

FedUP: Efficient Pruning-based Federated Unlearning for Model Poisoning Attacks

Nicolò Romandini*, Cristian Borcea, Rebecca Montanari, Luca Foschini

Abstract—Federated Learning (FL) can be vulnerable to attacks, such as model poisoning, where adversaries send malicious local weights to compromise the global model. Federated Unlearning (FU) is emerging as a solution to address such vulnerabilities by selectively removing the influence of detected malicious contributors on the global model without complete retraining. However, unlike typical FU scenarios where clients are trusted and cooperative, applying FU with malicious and possibly colluding clients is challenging because their collaboration in unlearning their data cannot be assumed. This work presents FedUP, a lightweight FU algorithm designed to efficiently mitigate malicious clients' influence by pruning specific connections within the attacked model. Our approach achieves efficiency by relying only on clients' weights from the last training round before unlearning to identify which connections to inhibit. Isolating malicious influence is non-trivial due to overlapping updates from benign and malicious clients. FedUP addresses this by carefully selecting and zeroing the highest magnitude weights that diverge the most between the latest updates from benign and malicious clients while preserving benign information. FedUP is evaluated under a strong adversarial threat model, where up to 50% – 1 of the clients could be malicious and have full knowledge of the aggregation process. We demonstrate the effectiveness, robustness, and efficiency of our solution through experiments across IID and Non-IID data, under label-flipping and backdoor attacks, and by comparing it with state-of-the-art (SOTA) FU solutions. In all scenarios, FedUP reduces malicious influence, lowering accuracy on malicious data to match that of a model retrained from scratch while preserving performance on benign data. FedUP achieves effective unlearning while consistently being faster and saving storage compared to the SOTA.

Index Terms—Federated Learning; Federated Unlearning; Malicious Clients; Pruning; Adversarial Attacks; Attack Mitigation.

I. INTRODUCTION

THE distributed nature of Federated Learning (FL) makes it susceptible to various attacks, including model poisoning attacks, where an adversary injects malicious local weights to disrupt the global model or poison it by introducing backdoors [1], [2]. Malicious actors are becoming increasingly skilled at executing stealth attacks [3]. The risk escalates significantly when critical public infrastructures like power plants are targeted [4]. Many techniques have been developed to detect these attacks [5]. However, after detection, recovering from such attacks can be costly and time-consuming, often necessitating retraining the model from scratch. While retraining

from scratch is a possible solution, it introduces substantial delays, as it requires restarting the entire training process. During this time, the compromised model cannot be safely used, and clients must wait for a new, clean version. This interruption can degrade service availability and user trust. This highlights the need to design mechanisms to alleviate the burden of dealing with such situations.

Federated Unlearning (FU) [6] has emerged as a solution to these problems. FU refers to the process of selectively removing the impact of specific data contributions from an FL model without the necessity of complete retraining. Unlike traditional Machine Unlearning (MU) [7], which is applied to centralized models, FU addresses the unique challenges posed by the decentralized nature of FL. Effective FU techniques aim to remove specific data contributions efficiently while retaining the valuable knowledge gained from the remaining data. However, applying FU algorithms in a malicious context differs from the typical exploitation of FU in scenarios where clients are considered trusted and cooperative [8], [9], [10]. Some work [11], [12] propose solutions in a malicious context, but they require the server to retain all client updates received during the training process, using them for performing unlearning. Storing gradients and model updates introduces for the entire duration of training both storage inefficiencies and substantial privacy risks, as a server breach could allow adversaries to exploit the stored updates to reconstruct private client data [13]. The challenge that we address is how to apply FU efficiently in a scenario with malicious clients, where it is not possible to rely on the collaboration of adversaries whose data needs to be unlearned. Many studies have already addressed detecting malicious clients [14], [15], [16], but detection alone, while important, is not sufficient. Once a malicious client is identified, the still open research question is how to remove its contribution from the FL model as soon as possible to minimize damage. Moreover, the unlearning algorithm itself could be the target of attacks [17], [18]. For example, an attacker could perform a Denial of Service (DoS) attack by repeatedly triggering the unlearning procedure to disrupt and potentially hijack the training process.

To address all these challenges, we present **FedUP**, an algorithm to efficiently mitigate the influence of multiple colluding malicious clients on the global FL model by pruning specific connections within the attacked model. FedUP is designed to start unlearning as soon as a malicious client is detected. Our approach is practical because the server uses only the weights of the local models sent by the clients to identify which connections to inhibit. To achieve efficiency, FedUP relies exclusively on the clients' weights from the last training round before the unlearning phase. Identifying and

Manuscript received ; ; .

Nicolò Romandini, Rebecca Montanari, and Luca Foschini are with the Department of Computer Science and Engineering (DISI), University of Bologna, Bologna, Italy (e-mail: {name.surname}@unibo.it). Cristian Borcea is with the New Jersey Institute of Technology, Newark, New Jersey, USA (email: borcea@njit.edu).

Asterisk indicates the corresponding author

removing malicious contributions is particularly challenging due to the complex overlap of weight updates across clients and the risk of destabilizing the model. Not all weights contribute equally to the model’s behavior, as some have a stronger influence than others. Simply choosing the highest magnitude weights of the malicious clients does not yield satisfactory results, as shown in Section VI-D4. Our algorithm addresses this by computing the difference between benign and malicious client updates, focusing on high-magnitude weights that exhibit conflicting modifications. These weights are more likely to affect the model’s decision boundaries and learning dynamics. Pruning is applied selectively to specific layers, since removing connections across the entire network would compromise stability and generalization. This careful targeting ensures that only the most harmful and influential connections are removed, allowing the algorithm to eliminate malicious influence while preserving benign knowledge and maintaining model performance. FedUP is executed directly by the server once malicious clients are detected. It also incorporates a rate limit mechanism against DoS attacks on the FU procedure.

Summing up, the main contributions of our work are the following:

- We propose an effective server-side-only FU algorithm that recovers a poisoned global model by simultaneously removing one or multiple malicious contributions.
- FedUP operates under a strong adversarial threat model in which up to 50% – 1 of the clients may be malicious. It is also resilient to emerging attacks that target FU, such as DoS attacks.
- The solution is lightweight and storage-efficient, relying solely on the model weights collected during the last training round before unlearning.
- Our algorithm is task-agnostic and can be seamlessly integrated into any FL framework without requiring modifications.
- We provide technical insights to support optimal fine-tuning of the algorithm’s parameters.
- We demonstrate the effectiveness and efficiency of our solution through extensive experiments involving three different models and two datasets, evaluated under both IID and Non-IID settings. Comparisons with State-of-the-Art (SOTA) FU approaches show that FedUP consistently achieves successful unlearning while being significantly faster than SOTA methods.

The rest of the paper is organized as follows. Section II section highlights the current state of the art in FU and explains how our solution differs from and improves upon it. Section III defines the threat model, detailing the attacker’s goals, knowledge, and capabilities. Section IV provides a detailed description of the FedUP algorithm. Section V elaborates on key factors and hyperparameters that influence the efficiency and effectiveness of the procedure. Section VI showcases the results of our extensive experiments. Finally, Section VII summarizes our findings.

II. RELATED WORK

Adversaries can launch targeted poisoning attacks by manipulating training to degrade model performance or inject

malicious behaviors. There are two common approaches. The first one is label-flipping [16], [19], where the attacker intentionally mislabels training data to bias the model’s decision boundary. The second approach is backdoor attacks [20], [21], in which a trigger pattern is embedded into training samples so that the model learns to associate this pattern with a target label. This enables the attacker to manipulate predictions at inference time whenever the trigger is present. In light of these attacks, this section focuses on related work topics relevant to contextualizing FedUP. We review techniques for attack detection, as FedUP depends on an effective detection mechanism. Additionally, we discuss FU approaches in both trusted settings, where unlearning is collaborative, and untrusted environments, where FU acts as a mitigation against adversarial contributions. Finally, we consider attacks targeting FU itself.

A. Detection Techniques

Detecting malicious clients in FL involves differentiating adversarial updates from legitimate ones to safeguard the global model. Broadly, detection strategies fall into two main groups [22], [23]: those that leverage clean validation datasets and those based on clustering or analyzing the similarity of model updates. Some approaches, such as FLTrust [24], maintain a trusted dataset on the server side to evaluate client updates. Clients whose updates closely match the server’s model receive higher trust scores, which helps filter out malicious contributions. Similarly, Li et al. [14] use variational autoencoders trained on clean validation samples to detect anomalous updates. Despite their effectiveness, these methods require access to clean and representative data, a condition often difficult to meet in real FL scenarios. Other methods focus on grouping or measuring the similarity between client updates. DnC [25] applies dimensionality reduction followed by spectral analysis to detect malicious clients but requires prior knowledge of the number of adversaries. Auror [26] uses K-means clustering, while FoolsGold [25] identifies attacks based on the similarity and diversity of client contributions. In [27], the authors propose FedCVAE, an unsupervised framework that utilizes a conditional variational autoencoder on the server side. By leveraging reconstruction errors as anomaly scores, FedCVAE dynamically filters out malicious updates. FedPRC [28] combines clustering with density-based anomaly detection to exclude malicious updates effectively. FLDetector [15] identifies malicious clients by examining the consistency of their model updates. A limitation of this method is its high computational cost. LFighter [19] is a detection method that analyzes gradient patterns in the output layer, exploiting discrepancies between malicious and honest updates. LFighter clusters and filters updates before aggregation, showing strong performance across different data distributions and model sizes. Finally, FedDMC [23] reduces dimensionality using PCA, employs binary tree clustering to mitigate noise, and incorporates a self-ensemble module to enhance detection robustness. Existing detection techniques become ineffective when malicious clients constitute the majority, as they rely on a benign majority to identify adversarial updates reliably

[14], [15], [22]. Moreover, in such scenarios, even existing robust aggregation methods may struggle to converge [29], [30], [23]. Therefore, FedUP also assumes a benign majority and is designed to operate effectively under this assumption.

