FAST: Adopting Federated Unlearning to Eliminating Malicious Terminals at Server Side

Xintong Guo, Pengfei Wang, *Member, IEEE,* Sen Qiu, *Member, IEEE,* Wei Song, *Student Member, IEEE,* Qiang Zhang, *Member, IEEE,* Xiaopeng Wei, *Member, IEEE,*and Dongsheng Zhou, *Member, IEEE*

Abstract—The emergence of the right to be forgotten has sparked interest in federated unlearning. Researchers utilize federated unlearning to address the issue of removing user contributions. However, practical implementation is challenging when malicious clients are involved. In this paper, we propose FAST, i.e., Adopting Federated unleArning to Eliminating MaliciouS Terminals at Server Side, a framework with three main components: 1) eliminating contributions of malicious clients: the central server records and subtracts updates of malicious clients from the global model, 2) judging unlearning efficiency: we model a mechanism to assess unlearning efficiency and prevent over-unlearning, and 3) remedying unlearning model performance: the central server utilizes a benchmark dataset to remedy model bias during unlearning. Experimental results demonstrate that FAST can achieve 96.98% accuracy on the MNIST dataset with 40% malicious clients, offering a 16x speedup to retraining from scratch. Meanwhile, it can recover model utility with high efficiency, and extensive evaluations of four real-world datasets demonstrate the validity of our proposed scheme.

Index Terms—Federated unlearning, malicious terminal, parameter update, federated learning, server side.

1 Introduction

With the growing emphasis on data privacy preservation [1], Google proposed Federated Learning (FL) [2], [3], [4] in 2016 as a machine learning paradigm, a distributed machine learning technique that provides reasonable preservation of data, which has since greatly developed and is widely applied. Also, relevant laws have introduced corresponding policies, such as the General Data Protection Regulation (GDPR) in the European Union [5] and the California Consumer Privacy Act (CCPA) [6] in the United States, proposing the right to be forgotten, where users can ask models to forget their historical contributions. This results in the model's need to utilize user data while simultaneously meeting user requirements. To address these challenges, an increasing number of researchers focus on federated unlearning. Aiming to successfully eliminate the historical contributions of clients requesting to be forgotten without compromising model performance and maintaining privacy among different clients.

- *This work was supported by the National Natural Science Foundation of China under grant 62202080, the China Postdoctoral Science Foundation under Grant 2023M733354, the 111 project under grant D23006, the Support Plan for Key Field Innovation Team of Dalian under grant 2021RT06, the Open Project of Key Laboratory for Advanced Design and Intelligent Computing of the Ministry of Education under grant ADIC20220001, and the Fundamental Research Funds for the Central Universities under grant DUT23YG122.*

- *Xintong Guo and Dongsheng Zhou are with the Key Laboratory of Advanced Design and Intelligent Computing (Ministry of Education), School of Software Engineering, Dalian University, Dalian, China, 116622. E-mail: {guoxintong@s.,zhouds@}dlu.edu.cn.*
- *Pengfei Wang, Wei Song, Xiaopeng Wei and Qiang Zhang are with the School of Computer Science and Technology, Dalian University of Technology, Dalian 116024, China, and Key Laboratory of Social Computing and Cognitive Intelligence (Dalian University of Technology), Ministry of Education, Dalian, 116024 China. E-mails: {wangpf@, songv@mail., zhangq, xpwei@}dlut.edu.cn.*
- *Sen Qiu is with the School of Control Science and Engineering, Dalian University of Technology, Dalian 116024, China. E-mail: qiu@dlut.edu.cn.*
- *(Corresponding authors: Pengfei Wang and Dongsheng Zhou)*

In recent years, researchers have resorted to federated unlearning to tackle the right-to-be-forgotten challenge, and despite tremendous advancements made in this area, the scenario of existing malicious users during FL goes unexplored. Among others, the traditional approach to federated unlearning is to train from scratch. This is done by eliminating the clients who request to be forgotten and then applying the remaining clients to a new FL process. But this method costs twice as much time. FedEraser [7] is the first flawed method for federated unlearning. It sets the index at fixed intervals so that the most recent model parameters are returned when the user makes an unlearning request. Although existing federated unlearning methods can partially address the aforementioned challenges contributed by unlearning clients, some remaining issues still need to be addressed.

Although some recent work [8], [9], [10] tried to design more novel federated unlearning methods because of experimental hypothesis concerns, they all assume that the data provided by the clients involved in training are correct. And, most of their methods perform unlearning operations from the client's perspective. Considering the client's willingness to cooperate in completing federated unlearning is necessary. [11] explored to complete the unlearning process at the central server, however, it did not consider the correctness of the data from the individual clients. If there are malicious clients among the participating clients, meaning they possess private data with significant errors, such as label inaccuracies or noisy data, the performance accuracy of the final global model generated by FL becomes extremely poor. To eliminate contributions from malicious clients, the existing methods are infeasible. As a result, solving the scenario in which malicious clients participate in training is critical for FL. How to effectively and efficiently eliminate target clients' contributions is not trivial.

Since the clients selected for FL are random, the phenomenon of malicious clients participating in FL is inevitable, and appropriate measures should be devised. This scenario holds value for exploration. It not only wastes the cost of conducting FL but also results in suboptimal global model accuracy. Therefore, we propose transferring the authority for federated unlearning to the central server. In our scenario, we assume that malicious clients have already participated in FL and the central server has generated the final global model. Even so, clearing the historical contributions of malicious clients and restoring the biases introduced by federated unlearning remain challenging.

To address the aforementioned challenges, we propose a federated unlearning algorithm to eliminate the historical contributions of malicious clients named FAST, i.e., Adopting Federated unleArning to Eliminating MaliciouS Terminals at Server Side. The basic idea of FAST is to erase malicious clients' influence by subtracting the historical parameter updates of each malicious client from the central server and to recover model performance biases through additional training of the unlearning model utilizing a small set of benchmark samples stored on the server. FAST achieves a similar model performance to completely retraining FL from scratch while requiring less running time. It is able to leverage the final global model results from previous FL without wasting resources. To effectively remove malicious clients' contributions, FAST integrates three major technical components: 1) eliminating contributions of malicious clients, 2) judging unlearning efficiency, and 3) remedying unlearning model performance. Initially, the process of clearing malicious client contributions builds upon the method outlined in [11], expanding its scope to include the removal of impacts from multiple clients, rather than being confined to just one. Second, the central server evaluates whether the desired outcome has been achieved during each unlearning operation, avoiding excessive unlearning that might produce adverse results. Lastly, correcting the performance of the unlearning model involves training the unlearning model and applying benchmark samples to counteract performance biases that arose during previous unlearning processes.

Our proposed approach is validated through empirical evaluations. We construct an experimental model utilizing real datasets and convolutional neural network architectures. We then perform extensive experiments on four image datasets, namely MNIST, FMNIST, CIFAR-10, and SVHN, utilizing these experimental models. The results of these experiments demonstrate that FAST achieves a 16x speedup in mitigating the impact of malicious clients, surpassing retraining from scratch. Particularly, FAST advances in both removing malicious client contributions and judging the unlearning effect, which contributes to superior performance in federated unlearning that significantly outperforms benchmark approaches.

The main contributions of this paper are summarized as follows:

- To the best of our knowledge, this is the first work to investigate the presence of malicious clients among FL participants, enabling FAST to effectively eliminate the impact of malicious clients and efficiently recover from performance bias.
- We choose the server to conduct FAST with superior computing ability, which can save time. FAST does not need to ask for permission from the client to execute unlearning progress and also supports unlearning multi-party malicious contributions.
- We design the unlearning operation to be executed at the end of FL and not limited to the current turn. A mechanism for judging the efficiency of unlearning is also added, which can prematurely end the process of eliminating the contributions of malicious clients.
- We perform several rounds of normal training on the unlearning model using a small benchmark dataset stored in advance on the central server. This is used to recover model performance bias incurred in the unlearning phase.

TABLE 1: Key notations.

Notation	Definition
N	The total number of FL participating clients
J	The number of malicious clients
F	The number of excellent clients
t	The index of FL time slots
T	The total number of FL
k	The index of eliminating contributions time slots
M	The FL final global model
C_n	The nth participating client
D_n	The local data of client C_n
D	The total number of all FL participating clients
C_J	The set of malicious clients
C_F	The set of excellent clients
M_t	The FL global model in iteration t
M_n^t	The FL local model of client C_n in iteration t
M'	The final unlearning global model
ΔM^t	The parameter updates of FL global model in iteration t
ΔM_n^t	The parameter updates of local model of client C_n in iteration t
M_t'	The FL unlearning global model in iteration t
$\Delta M_t'$	The parameter updates of unlearning model in iteration t
M_{cur}'	The current unlearning model
M_{pre}'	The previous unlearning model
α_t	The skew generated in unlearning stage
D_b	The benchmark dataset
M_{UL}'	The unlearning model after unlearning stage
$Test$	The function for testing model performance
ACC	The accuracy of model performance

This paper has the following structure. Section 2 presents the problem definition. In Section 3, we elaborate on FAST's design with theoretical analysis. Section 4 presents extensive experiments to evaluate FAST performance, followed by a review of related work in Section 5. Finally, Section 6 concludes this paper.

2 Problem Definition

In this section, we first describe the concept of federated unlearning, which is a derivative domain. We then proceed to formally define malicious clients and the problem of excessive unlearning. For ease of reference, we have listed the key notations in Table 1.

