w}}_{t+1} = \hat{\mathbf{w}}_t - \frac{\gamma \cdot \nabla L(\hat{\mathbf{w}}_t)}{n}$

The reconstructed gradient is used to update the reference model:

$$\bar{w}_{t+1} = \bar{w}_t - \frac{\gamma}{n} \cdot \nabla L(\bar{w}_t). \tag{14}$$

This involves calculating the precise global gradient and updating the exact model parameters, while simultaneously constructing local buffers that store global model differences and sub-secret gradient differences, denoted by $\widetilde{W}_t$ and $\widetilde{G}_t^{(x_i)}$.

Algorithm 2 SS-L-BFGS

Require: Local model difference buffer $\widetilde{W}^{(x_j)} = [\Delta \mathbf{w}_1^{(x_j)}, \cdots, \Delta \mathbf{w}_s^{(x_j)}]$, Local gradient difference buffer $\widetilde{G}^{(x_j)} = [\Delta \mathbf{g}_1^{(x_j)}, \cdots, \Delta \mathbf{g}_s^{(x_j)}]$, A direction vector $\mathbf{v}$

Ensure: Local approximated HVP $\widetilde{\mathbf{H}}^{(x_j)} \mathbf{v}$

$\mathbf{A} = (\widetilde{W}^{(x_j)})^T \widetilde{G}^{(x_j)}$
$\mathbf{D} = \text{diag}(\mathbf{A})$
$\mathbf{M} = \text{lower-triangular}(\mathbf{A})$
$\rho = (\Delta \mathbf{g}_s^{(x_j)})^T \Delta \mathbf{w}_s^{(x_j)} / (\Delta \mathbf{w}_s^{(x_j)})^T \Delta \mathbf{w}_s^{(x_j)}$
$\mathbf{p} = \begin{bmatrix} -\mathbf{D} & \mathbf{M}^T \\ \mathbf{M} & \rho(\widetilde{W}^{(x_j)})^T \widetilde{W}^{(x_j)} \end{bmatrix}^{-1} \begin{bmatrix} \widetilde{G}^{(x_j)} \mathbf{v} \\ \rho(\widetilde{W}^{(x_j)})^T \mathbf{v} \end{bmatrix}$
$\widetilde{\mathbf{H}}^{(x_j)} \mathbf{v} = \rho \mathbf{v} - \begin{bmatrix} \widetilde{G}^{(x_j)} & \rho \widetilde{W}^{(x_j)} \end{bmatrix} \cdot \mathbf{p}$
return $\widetilde{\mathbf{H}}^{(x_j)} \mathbf{v}$

Because the recovery process spans multiple iterations, the buffers initialized during the early stages of training may become stale. Stale buffers can lead to inaccurate approximations of the Hessian matrix, erroneous model update estimates, and ultimately compromise the accuracy of the recovered global models. To address this limitation, we require each client to repeat the preparation step every T_r iterations. This regular refresh of the buffer state ensures that second-order information remains representative of the current model landscape, thereby improving both the quality and stability of the recovery process.

To support approximate second-order recovery, each client C_j independently constructs local buffers. At each periodic iteration, the client stores its share of the gradient difference and the corresponding model difference:

$$\Delta G_t^{(x_j)} = \nabla L^{(x_j)}(\hat{w}_t) - \nabla L^{(x_j)}(\bar{w}_t), \quad \Delta W_t = \hat{w}_t - \bar{w}_t,$$

and updates:

$$\widetilde{G}_t^{(x_j)} \leftarrow \widetilde{G}_t^{(x_j)} \cup \{\Delta G_t^{(x_j)}\}, \quad \widetilde{W}_t^{(x_j)} \leftarrow \widetilde{W}_t^{(x_j)} \cup \{\Delta W_t\}.$$

Recovery. In the recovery phase, each client computes a local approximated update direction:

$$\hat{g}_t^{(x_j)} = \nabla L^{(x_j)}(\bar{w}_t) + \widetilde{H}^{(x_j)}(\hat{w}_t - \bar{w}_t), \tag{15}$$

where $\widetilde{H}^{(x_j)}$ is computed from $\left(\widetilde{W}^{(x_j)}, \widetilde{G}^{(x_j)}\right)$ using SS-L-BFGS, an extended version of the classical L-BFGS algorithm designed to support Hessian-vector product (HVP) computation over secret shares. This ensures that the approximated update direction can be securely reconstructed from distributed shares, without revealing any client's private information. The detailed procedure is described in Algorithm 2.

Once each client has computed its private direction estimate $\hat{g}_t^{(x_j)}$, the full global direction $\hat{g}_t$ is reconstructed using Lagrange interpolation:

$$\hat{g}_t = \sum_{j=1}^{n} \hat{g}_t^{(x_j)} \cdot \prod_{\substack{1 \le k \le n \\ k \ne j}} \frac{0 - x_k}{x_j - x_k}. \tag{16}$$

Using this aggregated update, the recovered model is updated as:

$$\hat{w}_{t+1} = \hat{w}_t - \frac{\gamma}{n} \cdot \hat{g}_t. \tag{17}$$

During recovery, the buffers $(\widetilde{W}^{(x_j)}, \widetilde{G}^{(x_j)})$ are not modified.

