Long-Short History of Gradients Is All You Need: Detecting Malicious and Unreliable Clients in Federated Learning

Ashish Gupta[1(✉)], Tie Luo[1(✉)], Mao V. Ngo[2], and Sajal K. Das[1]

[1] Missouri University of Science and Technology, Rolla, USA
{ashish.gupta,tluo,sdas}@mst.edu
[2] Singapore University of Technology and Design, Singapore, Singapore
vanmao_ngo@sutd.edu.sg

Abstract. Federated learning offers a framework of training a machine learning model in a distributed fashion while preserving privacy of the participants. As the server cannot govern the clients' actions, nefarious clients may attack the global model by sending malicious local gradients. In the meantime, there could also be *unreliable* clients who are *benign* but each has a portion of low-quality training data (e.g., blur or low-resolution images), thus may appearing similar as malicious clients. Therefore, a defense mechanism will need to perform a *three-fold* differentiation which is much more challenging than the conventional (two-fold) case. This paper introduces MUD-HoG, a novel defense algorithm that addresses this challenge in federated learning using *long-short history of gradients*, and treats the detected malicious and unreliable clients differently. Not only this, but we can also distinguish between *targeted* and *untargeted attacks* among malicious clients, unlike most prior works which only consider one type of the attacks. Specifically, we take into account sign-flipping, additive-noise, label-flipping, and multi-label-flipping attacks, under a non-IID setting. We evaluate MUD-HoG with six state-of-the-art methods on two datasets. The results show that MUD-HoG outperforms all of them in terms of accuracy as well as precision and recall, in the presence of a mixture of multiple (four) types of attackers as well as unreliable clients. Moreover, unlike most prior works which can only tolerate a low population of harmful users, MUD-HoG can work with and successfully detect a wide range of malicious and unreliable clients - up to 47.5% and 10%, respectively, of the total population. Our code is open-sourced at https://github.com/LabSAINT/MUD-HoG_Federated_Learning.

1 Introduction

In recent years, the proliferation of smart devices with increased computational capabilities have laid a solid foundation for training machine learning (ML)

V. Atluri et al. (Eds.): ESORICS 2022, LNCS 13556, pp. 445–465, 2022.
https://doi.org/10.1007/978-3-031-17143-7_22

models over a large number of distributed devices. Traditional ML approaches require the training data to reside at a central location; the distributed ML case requires a well-controlled data-center-like environment. Such approaches demand high network bandwidth and provoke great privacy concerns. To this end, Google introduced the concept of Federated Learning (FL) [21] which allows distributed clients to collaboratively train a global ML model without letting their data leave the respective devices. At a high level, it works as follows. A central server initiates the training process by disseminating an initial global model to a set of clients. Each client updates the received model using its local data and sends back the updated model (not data). The server aggregates the received model updates (weights or gradients) into a global model and disseminates it again back to the clients. This procedure repeats until the global model converges. FL is advantageous in preserving data privacy and saving communication bandwidth, and has been applied to a wide range of applications in the Internet of Things (IoT) [12], natural language processing [10,14], image processing [17], etc.

However, the uncontrolled and distributed nature of the clients, as well as the server's inaccessibility to clients' data, make FL vulnerable to adversarial attacks launched by clients [1–3,20,30]. In general, a malicious client (adversary) can launch two types of attacks: (1) an *untargeted attack*, sometimes referred to as a Byzantine attack [5,15,30], where the adversary attempts to corrupt the *overall* performance of the global model (e.g., degrade a classifier's accuracy on all classes); (2) a *targeted attack*, where the adversary aims to degrade the model performance only for some specific cases (e.g., misclassify all dogs to cat) while not affecting the other cases [9,20]. Untargeted attacks could be tackled by robust aggregation techniques [4,7,33] when data are independent and identically distributed (IID) among the clients, whereas targeted attacks are much harder to defend because their specific targets are often unknown to the defender.

Another category of clients, which are largely overlooked in the FL security literature, are *unreliable clients*. These are benign clients but some of their data are of low quality and hence may appear as if their model updates were malicious too. For example, IoT devices such as sensors, smartphones, wearables, and surveillance cameras, are often subject to rigid hardware limitations and harsh ambient environments and thus may produce low-quality and noisy data [11]. A simplified solution could be one that treats clients who do not improve classification performance over a number of rounds as unreliable, and excludes them from aggregation in subsequent rounds, like in [18,19]. However, firstly this does not differentiate between benign and malign clients; secondly, excluding unreliable clients is not always desirable because such clients may possess valuable data such as infrequent classes on which other clients have no or few samples.

In this paper, we tackle the challenge of detecting and distinguishing between malicious and unreliable clients, as well as between targeted and untargeted attackers (among malicious clients), in FL. The main idea of our approach is to use *long-short history of gradients* jointly with judiciously chosen distance and similarity metrics during the iterative model updating process. Unlike prior works in [1,4,7,9,20] which only consider attackers, we identify unreliable clients

and take advantage of their contributions. We further consider both targeted and untargeted attacks and more fine-grained attack types: (untargeted) additive-noise and sign-flipping attacks, and (targeted) single- and multi-label-flipping attacks. Moreover, unlike prior works in [4,7,33], we consider non-IID data settings which are more representative of real-world FL scenarios with heterogeneous clients.

The main contributions of this paper are summarized as follows:

- We propose a novel approach MUD-HoG that stands for **M**alicious and **U**nreliable Client **D**etection using **H**istory **o**f **G**radients. To the best of our knowledge, this is the first work that detects both malicious attackers and unreliable clients in FL, distinguishing between targeted and untargeted attackers. It allows the server to treat the clients in a more *fine-grained* manner, by exploiting unreliable clients' low-quality (but still useful) data.
- We introduce *short HoG* and *long HoG* and a sequential strategy that uses them in a carefully-designed way, allowing us to achieve the above goal. In addition, we achieve our goal in a non-IID setting which is more realistic and challenging, with the presence of mixed types of attackers.
- We conduct extensive experiments to evaluate MUD-HoG in terms of accuracy, precision, recall, and detection ratio, on two benchmark datasets in comparison with 6 prior FL security mechanisms. The results show that MUD-HoG withstands up to 47.5% clients being malicious with a negligible (∼1%) compromise of accuracy, and comprehensively outperforms all the baselines on the considered metrics.

The rest of the paper is organized as follows. Section 2 reviews the related literature while Sect. 3 define the problem statement with the types of clients and considered attacks. Section 4 presents the proposed MUD-HoG approach with novel concepts of short HoG and long HoG, and Sect. 5 evaluates the robustness of the approach by conducting extensive experiments. Finally, Sect. 6 concludes the paper with future research directions.

2 Related Work

2.1 Distributed ML with Malicious Clients

Defending against malicious clients has been explored in distributed ML [4,34,35]. It has been noted that the stochastic gradient descent (SGD) algorithm is vulnerable to untargeted (Byzantine) attacks where malicious clients send random/arbitrary gradients to the server to negatively affect the convergence or performance of the global model. Methods such as Krum and Multi-Krum [4], Medoid [33], and GeoMed [7] have been proposed to defend against Byzantine attacks by extending SGD with a robust aggregation function. In another work [26], the authors argued that the effect of malicious clients can be mitigated by gradient or norm clipping based on a threshold assuming that the attacks produce boosted gradients. However, these methods assume IID data,

which often does not hold in FL settings. In addition, they aim to *tolerate* malicious clients rather than *distinguishing* them from normal ones, and thus may lead to cumulative negative impact over time and is also less preferable.