B. FU in Benign Environments

Most existing works on FU are developed under benign settings, assuming honest participation from all clients. While our focus is primarily on unlearning in adversarial contexts, we briefly mention two notable approaches in benign scenarios. For a more exhaustive overview of FU in benign settings, we refer the reader to the survey [31]. The first solution is FedEraser [32]. It achieves client unlearning by leveraging historical parameter updates stored at the server to speed up the retraining process. Specifically, FedEraser conducts a calibration phase that iteratively sanitizes updates by only including the retained clients. Each round involves local training with calibrated updates and server-side aggregation, producing a sanitized global model to be broadcast in the next re-calibration round, continuing until all past updates are recovered. In addition to the need to store numerous client updates, the algorithm involves a recovery process that can require up to as many rounds as the training process itself. Unlike FedEraser, FedUP only requires the weights from the last round before unlearning and involves a recovery process that needs only a few additional rounds.

The second work in benign scenarios is [33], in which the authors present a method to forget specific categories in Convolutional Neural Networks (CNN) models within FL. Using Term Frequency-Inverse Document Frequency (TF-IDF) scores, the method quantifies channel discrimination, pruning those highly associated with target categories to achieve unlearning. Fine-tuning follows pruning to restore model performance. This approach is specifically designed for CNNs, which limits its applicability to other model types. Additionally, it requires clients to compute and transmit extra information to the server to identify the most significant channels, making it unsuitable for use in malicious contexts. FedUP relies solely on updates from the most recent training round and is applicable to any neural network.

The first work is used in our experimental evaluation for comparison, as it is one of the few approaches that offer publicly available and functional code. Although the second work proposes a method similar to ours, it does not provide open-source code, which hinders a direct comparison.

C. FU as a Mitigation Strategy in Malicious Settings

Few works specifically address the unique challenges posed by a malicious context. In [34], the authors propose a pruning method that removes neurons responsible for triggering misbehavior when backdoor patterns are detected. Although the method is designed for adversarial settings, it requires clients to compute additional information, which malicious clients are typically unwilling to provide or may deliberately falsify. Furthermore, the method is applicable only to specific types of attacks and is limited in that it cannot effectively remove multiple malicious clients simultaneously. In contrast, FedUP

requires no additional information beyond the models from the final training round and can be applied across a wide range of scenarios, including non-adversarial contexts, as demonstrated by the experimental results. Moreover, FedUP is capable of removing multiple malicious clients at the same time.

FedRecover [11] aims to recover a poisoned model attacked by malicious clients using historical information. It employs retained local model updates to estimate the updates that would be produced during a retrain-from-scratch process, using the Cauchy mean theorem and the L-BFGS algorithm [35] to approximate the integrated Hessian matrix efficiently. To improve the model's performance, FedRecover implements fine-tuning strategies, including warm-up, periodic correction, abnormality fixing, and final tuning phases. Storing all client updates during training and running the L-BFGS algorithm could be inefficient both in terms of storage and time. In contrast, FedUP only requires storing the weights from the models received in the last round before unlearning. Additionally, the pruning process consists of simply setting the selected weights to zero, thereby ensuring efficiency. FAST [12] is a protocol designed to mitigate the influence of Byzantine participants on the global model in FL. The server retains all client updates and adjusts the model parameters by subtracting the contributions of malicious clients from the final global model in each training round. The server compares the accuracy of the current unlearning model with the previous one, continuing the unlearning process if the current model outperforms its predecessor. This process is repeated until the maximum number of attempts is reached. To address potential loss in performance, the server uses an additional small benchmark dataset for supplementary training, enhancing the overall accuracy. Similar to previous work, storing all updates is inefficient in terms of storage and raises privacy concerns due to model and gradient inversion attacks. Furthermore, in many cases, the server may not have access to additional datasets, which limits the applicability of such algorithms. In contrast, FedUP does not require any extra data sources.

In [36], the authors propose a novel approach designed explicitly for backdoor removal in FL systems. Their method combines historical update subtraction and knowledge distillation to eliminate malicious clients' influence while preserving the global model's performance. The approach does not require additional clients' participation rounds and is compatible with various neural network architectures. However, the server requires additional data, which is not always available. In contrast, FedUP does not need supplementary data, making it more suitable for scenarios where access to external data is not possible or desirable. Lastly, in [37] the authors propose NoT, a FU method that removes a participant's contribution from a model without requiring access to the original data or additional storage. The approach leverages weight negation to disrupt the learned parameters while retaining the model's capacity to recover. Although NoT is not explicitly designed for removing malicious updates, the authors demonstrate its effectiveness against backdoor attacks. However, because the negation is not targeted at specific contributions, it can excessively disrupt the model, resulting in a higher number of recovery rounds, as observed in our comparison in Section VI.

Work	Malicious Clients	No Historical Updates	Multiple Unlearning	No Extra Data	DoS Resilient
[33]	×	✓	×	×	×
[32]	×	×	×	✓	×
[34]	✓	×	×	✓	×
[11]	✓	×	×	✓	×
[12]	✓	×	×	✓	×
[36]	✓	×	✓	×	×
[37]	✓	✓	×	✓	×
Ours	✓	✓	✓	✓	✓

TABLE I: Comparison with related work.

No described method provides open-source code, but we include NoT in our comparison since its simple design enabled us to reimplement it for our experimental evaluation. Table I summarizes the main novelty of FedUP compared to the most relevant related work. The columns highlight key features, including the ability to handle malicious clients, avoid storing historical updates, unlearn multiple contributions simultaneously, operate without requiring extra data beyond the models, and resist DoS attacks. To the best of our knowledge, FedUP is the first algorithm that enables the simultaneous removal of contributions from one or more malicious clients, requiring only the models from the last training round before unlearning, and is inherently resistant to DoS attacks by design.

D. Attacks Targeting FU

Recent studies have investigated how unlearning mechanisms can be maliciously exploited [17], [18]. In these contexts, malicious clients may aim to trigger the unlearning algorithm to disrupt the global model and cause it to forget valuable knowledge. Alternatively, malicious clients might try to continuously activate the unlearning algorithm to hijack the training process, effectively launching a DoS attack that prevents stable model convergence. Most of the works in the literature still lack protection mechanisms against these threats. FedUP is the first work to present a mechanism for preventing DoS attacks on the FU process.

III. THREAT MODEL

This section describes the threat model considered in our setting and outlines the main assumptions. Therefore, we clarify the adversary's goals, knowledge, and capabilities. Our threat model is consistent with those presented in related works [11], [34], [36].

1) *Adversary and Adversary's Goals*: We consider a scenario where a group of malicious and possibly colluding clients collaborate to compromise the training process in an FL system. These clients can manipulate their local updates, such as gradients or model weights, before sending them to the central server. We considered targeted poisoning attacks, where the adversary aims to cause specific misclassifications in the global model. In our scenario, malicious clients have already participated in multiple training rounds, embedding malicious updates into the global model. The server is assumed to be honest and is equipped with a mechanism to detect malicious behavior during the training process. Our objective is to neutralize the malicious influence already embedded in

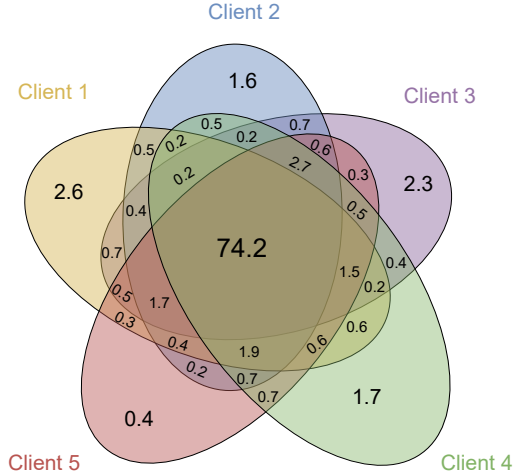


Fig. 1: Percentage of weights influenced by each client in a deep convolutional layer of MobileNetV2 after one training round on CIFAR-10.

the global model without degrading performance on benign tasks. Beyond the typical objective of poisoning the model, malicious clients can also perform a DoS attack that targets the unlearning mechanism. In this case, the adversary attempts to repeatedly trigger the unlearning process in order to hijack the training pipeline. This can result in disrupted convergence, excessive resource consumption, and overall system instability.

2) *Adversary's Knowledge*: We assume that adversaries possess knowledge of the FL process, the training algorithm, and the global model architecture. They are capable of crafting malicious updates tailored to the training objective. However, they do not have access to the local data or updates of other participants. This reflects a realistic threat model where attackers operate under limited information and cannot directly observe or influence the contributions of benign clients.

3) *Adversary's Capabilities*: Adversaries are assumed to control up to 50% – 1 of the total participants, which aligns with adopted threat models in the literature [38]. These malicious clients can upload manipulated model updates during the training process. However, adversaries cannot control the aggregation algorithm or the server's behavior, which is assumed to be honest and to execute the protocol correctly.