Periodic Step. To maintain the accuracy of the SS-L-BFGS approximation, a periodic exact step is performed every T_r iterations. In this step, each client re-evaluates its gradient share at the current model state $\hat{w}_t$, computes the corresponding gradient difference $\Delta G_t^{(x_j)} = \nabla L^{(x_j)}(\hat{w}_t) - \nabla L^{(x_j)}(\bar{w}_t)$ and model difference $\Delta W_t = \hat{w}_t - \bar{w}_t$, and subsequently refreshes its local buffers by updating $\widetilde{G}_t^{(x_j)} \leftarrow \widetilde{G}_t^{(x_j)} \cup \{\Delta G_t^{(x_j)}\}$ and $\widetilde{W}_t^{(x_j)} \leftarrow \widetilde{W}_t^{(x_j)} \cup \{\Delta W_t\}$. To limit memory usage and maintain relevance, the oldest entries are discarded, preserving a fixed buffer size s. This periodic update guarantees that the curvature information used for Hessian approximation remains fresh and accurately reflects the evolving optimization landscape.

Final Exact Update. To further stabilize the training process and eliminate accumulated approximation errors, PDLRecover concludes with T_f final exact update steps. These steps mirror the standard decentralized training procedure but omit the secret sharing step. The final exact updates restore full precision to the model and ensure that it reaches a stable and accurate final state.

VI. Experiment Results

In this section, we first introduce the datasets, the PDLRecover implementation details, and the baseline methods we compare with. Next, we illustrate the results that implement PDLRecover. Then, we demonstrate the performance of round number of preparation and exact training. Finally, we discuss the comparison between PDLRecover and existing methods.

A. Datasets

We use the following three datasets to implement PDLRecover.

MNIST. The MNIST dataset is a classic computer vision dataset widely used for image classification tasks. It contains 70,000 grayscale images of handwritten digits, each 28x28 pixels in size. Of these, 60,000 images are used for training and 10,000 for testing. Each image corresponds to a numerical label from 0 to 9. In this study, we used the ResNet50 model for the classification task on the MNIST dataset. To accommodate the input requirements of ResNet50, we converted the 28x28 pixel grayscale images into 224x224 pixel, three-channel images suitable for the model. We randomly distributed the MNIST dataset to 200 clients for training, where the independent homogeneity is set to 0.5, which usually ranges from 0.1 to 1.

FashionMNIST. The FashionMNIST dataset contains images of 10 different types of clothing and accessories. This dataset serves as an alternative to the MNIST dataset for more challenging image classification tasks. It consists of 70,000 grayscale images, each 28x28 pixels in size, with 60,000 used for training and 10,000 for testing. We also trained the model using 200 clients with the same method as MNIST.

HAR. The HAR dataset is a standard dataset used for human activity recognition tasks. It consists of signals captured by accelerometers and gyroscopes on a smartphone and is used for recognizing 6 different activities, including walking, walking up and down stairs, sitting, standing, and lying down. The dataset contains a total of 10,299 samples from 30 volunteers. We used 80% of the data as the training set for the clients and the remaining 20% as the test set.

B. Implementation Details

The fully decentralized learning model operates as a synchronized training and fully connected system. The clients utilize the stochastic gradient descent method to train local models, and each client employs the FedAvg algorithm to aggregate weight information for subsequent training rounds. After completing local training, clients must wait for neighboring nodes' iterations to synchronize with their own, ensuring that all clients train in unison.

For the model parameter set up, we utilize specific parameters tailored for each dataset because of the varying characteristics of different datasets. For example, MNIST and FashionMNIST are trained over 1000 rounds with a learning rate of 1.5×10^{-4} and a batch size of 32, whereas HAR is trained over 1000 rounds with a learning rate of 1×10^{-4} and a batch size of 16.

We also set up malicious clients to build a backdoor attack. We assume the number of participating clients is $n = 200$, and the number of malicious clients is k, where $n = 3k - 1$. Therefore, we set the number of malicious clients to be 10, 20, 30, 40, 50, and 60. In MNIST and FashionMNIST, we add red stripes as the trigger to the original images, and we expect the model to recognize all image data with red stripes as birds. In HAR, we set every 20th feature value to 0 as the trigger value, where 0 is the target label.

During the recovery step, the first 25 iterations are designated as the setup step, calibration is processed every 30 iterations, and the last 25 rounds are considered as the stabilization step. We set the SS-L-BFGS buffer size to 4.

C. Baseline

To evaluate the performance of PDLRecover, we setup two baseline methods:

- Drop client retrain: The malicious clients are removed, the remaining clients initialized and retrain the whole model with the default model parameters and rounds.
- Historical information: The malicious clients are removed and the remaining clients use historical information storing in the cache to reconstruct the global. They default a same initialized model, and recovery the model directly.

D. Experiment Results

Fig. 2 shows the accuracy of poison attacks, Drop Client Retrain, Model Before Recovery, and PDLRecover for the MNIST, FashionMNIST, and HAR datasets. It can be observed that the Drop Client Retrain and PDLRecover methods can recover the poisoned model, and PDLRecover achieves a similar effect as the Drop Client Retrain method. However, the Model Before Recovery results in lower accuracy, around 50%. To confirm the efficiency of PDLRecover, we also show the running time for different methods in Table II, where it

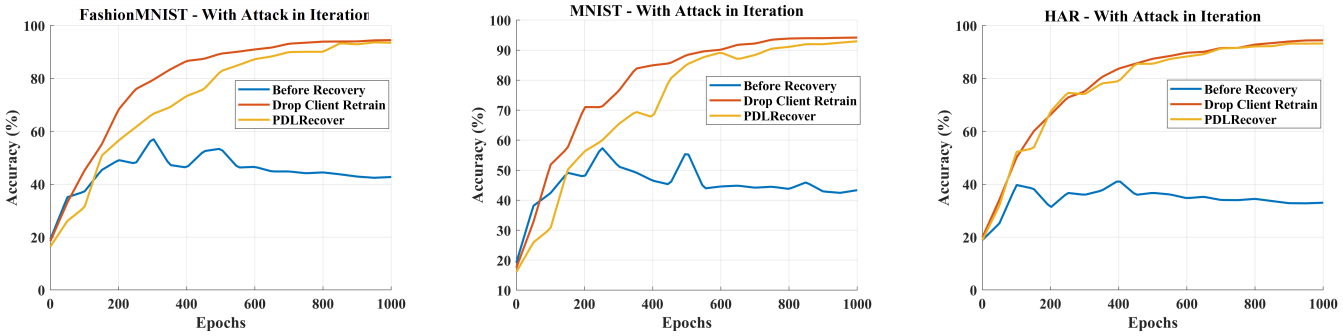

Fig. 2: Accuracy of the recovery strategy under attack on MNIST, FashionMNIST, HAR datasets