2.2 FL Under Untargeted Attacks

Various Byzantine-robust algorithms have been proposed for FL's non-IID settings in recent years. For example, a class of subgradient-based algorithms is proposed to defend malicious clients by robustifying the objective function with a regularization term [15]. However, these algorithms only consider simple attacks such as same-value and sign-flipping attacks. In another work [30], a variance reduction scheme inherited from [8] is combined with model aggregation to tackle untargeted attacks. In [6], the authors provided provable guarantees to ensure that the predicted label of a testing sample is not affected by the attack. They also proposed an ensemble method with a voting strategy to address the case of a bounded number of malicious clients. However, similar to some of the works discussed in the distributed ML case, this ensemble method cannot identify which clients are malicious. The above Byzantine-robust algorithms fail to stand against the attackers if they are present in high percentage. Moreover, all the above works are vulnerable to targeted attacks such as label flipping [27].

2.3 FL Under Targeted Attacks

As targeted attacks aim to reduce the model performance only on certain tasks while maintaining a good performance on others, they are elusive and harder to detect [20]. One of the popular defense methods, called FoolsGold [9], attempts to detect targeted attackers (e.g., label-flipping) based on the diversity of client contributions over the training rounds with an unknown number of attackers. With more realistic FL settings, Awan *et al.* [1] also exploited the clients' per-round contribution and cosine-similarity measure to defend against data poisoning attackers. In [16], an anomaly detection framework is proposed to differentiate anomalous gradients from normal ones in a low-dimensional embedding (spectral) using reconstruction errors. However, it requires a pre-trained model on a reference dataset at the server prior to start the training process, which is a strong requirement often not met in FL settings. Mao *et al.* [20] treated FL as a repeated game and introduced a robust aggregation model to defend against targeted and untargeted adversaries by designing a lookahead strategy based similarity measure. However, like many studies discussed earlier, it tolerates but does *not distinguish* adversaries from normal clients. Moreover, since most existing works [1,9,16,20] consider only two types of clients (normal and malicious), they may treat an unreliable client (who possesses lower-quality data) as malicious, which is not desirable.

In this work, we do not include *backdoor attacks* [1,24,29,32], which are a sub-category of data poisoning attack triggered by a particular pattern (e.g., pixel patch) embedded into data (e.g., images). However, unlike prior work, we include unreliable clients which are more likely to encounter in realistic FL deployments.

We also highlight that the term *unreliable* or *irrelevant* clients used in some studies [18,19,22] means clients whose contributions do not make any progress (i.e., improve model accuracy) over the past few rounds, which is considerably different from our definition of unreliable clients (see Sect. 3.2) which refers to clients who have low-quality data.

3 Model

We consider a typical FL framework with a central server and multiple clients participating in a collaborative model training process for a classification task using a deep learning model.

3.1 FL Preliminaries

Let N be the total number of clients participating in the FL model training process. Out of these N clients, m of them are malicious, and u of them are unreliable. Thus, there are $n = N-m-u$ normal clients. We consider a typical FL scenario for building a neural network model, where all clients share a common model structure under the same learning objective. The server initiates training by sending a global model $\boldsymbol{w}$ (e.g., random weights) to all clients. Each client updates the model $\boldsymbol{w}$ by training on its local dataset a certain number of epochs, and sends back the updated gradients. Note that sending gradients is equivalent to sending model parameters (weights). During training, each client learns the new weights $\boldsymbol{w}'$ by minimizing a loss function $\mathcal{L}(h_w(x), y)$ (e.g., cross-entropy loss function) over multiple epochs, where the function $h_w(\cdot)$ maps input data samples x to labels y. At a round τ, a client c_i computes the gradients as follows:

$$\nabla_{\tau,i} = \boldsymbol{w}_\tau - \underset{\boldsymbol{w}}{\operatorname{argmin}} \quad \mathcal{L}(h_{i,\boldsymbol{w}}(x), y). \tag{1}$$

Let the client c_i hold a local dataset $\mathcal{D}_i$ which can be non-IID as compared to other clients. When all clients are normal, the server aggregates all the gradients received from the clients, by

$$\boldsymbol{\nabla}_\tau = \sum_{i=1}^{N} \frac{|\mathcal{D}_i|}{|\boldsymbol{\mathcal{D}}|} \nabla_{\tau,i}, \tag{2}$$

where $|\boldsymbol{\mathcal{D}}| = \sum_{i=1}^{N} |\mathcal{D}_i|$. The weights of the global model for the next round $\tau+1$ are then updated as $\boldsymbol{w}_{\tau+1} = \boldsymbol{w}_\tau - \eta \boldsymbol{\nabla}_\tau$, where η is the learning rate.

3.2 Client Types

For generality, we consider a heterogeneous FL setting in which clients may be sensor boards, smartphones, surveillance cameras, laptops, connected vehicles, etc., owned by individuals or organizations. As a result, their data could be non-IID and thus each client could contribute to the global model training. We consider three types of clients and the last is further categorized in terms of attack types (see Sect. 3.3) the malicious client can launch.

1) *Normal clients* honestly participate in the model training process and have good-quality data.
2) *Unreliable clients* participate honestly in the FL but have some of its data are of low-quality. These data, however, could be exploited to improve diversity, especially if they capture distributions that normal clients fail to (or inadequately do). For example, A low-end camera does not produce high-resolution images but may capture some infrequent classes of images that other clients do not. Note that our definition of "unreliable client" is different from that in [18,19] and also from the "irrelevant client" in [22], where they mean a client who does not make progress (i.e., improve model accuracy) over the past few FL rounds, which therefore is a useless client.
3) *Malicious clients* are attackers who manipulate their local training data (i.e., *data poisoning*) or model weights/gradients (i.e., *model poisoning*) to generate adversarial impact on the global model being trained. For example, they may alter the labels of some of their data samples or perturb their local gradients before sending to the server.

With the presence of mixed types of clients having non-IID data, our problem is more realistic and challenging than prior work such as [1,7,9]. Figure 1 provides an overview of our problem setting, where MUD-HoG runs at the server.

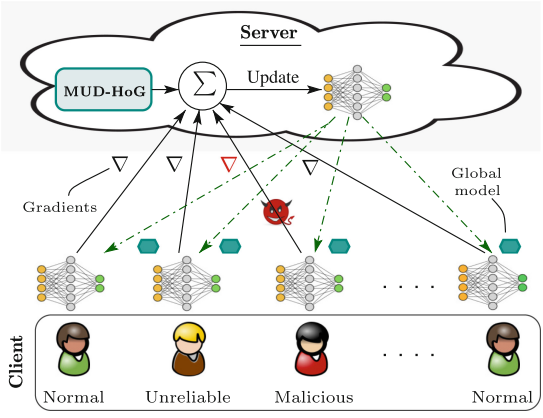


Fig. 1. Overview of FL with mixed types of clients. Malicious clients include targeted and untargeted attackers

Problem Statement. The problem in hand is two-fold: (1) How to identify and differentiate malicious clients (together with their attacks) from unreliable clients at the server while performing model aggregation? (2) How to mitigate the negative influence of malicious clients on the global model while still taking advantage of unreliable clients' updates? Let us reformulate Eq. (2) as:

$$\boldsymbol{\nabla}_\tau = \sum_{i \in \mathcal{C}_{norm}} \frac{|\mathcal{D}_i|}{|\boldsymbol{\mathcal{D}}|} \nabla_{\tau,i} + \alpha \sum_{i \in \mathcal{C}_{unrl}} \frac{|\mathcal{D}_i|}{|\boldsymbol{\mathcal{D}}|} \nabla_{\tau,i}, \tag{3}$$

where $\mathcal{C}_{norm}$ and $\mathcal{C}_{unrl}$ are the set of normal clients and that of unreliable clients, respectively, and the parameter $\alpha \in (0, 1)$ down-weights the gradients of unreliable clients. Note that malicious clients are excluded.