IV. FEDUP DESIGN

Our algorithm leverages one-shot soft pruning [39], [40], which allows the model to zero out specific weights while retaining the potential to retrain and recover them if needed. For simplicity, from this point, we will refer to soft pruning as pruning. However, unlike traditional pruning techniques that focus on identifying and removing less relevant connections to make the neural network more efficient, our algorithm reverses this principle entirely. Instead of discarding the less critical connections, we eliminate the most important ones that significantly influence the network's output, contributing more to the model's predictive power. Specifically, FedUP zeroes out

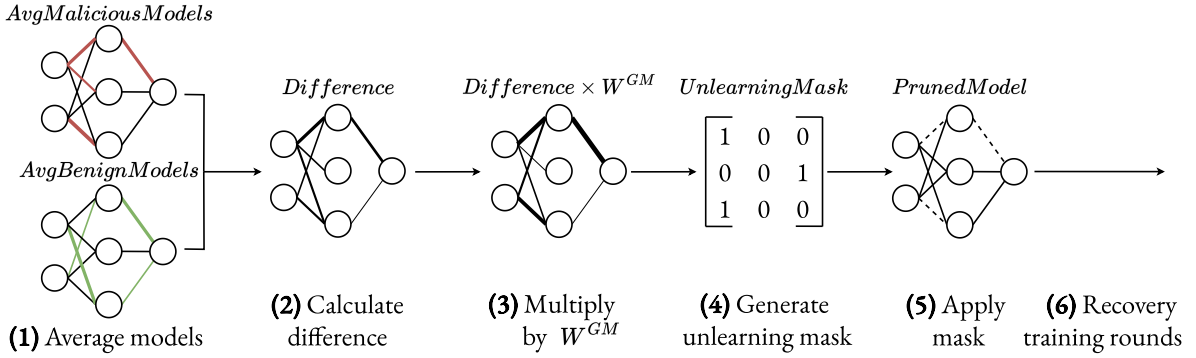


Fig. 2: Unlearning procedure after detecting malicious clients.

the salient weights identified in the local models sent by malicious clients from the global model. However, identifying the most salient weights is a challenging task. Figure 1 illustrates how each client influences different subsets of the model’s weights in each training round. As shown, there is a significant overlap among the weights affected by different clients, which complicates the isolation of individual contributions and the accurate identification of malicious updates. Nevertheless, pruning effectively removes any potentially malicious contributions contained in these connections, thereby neutralizing the influence of the malicious clients. Let us notice that FedUP can be applied to remove multiple malicious clients’ contributions simultaneously. The process remains unchanged regardless of the number of malicious clients.

Figure 2 presents the unlearning procedure. It is important to outline that the FedUP unlearning algorithm does not affect traditional training when not activated. The unlearning process is executed by the server only when it detects the presence of one or more malicious clients. First, the server creates two separate reference models: one by averaging benign models and the other by averaging malicious ones (1). The step is designed to enhance efficiency, especially when dealing with a large number of clients, by minimizing the need for extensive pairwise comparisons. Then, it computes the difference between the averaged models to identify the conflicting weights between the benign and malicious clients (2). However, weights in neural networks are not equally important. To account for this, weight differences are scaled by their global model values (3). After that, it selects the highest magnitude weights that differ the most to create an unlearning mask (4). The mask is then applied to the model produced by averaging the benign models (5). This step allows for the exclusion of the malicious clients’ models and their contribution in that round as well. However, some pruned weights might include benign information from non-malicious clients. As a result, pruning can generally lead to a reduction in the model’s performance. To recover the performance lost, FedUP incorporates additional training rounds after pruning. Finally, the server sends the pruned global model to the remaining clients to perform recovery training rounds to restore the performance (6).

FedUP seamlessly works whether clients send models, like in FedAVG [41], or gradients, like in FedSGD, as the infor-

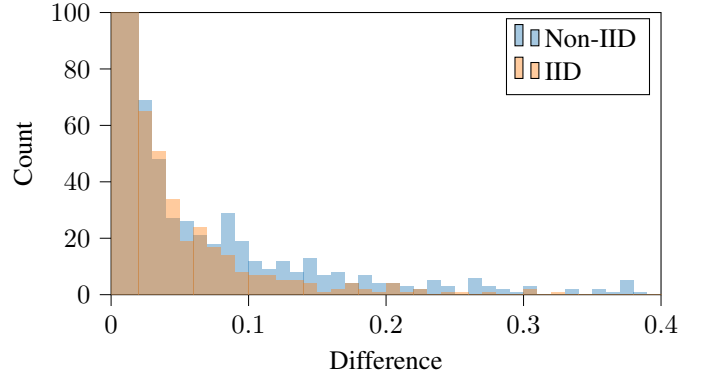


Fig. 3: Normalized squared difference between averaged benign and malicious weights with 20 clients and 6 malicious.

mation remains consistent in both cases. The most dissimilar weights are the same ones with the most dissimilar gradients. Moreover, our method does not require saving or receiving any additional information apart from the models or gradients from the clients. This allows for easy integration into any FL framework. The same goal of FedUP can be achieved through natural forgetting [42], but at the cost of numerous additional rounds that make the natural forgetting process inefficient when the model has already converged [43]. Therefore, the underlying intuition of our approach is to strategically perturb the model’s current state by pruning selected connections, thereby guiding it away from configurations influenced by malicious contributions. This accelerates natural forgetting and forces the model to reconfigure itself while discarding the data to be forgotten during the recovery rounds.

A. Identifying Most Important Connections

Our objective is to identify the most important weights for a specific client using only the latest updates sent by the clients. These identified weights determine the unlearning mask applied to the global model. Details of the procedure are provided in Algorithm 1. Many studies on weight importance are designed for centralized contexts [44], [45]. Identifying critical connections for specific clients in FL is challenging because multiple clients can update the same weight simul-

taneously. The variation in weights between rounds helps to understand the direction in which a client aims to drive the global model [46], [19]. By applying this principle to detect which weights are primarily influenced by each group of clients, we compute the squared difference between benign and malicious updates (line 5 of Algorithm 1). Figure 3 illustrates the squared differences between the averaged benign and malicious models in a CNN after one training round under both IID and Non-IID settings. In both cases, malicious clients also possess benign data that makes them more stealthy. As shown, most weights experience minimal changes. However, malicious clients are more likely to have weights that diverge significantly from benign clients, making their influence more noticeable. As said, weights in a neural network contribute differently. Those with higher magnitudes typically have a greater influence on the network’s behavior [47]. Therefore, the magnitude of the weight being modified must also be considered. For this reason, the difference between the weights is multiplied by the weight value in the global model (line 6 of Algorithm 1). This adjustment prioritizes the most significant weights both for the client to unlearn and for the model itself, making the pruning process more efficient and effective.

B. Unlearning Procedure

This procedure can be executed as soon as the server detects the presence of one or more malicious clients. This can also happen multiple times during the FL process. First, the server calculates the unlearning mask by selecting each layer’s most important weights. The steps are presented in Algorithm 1. The percentage of weights \mathcal{P} to be pruned could be estimated based on the complexity of the model and the data distribution. More details are discussed in Section V-B. Then, the server uses the unlearning mask to zero out the weights in the model produced by averaging the benign models (line 8 of Algorithm 2). As said, applying the mask on this model already removes the contribution sent by malicious clients in the last training round. Algorithm 2 shows the procedure to apply the mask. Another important consideration is regarding the layers on which the mask is applied. Not all layers have the same characteristics, and indiscriminately inhibiting weights in every layer can significantly degrade the network’s performance. For this reason, pruning is limited to fully connected and convolutional layers (line 7 of Algorithm 2), as seen in other works [48], [49], [50]. The result of this operation is a network that has become more or less sparse, depending on the percentage of the pruned weights. In centralized contexts, it has already been shown that sparsity can facilitate data unlearning [51]. At this stage, the network has lost part of its prediction capabilities and shifted away from the potential local minimum it had reached after convergence.

C. Restoring Performance

Pruning is inherently destructive, resulting in the loss of information stored in the zeroed out connections. Each connection stores information from multiple clients in a distributed training context such as FL. Therefore, pruning the weights, even if related to malicious clients, will likely erase some of

Algorithm 1 Generation of the Unlearning Mask

Input: $localModels^t, globalModel^{t-1}, \mathcal{P}$

Output: $mask$

```

1:  $maliciousModels \leftarrow [localModel_i \mid i \text{ is malicious}]$ 
2:  $avgMaliciousModels \leftarrow Avg(maliciousModels)$ 
3:  $benignModels \leftarrow [localModel_i \mid i \text{ is not malicious}]$ 
4:  $avgBenignClients \leftarrow Avg(benignModels)$ 
5:  $difference \leftarrow (avgMaliciousModels - avgBenignModels)^2$ 
6:  $rank \leftarrow difference \times globalModel^{t-1}$ 
7:  $mask \leftarrow []$ 
8: for  $i = 0$  to  $len(rank)$  do
9:    $rankSorted \leftarrow sortDescending(rank[i])$ 
10:   $weights \leftarrow choose(rankSorted, \mathcal{P})$ 
11:   $mask.append(weights)$ 
12: end for
13: return  $mask$ 
```

Algorithm 2 Apply Mask on Global Model

Input: $mask, avgBenignClients$

Output: $prunedGlobalModel$

```

1:  $prunedGlobalModel \leftarrow avgBenignClients.copy()$ 
2:  $layers \leftarrow prunedGlobalModel.getLayers()$ 
3: for  $i = 0$  to  $len(layers)$  do
4:   if  $layer[i] \in [Dense, Convolutional]$  then
5:      $layer[i][mask[i]] \leftarrow 0$ 
6:   end if
7: end for
8: return  $prunedGlobalModel$ 
```

the information learned from other clients. To mitigate this effect, a few additional FL training rounds performed by the remaining benign clients are sufficient to restore the performance lost during the unlearning procedure. The clients use the same local dataset and configuration as in the standard FL training process, and these recovery rounds remain transparent to them. This operation further strengthens the removal of the excluded client’s contribution, as the model will be more finely tuned to the remaining clients. Estimating the number of recovery rounds is a complex task, as it is equivalent to estimating the number of rounds a model needs to converge in a standard FL process. However, in Section V-C, we provide some considerations that offer insight into the estimation.

D. Unlearning Rate Limit

To prevent DoS attacks as described in Section II and Section III, FedUP enforces a rate limit on the unlearning process. In FedUP, unlearning is never triggered directly by clients but is instead initiated by the server upon detecting new malicious participants. To manage this process, the server defines an unlearning rate threshold \mathcal{T} , which sets the minimum number of training rounds that must pass between consecutive unlearning operations. This mechanism ensures that unlearning is not triggered too frequently, protecting the system from repeated and costly recovery operations that could be induced by adversarial activity. FedUP is designed

to unlearn the influence of multiple clients simultaneously. If several malicious clients are detected during the period in which unlearning is temporarily disabled due to rate limiting, the server accumulates these detections. Once the threshold \mathcal{T} is reached, FedUP performs a single unlearning step that prunes all identified malicious contributions at once. The threshold can be configured by the system owner to balance responsiveness with system stability, depending on the deployment context and threat landscape. Empirically, as shown in the experiments, the unlearned model tends to restore its original performance within a few rounds following the unlearning phase. Therefore, even a low threshold, such as $T = 10$ rounds, can provide an effective trade-off between mitigating DoS attacks and ensuring timely model recovery.