2.1 Federated Unlearning

Federated unlearning is a derivative field of FL and machine unlearning [12]. It not only enables the unlearning of target clients but also preserves the privacy of data from participating clients. Federated unlearning primarily stems from the right to be forgotten, as outlined in the GDPR regulations, presenting an innovative solution in the realm of FL. The primary objective of federated unlearning is to eliminate the contributions made by clients whose data is requested to be forgotten [13]. This also serves to mitigate legal risks. Moreover, federated unlearning is applicable to mitigating the impacts caused by malicious clients, and this paper primarily focuses on the application of federated unlearning in this context.

During the FL phase, the server randomly selects N participating training clients, represented as $\{C_1, C_2, ..., C_n\}$ from a set of K clients (K>N). The set of training datasets is represented as

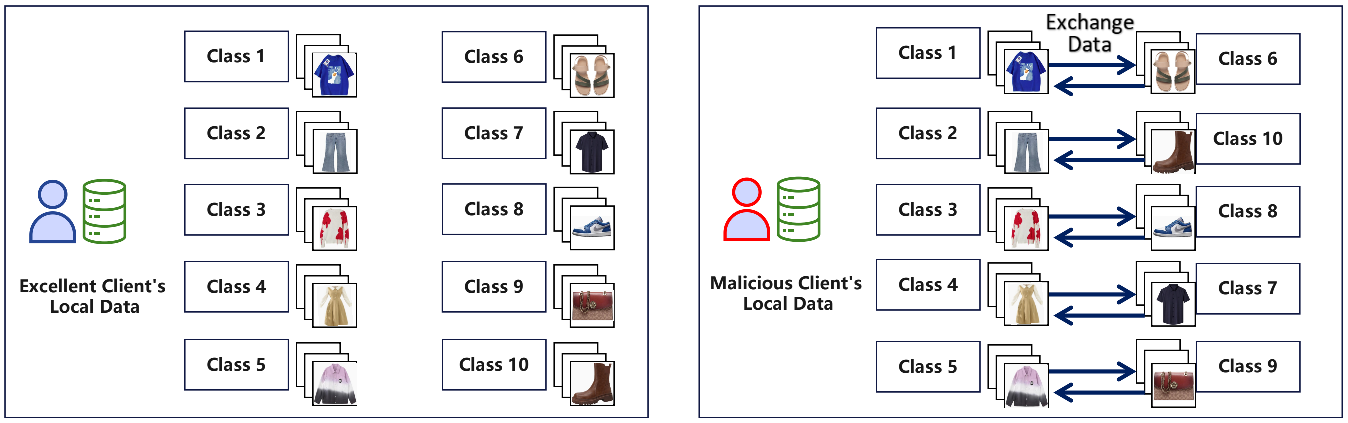


Fig. 1: Excellent clients data categories and targeted attacks of malicious clients.

$D = \{D_1, D_2, ..., D_n\}$. Suppose a client C_u requests the model to forget its contribution. We perform the unlearning operation on C_u, which results in the training dataset D/D_u.

In this paper, we employ the well-known FedAvg algorithm [14] as the FL aggregation function. For each iteration t, $t \in \{1, ..., T\}$, M_t represents the global model at iteration t. For each local model M_t, N participating clients. The local model updates ΔM_n^t of clients C_n, $n \in \{1, ..., N\}$ received from the central server and aggregated, the server updates the global model M_t. Therefore, the FL problem can be expressed as

$$M_t = M_{t-1} + \frac{1}{N}\sum_{n=1}^{N} \Delta M_n^t. \tag{1}$$

The traditional federated unlearning strategy, training from scratch, is similar to the FL process. The key difference is that during retraining, the clients involved do not include the ones that need to be forgotten, such as malicious clients. In theory, the problem of federated unlearning can be formulated as follows:

$$M^{'} = M_0 + \sum_{t=1}^{T} \Delta M_F^t, \tag{2}$$

$$\Delta M_F^t = \frac{1}{F}\sum_{n=1}^{F} \Delta M_n^t. \tag{3}$$

2.2 Setup

We can formalize the federated unlearning problem in terms of two key players: the central server S and the collaborating clients C. In our setting, the server S controls the entire training process. It generates the next round of global models by collecting and aggregating local model updates from participating clients. These participating clients $C = \{C_1, C_2, ..., C_n\}, n \in \{1, ..., N\}$ hold different local datasets D_n and form the training dataset $D = \{D_1, D_2, ..., D_n\}$, where N is the total number of all participating clients.

In this context, we utilize the term "excellent client" to refer to those clients with reasonable data who do not need data erasure. We also use the term "malicious client" to refer to those clients with poor-quality data who must perform data erasure.

2.3 Malicious Client

FL faces significant challenges due to malicious clients, posing a major roadblock to its practical deployment [15], [16], [17], [18]. The central server has no control over the local data held by the clients. This makes it challenging to judge the quality of participating clients' datasets since the clients have absolute control over the local data during FL. Moreover, the clients can act independently, and the server lacks control over their behavior. Malicious clients can easily corrupt the training process by injecting biased or poisoned data into their own datasets, leading to inaccurate global models. Malicious clients can deceive the central server by providing false information or impressions about their nodes' quality, e.g., providing overestimated node quality [19]. In this paper, we consider targeted attacks that aim to compromise the model's performance against malicious clients while preserving the model's performance for other high-quality clients. We demonstrate targeted attacks designed for malicious clients in the experiments, as shown in Fig. 1, e.g., the FMNIST dataset, these attacks are designed to modify the labels of data instances exclusively owned by the malicious client. . Targeted attacks can result in catastrophic consequences.

To mitigate the effects of malicious clients, it's crucial to detect them early and remove them from FL in a timely manner [20]. However, in cases where malicious clients are not detected in time, it is essential to have countermeasures that eliminate their impact on the final FL model. This is where Federated Unlearning comes in. Federated unlearning can remove such clients' contributions to the global model, enhancing system performance and integrity. As a result, we discuss the latter scenario and propose FAST.

We assume that there are malicious clients $C_J = \left\{C_1, C_2, ..., C_j\right\}, j \in \{1, ..., J\} \backslash F$, where F is the index of the excellent clients set C_F. The local data $D_J = \left\{D_1, D_2, ..., D_j\right\}$ of C_J violate the model training requirements for correctness. The participation of these clients in FL creates almost useless training models, so the central server needs to perform unlearning operations on these clients. Note that when $F = 0$, it represents a special case in which the server executes unlearning operations on all clients.

2.4 Over-unlearning

The concept of overfitting was introduced in Machine Learning [21], which refers to the phenomenon of closely matching a specific dataset but failing to generalize well to other data or predict future observations. We propose a similar concept of *over-unlearning*. Over-unlearning is a phenomenon in which the model focuses so much on the impact of deleted clients' contributions that need to be forgotten that it loses predictive power for other data that are not required to be forgotten. It is reasonable to assume that over-unlearning occurs when using federated unlearning methods. Therefore, it is essential to develop countermeasures to mitigate or tackle this phenomenon.

One way to address over-unlearning is to adopt a selective unlearning strategy that allows for more fine-grained control over the unlearning process. Selective unlearning involves identifying the specific data or clients that need to be removed from the model. It involves only unlearning their contributions, rather than blindly removing all their data. This approach ensures that the model retains its ability to predict skillfully for other data while improving its performance by eliminating undesirable elements.

In summary, addressing over-unlearning is a crucial consideration when utilizing federated unlearning methods. Selective unlearning is the potential strategy for mitigating the impact of over-unlearning and ensuring that the model retains its ability to predict well for a wide range of inputs.

3 Design of FAST

Algorithm 1 FAST algorithm

```
Input: FL final global model $M_T$, global FL epoch $T$, extra
    training epoch $L$, total number of clients $N$, total number
    of excellent clients $F$, total number of malicious clients $J$,
    historical updates $\Delta M_N^t$ of target clients at round $t$
Output: final unlearning model $M^{'}$
 1: Initializing current unlearning model $M_{cru}^{'}$, previous unlearn-
    ing model $M_{pre}^{'}$
 2: Initializing $i = 1, M_{cru}^{'} = M_{pre}^{'} = M$
 3: for epoch round $k$=1,2,...,$T$ do
 4:     if $i == 1$ then
 5:         $M_k^{'} = M_{cru}^{'} - \frac{J}{N}\Delta M_N^k$
 6:         $M_{pre}^{'} = M_{cur}^{'}$, $M_{cur}^{'} = M_k^{'}$
 7:         if $Test(M_{pre}^{'}) \geq Test(M_{cru}^{'})$ then
 8:             $M_{cru}^{'} = M_{pre}^{'}$
 9:             $i = 0$
10:         end if
11:     else
12:         exit
13:     end if
14: end for
15: for each round $r = 1, 2, ..., R$ do
16:     $M_{UL}^{'} = Train(M_{cru}^{'}, D_b)$
17: end for
18: $M^{'} = M_{UL}^{'}$
19: return $M^{'}$
```

In this section, we introduce FAST as a solution to overcome the challenges encountered by federated learning systems in the presence of malicious clients. FAST is a federated unlearning method for malicious clients participating in FL. It enables the removal of historical contributions from malicious clients, leverages the final global model in FL, and significantly reduces the overall unlearning cost. Algorithm 1 outlines the overall FAST workflow. FAST contains three parts: eliminating contributions of malicious clients (Algorithm 1, lines 3-5), judging unlearning efficiency (Algorithm 1, lines 6-10), and remedying unlearning model performance (Algorithm 1, lines 15-16). To implement this method, the server must record parameter updates from malicious clients in each round. It must also store a benchmark dataset in advance on the central server. The main idea is to initially apply the central server to eliminate parameter updates from malicious clients. It then determines whether the unlearning is excessive, and finally restores the unlearning model bias through the additional training method.