3.3 Threat Model

A malicious client can launch either of the following attacks:

- **Untargeted attack.** The objective here is to downgrade the *overall* performance of the global model. The following two model poisoning attacks are considered: (i) *Sign-flipping.* The malicious client flips the sign of its local gradients (from positive to negative and vice versa) before sending them to server, while the magnitude of the gradients remains unchanged. (ii) *Additive-noise.* The malicious client adds Gaussian or random noise to its local gradients before sending to the server.
- **Targeted attack.** The objective is to decrease model performance on particular cases while not affecting other cases. The following two data poisoning attacks are considered: (i) *Label-flipping.* The attacker changes the label of all the instances of one particular class (source label), say y_1, to another class (target label), say y_2, while (intentionally) keeping other classes intact to avoid being detected. (ii) *Multi-label-flipping.* The attacker flips multiple source labels to a particular target label. This will result in the target label has an increased accuracy while harming the accuracy on other classes.

We make the following *Assumptions*: (i) Each attacker can only manipulate its own data or model but not other clients' or modify the server's aggregation algorithm. (ii) Number of malicious clients (including untargeted and targeted attackers) is less than other clients (including normal and unreliable). (iii) Malicious clients are persistent, meaning that they attack in every round.

4 MUD-HoG Design

MUD-HoG runs at the server to defend the global model. Unlike existing work such as [4], MUD-HoG assumes that the number of malicious clients is *unknown* to the server.

Challenges. The design challenges come from the following factors: the mixed types and unknown distribution of clients, non-IID data, and the server's inaccessibility to client data. The only information that the server has is the gradients (Eq. 1) sent by the clients each round, as a result of their local optimization such as stochastic gradient descent (SGD) over the loss function $\mathcal{L}(\cdot)$.

With targeted attacks, the malicious clients share a common objective and thus will have similar gradients [9] between each other. On the other hand, gradients from untargeted attackers would be dissimilar from each other since they perturb gradients randomly or flip gradient signs. This gradient space is rather complex and irregular, insofar as there is no single appropriate similarity measure that can distinguish malicious clients from the normal ones. Furthermore, unreliable clients introduce another degree of complication as they would behave very similar to *untargeted* attackers and hence are hard to distinguish.

Long-Short History of Gradients (HoG). We propose two new notions of HoG, based on which we design a robust algorithm MUD-HoG to address the above challenges. Let $\nabla_i = \{\nabla_{1,i}, \nabla_{2,i}, \cdots, \nabla_{\tau-1,i}\}$ denote the collection of HoGs received by the server from client c_i prior to the τ^{th} round.

Definition 1 (Short HoG). The short HoG of client c_i at round τ, defined as,

$$\nabla_i^{sHoG} = \frac{1}{l} \sum_{t=\tau-l}^{\tau-1} \nabla_{t,i} \tag{4}$$

is a moving average of c_i's gradients of the last l rounds, where l is the sliding window size. The short HoG smooths a client's gradients to remove single-round randomness.

Definition 2 (Long HoG). The long HoG of client c_i at round τ is defined as

$$\nabla_i^{lHoG} = \sum_{t=1}^{\tau-1} \nabla_{t,i}, \tag{5}$$

which is the sum of all the gradients in the set ∇_i. Thus, the long HoG captures the *accumulated* influence of a client on the global model, which reflects its goal.

Note that, at any round τ, the server does not need to store all the previous gradient vectors $\{\nabla_{1,i}, \nabla_{2,i}, \cdots, \nabla_{\tau-1,i}\}$ received from the client c_i; instead, it only needs to keep l latest vectors for computing short HoG and the *sum* of all the previous vectors for long HoG. Hence, at each round, the server would keep only $l+1$ gradient vectors for each client. Therefore, the required memory is independent of the number of training rounds τ, and one should not have memory concerns when τ increases.

4.1 Sequential Strategy

By introducing short HoG and long HoG, MUD-HoG exploits two different gradient space and follows a sequential strategy to detect the type of each client in the following order: untargeted, targeted, unreliable, and normal, as depicted in Fig. 2. The key ideas are discussed in the following steps.

1) Untargeted attack. We can deduce the untargeted intention from the client's short HoG. Since an untargeted attacker aims to corrupt the whole

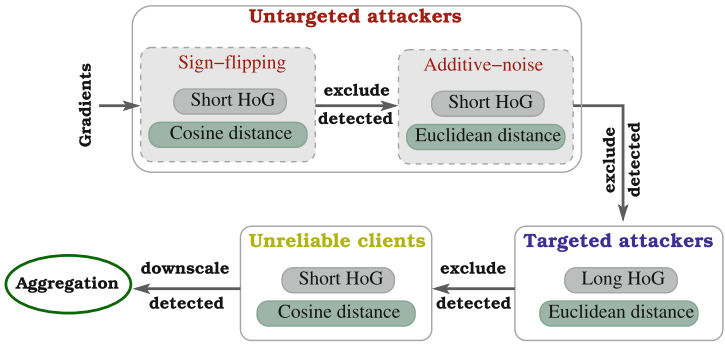


Fig. 2. Overview of MUD-HoG with the gradient space (short or long HoG) and similarity measures (Euclidean or cosine) used for detecting different types of clients

model, for example using sign flipping or additive noise, its short HoG would differ substantially from normal clients. First, in the case of sign-flipping attack, a malicious client essentially changes its gradient to the opposite direction, which would result in a large angular deviation from the *median* gradient of all the clients, as depicted in Fig. 3a. This also justifies that using *cosine distance* in the space of short HoG would be an appropriate choice. Note that short HoG is more robust than a single-round gradients by reducing false alarms.

On the other hand, additive-noise attackers and unreliable clients (with low-quality data) would have similar short HoGs, but considered collectively, would be apart from other clients. Therefore, after excluding the sign-flipping attackers, we use a clustering method based on short HoG to distinguish the above two types of clients from other clients. Empirically, we choose DBSCAN [25] as the clustering method because it conforms to our intuition and yields the best results. Between these two types, additive-noise attackers tend to be *farther* away from other clients than unreliable clients as the attackers add deliberate perturbations; nevertheless, a separation boundary could be learned by finding the largest gap over Euclidean distances. We also note that this is not a clear-cut line and further processing is needed which we discuss below in Step 3. The above intuition is depicted in Fig. 3b.

2) Targeted attack. Targeted attackers intend to manipulate the global model toward a specific convergence point (e.g., misclassifying all dogs to cats). Such intention can be captured by our long HoG which *reinforces* their adversarial goal over the entire history and is also robust to short-term noises and *camouflage cases* in which some attackers may strategically behave benignly in some of the rounds in order to evade detection. In MUD-HoG, we use K-means clustering with K=2 over long HoG to separate out targeted attackers, *after* excluding untargeted attackers detected in Step 1.