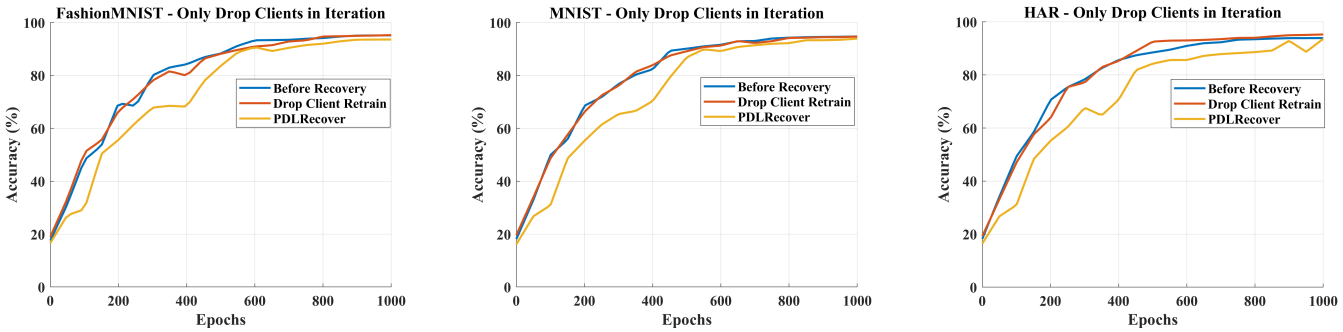

Fig. 3: Accuracy of the recovery strategy under client drop on MNIST, FashionMNIST, HAR datasets

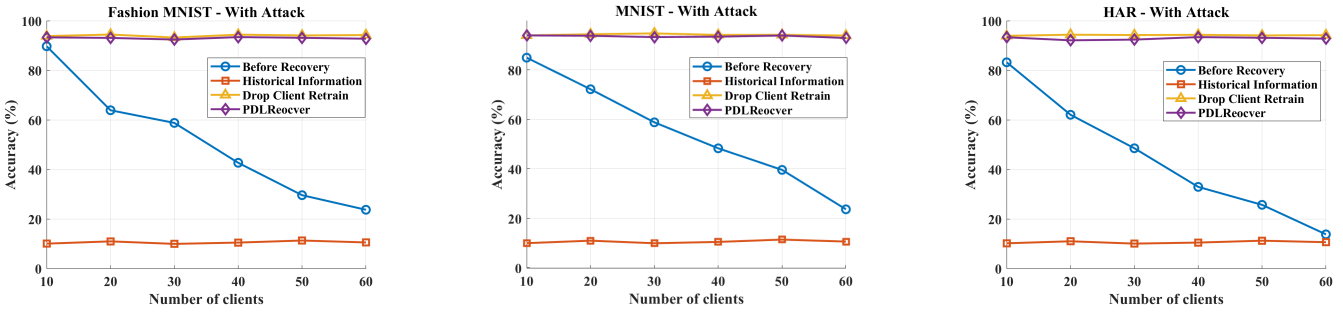

Fig. 4: Model accuracy under different recovery strategy with different number of malicious clients

		Number of clients					
Dataset Name		**10**	**20**	**30**	**40**	**50**	**60**
MNIST	Client Retrain	679	682	703	695	682	607
	PDLRecover	439	441	449	404	419	417
FashionMNIST	Client Retrain	707	706	707	711	708	710
	PDLRecover	433	423	438	416	462	465
HAR	Client Retrain	1235	1212	1227	1217	1246	1250
	PDLRecover	875	887	867	893	886	901

TABLE II: Running Time (seconds) for MNIST, FashionMNIST, and HAR with attack

		Number of clients					
Dataset Name		**10**	**20**	**30**	**40**	**50**	**60**
MNIST	Client Retrain	682	686	694	691	675	672
	PDLRecover	447	435	454	413	422	416
FashionMNIST	Client Retrain	702	712	704	705	707	718
	PDLRecover	426	414	428	426	461	461
HAR	Client Retrain	1241	1249	1223	1218	1248	1263
	PDLRecover	871	879	876	897	876	902

TABLE III: Running Time (seconds) for MNIST, FashionMNIST, and HAR only drop clients

can be noticed that PDLRecover saves 33% of the running time on average compared to the Drop Client Retrain method.

Fig. 3 show the accuracy of Model Before Recovery , Drop Client Retrain, and PDLRecover for the MNIST, FashionMNIST, and HAR datasets when all clients are honest. When the dropped clients are removed and the remaining clients apply Drop Client Retrain and PDLRecover, both methods maintain better performance, with accuracy similar to the Model Before Recovery result. However, Table III shows that the running time of PDLRecover is 35% lower on average compared to the Drop Client Retrain method.