V. TECHNICAL INSIGHTS

In this section, we give technical insights and practical guidelines to tune the algorithm properly.

A. Pruning Most Dissimilar Weights

The choice to prune the most dissimilar weights is supported by mathematical evidence, as detailed in the following section. In FL, each client starts a training round with the global model from the previous round, $W^{\text{GM},t-1}$. The client trains this model on its local data, computing updates ΔW^{C_i} , which adjust the global model to the client's data. The updated client weights are:

$$W^{C_i,t} = W^{\text{GM},t-1} + \Delta W^{C_i}. \quad (1)$$

The server aggregates these updates, typically using weighted averaging, to create the new global model for the next round. Here, ΔW^{C_i} reflects the client-specific contributions. In a converged model, $W^{\text{GM},t-1}$ is near-optimal for the global data distribution, minimizing global loss. For weights already optimized globally, updates $\Delta w_j^{C_i}$ are small. In contrast, updates for weights critical to reducing a client's local loss, $\Delta w_j^{C_i}$, are larger. At convergence, the difference between benign and malicious weights stabilizes, manifesting two distinct patterns:

- 1) Both parties are satisfied with the weight w_j value, so the updates are nearly zero. These weights usually capture the common knowledge shared by all clients. Malicious clients, for example, may possess benign data to complicate their identification. In this case, the difference is minimal.
- 2) Both parties have updated the weight w_j in a conflicting manner. These weights are the contested ones, used by clients to shape the network and minimize the loss on their data. In this case, the difference is more significant.

During training, clients will naturally adjust the weights to align with each other's updates, promoting consensus on those parameters. However, when clients possess malicious data, their update will diverge on a particular weight [19]. This divergence can persist at convergence, given that malicious clients are typically in the minority, as most of the model's knowledge reflects benign clients' contributions. To identify the weights most targeted by malicious clients, the server

computes the differences, Δw_j^{diff} , between all the weights from benign and malicious clients. Weights with the largest values of Δw_j^{diff} are the most likely to have been influenced by malicious clients.

B. Threshold Choice

One of the most crucial aspects for precise and efficient unlearning is choosing an appropriate percentage of weights to prune \mathcal{P} . Selecting a value that is too low risks leaving malicious knowledge intact within the model. Conversely, a value that is too high risks pruning too many weights, thereby completely deteriorating the model. However, determining the optimal pruning percentage is challenging. In most existing works [52], [45], [53], the pruning percentage is typically chosen empirically. To enable a more informed selection, the pruning percentage can be fine-tuned based on the following considerations:

- 1) **Weights Separability Between Malicious and Benign Clients (\mathcal{S}):** When the divergence between the contributions of benign and malicious clients is more evident, it becomes easier to identify and isolate harmful weights. In this case, slightly more aggressive pruning can be applied without compromising useful knowledge. When separability is low, there is more overlap between benign and malicious updates, making it difficult to prune malicious influence without also affecting benign knowledge. This requires more conservative pruning.
- 2) **Ease of Performance Recovery (\mathcal{R}):** Another important factor is how quickly the model can regain its performance after pruning. In scenarios where the model architecture and data distribution support efficient relearning, slightly more aggressive pruning can be tolerated since the model will recover in just a few rounds. Conversely, in more fragile setups where performance recovery is slower, pruning should be more conservative to avoid prolonged degradation.
- 3) **Natural Forgetting Efficacy (\mathcal{E}):** Neural networks naturally forget knowledge that is no longer reinforced through training. Excluding clients from the training process can help accelerate this forgetting. However, the speed remains relatively slow. Moreover, the effectiveness of this mechanism can vary significantly depending on the specific training dynamics. Factors such as model architecture, data distribution, and optimization strategy can all influence how quickly outdated or malicious information is overwritten. When natural forgetting is weak, more explicit pruning is necessary to effectively remove harmful contributions. Conversely, if natural forgetting is efficient, the model can reduce harmful influence with less pruning.

Assuming $\mathcal{S}, \mathcal{R}, \mathcal{E} \in [0, 1]$, where higher values represent greater separability, easier performance recovery, and stronger natural forgetting efficacy, respectively, we propose the following formula to guide the selection of the pruning percentage:

$$\mathcal{P} \approx \lambda_1 \cdot \mathcal{S} + \lambda_2 \cdot \mathcal{R} + \lambda_3 \cdot (1 - \mathcal{E}) \quad (2)$$

where $\lambda_1 + \lambda_2 + \lambda_3 = 1$ are weighting coefficients used to balance the influence of each factor, which can be chosen based on empirical observations or specific application needs. Although directly estimating \mathcal{S} , \mathcal{R} , and \mathcal{E} may be challenging, the pruning percentage can be approximated using the degree of data similarity among benign clients, based on the following insights.

In fact, the separability between benign and malicious updates tends to be higher in IID scenarios compared to Non-IID ones. This is because in IID settings, clients' data distributions are similar, so their model updates follow consistent patterns. Malicious updates therefore stand out more clearly as outliers or divergent patterns, making it easier to identify and isolate harmful weights. Conversely, in Non-IID scenarios, clients hold more diverse and heterogeneous data, leading to greater natural variability in their updates. This overlap between benign and malicious contributions reduces separability and complicates the pruning process.

Moreover, the model's ability to recover performance after pruning also differs between IID and Non-IID settings. In IID scenarios, the consistent and redundant nature of the data across clients allows the model to quickly relearn any slightly pruned benign knowledge, facilitating faster performance recovery. On the other hand, in Non-IID settings, data heterogeneity makes it more difficult for the model to recover lost information, as the removed knowledge might be unique to a specific client or data subset. This reinforces the need for more cautious pruning in such cases, as recovering lost benign contributions is inherently more challenging.

Finally, regarding natural forgetting, its efficacy is typically lower in IID environments. Since the global model continuously encounters similar updates from all clients, it reinforces learned parameters consistently, making it harder for the model to forget outdated or maliciously injected knowledge. In contrast, in Non-IID settings, the global model is exposed to more varied updates, which encourages parameter changes that can overwrite or diminish the impact of previous harmful contributions. This diversity acts as a natural regularizer, accelerating the forgetting of adversarial influence. Our ablation study in Section VI-D4 confirms this behavior, showing that natural forgetting is more effective in Non-IID training.

All these considerations suggest that the data similarity can provide an approximate guideline for selecting the pruning percentage. Therefore, Equation 2 can be rewritten as:

$$\mathcal{P} \propto \text{SIM} \quad (3)$$

where $\text{SIM} \in [0, 1]$ quantifies the degree of data similarity among clients, with higher values indicating more IID-like distributions and lower values corresponding to more Non-IID settings. While many similarity metrics exist, in this work we choose cosine similarity on the last layer of the model, as it is one of the most commonly adopted approaches in FL literature. To improve separation and control, we normalize the similarity into a variable $z \in [0, 1]$ through the following mapping:

$$z = \frac{\text{SIM}_{\text{orig}} - \text{SIM}_{\text{min}}}{\text{SIM}_{\text{max}} - \text{SIM}_{\text{min}}} \quad (4)$$

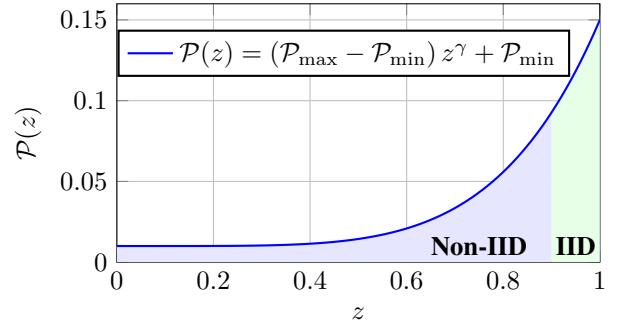


Fig. 4: Pruning rate \mathcal{P} as a function of normalized similarity z with $\gamma = 5$, $\mathcal{P}_{\min} = 0.01$, and $\mathcal{P}_{\max} = 0.15$. The shaded regions highlight typical ranges for Non-IID and IID scenarios.

where SIM_{\min} and SIM_{\max} define the range of the original similarity metric. Since we calculate similarity at convergence, we assume the original cosine similarity values fall within $[0.5, 1]$. This normalization helps standardize the similarity values, improving differentiation between Non-IID and IID cases.

To further improve separation, especially between low-to-medium and high similarity values, we introduce a nonlinear mapping for the pruning percentage \mathcal{P} . This nonlinear function produces small and relatively flat pruning values for low and medium similarity, capturing the Non-IID nature of the data, while steeply increasing pruning for very high similarity values corresponding to near-IID settings. We define the pruning percentage as

$$\mathcal{P} \approx (\mathcal{P}_{\max} - \mathcal{P}_{\min}) \cdot z^\gamma + \mathcal{P}_{\min}, \quad (5)$$

where \mathcal{P}_{\max} and \mathcal{P}_{\min} denote, respectively, the maximum and minimum pruning rates. The exponent γ controls the steepness of the curve: larger values keep the curve relatively flat for low-to-mid similarity scores and make it rise sharply as $z \rightarrow 1$. This behavior yields finer separation in non-IID scenarios while allowing more aggressive pruning when the data are highly similar (IID-like). The parameters \mathcal{P}_{\max} , \mathcal{P}_{\min} , and γ can be tuned to balance pruning aggressiveness and model stability for a given setting. In our experiments, we set $\mathcal{P}_{\max} = 0.15$, $\mathcal{P}_{\min} = 0.01$, and $\gamma = 5$, which we found to provide a good trade-off between sensitivity and robustness. Figure 4 illustrates the nonlinear mapping from the normalized similarity z to the pruning percentage $\mathcal{P}(z)$, with shaded regions indicating typical similarity ranges for IID and Non-IID data.