3) Unreliable clients. Finally, MUD-HoG identifies and separates unreliable clients from normal ones. After excluding all the detected malicious clients (tar-

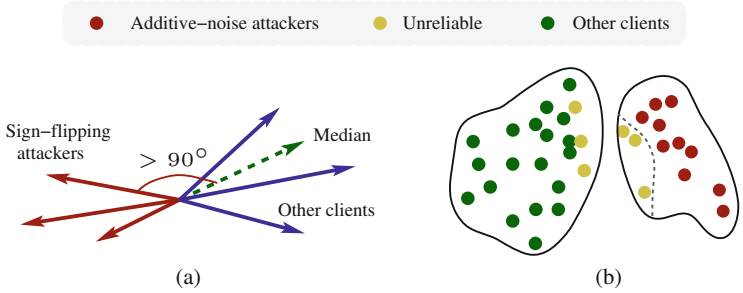


Fig. 3. Illustration of (a) the angular deviation of sign-flipping attackers from the median client (green), and (b) clustering of additive-noise attackers, unreliable clients, and other clients after excluding sign-flipping attackers (Color figure online)

geted and untargeted), the unreliable clients become farther from the *median* client in terms of their *short HoG*. Rather than using clustering, in this case we find that the *cosine distance* is the most effective to detect them and hence adopt it in MUD-HOG.

4.2 Detection of Malicious Clients

Based on the basic ideas discussed above, now we present all the technical details of how MUD-HoG detects different types of clients. The server starts detection from round τ_0 ($\tau_0 = l = 3$ in our experiments).

Detecting Untargeted Attackers Using Short HoG. MUD-HoG first computes the *median* short HoG over all clients, as $\nabla_{med}^{sHoG} = median\{\nabla_i^{sHoG} \mid 1 \leq i \leq N\}$. Then, it flags a client c_i as a sign-flipping attacker if

$$d_{cos}\left(\nabla_{med}^{sHoG}, \nabla_i^{sHoG}\right) < 0, \tag{6}$$

where the function $d_{cos}(\cdot)$ computes the cosine distance. We note that an existing algorithm CONTRA [1] also employs cosine distance to separate out *targeted attackers*. CONTRA computes the pair-wise distances between the gradients of all the clients, which therefore leads to a complexity of $O(N^2)$; in contrast, MUD-HoG uses median and thus the complexity is linear, $O(N)$, which is worth noting because FL often deals with a massive number of clients (e.g., IoT devices). Moreover, CONTRA does not handle unreliable clients.

Next, MUD-HoG proceeds to detecting additive-noise attackers after excluding the above detected sign-flipping attackers. We apply DBSCAN clustering on the short HoGs of all the remaining clients and obtain two groups – (i) a smaller group (g_l) consisting of the additive-noise attackers and unreliable clients and (ii) a larger group (g_h) consisting of the rest of the clients. Based on our above analysis that the additive-noise attackers are relatively farther from normal clients than unreliable clients (Fig. 3b), MUD-HoG attempts to learn a

separation boundary as follows. Recalculate ∇_{med}^{sHoG} as the median short HoG of group g_h, and construct $\mathbf{d} = \{d_{Euc}(\nabla_{med}^{sHoG}, \nabla_i^{sHoG})\}$ which is a set of Euclidean distances (denoted by $d_{Euc}(\cdot)$) between ∇_{med}^{sHoG} and each client $c_i \in g_l$. The reason we use Euclidean distance rather than cosine distance is that the former produces a larger separation over *unnormalized* short HoG (which we intend). Then, we find the largest gap between any two consecutive values in the *sorted* list of the set $\mathbf{d}$, and use the mid-point of this gap as the separation boundary d_ϕ. Thus, a client $c_i \in g_l$ is an additive-noise attacker if

$$d_{Euc}\left(\nabla_{med}^{sHoG}, \nabla_i^{sHoG}\right) > d_\phi \tag{7}$$

for $1 \leq i \leq |g_l|$. The remaining clients in g_l and the set g_h will be handled in the next step. The above detection of untargeted attackers is summarized as the pseudo-code of Lines $6-16$ in Algorithm 1.

Detecting Targeted Attackers Using Long HoG. After excluding the detected untargeted attackers as above, we compute the long HoG for each of the remaining clients, denoted by ∇_i^{lHoG}. Then, we apply K-means clustering with $K = 2$ on all the computed long HoGs to obtain two groups of clients: the smaller group will consist of the targeted attackers and the other (bigger) group of the normal clients, based on our assumption that normal clients constitute more than half of the entire population. In Algorithm 1, Lines 17-18 corresponds to the detection of targeted attackers.

4.3 Detection of Unreliable Clients

We are now left with a mixture of unreliable and normal clients. To distinguish them, MUD-HoG finds a new separation boundary d_ϕ as follows. Let N' be the number of remaining clients and ∇_{med}^{sHoG} be the (updated) median short HoG of them. Let $\mathbf{d}' = \{d_{cos}(\nabla_{med}^{sHoG}, \nabla_i^{sHoG})\}$ be a set of *cosine* distances between ∇_{med}^{sHoG} and each client c_j for $1 \leq i \leq N'$. The separation boundary d_ϕ is then determined from $\mathbf{d}'$ similarly as the above detection of additive-noise attackers (but here we use cosine distance). Then, a client c_i is deemed unreliable if it satisfies the condition

$$d_{cos}\left(\nabla_{med}^{sHoG}, \nabla_i^{sHoG}\right) < d_\phi. \tag{8}$$

Note that the cosine distance is smaller when the angle between two vectors is larger, and that is why the condition '$<$' used in (8) is opposite to that in (7). The unreliable clients are detected at Lines 19-24 in Algorithm 1 after exclusion of all types of attackers.

Thus finally (in each FL round), MUD-HoG obtains the set of normal clients $\mathcal{C}_{norm}$ and the set of unreliable clients $\mathcal{C}_{unrl}$, after filtering out $\mathcal{C}_{tar}$ and $\mathcal{C}_{untar}$. It then aggregates the gradients of normal and unreliable clients using (3) (or see Line 26 in Algorithm 1), where unreliable clients are downscaled, and then updates the global model as $\boldsymbol{w}_{\tau+1} = \boldsymbol{w}_\tau - \eta \boldsymbol{\nabla}_\tau$. Clearly, since the gradients of malicious clients have been discarded, their negative impact is eradicated from the global model.

Algorithm 1: MUD-HoG

Input: Gradients from round 1 to τ, for each client c_i, denoted by $\nabla_i = \{\nabla_{1,i}, \nabla_{2,i}, \cdots, \nabla_{\tau-1,i}\}$, $i = 1...N$. (Note that the server only keeps the latest l gradient vectors and the *sum* of all $\tau - 1$ gradients.)