3.1 Federated Unlearning Stage

3.1.1 Eliminating Contributions of Malicious Clients

To quickly eliminate the historical contributions of malicious clients, and comply with existing privacy laws, we mainly utilize a method that modifies the model parameters to remove these contributions of malicious clients in each round. This is achieved by having the server retain all parameter updates made by participating malicious clients during FL training. Then, by leveraging the global model parameters and the parameter updates from malicious clients, the server can generate an unlearning model by directly modifying the model parameters. Crucially, our approach does not rely on accessing the data owned by malicious clients, nor does it depend on their willingness to cooperate with the unlearning process.

We aim to nullify the contributions of malicious clients C_J to the final global model M. As Equation 1 demonstrates, the generated FL global model in each round relies on the previous round's global model. The initial global model is denoted M_0, and we calculate the subsequent global model based on it. In (4), the second term refers to the parameter updates of the global model in each iteration corresponding to each training round.

$$M_t = M_0 + \sum_{t=1}^{T} \Delta M^t. \tag{4}$$

It is imperative to treat the two types of clients differently when malicious clients are involved in FL. Specifically, we should separate parameter updates from excellent clients C_F and malicious clients C_J during training. This means that the parameter update of the global model for each round is the weighted sum of the parameter updates from excellent clients C_F, as well as the parameter updates from malicious clients C_J, which becomes the following equation.

$$\Delta M^t = \frac{1}{N}\sum_{n=1}^{N} \Delta M_n^t = \frac{1}{N}\sum_{n=1}^{F} \Delta M_n^t + \frac{1}{N}\sum_{n=1}^{J} \Delta M_n^t. \tag{5}$$

During the federated unlearning process, the ideal parameter update for the global model $\Delta M_t^{'}$ of training round t is obtained by aggregating the local models of excellent clients C_J. Alternatively, as shown in Equation 6, we can produce this ideal update by subtracting the updates of local models of malicious clients C_F and multiplying them by the corresponding scale factor.

$$\Delta M_t^{'} = \frac{1}{N-J}\sum_{n=1}^{N-J} \Delta M_n^t = \frac{N}{N-J}\Delta M^t - \frac{1}{N-J}\Delta M_N^t. \tag{6}$$

In the FAST algorithm, we assume that multiple malicious clients may participate in FL training. To simplify the calculations, we compute the average of the parameter updates generated by all

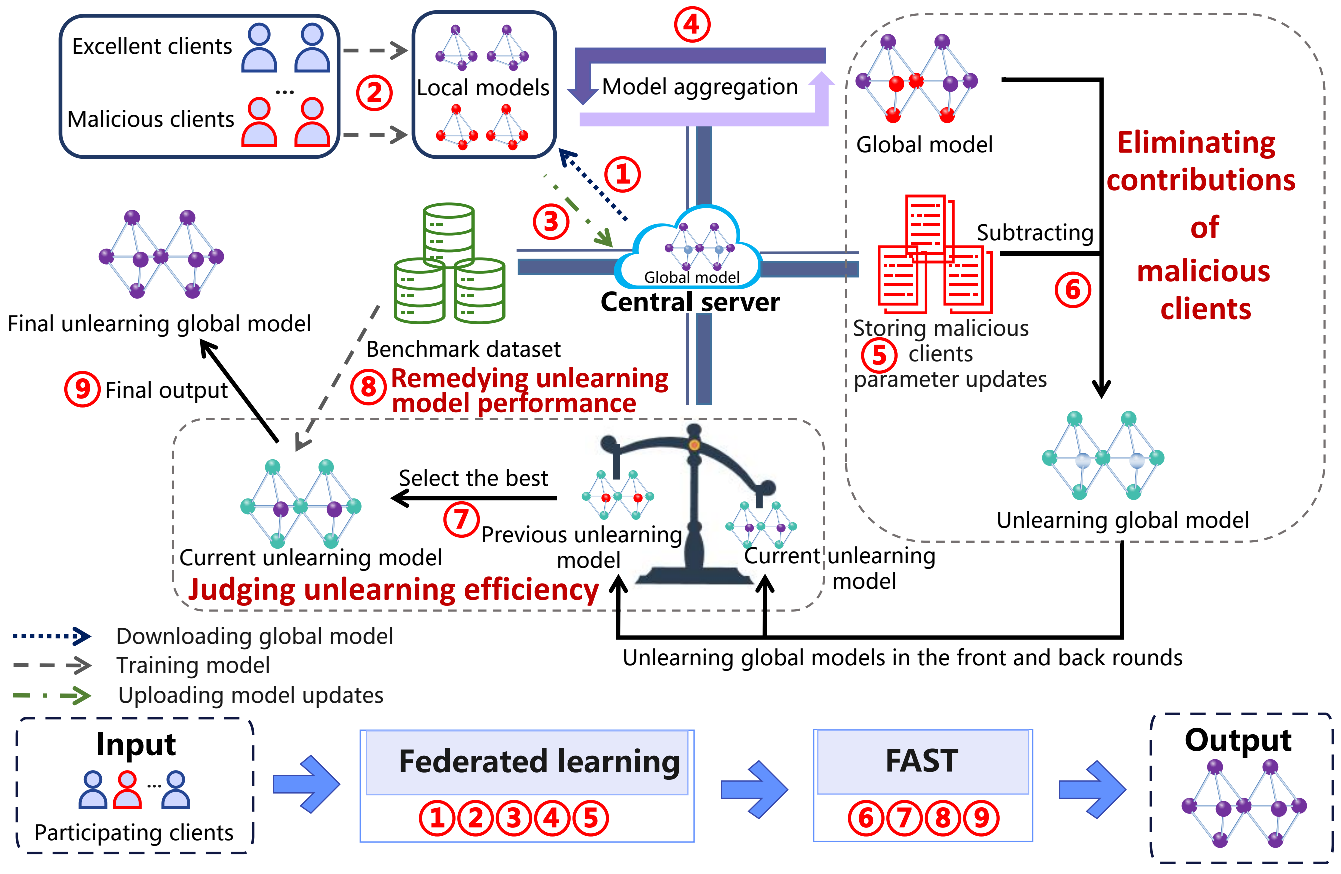


Fig. 2: Architecture of FAST

malicious clients C_F in each training round t, which we define as the parameter updates generated by malicious clients C_F in the t-th round. This can be expressed in the following equation.

$$\Delta M_N^t = \sum_{n=1}^{J} \Delta M_n^t. \quad (7)$$

Due to the incremental nature of FL, we cannot solely rely on the parameters from the current training round to update the model. In this regard, if we remove the historical contributions of malicious clients, a certain bias is inevitable. To account for this bias, we use α_t to present the global model bias at each round t. By combining the aforementioned equation with Equation 2, we derive the final unlearning model. However, the global model update $\Delta M_t'$ may still change a lot by multiplying itself with a factor of $\frac{N}{N-J}$ when malicious clients make minimal contributions M_N^t at round t. This can create additional bias in subsequent rounds, leading to an increased skew α_t of the global model.

$$M_t' = M_0 + \frac{N}{N-J}\sum_{t=1}^{t} \Delta M^t - \frac{1}{N-J}\sum_{t=1}^{t} \Delta M_N^t + \sum_{t=1}^{t} \alpha_t. \quad (8)$$

To mitigate this problem, we assume that the parameter updates of malicious clients are $\Delta M_n^t = 0$, where $n \in \{1, ..., J\}$ and malicious clients continue to participate in FL progress. This allows us to simplify the unlearning model updates, which can be expressed as follows

$$\Delta M_t' = \frac{1}{N}\sum_{n=1}^{N-J} \Delta M_n^t = \Delta M^t - \frac{J}{N}\Delta M_N^t. \quad (9)$$

The final unlearning model M' is obtained by combining Equation 9 with Equation 2.

$$\begin{aligned} M' &= M_0 + \sum_{t=1}^{T} \Delta M_t' + \sum_{t=1}^{T} \alpha_t \\ &= M_0 + \sum_{t=1}^{T} \Delta M^t - \frac{J}{N}\sum_{t=1}^{T} \Delta M_N^t + \sum_{t=1}^{T} \alpha_t \\ &= M - \frac{J}{N}\sum_{t=1}^{T} \Delta M_N^t + \sum_{t=1}^{T} \alpha_t. \end{aligned} \quad (10)$$

The unlearning model calculation becomes significantly simplified and more easily understandable. In contrast to the unlearning method proposed by [8] which can only remove the influence of forgotten nodes in the current round. FAST can eliminate the influence of malicious clients in the previous rounds after the model converges to M. Finally, a decision method can be established to detect over-unlearning.

3.1.2 *Judging Unlearning Efficiency*

In this subsection, we present a mechanism for judging unlearning efficiency by evaluating over-unlearning. The federated unlearning phase is divided into two parts in total: one is to eliminate malicious clients' contributions and the other is to evaluate the unlearning effect. During the process of eliminating malicious clients' contributions, we implement the mechanism for judging unlearning efficiency synchronously on the central server.

The key idea of the judge unlearning efficiency mechanism is that when the server removes each round of malicious clients' historical parameter updates based on the final FL global model, the server checks whether the phenomenon of over-unlearning has occurred after deleting that round of malicious clients' contributions. If this phenomenon occurs, the server stops removing the contributions of the remaining malicious clients who participated in other training rounds in advance. It returns the corresponding unlearning model.