In our experiments, we calculated mean cosine similarities of 0.99 for IID data and 0.89 for Non-IID data, which normalize to $z \approx 0.98$ and $z \approx 0.78$, respectively. These values produce pruning percentages \mathcal{P} of approximately 10% for IID and 5% for Non-IID cases, effectively adapting pruning aggressiveness based on data similarity.

C. Number of Recovery Rounds

The recovery process involves retraining the model to restore performance after zeroing out specific weights. While predicting the exact number of recovery rounds is challenging,

the upper bound is given by the number of rounds needed for retraining from scratch, denoted as \mathcal{R}^* . We hypothesize that an optimal pruning percentage, \mathcal{P}_{opt} , exists, which effectively removes the influence of malicious data and ensures $\mathcal{R}_{\mathcal{P}_{opt}} < \mathcal{R}^*$. Not all weights contribute equally to predicting a sample, implying a subset of weights, $\mathcal{W}_{malicious}$, is responsible for predicting malicious samples, $\mathcal{D}_{malicious}$. We can determine \mathcal{P}_{opt} by pruning one weight at a time, progressively increasing the pruning percentage with each iteration. As we prune more weights, the model's performance will degrade, increasing the number of recovery rounds, \mathcal{R} . Weights are pruned until $\mathcal{D}_{malicious}$ is no longer predictable. At \mathcal{P}_{opt} , all malicious weights are pruned, leaving only the benign ones, which still retain useful knowledge. As a result, the model requires fewer recovery rounds than retraining from scratch because it already possesses some knowledge. To estimate the number of recovery rounds required after pruning, we propose an upper bound that accounts for the extent of pruning. Specifically, the bound is defined as:

$$\mathcal{R}_{\mathcal{P}} \leq \lceil \mathcal{R}^* \times \mathcal{P} \rceil \quad (6)$$

where $\mathcal{R}_{\mathcal{P}}$ denotes the number of recovery rounds, \mathcal{R}^* is the number of rounds required to retrain the model from scratch, and \mathcal{P} is the pruning percentage. It is important to note that \mathcal{R}^* is typically higher in Non-IID scenarios due to the difficulty of aggregating heterogeneous client updates. At the same time, according to Equation 3, IID settings require higher pruning percentages \mathcal{P} . Conversely, Non-IID scenarios tend to necessitate lower \mathcal{P} values, reflecting the greater risk of mistakenly pruning useful information. These opposing trends naturally balance: when \mathcal{R}^* is high, \mathcal{P} is low, and vice versa. This makes $\lceil \mathcal{R}^* \times \mathcal{P} \rceil$ a practical and robust upper bound for estimating recovery rounds across data distributions.

D. Security Analysis

In this section, we evaluate the security guarantees of FedUP under the adversarial model defined in Section III.

Assumption 1. *In an FL system, the majority of the clients are benign. The current detection solutions (see Section II-A) cannot work if the majority of clients are malicious, because it is not possible to separate their updates from those of benign clients in the weight space.*

Assumption 2. *Malicious clients cannot access benign clients' updates, and all communication between clients and the server is encrypted, ensuring confidentiality and reducing the attack surface.*

Theorem 1. *FedUP effectively removes the influence of multiple malicious clients, as long as the number of malicious clients does not exceed $\lfloor \frac{N-1}{2} \rfloor$, where N is the total number of clients.*

Proof. Let $\mathcal{C} = \mathcal{B} \cup \mathcal{M}$ be the set of all clients, where \mathcal{B} is the set of benign clients and \mathcal{M} the set of malicious clients. Assume that $|\mathcal{M}| \leq \lfloor \frac{N-1}{2} \rfloor$, so that benign clients are in strict majority. Under Assumption 1, when benign clients are the majority, malicious updates can be detected. At convergence,

benign updates tend to be small and mutually consistent (as discussed in Section V-A), while malicious clients, attempting to alter the model toward their adversarial goal, produce updates that conflict with the benign trend. FedUP identifies such conflicting weights by computing update dissimilarities and pruning the most deviant ones, which are most likely to be influenced by malicious behavior. This dissimilarity-based pruning mechanism exploits the structural divergence created by a benign majority. Therefore, as long as $|\mathcal{B}| > |\mathcal{M}|$, FedUP can distinguish malicious from benign knowledge and successfully remove the adversarial influence. \square

Theorem 2. *All benign model updates in FedUP are protected from eavesdropping and tampering.*

Proof. Following Assumption 2, all communications between clients and the central server are encrypted. By maintaining confidentiality, adversaries are prevented from crafting more effective or targeted attacks. \square

Theorem 3. *FedUP is resistant to DoS attacks, preventing adversarial misuse of the unlearning mechanism.*

Proof. FedUP implements a rate-limiting mechanism, presented in Section IV-D, to prevent attackers from continuously triggering unlearning, which could degrade the model. \square

VI. PERFORMANCE EVALUATION

In this section, we present the experiments conducted to evaluate the effectiveness and efficiency of our algorithm. This evaluation aims to demonstrate how effectively FedUP removes the contribution of one or more malicious clients and efficiently restores the original performance in as few recovery rounds as possible. It also presents a comparison with two SOTA solutions in terms of storage requirements and the number of recovery rounds needed to restore performance.

A. Experimental Setup

1) Malicious Setting: To evaluate the effectiveness of our algorithm, we replicate SOTA adversarial settings, including label-flipping and backdoor attacks. Both attack scenarios are implemented using standard FL benchmarks and follow comparable configurations to ensure consistent evaluation. Malicious clients inject poisoned updates into the global model during training. Detection and removal occur in a round when malicious behavior is present, consistent with standard detection-based defense assumptions [15]. We consider two percentages of malicious clients: 30% (with 10 and 20 clients) and 50% – 1 (across all client numbers). We choose these proportions because at least 30% of malicious clients are required to execute a successful poisoning attack [1] and to be consistent with our threat model.

Label-Flipping Attack: We reproduce the experimental setup outlined in [1], which provides a robust framework for simulating adversarial clients in FL. We utilize the same standard datasets as the reference settings, namely CIFAR-10 [54] and FashionMNIST [55], which are commonly used in the literature for benchmarking FL algorithms. This enables

an assessment of unlearning performance across varying degrees of difficulty. The neural network models used in our experiments are the same as those in the reference setting. They are standard CNNs similar to the LeNet architecture [56]. Malicious clients execute a label-flipping attack, an adversarial attack where the attacker deliberately changes the labels of the training data to incorrect values. This attack aims to degrade the performance of the global model by misleading the training process, causing the model to learn incorrect patterns and make erroneous predictions. In our experiments, each malicious client swap 10% of its labels while keeping the remaining data correctly labeled. The labels to flip are chosen following the same criteria as the reference paper, namely, the most wrongly classified labels in a non-malicious scenario [1]. This approach is designed to make malicious activities more challenging to detect.

Backdoor Attack: We replicate the setting proposed in [57], which employs a ResNet-18 [58] model trained on the CIFAR-10 dataset. In this setup, each attacker embeds a 3×3 white pixel square in the bottom-right corner of 10% input images as a trigger, following [57]. The model learns to misclassify these triggered inputs while maintaining high accuracy on clean data, thereby evaluating the algorithm's ability to detect and remove subtle malicious contributions.

2) *False Positive Scenario:* FedUP relies on a detection mechanism to identify malicious clients. Although these algorithms are becoming increasingly sophisticated, it is important to consider situations in which the detection may produce incorrect results. One such case occurs when a benign client is mistakenly identified as malicious and is excluded from the training process through pruning. This is commonly known as a false positive scenario. A benign client is unjustifiably removed from the training process, which can negatively affect model performance.

3) *Comparison with SOTA:* To better demonstrate the capabilities of our algorithm, we include a comparison with SOTA solutions in both malicious and benign settings. The evaluation has been extended to the benign scenario primarily due to the limited availability of publicly released source code for methods designed to operate in adversarial contexts. Nonetheless, this comparison offers valuable insights into FedUP's performance in non-adversarial environments. In the malicious setting, we evaluate FedUP against NoT [37] under a backdoor attack scenario. NoT uses weight negation to disrupt the learned parameters to remove a participant's contribution without requiring access to the original data or additional storage. It has also been successfully tested against backdoor attacks. Although the source code is not publicly available, its simple design allowed us to reimplement it for our *backdoor attack* experimental evaluation. For the benign setting, we evaluate FedUP against NoT and FedEraser [32]. In FedEraser, performing unlearning requires the participation of the server, which retains all historical updates along with information from the remaining clients. However, it cannot operate in a malicious context, and removing multiple clients simultaneously is not supported. For additional details, please refer to Section II.

B. FL Parameters

In all experiments, we use the Adam optimizer with a learning rate of $1e-3$, a batch size of 32, and FedAVG as the aggregation strategy. In the malicious settings, local training is performed for 1 epoch in the label-flipping attack scenario and extended to 3 epochs in the backdoor attack scenario, following the reference settings. We vary the number of clients to 5, 10, and 20, which reflects standard experimental setups commonly used in the FU literature [31]. Additionally, we explore both IID and Non-IID data distributions, the latter simulated using a Dirichlet distribution ($\alpha = 1$), a widely adopted approach for representing realistic data heterogeneity across clients. In the benign setting, we adopt the same experimental configuration as used in FedEraser, which involves a small custom CNN, the CIFAR-10 dataset split in an IID manner among 10 clients, and 5 local training epochs.

C. Evaluation Metrics

To evaluate the efficacy of the unlearning performance, we measure the accuracy on the to-be-forgotten data before and after the unlearning procedure. This metric is a SOTA way to understand how effectively the model has unlearned the specific data [31]. After unlearning, the new model's performance should be comparable to a baseline model \mathcal{B} , which is obtained by fully retraining the model without any exposure to the malicious data. This represents the standard performance of the model, unaffected by malicious influences. To assess the restoration performance, we evaluate the model on the same test dataset before and after the unlearning process. The new model should maintain equivalent performance. To evaluate efficiency, we measure the number of recovery rounds, \mathcal{R}_{rec} , required to restore the global model's performance, compared to the rounds needed for a retrain from scratch, \mathcal{R}^* . Lower values indicate faster performance recovery and, consequently, greater efficiency.