Fig. 4 shows the effect of different numbers of malicious clients on the models. It can be observed that the accuracy of the poisoned model linearly decreases with the increase

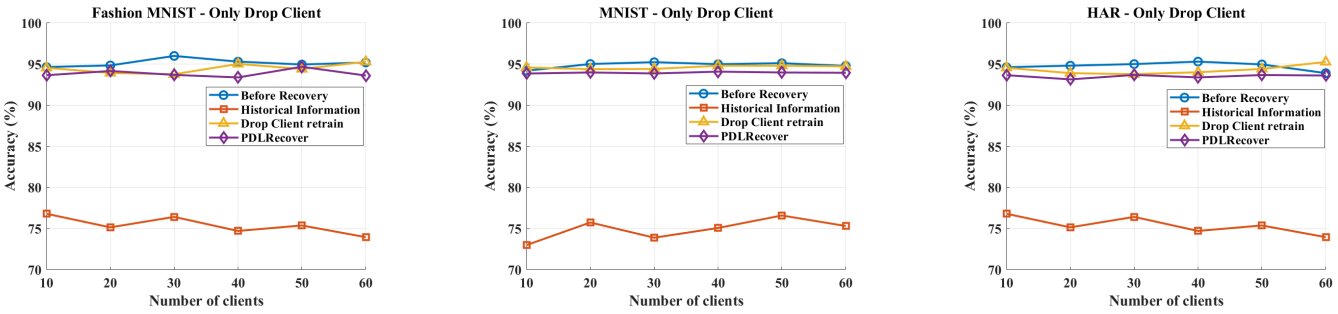

Fig. 5: Model accuracy under different recovery strategy with different drop client number

Datasets	Retrain	[14]	[40]	[35]	Ours
MNIST	95.4	95.2	95.3	94.7	**93.9**
FashionMNIST	94.9	86.5	87.4	85.8	**93.2**
HAR	94.6	84.4	86.5	87.2	**93.4**

TABLE IV: Accuracy(%) of unlearning methods comparison

in the number of malicious clients. When the number of malicious clients reaches its maximum, the training accuracy of the MNIST and FashionMNIST training sets stays around 24%, while the HAR training set accuracy is around 13%, indicating that the increase in malicious clients greatly impacts the model's performance.

At the same time, the number of malicious clients does not affect the model's final performance when we recover the poisoned model using retraining and PDLRecover methods. The accuracy of the Drop Client Retrain method stays around 93% in the MNIST and FashionMNIST datasets, while it remains around 94% in HAR. PDLRecover stays around 91% in the MNIST and FashionMNIST datasets and around 92% in HAR. However, according to Table II, PDLRecover can save 35% of the running time.

Fig. 5 show all the clients are honest, maximum 60 clients want to dropped out, the remaining clients use the methods of drop client retrain, historical information, or PDLRecover to remove the impact of the dropped clients. Our experiments show that the drop client retrain and PDLRecover can keep the performance of the model, but the historical information method decreases the accuracy of the model to around 75%. At the same time, Table III shows the running time of drop client retrain and PDLRecover, and PDLRecover still can save around 30% of the running time. Therefore, PDLRecover can more efficiently eliminate the impact of dropped clients on the whole model and maintain the performance.

E. Round of Recovery Preparation and Exact Training

Fig. 6 illustrates the accuracy variations of the final model across different epochs during the preparation step and the final training step. As shown in Fig. 6, the accuracy curves for both steps are close in most epochs, fluctuating between 90% and 95%. While the accuracy during the preparation step shows minor fluctuations, it remains generally sTable The accuracy during the final training step is slightly more stable, but the difference is minimal. Overall, the model maintains high accuracy throughout the training process, indicating that

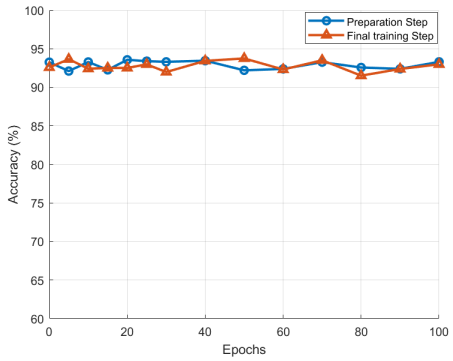

Fig. 6: Effect of preparation Step and final training step in PDLRecover

both the preparation step and the final training step enable the model to effectively learn and sustain high predictive performance.

Furthermore, the number of epochs in both steps has little impact on overall model performance, with excessive epochs merely increasing memory usage, computational resources, and time. Thus, only minimal additional training is required to achieve optimal performance, making it unnecessary to set a large number of epochs to ensure model stability. PDLRecover sets both the preparation step and the final training step to 25 epochs each, ensuring model performance stability while optimizing resource and time efficiency.

F. Comparisons

Table IV provides a comparative accuracy analysis for various unlearning methods across three datasets: MNIST, FashionMNIST, and HAR. It can be seen that our PDLRecover demonstrates significant improvement in accuracy, particularly on the FashionMNIST and HAR datasets. Specifically, for FashionMNIST, PDLRecover achieves an accuracy of 93.2%, which is markedly higher than the accuracies reported by [14] (86.5%), [40] (87.4%), and [35] (85.8%), and closely matches the retrain method's accuracy of 94.9%. Similarly, for HAR dataset, PDLRecover achieves an accuracy of 93.4%, significantly surpassing the accuracies from [14] (84.4%), [40] (86.5%), and [35] (87.2%), and closely matching the retrain method's accuracy of 94.6%. For MNIST dataset, PDLRecover attains an accuracy of 93.9%, which is slightly lower than the retrain method (95.4%) and comparable to the accuracies of [14] (95.2%) and [40] (95.3%).

These results indicate that PDLRecover is highly effective in restoring model performance, achieving accuracies that are competitive or superior to existing methods. Moreover, PDLRecover demonstrates robustness against poison attacks, ensuring stable and accurate model recovery across different datasets. This highlights the efficacy and reliability of PDLRecover in practical scenarios, making it a valuable contribution to decentralized learning.