Output: Normal clients ($\mathcal{C}_{norm}$), targeted attackers ($\mathcal{C}_{tar}$), untargeted attackers ($\mathcal{C}_{untar}$), and unreliable clients ($\mathcal{C}_{unrl}$)

Initialize $\mathcal{C}_{norm}, \mathcal{C}_{tar}, \mathcal{C}_{untar} = \emptyset, \mathcal{C}_{all} = \{c_i\}, 1 \leq i \leq N$

for *round* $\tau = 1$ *to* τ_0 **do**

  Aggregate gradients of all clients

for *round* $\tau = \tau_0 + 1$ *to* $\mathcal{T}$ **do**

  Compute short HoG ∇_i^{sHoG} and long HoG ∇_i^{lHoG} for each client c_i

  `/* Detecting untargeted attackers */`

  Computer median short HoG ∇_{med}^{sHoG} over all N clients

  for $i = 1$ *to* N **do**

    if (6) *holds* **then**

      $\mathcal{C}_{untar} = \mathcal{C}_{untar} \cup \{c_i\}$; `// Sign-flipping attackers`

  Apply DBSCAN clustering on short HoGs of $\mathcal{C}_{all} \setminus \mathcal{C}_{untar}$ to obtain two groups g_l and g_h

  Compute ∇_{med}^{sHoG} of the larger group g_h

  Compute d_{Euc} between ∇_{med}^{sHoG} and each ∇_i^{sHoG} of the smaller group g_l

  Find the separation boundary d_ϕ per Section 4.2

  for $i = 1$ *to* N *and* $c_i \notin \mathcal{C}_{untar}$ **do**

    if (7) *holds* **then**

      $\mathcal{C}_{untar} = \mathcal{C}_{untar} \cup \{c_i\}$; `// Additive-noise attackers`

  `/* Detecting targeted attackers */`

  Apply K-means clustering with $K = 2$ on long HoGs of $\mathcal{C}_{all} \setminus \mathcal{C}_{untar}$

  $\mathcal{C}_{tar}$ = clients who belong to the smaller cluster

  `/* Detecting unreliable clients */`

  Recompute ∇_{med}^{sHoG} over $\mathcal{C}_{all} \setminus \{\mathcal{C}_{tar} \cup \mathcal{C}_{untar}\}$

  Compute d_{cos} between ∇_{med}^{sHoG} and each ∇_i^{sHoG} of $\mathcal{C}_{all} \setminus \{\mathcal{C}_{tar} \cup \mathcal{C}_{untar}\}$

  Recompute the separation boundary d_ϕ per Section 4.3

  for $i = 1$ *to* N *and* $c_i \notin \{\mathcal{C}_{tar} \cup \mathcal{C}_{untar}\}$ **do**

    if (8) *holds* **then**

      $\mathcal{C}_{unrl} = \mathcal{C}_{unrl} \cup \{c_i\}$

  $\mathcal{C}_{norm} = \mathcal{C}_{all} \setminus \{\mathcal{C}_{tar} \cup \mathcal{C}_{untar} \cup \mathcal{C}_{unrl}\}$

  `/* Aggregate gradients over` $\mathcal{C}_{norm}$ `and` $\mathcal{C}_{unrl}$ `*/`

  $\boldsymbol{\nabla}_\tau = \sum_{i \in \mathcal{C}_{norm}} \frac{|\mathcal{D}_i|}{|\mathcal{D}|} \nabla_{\tau,i} + \alpha \sum_{i \in \mathcal{C}_{unrl}} \frac{|\mathcal{D}_i|}{|\mathcal{D}|} \nabla_{\tau,i}$

  Update global model as $\boldsymbol{w}_{\tau+1} = \boldsymbol{w}_\tau - \eta \boldsymbol{\nabla}_\tau$

  Send $\boldsymbol{w}_{\tau+1}$ back to all clients

return $\mathcal{C}_{norm}, \mathcal{C}_{tar}, \mathcal{C}_{untar}, \mathcal{C}_{unrl}$

5 Performance Evaluation

In this section, we evaluate MUD-HoG in comparison with six state-of-the-art methods on two real datasets with various type of attacks.

5.1 Experiment Setup

We consider a classification task on two datasets: (i) MNIST [13]: Our FL task is to train a deep model with 2 convolutional neural networks (CNN) followed by 3 fully connected layers[1] to classify 10 digits. (ii) Fashion-MNIST [31]: We build a deep model with 6 CNN layers followed by two fully connected layers to classify 10 fashion classes.

Hyper-parameters. We train the FL model with SGD optimizer (learning rate = 1e-2, momentum = 0.5 for MNIST and 0.9 for Fashion-MNIST, and weight-decay = 1e-4 for Fashion-MNIST) over 40 communication rounds, 4 local epochs; other setup details are similar to [28]. We use the window size of $l = 3$ for calculating the moving average short HoG. Our algorithm triggers only after $\tau_0 = 3$ rounds to accumulate enough HoGs. Since, the server stores only $l + 1$ gradient vectors (l latest and a *sum* of all previous vectors) to compute HoGs, it never runs into storage related issues. Moreover, we make a firm decision about malicious clients if they are detected in two consecutive rounds. Therefore, our algorithm can only detect malicious clients at least after 4 rounds.

To simulate non-IID data, we divide the datasets into 40 clients as disjoint portions that follows *Dirichlet distribution* with hyperparameter 0.9, as also adopted by [2,28]. Besides normal clients, our FL system consists of unreliable clients (up to 10% of total clients), and malicious clients (up to 47.5% of total clients), as detailed below.

Untargeted Attacks. (i) *Sign-flipping (SF)* – We flip the sign of gradients of the malicious clients without enlarging the magnitudes in our FL setup, which makes the detection more challenging. (ii) *Additive-noise (AN)* – We add a Gaussian noise with $\mu = 0$ and $\sigma = 0.01$ to the gradients of attackers.

Targeted Attacks. (i) *Label-flipping (LF)* – Before training the local model, attacker flips label of digit "1" to "7" in its local MNIST dataset, and label ("1-Trouser‘') to ("7-Sneaker") in Fashion-MNIST dataset. (ii) *Multi-label-flipping (MLF)* – Attacker flips the labels of few source classes to a targeted class in its local dataset. For MNIST and Fashion-MNIST (in brackets) datasets, we flip three source labels of digits "1" ("1-Trouser"), "2" ("2-Pullover"), and "3" ("3-Dress") to a target label "7" ("7-Sneaker").

Unreliable Clients. We simulate them to mimic a real-life scenario of low-end smartphone with poor-resolution camera and computing power. We use *Gaussian smoothing* (kernel size= 7, $\sigma = 50$) to blur 50% of the local image dataset; and simulate low computing power by training over randomly selected portion of 30% of local dataset. We set $\alpha = 0.5$ to downscale the unreliable clients.

[1] Adopt the model from PyTorch tutorial.

To simulate heterogeneous FL scenarios, we consider two different series of experiments with upto 47.5% malicious clients (including untargeted and targeted attack) and upto 10% unreliable clients. We configure 12 different experimental setups with increasing numbers of unreliable and malicious clients as follows.

- *Series of Exp1* consists of $a = \min\{i, 4\}$ unreliable clients, $b = \min\{i, 6\}$ additive-noise attackers, $c = \min\{i, 5\}$ sign-flipping attackers, $d = (i + 2)$ label-flipping attackers, and $(40 - a - b - c - d)$ normal clients; where $i = \{1, 2, 3, 4, 5, 6\}$.
- *Series of Exp2* consists of $a = \min\{i, 4\}$ unreliable clients, $b = \min\{i, 6\}$ additive-noise attackers, $c = \min\{i, 5\}$ sign-flipping attackers, $d = (i + 2)$ multi-label-flipping attackers, and $(40 - a - b - c - d)$ normal clients; where $i = \{1, 2, 3, 4, 5, 6\}$.