D. Results

This section presents the results of FedUP under malicious settings and false flag scenarios, and compares its performance with SOTA solutions.

1) *Performance under Malicious Setting:* Tables II and III show the results of FedUP with CIFAR-10 and Fashion-MNIST in both IID and Non-IID settings under label-flipping attack. Table IV instead shows the results of FedUP with CIFAR-10 in both IID and Non-IID settings under backdoor attack. These tables present the performance metrics of the global model before and after applying FedUP, specifically the accuracy obtained on the test set and on the malicious data. They also highlight the number of recovery rounds, \mathcal{R}_{rec} , required to achieve these results. As shown, FedUP effectively eliminates the influence of malicious data, as evidenced by the significant drop in the accuracy on their data. Meanwhile, the performance on the test set is almost completely restored in nearly all cases. The performance achieved is comparable to that of the baseline. The results are consistent with those observed across all settings and datasets used. Specifically,

No. Clients (Malicious)	IID									Non-IID								
	Test Data		Malicious Data			\mathcal{R}^*	\mathcal{R}_{rec}	\mathcal{P}_{opt}		Test Data		Malicious Data			\mathcal{R}^*	\mathcal{R}_{rec}	\mathcal{P}_{opt}	
	Before	After	\mathcal{B}	Before	After					Before	After	\mathcal{B}	Before	After				
5 (2)	78	79	4	57	5	24	1	10		79	80	4	44	4	32	1	5	
10 (3)	79	77	3	51	2	26	1	11		78	78	8	30	6	37	2	4	
10 (4)	78	78	4	59	3	27	1	11		77	77	10	23	9	43	2	4	
20 (6)	80	78	4	60	5	42	2	11		79	77	11	37	11	75	4	3	
20 (9)	79	79	5	42	4	67	2	10		79	79	7	29	8	66	2	3	

TABLE II: Accuracy (%) before and after unlearning on test and malicious clients' data, the number of recovery rounds, and the percentage of weights pruned (%) used with CIFAR-10 under label-flipping attack

No. Clients (Malicious)	IID									Non-IID								
	Test Data		Malicious Data			\mathcal{R}^*	\mathcal{R}_{rec}	\mathcal{P}_{opt}		Test Data		Malicious Data			\mathcal{R}^*	\mathcal{R}_{rec}	\mathcal{P}_{opt}	
	Before	After	\mathcal{B}	Before	After					Before	After	\mathcal{B}	Before	After				
5 (2)	84	88	4	73	5	22	1	10		83	82	21	54	20	19	2	6	
10 (3)	85	87	4	65	3	34	1	10		89	88	9	22	7	36	1	5	
10 (4)	84	88	5	72	3	19	1	10		86	87	11	36	12	19	1	5	
20 (6)	86	88	6	58	8	51	1	10		90	89	5	16	4	64	2	5	
20 (9)	83	82	5	69	6	35	1	11		89	88	5	24	6	41	2	5	

TABLE III: Accuracy (%) before and after unlearning on test and malicious clients' data, the number of recovery rounds, and the percentage of weights pruned (%) used with Fashion-MNIST under label-flipping attack

No. Clients (Malicious)	IID									Non-IID								
	Test Data		Malicious Data			\mathcal{R}^*	\mathcal{R}_{rec}	\mathcal{P}_{opt}		Test Data		Malicious Data			\mathcal{R}^*	\mathcal{R}_{rec}	\mathcal{P}_{opt}	
	Before	After	\mathcal{B}	Before	After					Before	After	\mathcal{B}	Before	After				
5 (2)	75	75	1	88	2	33	2	10		75	73	4	93	5	40	2	5	
10 (3)	76	75	3	81	4	31	2	10		75	74	2	82	2	32	2	4	
10 (4)	75	73	1	89	2	26	1	10		74	73	3	90	3	23	1	5	
20 (6)	75	73	3	83	4	46	3	10		73	73	2	73	3	56	2	3	
20 (9)	75	74	2	93	3	37	3	12		73	71	2	90	3	47	2	3	

TABLE IV: Accuracy (%) before and after unlearning on test and malicious clients' data, the number of recovery rounds, and the percentage of weights pruned (%) used with CIFAR-10 under backdoor attack

a notable decrease in performance on data from malicious clients is evident, while the performance on the test set remains essentially unchanged.

Regarding the number of recovery rounds, the recovery process is generally very fast in all cases. However, the Non-IID case typically requires more recovery rounds on average. This is due to the highly heterogeneous data distribution across clients in Non-IID settings, which makes the updates less stable and the contributions more overlapped. As a result, pruning also removes more benign knowledge, necessitating additional iterations to recover the model effectively. Nevertheless, the number of recovery rounds is considerably smaller than those needed to retrain from scratch. In all cases, the bound suggested in Section V-C holds.

The selection of the percentage of weights to prune follows the guidelines presented in Section V-B, demonstrating their effectiveness. In fact, the IID settings require higher percentages of pruning to remove more weights in all cases. In contrast, the Non-IID settings demand lower percentages of pruning due to the presence of more conflicting weights and a more efficient natural forgetting. In the tables, we report the optimal pruning percentage \mathcal{P}_{opt} that best balances FedUP's performance in each specific scenario to highlight its capabilities at their best. As shown, these values closely align with the general guidelines, and in many cases, they are

identical to them. Nevertheless, FedUP is resilient to variations in the pruning percentage. Notably, using a percentage slightly higher or lower than optimal does not significantly affect the overall outcome. In the former case, a little too much benign knowledge might be removed, requiring a few recovery rounds to restore performance. In the latter case, some malicious knowledge may remain, and additional rounds would be needed to ensure it is fully forgotten through natural training dynamics.

2) *False Positive Scenario*: We test this scenario in the backdoor attack setting described earlier, ensuring that the removal of a benign client does not result in an even split between benign and malicious clients, which would completely disrupt the training process and violate Assumption 1.

Table V presents the results for a false positive scenario. The forgetting data refers to the data held by the benign client that was mistakenly excluded from the process. The pruning percentages used are the same as in the respective settings of Table IV. As shown, pruning the weights associated with a benign client does not completely disrupt the model. On the contrary, performance is recovered after just a few additional training rounds. Notably, the accuracy on the forgetting data, although it decreases, still remains higher than the accuracy on the test data. This indicates that the data is not entirely forgotten, which is expected and desirable, as the information

	No. Clients (Malicious)	Test Data Before/After	Forgetting Data Before/After	\mathcal{R}_{rec}
IID	10 (4) \rightarrow 9 (4)	76 / 76	99 / 86	2
	20 (9) \rightarrow 19 (9)	75 / 74	98 / 83	3
Non-IID	10 (4) \rightarrow 9 (4)	75 / 74	91 / 80	2
	20 (9) \rightarrow 19 (9)	75 / 73	87 / 82	5

TABLE V: Model accuracy (%) before and after a pruning benign client contribution in false positive scenario.

No. Clients (Malicious)	Work	IID		Non-IID	
		Test Data Before/After	\mathcal{R}_{rec}	Test Data Before/After	\mathcal{R}_{rec}
5 (2)	FedUP	75 / 75	2	75 / 73	2
	NoT	75 / 75	10	75 / 71	20
10 (3)	FedUP	76 / 75	2	75 / 74	2
	NoT	76 / 75	5	75 / 73	20
10 (4)	FedUP	75 / 75	1	74 / 73	1
	NoT	75 / 75	6	74 / 72	20
20 (8)	FedUP	75 / 75	2	75 / 73	2
	NoT	75 / 75	15	75 / 72	20
20 (9)	FedUP	75 / 74	3	71 / 71	2
	NoT	75 / 73	10	71 / 71	12

TABLE VI: Comparison with NoT showing the accuracy (%) on test and forgetting data before and after unlearning, the number of recovery rounds, and the storage needed (MB) under backdoor attack setting.

is valid and beneficial. In fact, FedUP prunes a small percentage of weights that exhibit significant deviation from the collective average. In the case of a false positive, the weights of the benign client would, on average, be close to those of the benign cluster. Thus, pruning would predominantly remove only those weights with substantial deviations, which are likely to be less critical to the model's general performance. As shown, experimental evidence illustrates that the model can recover its utility within a few additional training rounds, even when benign contributions are pruned. Therefore, the impact of accidental benign client removal is limited and recoverable.

3) *Comparison with SOTA*: Table VI shows the comparison with NoT under the backdoor attack setting. The accuracy column on malicious data is omitted because both algorithms effectively remove malicious contributions (for FedUP results, refer to Table IV). The main difference lies in how efficiently each solution recovers the original model performance. FedUP restores the performance lost during pruning within a few rounds, while NoT struggles to do so and, in some cases, fails to fully recover even after many rounds. This trend is especially evident in the Non-IID setting. This is because FedUP targets specific malicious weights, whereas NoT disrupts the entire first layer indiscriminately. Such broad disruption is highly detrimental, requiring significantly more recovery rounds to regain original performance.

Table VII presents, instead, the comparison with NoT and FedEraser in the benign setting. The pruning percentage selected for FedUP is $\mathcal{P} = 30\%$. This is a particularly unique configuration, as it is both IID and does not involve malicious clients or clients with distinct data. Even in this context, the percentage of weights to prune is consistent with

Work	Test Data Before/After	Forgetting Data Before/After	\mathcal{R}_{rec}	Storage (MB)
FedEraser	56 / 56	76 / 56	20	54
NoT	56 / 50	74 / 50	20	0.2
FedUP	56 / 56	74 / 56	3	2.7

TABLE VII: Comparison with FedEraser and NoT, reporting test and forgetting accuracy (%) before and after unlearning, number of recovery rounds, and storage requirements (MB) in a benign setting with 10 clients and IID data.

expectations. All algorithms successfully unlearn the client's data, as evidenced by the significant drop in the global model's performance on that client's data. Additionally, they can recover the performance lost during the unlearning process, restoring the model's effectiveness on retained data. However, the key difference is that while FedEraser and NoT fail to rapidly recover the original performance, FedUP fully restores the model's performance in just a few recovery rounds. In contrast, both NoT and FedEraser require at least 20 additional training rounds to achieve similar results. NoT even fails to fully recover the lost performance, as the test accuracy after the recovery rounds remains below the original level. Moreover, FedUP requires only 1/20 of the storage needed by FedEraser to execute, as it only requires the updates and the global model from the last training round. In this regard, NoT is the most efficient solution since it requires only the global model, although its efficiency is not significantly different from that of FedUP. This comparison highlights how our algorithm is storage-efficient and significantly faster than SOTA solutions while remaining equally effective. Furthermore, this demonstrates that, despite being specifically designed for malicious environments, FedUP can also be effectively applied in benign settings as well.