VII. Security Analysis

A. Correctness

Theorem: Let there be k clients, each holding a secret $a_{i,0}$ encoded as the constant term of a degree-$(t_{\text{th}}-1)$ polynomial $f_i(x) = a_{i,0}+a_{i,1}x+\ldots+a_{i,t_{\text{th}}-1}x^{t_{\text{th}}-1}$ defined over a finite field $\mathbb{F}q$. Each client i generates n shares $si, 1, \ldots, s_{i,n}$ by evaluating f_i at $n \geq t_{\text{th}}$ mutually distinct nonzero points $x_1, \ldots, x_n \in \mathbb{F}_q$.

We prove that the result of aggregating all clients' secret shares followed by Lagrange interpolation is equal to the sum of the individually reconstructed secrets

$$\text{Lagrange}\left(\sum_i \text{shares}_i\right) = \sum_i \text{Lagrange}(\text{shares}_i). \tag{18}$$

Step 1: Reconstructing Each Client's Secret: Each client's secret can be recovered via Lagrange interpolation at $x = 0$

$$a_{i,0} = f_i(0) = \sum_{j=1}^{n} s_{i,j} \cdot \ell_j(0), \tag{19}$$

where $\ell_j(0) = \prod_{1\leq m\leq n\ m\neq j} \frac{-x_m}{x_j-x_m}$ denotes the j-th Lagrange basis polynomial evaluated at zero.

Step 2: Aggregating Before Reconstruction: Define the pointwise sum of all clients' polynomials as

$$F(x) = \sum_{i=1}^{k} f_i(x) = \sum_{\ell=0}^{t_{\text{th}}-1} \left(\sum_{i=1}^{k} a_{i,\ell}\right) x^{\ell}. \tag{20}$$

Evaluating F at $x = 0$ yields the aggregate secret

$$F(0) = \sum_{i=1}^{k} a_{i,0}. \tag{21}$$

Step 3: Verifying Linearity of Interpolation: Rewriting the expression for the total reconstructed secret:

$$\sum_{i=1}^{k} a_{i,0} = \sum_{i=1}^{k} \left(\sum_{j=1}^{n} s_{i,j} \cdot \ell_j(0)\right) = \sum_{j=1}^{n} \left(\sum_{i=1}^{k} s_{i,j}\right) \cdot \ell_j(0). \tag{22}$$

This matches the result of interpolating the aggregated shares $\sum_{i=1}^{k} s_{i,j}$ evaluated at $x = 0$, that is,

$$F(0) = \sum_{j=1}^{n} \left(\sum_{i=1}^{k} s_{i,j}\right) \cdot \ell_j(0). \tag{23}$$

Therefore, Lagrange interpolation is linear with respect to share aggregation.

Extension to SS-L-BFGS Approximation: In PDLRecover, each client C_j computes a local directional gradient approximation using SS-L-BFGS, the extended L-BFGS method,

$$\hat{g}_t^{(x_j)} = \nabla L^{(x_j)}(\bar{w}_t) + \widetilde{H}^{(x_j)}(\hat{w}_t - \bar{w}_t), \tag{24}$$

where $\widetilde{H}^{(x_j)}$ denotes the local approximation HVP, computed using the client's curvature buffer $(\widetilde{W}^{(x_j)}, \widetilde{G}^{(x_j)})$.

Each component of $\hat{g}_t^{(x_j)}$ is secret-shared across clients. The server aggregates these shares and performs Lagrange interpolation to recover the global gradient approximation

$$\hat{g}_t = \sum_{j=1}^{n} \hat{g}_t^{(x_j)} \cdot \ell_j(0) = \sum_{j=1}^{n} \left[\nabla L^{(x_j)}(\bar{w}_t) + \widetilde{H}^{(x_j)}(\hat{w}_t - \bar{w}_t)\right] \cdot \ell_j(0). \tag{25}$$

By linearity of interpolation,

$$\hat{g}_t = \underbrace{\sum_{j=1}^{n} \nabla L^{(x_j)}(\bar{w}_t) \cdot \ell_j(0)}_{\nabla L(\bar{w}_t)} + \underbrace{\sum_{j=1}^{n} \widetilde{H}^{(x_j)}(\hat{w}_t - \bar{w}_t) \cdot \ell_j(0)}_{\widetilde{H}(\hat{w}_t - \bar{w}_t)}, \tag{26}$$

which produces

$$\hat{g}_t = \nabla L(\bar{w}_t) + \widetilde{H}(\hat{w}_t - \bar{w}t), \tag{27}$$

where the global Hessian approximation is defined as $\widetilde{H} = \sum j = 1^n \widetilde{H}^{(x_j)} \cdot \ell_j(0)$.

Remarks on Quantization and Field Arithmetic: To enable arithmetic over a finite field, all real-valued vectors are quantized locally into fixed-point integers using k-bit precision and subsequently mapped into the finite field $\mathbb{F}_q$. Both secret sharing and Lagrange interpolation are then performed over $\mathbb{F}_q$ to ensure algebraic consistency.

Conclusion: Both scalar secrets and vector-valued approximations, such as those produced by SS-L-BFGS, preserve the linear homomorphism of shamir secret sharing and the linearity of Lagrange interpolation at $x = 0$. Therefore, we have

$$\text{Lagrange}\left(\sum_j \text{shares}_j\right) = \sum_j \text{Lagrange}(\text{shares}_j). \tag{28}$$

This confirms that the secure aggregation of local SS-L-BFGS approximations within the PDLRecover framework is mathematically sound and preserves gradient fidelity, without requiring the clients to disclose their individual updates.