Evaluation Metrics. The performance of MUD-HoG is measured in terms of precision, recall, accuracy, and detection ratio. We define *detection ratio* (r) as

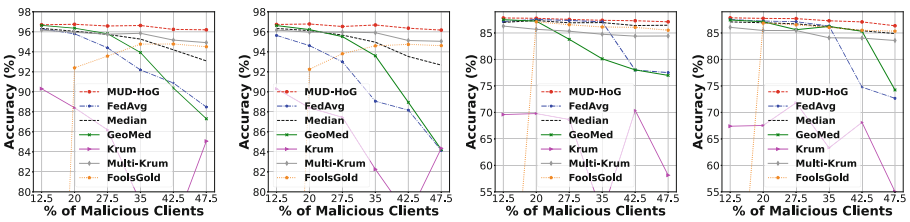
$$r = \frac{\sum_{\tau=1}^{\mathcal{T}} \sum_{i \in \mathcal{C}_x} \mathbb{1}(c_i \text{ detected at } \tau)}{\mathcal{T} \sum_x |\mathcal{C}_x|} \tag{9}$$

where $\mathcal{C}_x$ is either $\mathcal{C}_{tar}$, $\mathcal{C}_{untar}$ or $\mathcal{C}_{unrl}$, and not all of them are empty. The higher the detection ratio (closer to 100%), the better algorithm is.

Benchmark Algorithms. In addition to FedAvg [21], a popular algorithm in FL, we compare our proposed MUD-HoG algorithm with five other algorithms. They are: (i) coordinate-wise Median (or Median for short) [35], (ii) GeoMed [7], (iii) Krum [4], (iv) Multi-krum (or MKrum for short) [4], and (v) FoolsGold [9]. We borrowed the source code of these existing algorithms from [28].

5.2 Experimental Results

Overall Performance. Figure 4 shows the accuracy of 12 setups for series of Exp1 and Exp2 for MNIST and Fashion-MNIST datasets under the above seven benchmark algorithms. We observe that over all 12 setups with multiple types of attacks, MUD-HoG always achieves the best accuracy.



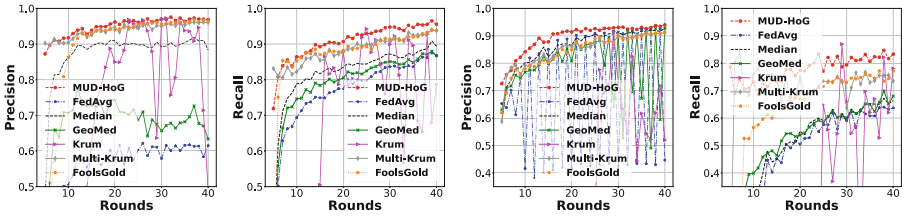
(a) Series of Exp1 (b) Series of Exp2 (c) Series of Exp1 (d) Series of Exp2

Fig. 4. Accuracy vs. the percentage of malicious clients. (a) and (b) are results on the *MNIST* dataset. (c) and (d) are results on the *Fashion-MNIST* dataset

It is consistently observed that when increasing percentage of malicious clients from 12.5% to 47.5% of the total number of clients, Krum and FoolsGold show fluctuated performance and poor performance at a certain level of attacks, some other algorithms such as FedAvg, GeoMed, Median, and MKrum continuously drop their accuracy. In contrast, our proposed MUD-HoG maintains robust performance against multiple levels of heterogeneous attacks.

For **MNIST** dataset shown in parts (a) and (b) of Fig. 4, initially GeoMed performs as good as MUD-HoG, but when the level of attacks are increased more than 35%, GeoMed drops its accuracy by 9.33% and 12.39% while MUD-HoG only drops 0.5% and 0.56% in series of Exp1 and Exp2, respectively. When compared to the second-best algorithm, i.e. MKrum, the proposed algorithm gained upto 1.28% and 1.12% higher accuracy in series of Exp1 and Exp2, respectively.

For **Fashion-MNIST** dataset shown in parts (c) and (d) of Fig. 4, GeoMed achieves comparative results as MUD-HoG at a low level of attacks for both series; however, GeoMed drops performance significantly at the high level of attacks. For instance, in series of Exp1 and Exp2, while MUD-HoG's accuracy only drops by 0.72% and 1.5% (when increasing percentage of attacks from 12.5% to 47.5%), GeoMed's accuracy drops by 10.52% and 13.21%, respectively. When compared to the second-best algorithm, i.e., Median, MUD-HoG gains upto 0.65% and 1.47% accuracy in series of Exp1 and Exp2, respectively.



(a) Precision of class "7" (b) Recall of class "2" (c) Precision of class "7" (d) Recall of class "2"

Fig. 5. Results for *Series of Exp2* with 42.5% malicious clients. "2" and "7" are the source and target classes, respectively. (a) and (b) are results on the *MNIST* dataset. (c) and (d) are results on the *Fashion-MNIST* dataset

Precision and Recall. To make a fair comparison with other algorithms (i.e., Krum, MKrum, FoolsGold) that ware designed specifically for targeted attacks, we plot *precision* of the targeted class (i.e., number of samples correctly classified as the targeted class over all samples predicted as the targeted class), and *recall* of a source class (i.e., number of samples correctly classified as the source class over all ground-truth samples of the source class) for MNIST and Fashion-MNIST datasets in Fig. 5. Here, FedAvg, GeoMed, Median or even Krum obtain poor performance and highly fluctuated precision of targeted class and recall of source class because they could not defend targeted attacks. On the flip side, though MKrum and FoolsGold show quite good precision, their values are lower than MUD-HoG for both the datasets.

Detection Ratio. We keep track of detected rounds for each type of clients during the course of FL training with MUD-HoG algorithm. Table 1 reports detection ratio (defined in Eq. 9) for each type of clients, and their first round of detection (presented inside brackets) for a setup in series of Exp1 and Exp2 with 27.5% malicious clients. We observe that the sign-flipping and additive-noise attackers are detected immediately at round 4, which is the earliest round when the MUD-HoG algorithm could provide a firm decision.

Table 1. Detection ratio r (%) and the earliest round (**1strnd**) that detects the client type (round number in brackets), with 27.5% malicious clients. [**SF**: Sign-flipping, **AN**: Additive-noise, **LF**: Label-flipping, **MLF**: Multi-label-flipping, **UR**: Unreliable]

Type	Detection	MNIST		Fashion-MNIST	
		Exp1	Exp2	Exp1	Exp2
SF	r (**1strnd**)	90.0 (4)	90.0 (4)	90.0 (4)	90.0 (4)
AN	r (**1strnd**)	90.0 (4)	90.0 (4)	90.0 (4)	90.0 (4)
LF	r (**1strnd**)	87.5 (5)	–	85.0 (6)	–
MLF	r (**1strnd**)	–	90.0 (4)	–	85.0 (6)
Overall rate r (%)		88.9	90.0	87.7	87.7
UR	r (**1strnd**)	87.5 (5)	87.5 (5)	85.0 (6)	85.0 (6)

For MNIST dataset, overall, we can detect all malicious clients at detection ratio (calculated over all types of clients) 88.9% and 90.0% for a setup in series of Exp1 and Exp2, respectively. Since FL training is done over 40 rounds and the earliest detection round is 4, upper bound of detection ratio can be at most 90.0%. And we can see in Exp2 of MNIST, MUD-HoG can detect MLF at round 4, which is as early as SF or AN, resulting in 90.0% of detection ratio. Next, for Fashion-MNIST dataset, our algorithm detects targeted attacks (i.e., LF and MLF) a bit slower than the case in MNIST, but the overall detection ratio is still above 87%. Finally, for unreliable clients (last two rows in Table 1), in all experiments, MUD-HoG achieves firm results of all unreliable clients from round 5 and round 6 for MNIST and Fashion-MNIST datasets, respectively. As a result, the detection ratio for unreliable clients is above 85.0%.