From the aforementioned analysis, we can deduce that the unlearning model can be obtained theoretically by subtracting the contributions of malicious clients per training round from the FL final global model theoretically, as shown in Equation 10. However, this may result in an unexpected outcome by changing the unlearning model parameter updates beyond the expected range during the federated unlearning process. As a result, if the impact of malicious clients is negligible during the first few rounds of removing malicious clients, subsequent rounds of removing malicious clients will be meaningless. This over-unlearning phenomenon can result in a substantial decrease in the unlearning model's accuracy. Hence, after eliminating each round of malicious client contributions, we additionally set up an over-unlearning-based evaluate unlearning efficiency mechanism to determine whether the malicious clients' contributions have been maximally eliminated and terminate the federated unlearning process in advance, preventing over-unlearning from occurring.

Our focus lies on designing a mechanism for judging unlearning efficiency. Specifically, in each iteration where the server removes the malicious clients' contributions at t round, we model a judgment method on the server that compares the accuracy of the current unlearning model $M^{'}_{cur}$ with the previous unlearning model $M^{'}_{pre}$ and then performs the corresponding decision of the judgment result. First, we consider the unlearning model $M^{'}_{k}$ of the k round as the current unlearning model and the unlearning model $M^{'}_{k-1}$ of the $k-1$ round as the previous unlearning model, where $k \in \{0, ..., K\}$, K is the max number of training rounds in the unlearning phase. To avoid over-unlearning, we record in advance the unlearning model $M^{'}_{k}$ corresponding to each round in the federal unlearning phase. After eliminating the contributions of malicious clients in the k round, we iterate the current unlearning model so that the current model $M^{'}_{k}$ becomes the previous unlearning model $M^{'}_{pre}$ in the $k+1$ round, and then the unlearning model $M^{'}_{k+1}$ with the contributions of malicious clients removed in the $k+1$ round is applied as the current unlearning model $M^{'}_{cur}$ in the $k+1$ round (Algorithm 1, line 5-6).

However, when malicious clients' contributions are largely eliminated, continuing to eliminate the effect can lead to results that run counter to the expected results. Therefore, we record the current unlearning model $M^{'}_{cur}$ and the previous unlearning model $M^{'}_{pre}$. We set up a test function $Test$ to express the accuracy of the model ACC by calculating the ratio of successful predictions to the total number of data involved in the test, as shown in Equation11. If the accuracy of the current unlearning model consistently surpasses the accuracy of the previous unlearning model, as shown in Equation 12, it means that the historical influence of the malicious client is not completely eliminated, and the next round of deleting contributions step should be continued. If the accuracy of the current unlearning model consistently exceeds that of the previous unlearning model, the unlearning operation will continue to be executed until the K round. This is where K is equal to T, representing the maximum number of times malicious clients' contributions can be eliminated. However, if the accuracy of the current model becomes less than the accuracy of the preceding model, this suggests that over-unlearning has occurred and the federated unlearning process should be stopped. In this case, the current model is reset to the previous unlearning model, and the current unlearning model is returned.

$$\begin{cases} Test(Model) \rightarrow ACC(Model) \\ Test(Model) = \frac{len(pre(D_t)==lab(D_t))}{len(lab(D_t))} \end{cases}, \tag{11}$$

$$ACC(M^{'}_{pre}) < ACC(M^{'}_{cru}). \tag{12}$$

3.2 Remedying Model Performance

There is currently no method to compute the skew during FL training without retraining the original dataset again. However, one of the challenges in FL arises from the lack of access to the local data of participating clients. Therefore, finding a solution to address this skew without relying on the client's local data becomes crucially important.

To mitigate the impact of malicious clients, we have successfully minimized a significant portion of their influence in the initial two components of our approach. However, there is still a residual skew α, as shown in Equation 10. Therefore, an additional method is required to be set up to restore the overall performance of the unlearning model. Previous research has suggested using knowledge distillation for this purpose, as mentioned in [11], but we find this approach unsuitable for our scenario. To address this issue, we propose utilizing the benchmark dataset D_b to train the unlearning model and optimize its performance.

3.2.1 Benchmark Dataset

We maintain an additional small benchmark dataset D_b on the central server for supplementary training of the unlearning model. This is done to enhance the accuracy of the unlearning model. In comparison to the total data contributed by all participating clients, the benchmark dataset stored on the server is relatively small. Moreover, the benchmark dataset exhibits a balanced data distribution and accurate labels, thereby serving as a benchmark dataset [22]. Each entry in the benchmark dataset consists of a tuple (x, y), where x represents the content of training data, and y represents the correct category to which x belongs.

The experimental datasets have already been divided into training sets and test sets. We partition the benchmark dataset from the original test dataset, constituting half of the original test set, with the remaining half serving as the test set for our experiments. In FOCUS [23], the benchmark dataset is employed in federated learning to quantify the credibility of user-local data. FAST utilizes the benchmark dataset to enhance the performance of the unlearning model, facilitating the creation of a higher-quality unlearning model and mitigating the skew introduced during the federated unlearning stage.

3.2.2 Extra Training

After completing the two steps of the unlearning process, eliminating malicious clients' contributions and judging the unlearning efficiency, the central server generates the unlearning model $M^{'}_{UL}$

after federated unlearning. To remedy the skew introduced during federated unlearning, we employ a straightforward yet effective method by performing extra training on $M_{UL}^{'}$ using the benchmark dataset.

The improvement in the performance of the recovered unlearning model depends on the number of extra training rounds conducted. Importantly, the extra training time required for these rounds is considerably smaller than the time allocated for federated unlearning. This is because the overall dataset D is eight times larger than the benchmark dataset. The extra training iterations compensate for the relatively small benchmark dataset. Through this optimization, we can efficiently restore the performance of the unlearning model while minimizing the computational resources required.

4 Experimental Evaluation

In this section, we present the results of extensive experiments conducted to assess FAST performance using various model architectures on four different datasets. Specifically, we first elaborate on the experimental settings, learning models, and the four datasets employed. We provide empirical results demonstrating the performance and efficiency of the models. In our experiment, all participating clients and the central server are simulated by processes on a physical device equipped with Intel(R)Core(TM) i9-12900H CPU and NVIDIA GeForce RTX 3060 GPU.

4.1 Experimental Setting

1) Datasets and Models. Our experiments encompass four diverse datasets, namely MNIST, FMNIST, CIFAR-10, and SVHN. Each dataset presents unique characteristics and classification tasks, requiring different model architectures for optimal performance.

To accommodate the varying sizes and features of each dataset, we employ four distinct global models with different structures for the respective classification tasks. Specifically, we utilize MLP [24] on the MNIST dataset. As for FMNIST, the FMNIST model comprised 2 convolutional layers, 2 max pool layers, and 1 fully connected layer at the end to generate prediction output. To evaluate the CIFAR-10, we apply the famous VGG11 network [25]. Lastly, on the SVHN dataset, we use another famous MobileNet [26] model.

MNIST. This is a dataset for counting handwritten digits [27]. The training and validation sets contain a total of 70,000 images. The handwritten numbers were written by half of the high school students and half of the Census Bureau staff. MNIST has 10 categories for 0 to 9. Each is a binary image, consisting of black and white, and is 28×28 in size.

FMNIST. FMNIST dataset is obtained from Zalando's fashion article images [28] and consists of a training set with 60,000 examples and a test set with 10,000 examples. Each example in the dataset has a size of 28×28 pixels and belongs to one of the 10 classes, such as "ankle boot", "bag", "coat", etc.

CIFAR-10. CIFAR-10 is applied to evaluate image recognition models [29]. It contains 50000 32×32 colored images of 10 classes such as "airplane", "dog", "cat", etc. Each pixel point includes three RGB values, ranging from 0-255.

SVHN. The Street View House Numbers (SVHN) dataset [30] is an image digit recognition dataset comprising over 600,000 digit images extracted from real-world data. These images are obtained from house numbers captured in Google Street View images and are cropped to 32×32 pixels.

2) Comparison. We compare our proposed FAST algorithm with other relevant approaches as follows:

- ***RF (Retrain from Scratch):*** In this method, only excellent clients are retained, malicious clients are removed, and the FL model is retrained from scratch. It follows a traditional federated unlearning approach.
- ***FL (General Federated Learning):*** This represents the conventional Federated Learning algorithm that incorporates malicious clients. It does not focus specifically on removing or handling malicious contributions.
- ***KD (Knowledge Distillation):*** A revised approach, mentioned in [11]. This approach applies knowledge distillation to enhance model performance following the removal of historical impact from forgotten clients.
- ***OU (Only Unlearning):*** Our first step in federated unlearning is to eliminate contributions of malicious clients without any other remedies. It focuses solely on removing malicious contributions.
- ***OB (Only Boosting):*** This compared algorithm continues training the final FL model utilizing the benchmark dataset without actively addressing the malicious extent of the model, even when malicious clients are present.
- ***RR (Rapid Retraining):*** The approach [31] aims to completely remove data samples from a trained FL model by employing rapid retraining. It focuses on data removal rather than malicious contributions.
- ***FAST (Proposed algorithm):*** Our proposed algorithm FAST, in each unlearning round, eliminates contributions of malicious clients while evaluating the efficacy of the unlearning process and corrects unlearning model performance.

3) Metrics. The objective is to restore the accuracy of the model to its regular metrics while simultaneously minimizing the total running time. Under different experimental variables settings, the efficiency of the proposed FAST algorithm and the comparative algorithms is quantified by measuring the model's accuracy. In addition, we introduce five metrics to comprehensively assess the algorithm's performance: the unlearning rate, F1-Score, total running time, CPU usage, and memory consumption.

Accuracy: Accuracy of the FL global model and unlearning global model is applied as an evaluation metric, calculated as follows:

$$ACC = 1 - \frac{\sum_{i=1}^{D}[y_i \neq \overline{y}_i]}{D}, \tag{13}$$

where $\overline{y}_i$ denotes the predicted label of sample i.