B. Privacy Analysis

In our security analysis, we categorize attackers into two types. The first type is an external attacker. External attackers may attempt to attack buffers stored by clients, which they are not authorized to access. This type of attacks concerns the confidentiality of client content. We will demonstrate that external attackers gain no advantage from such attacks, as they cannot access complete local update information. The second type is an internal attacker, who is authorized to know the subsecrets of other clients. Internal attackers may attempt to attack local updates from honest clients or tamper with the recovered data. Since any client can request data

reconstruction, we must ensure that internal attackers cannot access the personal local update information of other clients.

The following theorems prove that our protocol meets the security goals.

Theorem 1 (External Attacks). Suppose an attacker steals the local update subsecret stored by a client. In that case, the attacker cannot obtain complete local update information nor can they reconstruct the local information with any client.

Proof. An attacker can target the local update sets stored by individual clients, only a collusion attack involving the compromise of buffers from k clients would allow them to recover the sum of the local updates. However, they cannot recover the individual local updates of any single client. This security feature theoretically ensures the security of local updates.

Theorem 2 (Internal Attacks). Suppose malicious clients attempt to attack local updates from other honest clients or try to tamper with the recovered data. In that case, the malicious clients cannot obtain complete local update information nor they can reconstruct the local information with any client.

Proof. Although any client can attack the local update sets stored by other clients, only if more than k malicious clients collude by separately sending the local update subsecret of individual clients can they compute the local updates of honest clients. For the local update secret sharing algorithm in this method, each client saves the local update set $\{(x_i, f_j(x_i))\}$ during training. However, during the reconstruction of local update information, each client calculates the sum of the subsecrets of the remaining clients $f(x_i) = \sum_{j\in n} f_j(x_i)$, without revealing the subsecret $f_j(x_i)$ shared by an individual client. Therefore, even though a client can reconstruct the polynomial $f_j(x)$ to obtain the personal local update of client j, it requires the collusion of more than k malicious clients.

C. Computation and Communication Costs for Clients

Updating the client computational model incurs both computational and communication costs. These costs can be considered fixed unit expenses, as they do not vary based on the specific round in which the client calculates the model update. In the drop client retraining scenario, the average computational and communication cost is $O(T)$, where T represents the total number of iterations. This is due to the requirement for each client to retrain and update the model during every iteration round.

The number of preparation rounds T_p, the periodic round T_r, and the number of final training rounds T_f determine the cost of the method we propose. It can be deduced that the average computation and communication cost per client in PDLRecover is $O(T_p + T_f + \left\lceil \frac{T-T_p-T_f}{T_r} \right\rceil)$.

1) Bounding the Difference between PDLRecover and Drop Client Retrain in Global Model Recovery: We outline the assumptions that guide our theoretical evaluation. Next, we display the bound on the difference between the global model that our technique recovered and the drop client retrain model.

Assumption 1: The loss function is μ-strongly convex and L-smooth. Formally, for each client i, and for any $\mathbf{w}$ and $\mathbf{w}'$, we have the following inequalities:

$$\langle \mathbf{w} - \mathbf{w}', \nabla L_i(\mathbf{w}) - \nabla L_i(\mathbf{w}') \rangle \geq \mu \|\mathbf{w} - \mathbf{w}'\|^2, \quad (29)$$

$$\langle \mathbf{w} - \mathbf{w}', \nabla L_i(\mathbf{w}) - \nabla L_i(\mathbf{w}') \rangle \geq \frac{1}{\mathcal{L}} \|\nabla L_i(\mathbf{w}) - \nabla L_i(\mathbf{w}')\|^2, \quad (30)$$

where L_i is the loss function for client i, $\langle \cdot, \cdot \rangle$ denotes the inner product of two vectors, and $\| \cdot \|$ represents the ℓ_2 norm of a vector.

Assumption 2: The approximation error of the Hessian-vector product in SS-L-BFGS algorithm is bounded. Formally, each approximate Hessian-vector product satisfies the following condition:

$$\forall i, \forall t, \|\tilde{\mathbf{H}}_t^i(\hat{\mathbf{w}}_t - \bar{\mathbf{w}}_t) + \nabla L_i(\hat{\mathbf{w}}) - \nabla L_i(\bar{\mathbf{w}})\| \leq Z, \quad (31)$$

where Z is a finite positive value.

Theorem 1: Assume the following two conditions are met: all malicious clients have been identified, FedAvg is utilized as an aggregation rule, and the learning rate γ fulfills $\gamma \leq \min\left(\frac{1}{\mu}, \frac{1}{\mathcal{L}}\right)$. The global model recovered by our method and the global model obtained by deleting clients for retraining can therefore be distinguished at any iteration $t > 0$ as follows:

$$\|\hat{\mathbf{w}}_t - \bar{\mathbf{w}}_t\| \leq (\sqrt{1-\gamma\mu})^t \|\hat{\mathbf{w}}_0 - \bar{\mathbf{w}}_0\| + \frac{1-(\sqrt{1-\gamma\mu})^t}{1-\sqrt{1-\gamma\mu}} \gamma M, \quad (32)$$

where $\hat{\mathbf{w}}_t$ and $\bar{\mathbf{w}}_t$ are the global models recovered by PDLRecover and drop client retrain, respectively, in iteration t.

Proof: PDLRecover is to recursively bound the difference in each iteration.