5.3 Discussions and Limitations

Convergence Analysis. Based on our experimental results (see Fig. 6 and Fig. 7), the loss of the global model stabilizes in 40 FL rounds for both the datasets even in the presence of 42.5% clients posing different types of attacks and having non-IID data. This indicates that MUD-HoG can achieve convergence in rather adversarial scenarios. Although the presence of malicious clients initially diverges the global model from its objective, excluding them from aggregation, as MUD-HoG did, rectifies the SGD process back to normal as defined

in [23]. In future work, we plan to incorporate a rigorous theoretical analysis of convergence for our approach.

More Strategic Attacks. While we have experimentally shown that MUD-HoG is robust to various untargeted and targeted attacks in the presence of a large number of malicious clients, it may still miss out attackers who perform stealthy or highly strategic targeted attacks (some are formally defined in [7]). Besides, an attacker may implant a certain trigger pattern into some training/test data to inject corruption [3,29], known as *backdoors*. Such attacks are more evasive since they are only triggered when the particular pattern arises, while the overall performance is almost not affected. Currently, MUD-HoG has not been specifically designed to defend backdoor attacks but this would be an interesting direction to explore.

6 Conclusion

While federated learning (FL) offers a privacy-preserving framework for collaborative training of ML models, it is susceptible to adversarial attacks. This paper has proposed a new approach called MUD-HoG to detect malicious clients who launch untargeted or targeted attacks and unreliable clients who possess low-quality data, and offers a fine-grained classification of four types of participants. We introduce the concept of long-short HoG and select appropriate distance and similarity measures to identify different types of attacks and clients. MUD-HoG excludes malicious contributions but exploits unreliable clients' contributions to maximize the utility of the final global model. Experimental results confirm that MUD-HoG is robust against malicious and unreliable clients and produces a global model with higher accuracy than state-of-the-art baselines. It can detect a mixture of multiple types of attackers and unreliable clients in non-IID settings even when the ratio of attackers is close to half. In future work, we plan to investigate more challenging and dynamic settings where attackers may vary attack types and clients may even switch roles (attackers, unreliable, normal, etc.) over time. More extensive experiments will also be conducted.

Acknowledgements. This work is partially supported by the NSF grant award #2008878 (FLINT: Robust Federated Learning for Internet of Things) and the NSF award #2030624 (TAURUS: Towards a Unified Robust and Secure Data Driven Approach for Attack Detection in Smart Living).

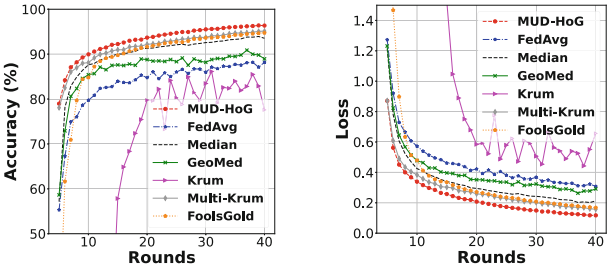
A Additional Experimental Results

A.1 Performance Improvement over Rounds

We consider a specific setup with 42.5% malicious clients, for both the datasets to evaluate the improvement of the accuracy of all the algorithms over FL rounds.

We plot test accuracy and loss from round 5 to the final round 40 for MNIST dataset in Fig. 6 using global model. It is obvious to see that MUD-HoG obtains

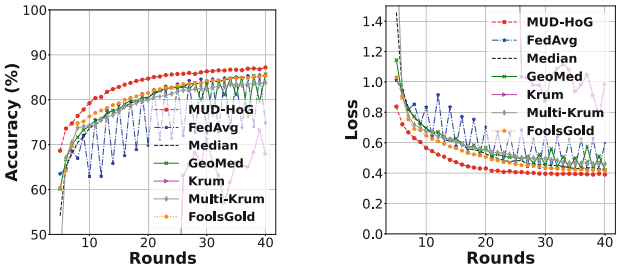
an upper bound of test accuracy and an lower bound of test loss over the course of FL training. While some algorithms show fluctuated performance during training such as Krum with a high fluctuation, or FedAvg and GeoMed with smaller fluctuations, the other state-of-the-art algorithms designed against attackers such as Median, MKrum, FoolsGold and MUD-HoG show smooth improvement as training progresses. Among these algorithms, we also observe in Fig. 6 that the gap of test loss between MUD-HoG and the second-best algorithm is increasing over the course of FL training.



(a) Accuracy (b) Loss

Fig. 6. Performance improvement of global model on *MNIST* in *Series of Exp2* with 42.5% malicious clients

Figure 7 shows test accuracy and loss for Fashion-MNIST dataset. Similar to MNIST's results, we can see that among all evaluated algorithms, MUD-HoG obtains the highest accuracy and the lowest loss for all training rounds. The fluctuation of FedAvg and GeoMed is more severe with high variance, so the final accuracy of these algorithms are not really reliable. This is the reason why FedAvg and GeoMed can obtain accuracy close to MUD-HoG (see Fig. 4) in the setups of 12.5% and 20% of malicious clients.

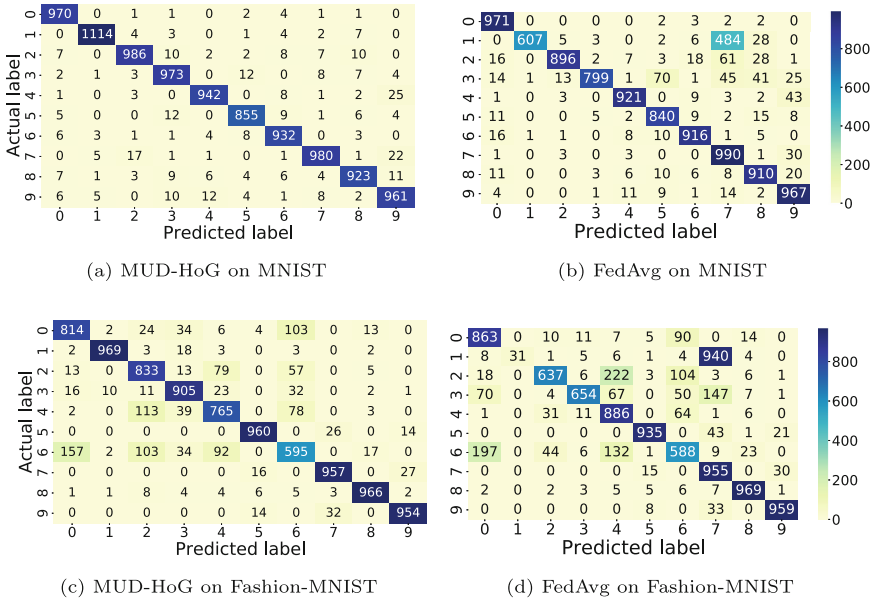


(a) Accuracy (b) Loss

Fig. 7. Performance improvement of global model on *Fashion-MNIST* in *Series of Exp2* with 42.5% malicious clients

A.2 Confusion Matrix

In Fig. 8, we show confusion matrices for MUD-HoG and FedAvg obtained from the completely trained model for MNIST and Fashion-MNIST datasets using a setup of series Exp2 with 42.5% malicious clients. As multi-label-flipping attackers flip their local samples with source labels of "1", 2‘', and "3‘' to the target label "7", we can clearly see in parts (b) and (d) of Fig. 8, FedAvg confuses with several samples actually having the source labels as the target label while it is not the case for MUD-HoG. In addition, we see an interesting observation in part (d) of Fig. 8, where FedAvg completely fails as it predicts nearly all samples of source label "1" as the target label "7" (i.e., 940 samples of label "1" are predicted as label "7").