4) *Ablation Study*: To assess the impact of FedUP, we conducted three ablation studies, shown in Figure 5. In the first study, malicious clients are excluded from training, but no pruning is applied, so the only mitigating factor is natural forgetting. In the second study, a percentage of weights is randomly pruned, following the same threshold guidelines used in our experiments, but without selecting weights based on their contribution. The third study applies the same pruning percentage but targets the weights with the highest estimated malicious magnitude, simulating a basic yet focused unlearning strategy. All experiments use the previously described backdoor attack setting and evaluate various proportions of malicious clients under both IID and Non-IID data distributions. Due to space constraints, results are aggregated, with variance across the different configurations shown separately for the IID and Non-IID cases. In the figures, red indicates accuracy on test data, blue shows accuracy on malicious data, and green represents the baseline \mathcal{B} accuracy on malicious data achieved through full retraining. At epoch 0, the malicious clients are removed, and 10 subsequent epochs represent the recovery rounds.

As shown, natural forgetting can initially reduce some of the malicious influence on the model weights. However, consistent with findings in prior work [31], it is not sufficient

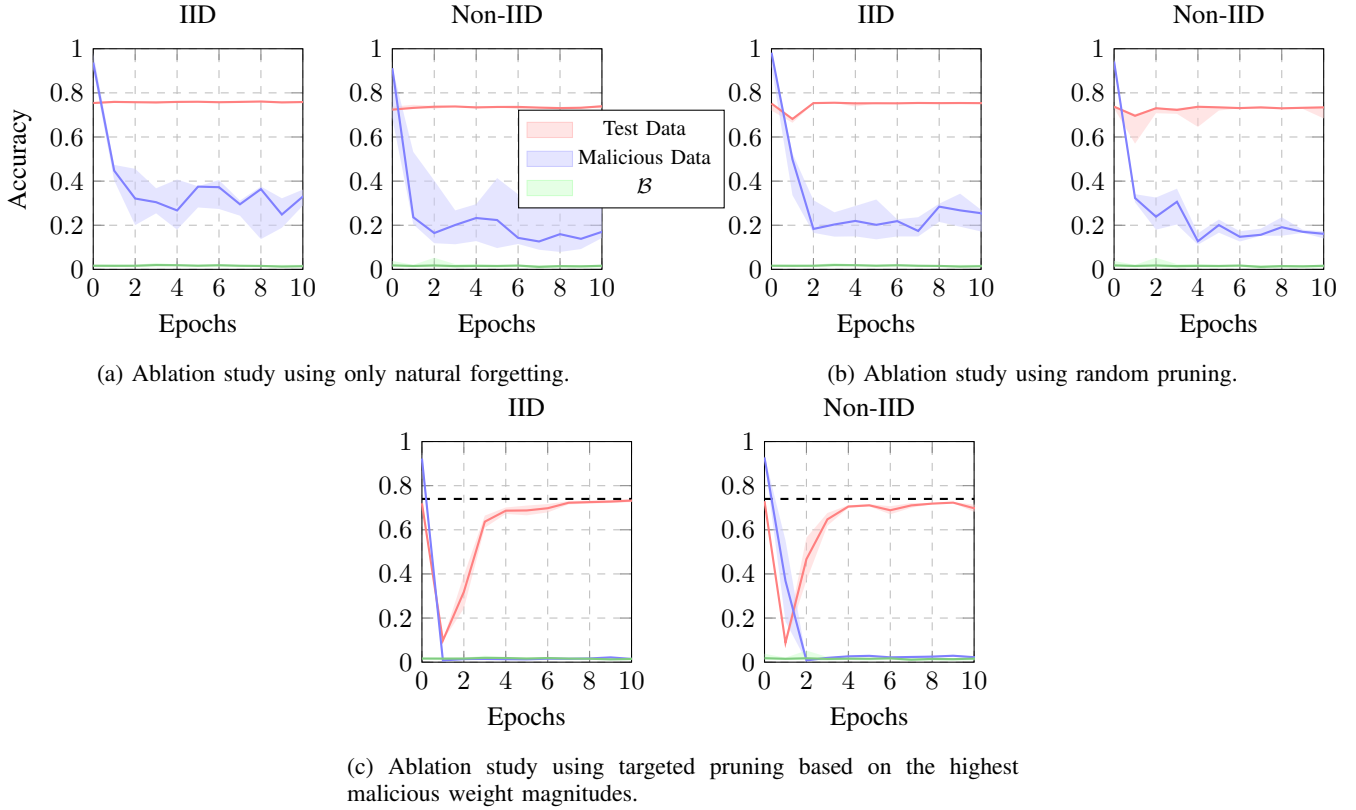


Fig. 5: (a) Effect of natural forgetting after removing malicious clients without using FedUP. (b) Effect of random pruning for unlearning. (c) Effect of targeted pruning based on the highest malicious magnitude weights. The plots show accuracy on test data (red), accuracy on malicious data (blue), and baseline malicious accuracy after full retraining (green). Results are averaged across varying percentages of malicious clients.

to completely eliminate the impact of malicious updates. Random pruning can help improve unlearning performance, but its effectiveness remains limited. This is likely because the pruning percentage is limited and the weights removed are often either benign or insignificant. Nevertheless, these two approaches fail to fully eliminate malicious contributions. Targeted pruning based on the highest magnitude weights produces better results, confirming that selecting meaningful weights is crucial. Although this is the only method that effectively removes malicious contributions, it struggles to quickly restore performance, typically requiring between 8 and 10 rounds, which is significantly more than FedUP and sometimes failing to reach the original accuracy. This is because it focuses solely on the malicious weights without considering benign ones, which risks removing weights that are important for benign clients. These experiments demonstrate that each component of FedUP is essential for effective and reliable unlearning of malicious contributions.

VII. CONCLUSION

In this paper, we introduced FedUP, a novel FU algorithm specifically designed to effectively and efficiently remove malicious contributions in adversarial FL environments. Unlike conventional FU methods that assume cooperative clients, FedUP operates under a strong adversarial threat model where up to 50% - 1 of clients may be malicious and colluding.

Our approach leverages a lightweight pruning mechanism that zeros out the highest-magnitude weights most dissimilar between benign and malicious client updates, effectively isolating malicious influence despite overlapping updates. To recover from the performance loss caused by pruning, FedUP performs a limited number of training rounds, avoiding costly retraining from scratch. Extensive experiments across IID and Non-IID settings, multiple models, and two attack types (label-flipping and backdoor) show that FedUP reliably reduces malicious impact and restores performance on benign data. Additionally, it outperforms SOTA FU techniques in both speed and storage efficiency, making it a practical solution for real-world FL deployments.

REFERENCES

- [1] V. Tolpegin, S. Truex, M. E. Gursoy, and L. Liu, "Data poisoning attacks against federated learning systems," in *Computer security—ESORICS 2020: 25th European symposium on research in computer security, ESORICS 2020, guildford, UK, September 14–18, 2020, proceedings, part i* 25, pp. 480–501, Springer, 2020.
- [2] G. Xia, J. Chen, C. Yu, and J. Ma, "Poisoning attacks in federated learning: A survey," *Ieee Access*, vol. 11, pp. 10708–10722, 2023.
- [3] Microsoft Threat Intelligence, "Volt typhoon targets us critical infrastructure with living-off-the-land techniques." URL <https://www.microsoft.com/en-us/security/blog/2023/05/24/volt-typhoon-targets-us-critical-infrastructure-with-living-off-the-land-techniques/>, 2023. Accessed: 2024-07-01.