According to Theorem 1, we have

$$\lim_{t\to\infty} \|\hat{\mathbf{w}}_t - \bar{\mathbf{w}}_t\| \leq \frac{\gamma Z}{1-\sqrt{1-\gamma\mu}}. \quad (33)$$

Additionally, we derive the following corollary:

Corollary 1: When the SS-L-BFGS algorithm can accurately compute the Hessian-vector product , the bound on the difference between the global model recovered by PDLRecover and the one recovered by drop client retrain is given by

$$\|\hat{\mathbf{w}}_t - \bar{\mathbf{w}}_t\| \leq (\sqrt{1-\gamma\mu})^t \|\hat{\mathbf{w}}_0 - \bar{\mathbf{w}}_0\|. \quad (34)$$

Therefore, the global model recovered by PDLRecover converges to the one recovered by drop client retrain, i.e., $\lim_{t\to\infty} \hat{\mathbf{w}}_t = \lim_{t\to\infty} \bar{\mathbf{w}}_t$.

2) Trade-off between Difference Bound and Computation/Communication Costs: Based on Corollary 1, we have

$$\|\hat{\mathbf{w}}_T - \mathbf{w}_T\| \leq (\sqrt{1-\gamma\mu})^T \|\hat{\mathbf{w}}_0 - \mathbf{w}_0\|, \quad (35)$$

when PDLRecover runs for T rounds. As T increases, the difference bound decreases exponentially. The computation and communication costs of PDLRecover are linear with T. Therefore, as costs increase, the difference bound decreases exponentially. In other words, we observe an accuracy-cost trade-off for PDLRecover: The recovered global model becomes more accurate (i.e., closer to the drop client retrain model) if more computational and communication cost are spent, that is, the computation and communication costs for clients increase accordingly.

VIII. Conclusion

In this paper, we proposed PDLRecover, a novel decentralized unlearning framework that enables secure and efficient recovery from poison attacks without requiring clients to access or reveal each other's model updates. PDLRecover reconstructs the global model via Lagrange interpolation while ensuring complete local privacy. Each client independently computes its contribution to the recovery direction using only private gradient and curvature information, and the global model update is collaboratively reconstructed without exposing any individual values. Our theoretical analysis and empirical results demonstrate that PDLRecover not only preserves the confidentiality of historical updates but also enables accurate reconstruction of the global model in the presence of malicious or dropped clients. Unlike traditional detection-based defenses, PDLRecover provides a proactive, privacy-preserving solution to mitigating poison attacks in decentralized environments.

In future work, we will extend PDLRecover with advanced secure aggregation techniques to better address a wider variety of attack types and operating environments. We will also focus on developing multi-layered defense mechanisms to further bolster the security and recoverability of the global model. Additionally, we aim to investigate the potential for model recovery with reduced dependency on historical data, which could lower storage requirements and enhance the efficiency of model updates.

References

[1] E. T. M. Beltrán and M. Q. e. Pérez, "Decentralized federated learning: Fundamentals, state of the art, frameworks, trends, and challenges," *IEEE Communications Surveys & Tutorials*, 2023.

[2] A. Hard, K. Rao, R. Mathews, S. Ramaswamy, F. Beaufays, S. Augenstein, H. Eichner, C. Kiddon, and D. Ramage, "Federated learning for mobile keyboard prediction," *arXiv preprint arXiv:1811.03604*, 2018.

[3] M. Fredrikson, E. Lantz, S. Jha, S. Lin, D. Page, and T. Ristenpart, "Privacy in pharmacogenetics: An end-to-end case study of personalized warfarin dosing," in *Proc. of USENIX Security 14*, 2014, pp. 17–32.

[4] R. Shokri, M. Stronati, C. Song, and V. Shmatikov, "Membership inference attacks against machine learning models," in *Proc. of 2017 S&P*, 2017, pp. 3–18.

[5] M. Goldblum, D. Tsipras, C. Xie, X. Chen, A. Schwarzschild, D. Song, A. Mądry, B. Li, and T. Goldstein, "Dataset security for machine learning: Data poisoning, backdoor attacks, and defenses," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 45, no. 2, pp. 1563–1580, 2022.

[6] J. Xu, Z. Wu, C. Wang, and X. Jia, "Machine unlearning: Solutions and challenges," *IEEE Transactions on Emerging Topics in Computational Intelligence*, 2024.

[7] D. Haink, "Hessian eigenvectors and principal component analysis of neural network weight matrices," *arXiv preprint arXiv:2311.00452*, 2023.

[8] A. Shamir, "How to share a secret," *Communications of the ACM*, vol. 22, no. 11, pp. 612–613, 1979.

[9] A. S. Berahas, J. Nocedal, and M. Takác, "A multi-batch l-bfgs method for machine learning," *Advances in Neural Information Processing Systems*, vol. 29, 2016.

[10] L. Bourtoule, V. Chandrasekaran, C. A. Choquette-Choo, H. Jia, A. Travers, B. Zhang, D. Lie, and N. Papernot, "Machine unlearning," in *Proc. of 2021 S&P*, 2021, pp. 141–159.

[11] R. N. Zaeem and K. S. Barber, "The effect of the gdpr on privacy policies: Recent progress and future promise," *ACM Transactions on Management Information Systems (TMIS)*, vol. 12, no. 1, pp. 1–20, 2020.

[12] A. Thudi, H. Jia, I. Shumailov, and N. Papernot, "On the necessity of auditable algorithmic definitions for machine unlearning," in *Proc. of USENIX Security 22*, 2022, pp. 4007–4022.

[13] M. Chen, Z. Zhang, T. Wang, M. Backes, M. Humbert, and Y. Zhang, "When machine unlearning jeopardizes privacy," in *Proc. of CCS '21*, 2021, pp. 896–911.

[14] C. Guo, T. Goldstein, A. Hannun, and L. Van Der Maaten, "Certified data removal from machine learning models," *arXiv preprint arXiv:1911.03030*, 2019.

[15] Y. Wu, E. Dobriban, and S. Davidson, "Deltagrad: Rapid retraining of machine learning models," in *Proc. of ICML 2020*, 2020, pp. 10 355–10 366.