F1-Score: The F1-Score is a measure that balances precision and recall. It is applied to evaluate overall model performance, particularly in scenarios with imbalanced class distributions. The F1-Score is calculated as follows:

$$F1 - Score = \frac{2 \cdot (Precision \cdot Recall)}{Precision + Recall}, \tag{14}$$

where *Precision* is the ratio of true positive predictions to all positive predictions, and *Recall* is the ratio of true positive predictions to all actual positive samples.

4.2 Experimental Results

In this section, our objective is to evaluate the performance of various algorithms in mitigating the contributions of malicious

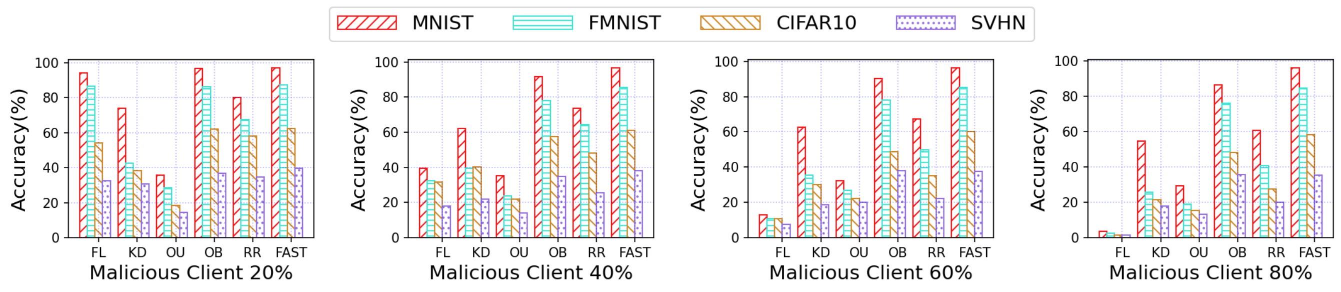


Fig. 3: Accuracy comparison of various methods on four datasets and different proportions of malicious clients.

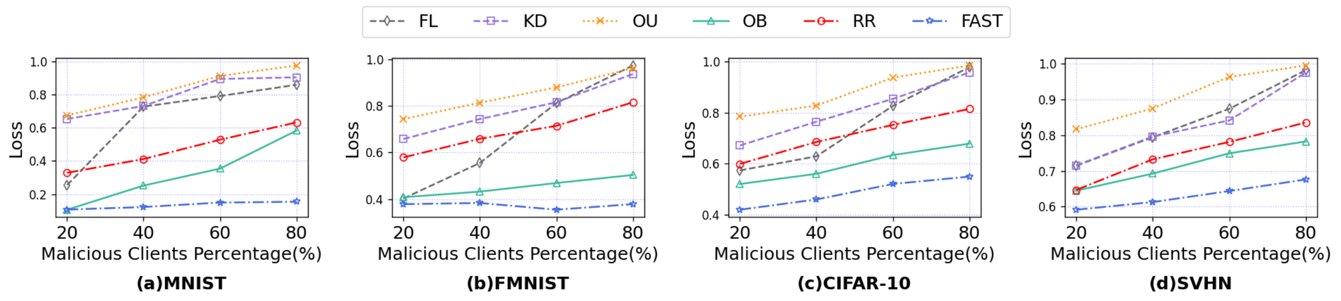


Fig. 4: Loss versus different malicious clients percentage on four datasets.

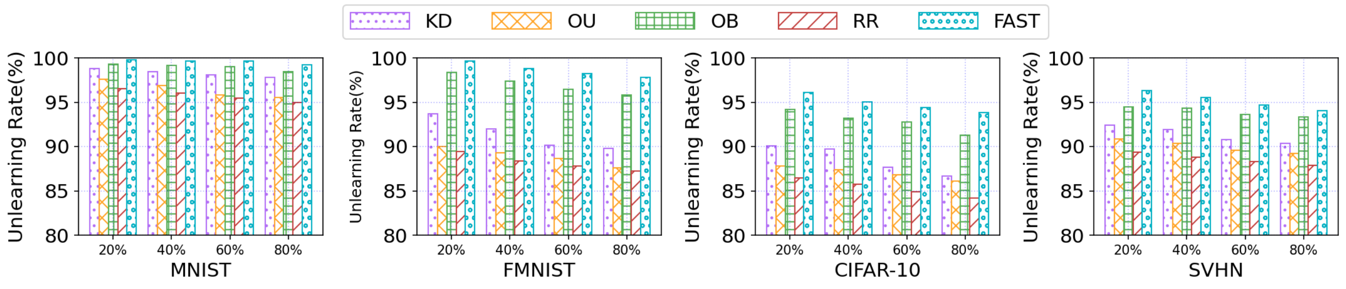


Fig. 5: The unlearning rate of prediction tasks on local data of malicious clients.

clients and analyze the key experimental results of eliminating the impact of malicious clients. For each framework, we record the accuracy, the loss value, the unlearning rate, and the elapsed time.

4.2.1 Impact of Different Malicious Levels

In our experiments, our main focus is to investigate the impact of the level of malicious participation by clients on FAST and other models across different datasets. For simplicity of calculation and presentation, we use percentages to indicate model accuracy and the loss value to indicate the degree of loss.

We first investigate the performance of the federated unlearning model under multiple levels of maliciousness among participating clients. We assume that there are 10 participating clients in each iteration, each with the same data size. Fig. 3 displays the accuracy of each method under varying proportions of malicious clients, while Fig. 4 illustrates the loss value of each method under different proportions of malicious clients. For all training tasks, we consider different proportions of malicious client shares, 20%, 40%, 60%, and 80%. Each algorithm is tested on four datasets according to the number of malicious clients shared in each experiment. Among the 10 participating clients, e.g., 20% are malicious clients, we randomly select 2 participating clients to corrupt their local data. For example, we may mislabel the number “6” as “9” in MNIST and SVHN, or label “pullover” as “sandal” in FMNIST, and “bird” labeled as “deer” in CIFAR-10. Additionally, we add the non-IID data samples to the rest of the excellent clients, and the numbers of some categories do not match the numbers of other categories, even almost zero. This intentional variation enhances the significance of the experimental results.

Obviously, the results are in Fig. 3 and Fig. 4 clearly indicate that the proposed FAST algorithm consistently outperforms the comparison algorithms across all experimental configurations. Regardless of four different malicious levels, FAST achieves higher utility, proving the effectiveness of our proposed algorithm. As we observed from the figures, the training model performance of these methods utilizing different datasets is more fixed in terms of advantages and disadvantages. However, the accuracy of each model has a trend of slightly decreasing as the malicious level increases. Moreover, it can be seen that with increasing the malicious

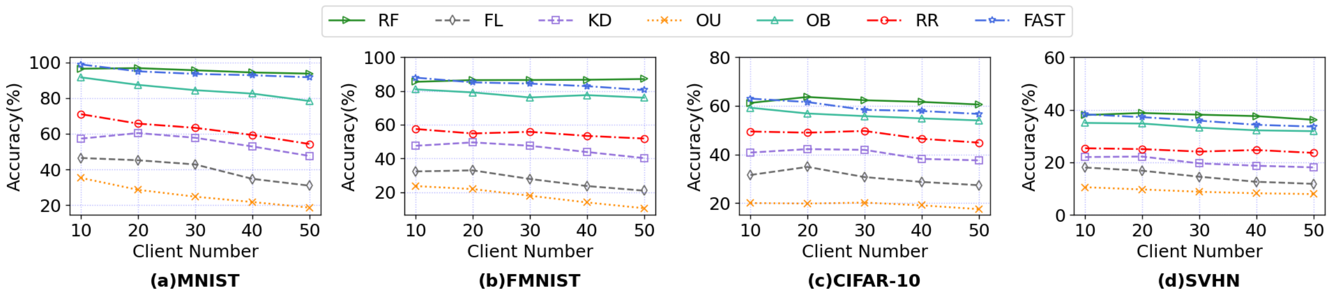


Fig. 6: Performance comparison under different client numbers with 40% malicious level.

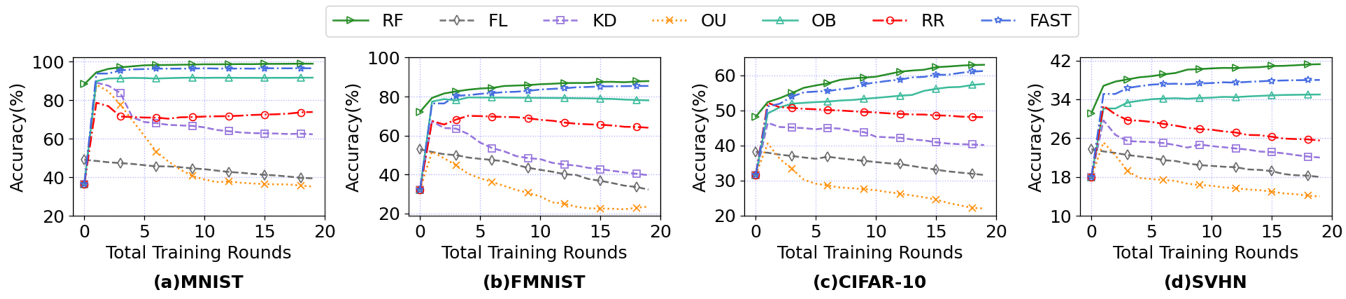


Fig. 7: Performance comparison with different methods and datasets.

level of participating clients, the FAST algorithm exhibits strong performance. This surpasses the comparison methods in terms of accuracy and loss value. For instance, when evaluating the MNIST task including 20% malicious clients, after 20 total training rounds (iterations), the FL, RR, KD, and OU techniques achieve accuracy scores of 94.34%, 80.12%,74.05%, and 35.74% respectively. In contrast, OB achieves a score of 96.82% while FAST further improves accuracy to 96.98%. Although there is a slight decrease in the accuracy of FAST as the malicious level increases to 40%, 60%, and 80%, it manages to maintain an accuracy of around 96%. We apply the cross-entropy loss function in our experiments, and as can be seen in Fig. 4, the loss values of all approaches rise along with the increase in the number of malicious clients. Among them, the loss of OU is the largest, and the loss value of FAST is always the smallest. Their loss values achieve 0.9756 and 0.1536 respectively at the malicious level of 80% in the MNIST task. It is worth noting that, since normal federated learning training has no measure for malicious clients, the loss of FL grows dramatically when the malicious level increases. This crosses over with the loss of other approaches.