(a) MUD-HoG on MNIST

(b) FedAvg on MNIST

(c) MUD-HoG on Fashion-MNIST

(d) FedAvg on Fashion-MNIST

Fig. 8. Confusion matrices in *Series of Exp2* with 42.5% malicious clients

References

1. Awan, S., Luo, B., Li, F.: CONTRA: defending against poisoning attacks in federated learning. In: Bertino, E., Shulman, H., Waidner, M. (eds.) ESORICS 2021. LNCS, vol. 12972, pp. 455–475. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-88418-5_22
2. Bagdasaryan, E., Veit, A., Hua, Y., Estrin, D., Shmatikov, V.: How to backdoor federated learning. In: International Conference on Artificial Intelligence and Statistics, pp. 2938–2948. PMLR (2020)

3. Bhagoji, A.N., Chakraborty, S., Mittal, P., Calo, S.: Analyzing federated learning through an adversarial lens. In: International Conference on Machine Learning, pp. 634–643. PMLR (2019)
4. Blanchard, P., El Mhamdi, E.M., Guerraoui, R., Stainer, J.: Machine learning with adversaries: byzantine tolerant gradient descent. In: 31st International Conference on Neural Information Processing Systems. pp. 118–128 (2017)
5. Cao, X., Fang, M., Liu, J., Gong, N.Z.: Fltrust: byzantine-robust federated learning via trust bootstrapping. In: ISOC Network and Distributed System Security Symposium (NDSS) (2021)
6. Cao, X., Jia, J., Gong, N.Z.: Provably secure federated learning against malicious clients. In: AAAI Conference on Artificial Intelligence, vol. 35, pp. 6885–6893 (2021)
7. Chen, Y., Su, L., Xu, J.: Distributed statistical machine learning in adversarial settings: Byzantine gradient descent. ACM Measur. Anal. Comput. Syst. **1**(2), 1–25 (2017)
8. Defazio, A., Bach, F., Lacoste-Julien, S.: Saga: a fast incremental gradient method with support for non-strongly convex composite objectives. In: Advances in Neural Information Processing Systems (2014)
9. Fung, C., Yoon, C.J., Beschastnikh, I.: 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)
10. Hard, A., et al.: Federated learning for mobile keyboard prediction. arXiv (2018)
11. Jiang, Y., Cong, R., Shu, C., Yang, A., Zhao, Z., Min, G.: Federated learning based mobile crowd sensing with unreliable user data. In: IEEE International Conference on High Performance Computing and Communications, pp. 320–327 (2020)
12. Khan, L.U., Saad, W., Han, Z., Hossain, E., Hong, C.S.: Federated learning for internet of things: recent advances, taxonomy, and open challenges. IEEE Commun. Surv. Tutor. **23**(3), 1759–1799 (2021)
13. LeCun, Y.: The MNIST database of handwritten digits (1998). http://yann.lecun.com/exdb/mnist/
14. Leroy, D., Coucke, A., Lavril, T., Gisselbrecht, T., Dureau, J.: Federated learning for keyword spotting. In: IEEE International Conference on Acoustics, Speech and Signal Processing, pp. 6341–6345 (2019)
15. Li, L., Xu, W., Chen, T., Giannakis, G.B., Ling, Q.: RSA: byzantine-robust stochastic aggregation methods for distributed learning from heterogeneous datasets. In: AAAI Conference on Artificial Intelligence, vol. 33, pp. 1544–1551 (2019)
16. Li, S., Cheng, Y., Wang, W., Liu, Y., Chen, T.: Learning to detect malicious clients for robust federated learning. arXiv (2020)
17. Liu, Y., et al.: Fedvision: an online visual object detection platform powered by federated learning. In: AAAI Conference on Artificial Intelligence, vol. 34, pp. 13172–13179 (2020)
18. Ma, C., Li, J., Ding, M., Wei, K., Chen, W., Poor, H.V.: Federated learning with unreliable clients: performance analysis and mechanism design. IEEE Internet Things J. **8**, 17308–17319 (2021)
19. Mallah, R.A., Lopez, D., Farooq, B.: Untargeted poisoning attack detection in federated learning via behavior attestation. arXiv (2021)
20. Mao, Y., Yuan, X., Zhao, X., Zhong, S.: Romoa: robust model aggregation for the resistance of federated learning to mdodel poisoning attacks. In: Bertino, E., Shulman, H., Waidner, M. (eds.) ESORICS 2021. LNCS, vol. 12972, pp. 476–496. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-88418-5_23

21. McMahan, B., Moore, E., Ramage, D., Hampson, S., Arcas, B.A.: Communication-efficient learning of deep networks from decentralized data. In: Artificial Intelligence and Statistics, pp. 1273–1282. PMLR (2017)
22. Nagalapatti, L., Narayanam, R.: Game of gradients: mitigating irrelevant clients in federated learning. In: AAAI Conference on Artificial Intelligence, vol. 35, pp. 9046–9054 (2021)
23. Nguyen, L.M., Nguyen, P.H., Richtárik, P., Scheinberg, K., Takáč, M., van Dijk, M.: New convergence aspects of stochastic gradient algorithms. J. Mach. Learn. Res. **20**, 1–49 (2019)
24. Ozdayi, M.S., Kantarcioglu, M., Gel, Y.R.: Defending against backdoors in federated learning with robust learning rate. In: AAAI Conference on Artificial Intelligence, vol. 35, pp. 9268–9276 (2021)
25. Schubert, E., Sander, J., Ester, M., Kriegel, H.P., Xu, X.: DBSCAN revisited, revisited: why and how you should (still) use DBSCAN. ACM Trans. Database Syst. (TODS) **42**(3), 1–21 (2017)
26. Sun, Z., Kairouz, P., Suresh, A.T., McMahan, H.B.: Can you really backdoor federated learning? arXiv (2019)
27. Tolpegin, V., Truex, S., Gursoy, M.E., Liu, L.: Data poisoning attacks against federated learning systems. In: Chen, L., Li, N., Liang, K., Schneider, S. (eds.) ESORICS 2020. LNCS, vol. 12308, pp. 480–501. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-58951-6_24
28. Wan, C.P., Chen, Q.: Robust federated learning with attack-adaptive aggregation. ArXiv:abs/2102.05257 (2021)
29. Wang, H., et al.: Attack of the tails: Yes, you really can backdoor federated learning. arXiv (2020)
30. Wu, Z., Ling, Q., Chen, T., Giannakis, G.B.: Federated variance-reduced stochastic gradient descent with robustness to byzantine attacks. IEEE Trans. Signal Process. **68**, 4583–4596 (2020)
31. Xiao, H., Rasul, K., Vollgraf, R.: Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms (2017)
32. Xie, C., Chen, M., Chen, P.Y., Li, B.: CRFL: certifiably robust federated learning against backdoor attacks. In: International Conference on Machine Learning, pp. 11372–11382. PMLR (2021)
33. Xie, C., Koyejo, O., Gupta, I.: Generalized byzantine-tolerant SGD. arXiv (2018)
34. Xie, C., Koyejo, S., Gupta, I.: Zeno: Distributed stochastic gradient descent with suspicion-based fault-tolerance. In: International Conference on Machine Learning, pp. 6893–6901. PMLR (2019)
35. Yin, D., Chen, Y., Kannan, R., Bartlett, P.: Byzantine-robust distributed learning: towards optimal statistical rates. In: International Conference on Machine Learning, pp. 5650–5659. PMLR (2018)