[16] A. Golatkar, A. Achille, and S. Soatto, "Eternal sunshine of the spotless net: Selective forgetting in deep networks," in *Proc. of CVPR 2020*, 2020, pp. 9304–9312.

[17] N. Su and B. Li, "Asynchronous federated unlearning," in *Proc. of INFOCOM 2023*, 2023, pp. 1–10.

[18] Y. Liu, L. Xu, X. Yuan, C. Wang, and B. Li, "The right to be forgotten in federated learning: An efficient realization with rapid retraining," in *Proc. of INFOCOM 2022*, 2022, pp. 1749–1758.

[19] J. Martens, "New insights and perspectives on the natural gradient method," *Journal of Machine Learning Research*, vol. 21, no. 146, pp. 1–76, 2020.

[20] C. Wu, S. Zhu, and P. Mitra, "Federated unlearning with knowledge distillation," *arXiv preprint arXiv:2201.09441*, 2022.

[21] G. Liu, X. Ma, Y. Yang, C. Wang, and J. Liu, "Federaser: Enabling efficient client-level data removal from federated learning models," in *Proc. of 2021 IWQOS*, 2021, pp. 1–10.

[22] Z. Zhang, S. Umar, A. Y. A. Hammadi, S. Yoon, E. Damiani, and C. Y. Yeun, "Data poisoning attacks on eeg signal-based risk assessment systems," *arXiv preprint arXiv:2302.04224*, 2023.

[23] P. R. Ovi, A. Gangopadhyay, R. F. Erbacher, and C. Busart, "Confident federated learning to tackle label flipped data poisoning attacks," in *Artificial Intelligence and Machine Learning for Multi-Domain Operations Applications V*, vol. 12538, 2023, pp. 263–272.

[24] J. Li, Z. Li, H. Zhang, G. Li, Z. Jin, X. Hu, and X. Xia, "Poison attack and defense on deep source code processing models," *arXiv preprint arXiv:2210.17029*, 2022.

[25] J. Zhang, J. Chen, D. Wu, B. Chen, and S. Yu, "Poisoning attack in federated learning using generative adversarial nets," in *Proc. of TrustCom/BigDataSE 2019*, 2019, pp. 374–380.

[26] S. Shan, W. Ding, J. Passananti, S. Wu, H. Zheng, and B. Y. Zhao, "Nightshade: Prompt-specific poisoning attacks on text-to-image generative models," in *Proc. of 2024 S& P*, 2024, pp. 212–212.

[27] J. Hayase, W. Kong, R. Somani, and S. Oh, "Spectre: Defending against backdoor attacks using robust statistics," in *Proc. of ICML 2021*, 2021, pp. 4129–4139.

[28] D. Tang, X. Wang, H. Tang, and K. Zhang, "Demon in the variant: Statistical analysis of dnns for robust backdoor contamination detection," in *Proc. of USENIX Security 21*, 2021, pp. 1541–1558.

[29] Y. Li, X. Lyu, N. Koren, L. Lyu, B. Li, and X. Ma, "Neural attention distillation: Erasing backdoor triggers from deep neural networks," *arXiv preprint arXiv:2101.05930*, 2021.

[30] K. Liu, B. Dolan-Gavitt, and S. Garg, "Fine-pruning: Defending against backdooring attacks on deep neural networks," in *Proc. of RAID 2018*, 2018, pp. 273–294.

[31] P. Zhao, P.-Y. Chen, P. Das, K. N. Ramamurthy, and X. Lin, "Bridging mode connectivity in loss landscapes and adversarial robustness," *arXiv preprint arXiv:2005.00060*, 2020.

[32] E. Borgnia, V. Cherepanova, L. Fowl, A. Ghiasi, J. Geiping, M. Goldblum, T. Goldstein, and A. Gupta, "Strong data augmentation sanitizes poisoning and backdoor attacks without an accuracy tradeoff," in *Proc. of ICASSP 2021-2021*, 2021, pp. 3855–3859.

[33] M. Fang, X. Cao, J. Jia, and N. Gong, "Local model poisoning attacks to byzantine-robust federated learning," in *Proc. of USENIX Security 20*, 2020, pp. 1605–1622.

[34] Z. Li, H. Yu, T. Zhou, L. Luo, M. Fan, Z. Xu, and G. Sun, "Byzantine resistant secure blockchained federated learning at the edge," *IEEE Network*, vol. 35, no. 4, pp. 295–301, 2021.

[35] X. Cao, J. Jia, Z. Zhang, and N. Z. Gong, "Fedrecover: Recovering from poisoning attacks in federated learning using historical information," in *Proc. of 2023 S&P*, 2023, pp. 1366–1383.

[36] Y. Jiang, J. Shen, Z. Liu, C. W. Tan, and K.-Y. Lam, "Towards efficient and certified recovery from poisoning attacks in federated learning," *arXiv preprint arXiv:2401.08216*, 2024.

[37] A. G. Roy, S. Siddiqui, S. Pölsterl, N. Navab, and C. Wachinger, "Braintorrent: A peer-to-peer environment for decentralized federated learning," *arXiv preprint arXiv:1905.06731*, 2019.

[38] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas, "Communication-efficient learning of deep networks from decentralized data," in *Proc. of 20th AISTATS*, 2017, pp. 1273–1282.

[39] H. Matthies and G. Strang, "The solution of nonlinear finite element equations," *International journal for numerical methods in engineering*, vol. 14, no. 11, pp. 1613–1626, 1979.

[40] L. Zhang, T. Zhu, H. Zhang, P. Xiong, and W. Zhou, "Fedrecovery: Differentially private machine unlearning for federated learning frameworks," *IEEE Transactions on Information Forensics and Security*, vol. 18, pp. 4732–4746, 2023.