4.2.2 *Performance of Unlearning Efficiency*

In this section, we aim to investigate whether FAST successfully eliminates the impact caused by malicious clients. To assess this, we evaluate the unlearning rate of the final unlearning model applying local data from malicious clients at various malicious levels. Although the final unlearning model generated by FAST can obtain a satisfactory score, it is crucial to verify whether the model removes parameters associated with malicious clients or if it merely performs well on the correct test dataset. Therefore, we introduce the concept of the unlearning rate. This is calculated as 100% minus the accuracy of the unlearning model utilizing the local dataset of malicious clients at different malicious levels. This metric helps us determine the extent to which the model has successfully eliminated the impact of malicious clients. A higher unlearning rate indicates a more thorough removal of the influence of malicious clients.

We explore different malicious levels, datasets, and frameworks to showcase the unlearning rate in different malicious environments. From Fig. 5 we can observe that as the malicious level decreases, the unlearning rate tends to increase. With the 20% malicious level, the remaining 80% of excellent clients can still train the model effectively. Therefore, it is worthwhile to consider other results. When the malicious level rises, unlearning rates generally decrease, but FAST significantly outperforms the other benchmarks. For instance, when evaluating the unlearning rate against FMNIST at an 80% malicious level. In this scenario, the KD, OU, and RR mechanisms achieve unlearning rates of 89.78%, 87.62%, and 87.23% respectively. In contrast, OB achieves an unlearning rate of 95.84%. In contrast, FAST demonstrates superior performance by improving the unlearning rate to 97.78%.

4.2.3 *Impact of Multiple Client Numbers*

After ensuring overall performance, we investigate the influence of the number of active clients. We compare FAST and other methods with different numbers of participating clients. Specifically, when considering the impact of the number of clients, we consider five distinct scenarios characterized by different numbers of active clients, namely $N = 10, 20, 30, 40, 50$. While maintaining the malicious level of the participating clients at 40%, we only vary the number of participating clients. Likewise, the previous experiment setting is adopted and we carefully control the proportion of non-IID local data of participating clients based on the malicious level and the number of clients. By exploring these scenarios, we aim to gain a deeper understanding of how the number of

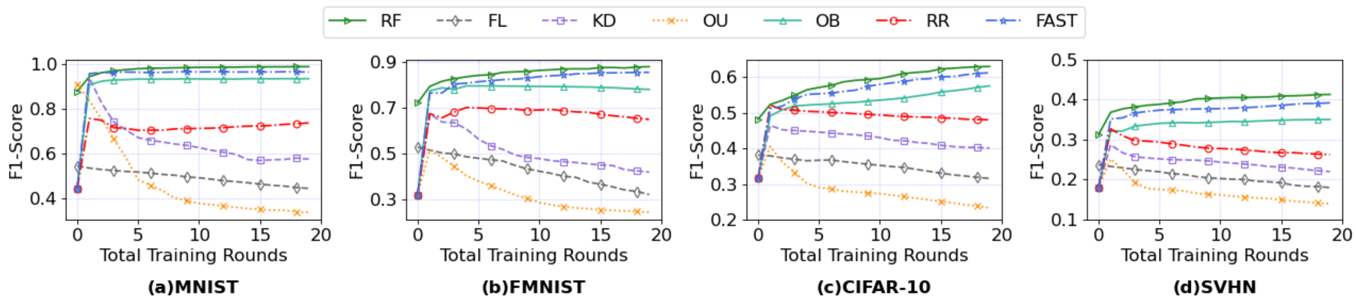


Fig. 8: Comparison of F1-Scores under 20 training rounds with 40% malicious level.

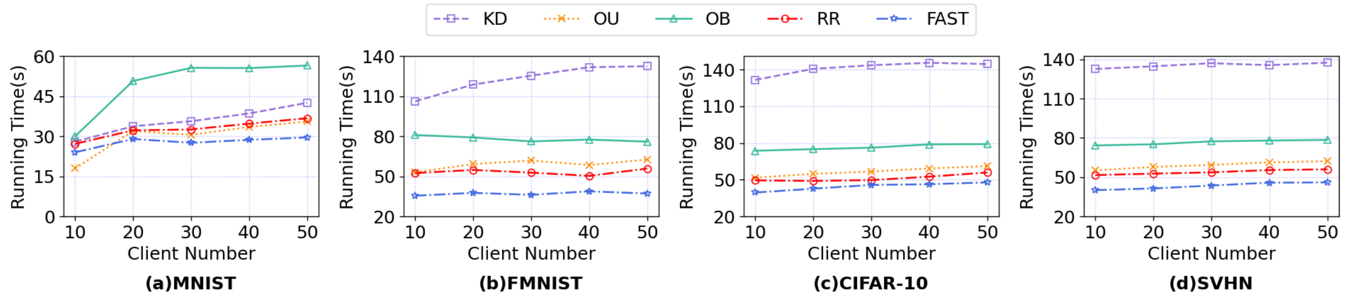


Fig. 9: Comparison of the running time under different client numbers with 40% malicious level.

participating clients impacts the overall performance of FAST and its comparison to other methods.

In Fig. 6, we present the accuracy of the final unlearning model over 20 iterations under different numbers of participating clients and a fixed 40% malicious level. Based on the experimental results, we observe a consistent trend where the model accuracy of each algorithm decreases as the number of active clients increases. For instance, on the learning task based on the MNIST dataset, the final unlearning model generated by the FAST algorithm achieves an accuracy of 93.63% when 30 participating clients are involved in FL training. However, as the number of participating clients increases to 50, the model accuracy drops to 91.74%. The reason can be attributed to the large number of clients participating in training at a fixed malicious level. This consequently increases the likelihood of encountering malicious clients. Thus, while the FAST algorithm effectively eliminates the contributions of malicious clients, the overall accuracy of the final unlearning model still experiences a decline. By Comparing FAST with other methods under the varying number of participating clients, we affirm that FAST still has exceptional performance when the number of participating clients increases.

4.2.4 *ACC Value and F1-Score*

To visualize the performance of the proposed FAST algorithm, we track the accuracy changes of each training round for other algorithms at a malicious level of 40% during the FL training process. These results are shown in Fig. 7. Individual model performance remains relatively consistent across diverse datasets. Retraining the models from scratch yields the highest accuracy for any dataset because it excludes malicious clients from participating clients. It trains the models with more high-quality data leading to the aggregation of near-ideal models at the central server. In terms of model accuracy, retraining outperforms all other optimization methods. However, retraining has the drawback of requiring the complete FL training process to be repeated, resulting in nearly double the time cost. Additionally, the clients involved in retraining have been carefully selected and are brand-new, therefore retraining from scratch has little to no correlation with the previous federated learning. In contrast, FAST works robustly in both scenarios with a large number of mislabeled data samples (60%, 80%) or non-IID data distributions. We observe a rapid recovery in the accuracy of the FAST model within about three epochs of unlearning training, gradually approaching that of the RF model.

Since most of the methods are federated unlearning algorithms, model accuracy improves dramatically in the first few training rounds, except for RF and FL. Fig. 7 illustrates that the accuracy of the MNIST study task can be easily raised because of the high recognizability of handwritten digits. Because of the complexity of the CIFAR-10 and SVHN environments, even these unlearning methods do not achieve the desired results. Since RF and FL have excellent and malicious clients from the early stages of training respectively, their models do not change much in terms of accuracy.

As shown in Fig.8, we have plotted the change in F1-Score over 20 training rounds with 10 clients participating in FL, with 40% malicious level. F1-Score combines precision and recall, providing a more comprehensive evaluation of model performance. It is evident that FAST can outperform the other benchmarks except RF significantly in each dataset. For F1-Score, RF and FAST show better performances, as RF itself does not contain malicious clients, so FAST's F1-Score does not surpass that of RF. On the MNIST dataset, FAST reaches its peak at 0.9664 in the 18th round of training, and although the F1-Score decreases slightly in the subsequent two rounds, the difference is very small, approximately 0.005, indicating normal behavior. While training

TABLE 2: CPU utilization per rounds of training(%).