- [4] U.S. Department of Homeland Security, "Secure cyberspace and critical infrastructure." URL <https://www.dhs.gov/secure-cyberspace-and-critical-infrastructure>, 2023. Accessed: 2024-07-01.
- [5] S. Li, Y. Cheng, Y. Liu, W. Wang, and T. Chen, "Abnormal client behavior detection in federated learning," *arXiv preprint arXiv:1910.09933*, 2019.
- [6] L. Wu, S. Guo, J. Wang, Z. Hong, J. Zhang, and Y. Ding, "Federated unlearning: Guarantee the right of clients to forget," *IEEE Network*, vol. 36, no. 5, pp. 129–135, 2022.
- [7] L. Bourtole, V. Chandrasekaran, C. A. Choquette-Choo, H. Jia, A. Travers, B. Zhang, D. Lie, and N. Papernot, "Machine unlearning," in *2021 IEEE Symposium on Security and Privacy (SP)*, pp. 141–159, IEEE, 2021.
- [8] C. Wu, S. Zhu, and P. Mitra, "Federated unlearning with knowledge distillation," *arXiv preprint arXiv:2201.09441*, 2022.
- [9] R. Kassab and O. Simeone, "Federated generalized bayesian learning via distributed stein variational gradient descent," *IEEE Transactions on Signal Processing*, vol. 70, pp. 2180–2192, 2022.
- [10] Y. Zhao, P. Wang, H. Qi, J. Huang, Z. Wei, and Q. Zhang, "Federated unlearning with momentum degradation," *IEEE Internet of Things Journal*, vol. 11, no. 5, pp. 8860–8870, 2024.
- [11] X. Cao, J. Jia, Z. Zhang, and N. Z. Gong, "Fedrecover: Recovering from poisoning attacks in federated learning using historical information," in *2023 IEEE Symposium on Security and Privacy (SP)*, pp. 1366–1383, 2023.
- [12] X. Guo, P. Wang, S. Qiu, W. Song, Q. Zhang, X. Wei, and D. Zhou, "Fast: Adopting federated unlearning to eliminating malicious terminals at server side," *IEEE Transactions on Network Science and Engineering*, 2023.
- [13] J. Geiping, H. Bauermeister, H. Dröge, and M. Moeller, "Inverting gradients-how easy is it to break privacy in federated learning?," *Advances in neural information processing systems*, vol. 33, pp. 16937–16947, 2020.
- [14] S. Li, Y. Cheng, W. Wang, Y. Liu, and T. Chen, "Learning to detect malicious clients for robust federated learning," *arXiv preprint arXiv:2002.00211*, 2020.
- [15] Z. Zhang, X. Cao, J. Jia, and N. Z. Gong, "Fldetector: Defending federated learning against model poisoning attacks via detecting malicious clients," in *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pp. 2545–2555, 2022.
- [16] Y. Jiang, W. Zhang, and Y. Chen, "Data quality detection mechanism against label flipping attacks in federated learning," *IEEE Transactions on Information Forensics and Security*, vol. 18, pp. 1625–1637, 2023.
- [17] X. Sheng, W. Bao, and L. Ge, "Robust federated unlearning," in *Proceedings of the 33rd ACM International Conference on Information and Knowledge Management*, pp. 2034–2044, 2024.
- [18] J. Chen, Z. Lin, W. Lin, W. Shi, X. Yin, and D. Wang, "Fedmua: Exploring the vulnerabilities of federated learning to malicious unlearning attacks," *IEEE Transactions on Information Forensics and Security*, 2025.
- [19] N. M. Jebreel, J. Domingo-Ferrer, D. Sánchez, and A. Blanco-Justicia, "Lfighter: Defending against the label-flipping attack in federated learning," *Neural Networks*, vol. 170, pp. 111–126, 2024.
- [20] C. Xie, K. Huang, P.-Y. Chen, and B. Li, "Dba: Distributed backdoor attacks against federated learning," in *International conference on learning representations*, 2019.
- [21] E. Bagdasaryan, A. Veit, Y. Hua, D. Estrin, and V. Shmatikov, "How to backdoor federated learning," in *International conference on artificial intelligence and statistics*, pp. 2938–2948, PMLR, 2020.
- [22] W. Huang, M. Ye, Z. Shi, G. Wan, H. Li, B. Du, and Q. Yang, "Federated learning for generalization, robustness, fairness: A survey and benchmark," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2024.
- [23] X. Mu, K. Cheng, Y. Shen, X. Li, Z. Chang, T. Zhang, and X. Ma, "Feddmc: Efficient and robust federated learning via detecting malicious clients," *IEEE Transactions on Dependable and Secure Computing*, 2024.
- [24] X. Cao, M. Fang, J. Liu, and N. Z. Gong, "Fltrust: Byzantine-robust federated learning via trust bootstrapping," *arXiv preprint arXiv:2012.13995*, 2020.
- [25] C. Fung, C. J. Yoon, and I. Beschastnikh, "The limitations of federated learning in sybil settings," in *23rd International Symposium on Research in Attacks, Intrusions and Defenses (RAID 2020)*, pp. 301–316, 2020.
- [26] S. Shen, S. Tople, and P. Saxena, "Auror: Defending against poisoning attacks in collaborative deep learning systems," in *Proceedings of the 32nd annual conference on computer security applications*, pp. 508–519, 2016.
- [27] Z. Gu and Y. Yang, "Detecting malicious model updates from federated learning on conditional variational autoencoder," in *2021 IEEE international parallel and distributed processing symposium (IPDPS)*, pp. 671–680, IEEE, 2021.
- [28] J. Ma, M. Xie, and G. Long, "Personalized federated learning with robust clustering against model poisoning," in *International conference on advanced data mining and applications*, pp. 238–252, Springer, 2022.
- [29] X. Cao, J. Jia, and N. Z. Gong, "Provably secure federated learning against malicious clients," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 35, pp. 6885–6893, 2021.
- [30] J. Le, D. Zhang, X. Lei, L. Jiao, K. Zeng, and X. Liao, "Privacy-preserving federated learning with malicious clients and honest-but-curious servers," *IEEE Transactions on Information Forensics and Security*, vol. 18, pp. 4329–4344, 2023.
- [31] N. Romandini, A. Mora, C. Mazzocca, R. Montanari, and P. Bellavista, "Federated unlearning: A survey on methods, design guidelines, and evaluation metrics," *IEEE Transactions on Neural Networks and Learning Systems*, 2024.
- [32] G. Liu, X. Ma, Y. Yang, C. Wang, and J. Liu, "Federaser: Enabling efficient client-level data removal from federated learning models," in *2021 IEEE/ACM 29th International Symposium on Quality of Service (IWQOS)*, pp. 1–10, 2021.
- [33] J. Wang, S. Guo, X. Xie, and H. Qi, "Federated unlearning via class-discriminative pruning," in *Proceedings of the ACM Web Conference 2022*, pp. 622–632, 2022.
- [34] C. Wu, X. Yang, S. Zhu, and P. Mitra, "Toward cleansing backdoored neural networks in federated learning," in *2022 IEEE 42nd International Conference on Distributed Computing Systems (ICDCS)*, pp. 820–830, IEEE, 2022.
- [35] J. Nocedal, "Updating quasi-newton matrices with limited storage," *Mathematics of computation*, vol. 35, no. 151, pp. 773–782, 1980.
- [36] C. Wu, S. Zhu, P. Mitra, and W. Wang, "Unlearning backdoor attacks in federated learning," in *2024 IEEE Conference on Communications and Network Security (CNS)*, pp. 1–9, IEEE, 2024.
- [37] Y. H. Khalil, L. Brunswic, S. Lamghari, X. Li, M. Beitollahi, and X. Chen, "Not: Federated unlearning via weight negation," *arXiv preprint arXiv:2503.05657*, 2025.
- [38] Y. Jiang, J. Shen, Z. Liu, C. W. Tan, and K.-Y. Lam, "Towards efficient and certified recovery from poisoning attacks in federated learning," *IEEE Transactions on Information Forensics and Security*, 2025.
- [39] Y. He, G. Kang, X. Dong, Y. Fu, and Y. Yang, "Soft filter pruning for accelerating deep convolutional neural networks," *arXiv preprint arXiv:1808.06866*, 2018.
- [40] L. Cai, Z. An, C. Yang, and Y. Xu, "Softer pruning, incremental regularization," in *2020 25th international conference on pattern recognition (ICPR)*, pp. 224–230, IEEE, 2021.
- [41] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas, "Communication-efficient learning of deep networks from decentralized data," in *Artificial intelligence and statistics*, pp. 1273–1282, PMLR, 2017.
- [42] R. M. French, "Catastrophic forgetting in connectionist networks," *Trends in cognitive sciences*, vol. 3, no. 4, pp. 128–135, 1999.
- [43] X. Gao, X. Ma, J. Wang, Y. Sun, B. Li, S. Ji, P. Cheng, and J. Chen, "Verifi: Towards verifiable federated unlearning," *IEEE Transactions on Dependable and Secure Computing*, 2024.
- [44] R. Yu, A. Li, C.-F. Chen, J.-H. Lai, V. I. Morariu, X. Han, M. Gao, C.-Y. Lin, and L. S. Davis, "Nisp: Pruning networks using neuron importance score propagation," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 9194–9203, 2018.
- [45] P. Molchanov, A. Mallya, S. Tyree, I. Frosio, and J. Kautz, "Importance estimation for neural network pruning," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 11264–11272, 2019.
- [46] W. Gill, A. Anwar, and M. A. Gulzar, "Provfl: Client-driven interpretability of global model predictions in federated learning," *arXiv preprint arXiv:2312.13632*, 2023.
- [47] Y. LeCun, J. Denker, and S. Solla, "Optimal brain damage," in *Advances in Neural Information Processing Systems* (D. Touretzky, ed.), vol. 2, Morgan-Kaufmann, 1989.
- [48] S. Han, J. Pool, J. Tran, and W. Dally, "Learning both weights and connections for efficient neural network," *Advances in neural information processing systems*, vol. 28, 2015.
- [49] J. Frankle and M. Carbin, "The lottery ticket hypothesis: Finding sparse, trainable neural networks," *arXiv preprint arXiv:1803.03635*, 2018.
- [50] T. Wu, X. Li, D. Zhou, N. Li, and J. Shi, "Differential evolution based layer-wise weight pruning for compressing deep neural networks," *Sensors*, vol. 21, no. 3, p. 880, 2021.

- [51] J. Liu, P. Ram, Y. Yao, G. Liu, Y. Liu, P. SHARMA, S. Liu, *et al.*, “Model sparsity can simplify machine unlearning,” *Advances in Neural Information Processing Systems*, vol. 36, 2024.
- [52] Z. Liu, M. Sun, T. Zhou, G. Huang, and T. Darrell, “Rethinking the value of network pruning,” *arXiv preprint arXiv:1810.05270*, 2018.
- [53] D. Blalock, J. J. Gonzalez Ortiz, J. Frankle, and J. Gutttag, “What is the state of neural network pruning?,” *Proceedings of machine learning and systems*, vol. 2, pp. 129–146, 2020.
- [54] A. Krizhevsky, “Learning multiple layers of features from tiny images,” tech. rep., 2009.
- [55] H. Xiao, K. Rasul, and R. Vollgraf, “Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms,” *arXiv preprint arXiv:1708.07747*, 2017.
- [56] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, “Gradient-based learning applied to document recognition,” *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.
- [57] N. M. Jebreel and J. Domingo-Ferrer, “Fl-defender: Combating targeted attacks in federated learning,” *Knowledge-Based Systems*, vol. 260, p. 110178, 2023.
- [58] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.