Algorithm	Training rounds																			
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
FL	0.2	0.3	0.5	0.7	0.9	**2.5**	**4.6**	**1.3**	0.2	0.7	**1.0**	**4.7**	0.7	0.5	1.3	**5.2**	1.8	1.6	0.8	1.4
KD	1.1	**1.7**	**1.8**	**1.8**	0.4	0.5	0.4	0.6	0.4	0.6	**1.0**	0.6	0.6	0.6	**2.3**	0.6	**3.4**	**1.9**	0.5	**1.8**
OU	**4.0**	0.6	1.5	0.9	**3.2**	0.5	1.2	**1.3**	**1.6**	**1.9**	0.6	0.6	0.5	0.8	0.3	0.4	2.4	1.4	**1.0**	1.1
OB	2.7	0.7	1.4	1.2	0.4	1.8	0.6	0.1	0.8	0.2	0.7	0.8	**3.5**	1.1	0.9	0.6	0.5	0.3	0.2	0.2
RR	1.0	0.7	0.3	1.2	1.0	0.8	1.7	1.2	0.6	0.9	**1.0**	1.1	0.8	**1.2**	1.3	0.9	1.4	0.7	0.4	0.6
FAST	0.8	0.3	0.7	0.8	0.5	1.6	0.5	0.7	0.5	0.5	0.5	0.4	0.8	0.8	0.4	0.8	1.1	0.6	0.5	0.9

TABLE 3: Memory consumption per rounds of training (MB).

Algorithm	Training rounds									
	1	2	3	4	5	6	7	8	9	10
FL	324.98	333.64	347.31	351.05	**371.60**	**373.61**	**378.22**	**382.92**	**419.50**	**424.1**1
KD	**365.52**	**365.71**	**366.30**	**367.68**	367.82	367.55	367.12	368.27	368.49	368.61
OU	365.23	365.49	364.63	366.06	367.24	367.40	367.05	367.53	368.35	366.01
OB	320.93	323.25	323.52	323.87	324.18	324.54	324.84	327.16	327.48	327.74
RR	331.78	332.01	333.54	333.97	335.04	335.46	335.91	336.40	336.78	338.30
FAST	346.79	323.66	324.02	326.32	326.64	326.94	327.27	327.57	327.83	328.22
	11	12	13	14	15	16	17	18	19	20
FL	**428.76**	**433.36**	**438.01**	**442.59**	**447.27**	**451.93**	**520.52**	**525.18**	**528.88**	**533.56**
KD	405.80	406.13	408.71	409.13	409.34	409.84	409.91	410.33	410.55	411.02
OU	366.23	366.24	366.59	366.70	366.68	369.00	369.02	369.19	369.44	369.43
OB	328.05	328.37	328.68	328.99	329.37	333.68	333.99	334.38	334.73	335.05
RR	338.67	339.13	341.64	342.16	342.71	343.45	343.83	345.51	345.82	346.12
FAST	328.54	332.80	333.12	333.47	333.79	334.09	334.38	334.72	335.03	335.34

on other datasets, all methods exhibit a declining trend in F1-Score. However, FAST maintains its leading position. On MNIST and FMNIST, during the initial stages, the F1-Score drops quickly due to the significant impact of malicious clients, but it gradually stabilizes with continued training. In contrast, CIFAR-10 and SVHN exhibit lower initial F1 scores, both due to the inherent lower prediction accuracy of the datasets and the presence of malicious clients, resulting in lower fluctuations during subsequent training.

4.2.5 *Computational Cost and Memory Cost*

Because FAST involves storing parameter updates from malicious clients during the FL phase and additional training during the model performance recovery phase, we conduct experiments to access computational and memory costs. We record the CPU utilization, memory consumption, and training time for FAST and compare these metrics with other algorithms.

In Table 2, we select the results for training on the MNIST dataset and display the CPU utilization for each training round. We highlight the maximum and minimum CPU utilization for each training round applying bold and underlined formatting. It can be observed that OB exhibits relatively lower CPU utilization, possibly due to its simpler algorithm that does not require extensive CPU resources. In contrast, KD, which utilizes knowledge distillation to recover model performance, shows an increasing trend in CPU demand in the later rounds of training. Since FL and RF have similar performance, we only display the results for FL. It is worth noting that FAST does not incur significant additional computational costs despite the extra training and exhibits the lowest CPU utilization within the first three rounds of training. This suggests that while FAST introduces some computational cost, it does not significantly increase CPU requirements.

Additionally, due to the requirement of FAST to store parameter updates from malicious clients, it incurs additional storage costs. Table3 presents the memory consumption for FAST and the compared algorithms for each training round. FL requires significant memory since multiple rounds of training with large amounts of data, while other benchmarks build upon FL, resulting in lower memory consumption compared to FL. Specifically, OB exhibits the lowest memory consumption, even reducing memory usage by approximately 45MB in the first training round compared to KD. Although the memory consumption of FAST is not the lowest, it remains relatively low. It denotes that the minimal memory required for storing the additional parameter updates can lead to excellent model performance.

In the end, we analyze a study of the time cost of the different methods involved. Given that minimizing total training time is another objective, we record the required total running time for our proposed FAST algorithm and other algorithms at the specified malicious level. Fig.9 presents the running time needed for five algorithms on the MNIST dataset at a 40% malicious level, considering different numbers of participating clients. The results show that while the difference in execution time of the individual algorithms is not very prominent when using the MNIST dataset, the time advantage of FAST becomes more prominent as training with the other three datasets. For instance, when there are 50 participating clients, the running times of FAST for MNIST, FMNIST, CIFAR-10, and SVHN datasets are 29.57s, 37.12s, 47.79s, and 45.98s, respectively. Besides, compared with the OB method, FAST times are reduced by 26.96s, 38.89s, 31.32s, and 32.5s, respectively.

Furthermore, we should also compare running times between FL and FAST. The reason why FL and RF running times are not available in Fig.9 is that these models require approximately 16 times more time to train than FAST. The difference in this data appears striking, as the retraining from scratch approach would incur a comparable running time as FL. In contrast, FAST not only capitalizes on the final global model generated by FL but also completes the unlearning task in significantly less running time. From the results, it becomes evident that our proposed FAST algorithm outperforms the comparison algorithms, highlighting its robustness and superior performance. Despite the slight increases in computational and memory costs for FAST, the magnitude of this increase is not substantial. However, in comparison to this minor cost increase, our algorithm demonstrates a significant improvement in accuracy. It is worth mentioning that our algorithm achieves outstanding model performance while maintaining reasonable computational and memory resource usage.

5 Related Work

Most of the existing studies related to federated attention have been dedicated to eliminating the contributions of deleted clients and the performance optimization of federated unlearning. In our paper, we focus on various federated unlearning methods and scenarios where malicious clients are involved in training. Therefore, we review the literature on these two areas in this section.

5.1 Federated Unlearning Methods

The concept of unlearning first emerged when the advent of machine unlearning [32]. In their groundbreaking research, the Sliced, Isolated, Sliced, and Aggregated (SISA) procedure was proposed. SISA leverages the idea of training models on sliced and isolated datasets to address machine unlearning challenges. It divides the raw dataset into several sliced datasets. It slices and splits each sliced dataset and starts training, and finally aggregates the models of each sliced dataset. When unlearning forgotten data in round t is needed, training is started from the previous round state. This reduces the amount of data needed for forgetting. Compared with retraining, SISA reduces the computational cost of unlearning but increases the storage cost.

FedEraser is indeed one of the pioneering studies on federated unlearning. It addresses the problem of eliminating the influence of a specific client on the global model in a federated learning setting. FedEraser eliminates the influence of a particular client on the global model during training by appending calibration training after standard FL. The server keeps clients updated regularly in standard FL. In calibration training, calibrated clients are retrained, FedEraser determines the updated values based on the collected updates, and finally, the server aggregates the updated values to obtain a calibrated global model. FedEraser has the following drawbacks: first, only the influence of the current round can be forgotten; second, other clients still retain the contributions of the target client; third, it occupies storage space because the model parameters of each client need to be saved.

Liu et al. utilized a virtual gradient generator to clear the memory at the kth iteration to eliminate the data impact of the kth-1st round. However, they were only able to modify the data memory of the previous round [8]. In [33], a gradient ascent method is mainly used for data elimination from the forgotten client. The gradient mean of other clients is used to constrain the ascent range. Gong et al. designed an unlearning method based on the Bayesian theorem [9]. Wang et al. pruned from the class-discriminative to achieve unlearning [10].

5.2 Malicious Client Scenarios

Zhang et al. considered the presence of malicious clients at FL [34], if the client uploads a model update that is inconsistent with the predicted model, it is marked as a malicious client, and applies FLDetector to detect and remove malicious clients, but can only resist a few malicious clients [34]. Malicious users are also considered in other areas of research, such as attacking cryptosystems [35], system behavior [36], digital signatures [37]. In the current age of data security, there will be more and more malicious clients and malicious attacks attacking the model results, so we should pay more attention to this.

In general, existing studies have paid scant attention to scenarios where malicious clients are involved in federated learning. The federated unlearning phase is performed on a central server. To address this issue, we propose FAST, an approach that aims to eliminate the contributions of malicious clients. It also sets up a judging unlearning efficiency mechanism to complete federated unlearning quickly and efficiently.

6 Conclusion

In this paper, we present FAST, a federated unlearning algorithm aimed at eliminating malicious clients' historical contributions on the central server. Specifically, it removes malicious clients' influence by subtracting the historical parameter updates made by each malicious client from the central server's aggregated model. Additionally, FAST employs additional training of the unlearning model by utilizing a benchmark dataset to recover performance bias. Particularly, to enhance FAST performance, we have developed and implemented three key technical components in FAST: 1) eliminating contributions of malicious clients, 2) judging unlearning efficiency, and 3) remedying unlearning model performance. We have conducted numerous empirical studies on four canonical datasets that have demonstrated the efficacy of our proposed FAST algorithm in effectively removing the influence of malicious clients. One of the advantages of our method is that it operates solely on the central server side, where the server records the parameter updates of malicious clients. It is not necessary to account for the randomness of participating clients or the voluntariness of malicious clients in federated unlearning. Notably, our method significantly enhances FAST model accuracy and running time compared to alternative approaches. In the future, we plan to work towards multiple interesting topics, such as unlearning solutions that address the presence of multiple malicious clients participating in vertical federated learning.

References

[1] R. Kitchin, "Getting smarter about smart cities: Improving data privacy and data security," 2016.

[2] J. Konečnỳ, H. B. McMahan, D. Ramage, and P. Richtárik, "Federated optimization: Distributed machine learning for on-device intelligence," *arXiv preprint arXiv:1610.02527*, 2016.

[3] J. Konečnỳ, H. B. McMahan, F. X. Yu, P. Richtárik, A. T. Suresh, and D. Bacon, "Federated learning: Strategies for improving communication efficiency," *arXiv preprint arXiv:1610.05492*, 2016.

[4] H. B. McMahan, E. Moore, D. Ramage, and B. A. y Arcas, "Federated learning of deep networks using model averaging," *arXiv preprint arXiv:1602.05629*, vol. 2, 2016.

[5] P. Voigt and A. Von dem Bussche, "The eu general data protection regulation (gdpr)," *A Practical Guide, 1st Ed., Cham: Springer International Publishing*, vol. 10, no. 3152676, pp. 10–5555, 2017.

[6] S. L. Pardau, "The california consumer privacy act: Towards a european-style privacy regime in the united states," *J. Tech. L. & Pol'y*, vol. 23, p. 68, 2018.

[7] 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, IEEE, 2021.

[8] Y. Liu, Z. Ma, X. Liu, and J. Ma, "Learn to forget: User-level memorization elimination in federated learning," *arXiv preprint arXiv:2003.10933*, vol. 1, 2020.

[9] J. Gong, J. Kang, O. Simeone, and R. Kassab, "Forget-svgd: Particle-based bayesian federated unlearning," in *2022 IEEE Data Science and Learning Workshop (DSLW)*, pp. 1–6, IEEE, 2022.

[10] 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.

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

[12] S. Shintre, K. A. Roundy, and J. Dhaliwal, "Making machine learning forget," in *Annual Privacy Forum*, pp. 72–83, Springer, 2019.

[13] Y. Cao and J. Yang, "Towards making systems forget with machine unlearning," in *2015 IEEE symposium on security and privacy*, pp. 463–480, IEEE, 2015.

[14] 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.

[15] 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.

[16] A. N. Bhagoji, S. Chakraborty, P. Mittal, and S. Calo, "Analyzing federated learning through an adversarial lens," in *International Conference on Machine Learning*, pp. 634–643, PMLR, 2019.

[17] P. Kairouz, H. B. McMahan, B. Avent, A. Bellet, M. Bennis, A. N. Bhagoji, K. Bonawitz, Z. Charles, G. Cormode, R. Cummings, *et al.*, "Advances and open problems in federated learning," *Foundations and Trends® in Machine Learning*, vol. 14, no. 1–2, pp. 1–210, 2021.

[18] Z. Wu, Q. Ling, T. Chen, and G. B. Giannakis, "Federated variance-reduced stochastic gradient descent with robustness to byzantine attacks," *IEEE Transactions on Signal Processing*, vol. 68, pp. 4583–4596, 2020.

[19] Y. Deng, F. Lyu, J. Ren, Y.-C. Chen, P. Yang, Y. Zhou, and Y. Zhang, "Improving federated learning with quality-aware user incentive and auto-weighted model aggregation," *IEEE Transactions on Parallel and Distributed Systems*, vol. 33, no. 12, pp. 4515–4529, 2022.

[20] 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.

[21] D. M. Hawkins, "The problem of overfitting," *Journal of chemical information and computer sciences*, vol. 44, no. 1, pp. 1–12, 2004.

[22] T. Tuor, S. Wang, B. J. Ko, C. Liu, and K. K. Leung, "Overcoming noisy and irrelevant data in federated learning," in *2020 25th International Conference on Pattern Recognition (ICPR)*, pp. 5020–5027, IEEE, 2021.

[23] Y. Chen, X. Yang, X. Qin, H. Yu, B. Chen, and Z. Shen, "Focus: Dealing with label quality disparity in federated learning," *arXiv preprint arXiv:2001.11359*, 2020.

[24] I. O. Tolstikhin, N. Houlsby, A. Kolesnikov, L. Beyer, X. Zhai, T. Unterthiner, J. Yung, A. Steiner, D. Keysers, J. Uszkoreit, *et al.*, "Mlp-mixer: An all-mlp architecture for vision," *Advances in neural information processing systems*, vol. 34, pp. 24261–24272, 2021.

[25] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[26] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "Mobilenetv2: Inverted residuals and linear bottlenecks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 4510–4520, 2018.

[27] 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.

[28] H. Xiao, K. Rasul, and R. Vollgraf, "Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms," *arXiv preprint arXiv:1708.07747*, 2017.

[29] Y. Abouelnaga, O. S. Ali, H. Rady, and M. Moustafa, "Cifar-10: Knn-based ensemble of classifiers," in *2016 International Conference on Computational Science and Computational Intelligence (CSCI)*, pp. 1192–1195, IEEE, 2016.

[30] Y. Netzer, T. Wang, A. Coates, A. Bissacco, B. Wu, and A. Y. Ng, "Reading digits in natural images with unsupervised feature learning," 2011.

[31] 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," *arXiv preprint arXiv:2203.07320*, 2022.

[32] L. Bourtoule, V. Chandrasekaran, C. A. Choquette-Choo, H. Jia, A. Travers, B. Zhang, D. Lie, and N. Papernot, "Machine unlearning," *arXiv preprint arXiv:1912.03817*, 2019.

[33] A. Halimi, S. Kadhe, A. Rawat, and N. Baracaldo, "Federated unlearning: How to efficiently erase a client in fl?," *arXiv preprint arXiv:2207.05521*, 2022.

[34] 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.

[35] A. T. Mahdad, M. Jubur, and N. Saxena, "Analyzing the security of otp 2fa in the face of malicious terminals," in *Information and Communications Security: 23rd International Conference, ICICS 2021, Chongqing, China, November 19-21, 2021, Proceedings, Part I 23*, pp. 97–115, Springer, 2021.

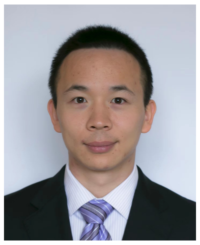
[36] R. Luh, G. Schramm, M. Wagner, and S. Schrittwieser, "Sequitur-based inference and analysis framework for malicious system behavior.," in *ICISSP*, pp. 632–643, 2017.

[37] I. Z. Berta, L. Buttyán, and I. Vajda, "A framework for the revocation of unintended digital signatures initiated by malicious terminals," *IEEE Transactions on Dependable and Secure Computing*, vol. 2, no. 3, pp. 268–272, 2005.

Xintong Guo received the B.S from Changchun University, China, in 2022. She is currently a master student in the School of Software Engineering, Dalian University(DLU), China. Her current research interest is federated learning.

Pengfei Wang (Member, IEEE) received the B.S., M.S. and Ph.D. degrees in software engineering from Northeastern University (NEU), China, in 2013, 2015 and 2020, respectively. From 2016 to 2018, He was a visiting Ph.D. with the Department of Electrical Engineering and Computer Science, Northwestern University, IL, USA.

He is currently an associate professor with the School of Computer Science and Technology, Dalian University of Technology (DUT), China. He has authored nearly 50 papers on high-quality journals and conferences, such as IEEE JSAC, IEEE/ACM TON, IEEE INFOCOM, IEEE TITS, DAC, IEEE ICNP, IEEE ICDCS, IEEE IoT-J, and JSA etc. He also holds a series of patents in US and China. His research interests are distributed artificial intelligence, big data, and AIoT.



Sen Qiu (Member, IEEE) received the B.Sc. and Ph.D. degrees in control theory and control engineering from Dalian University of Technology, Dalian, China, in 2009 and 2016, respectively. He was a Visiting Researcher with the Department of Computer Science and Electronic Engineering, University of Essex, Colchester, U.K. from 2013 to 2014.

He is currently an Associate Professor with the Dalian University of Tech- nology. His current research interests include Internet of Things, Body sensor networks, and Pattern recognition.

Wei Song (Student Member, IEEE) received the B.S. from Anhui University, China, in 2022. She is currently a master student in the School of Computer Science and Technology, Dalian University of Technology(DUT), China. Her current research interest is federated learning.

Qiang Zhang (Member, IEEE) received the B.S. degree in electronic engineering and M.S. and Ph.D. degrees in circuits and systems from the School of Electronic Engineering, Xidian University, Xi'an, China, in 1994, 1999, and 2002, respectively.

He is currently the dean and professor in the College of Computer Science and Technology, Dalian University of Technology, Dalian, China. His research interests are artificial intelligence, neural networks, DNA computing, and big data.

Xiaopeng Wei (Member, IEEE)received the B.S. and M.S. degrees in mechanical manufacturing and automation and the Ph.D. degree in computational mechanics from Dalian University of Technology, China, in June 1982, 1986, and 1993, respectively.

He is currently a Professor with the School of Computer Science and Technology, Dalian University of Technology. His current research interests include big data processing and analysis, intelligent CAD, computer graphics and robotics technology, and knowledge automation.

Dongsheng Zhou (Member, IEEE) received the Ph.D. degree in Dalian University of Technology. Currently, he is a Distinguish Professor of Liaoning Province. In 2019, he established the School of Software Engineering of Dalian University and served as the first dean. His research interests include computer graphics, intelligence computing, and human-robot interaction. He was appointed as a member of the Computer Science and Technology Discipline Evaluation Group of the Academic Degrees Committee of the Sixth People's Government of Liaoning Province. He is a member of IEEE, ACM, CGS and CCF.