FLVoogd: Robust And Privacy Preserving Federated Learning

Yuhang Tian Y.TIAN-13@STUDENT.TUDELFT.NL
Rui Wang R.WANG-8@TUDELFT.NL
Yanqi Qiao Y.QIAO@TUDELFT.NL
Delft University of Technology

Emmanouil Panaousis E.PANAOUSIS@GREENWICH.AC.UK
University of Greenwich

Kaitai Liang KAITAI.LIANG@TUDELFT.NL
Delft University of Technology

Editors: Emtiyaz Khan and Mehmet Gönen

Abstract

In this work, we propose FLVoogd, an updated federated learning method in which servers and clients collaboratively eliminate Byzantine attacks while preserving privacy. In particular, servers use automatic Density-based Spatial Clustering of Applications with Noise (DBSCAN) combined with Secure Multi-party Computation (SMPC) to cluster the benign majority without acquiring sensitive personal information. Meanwhile, clients build dual models and perform test-based distance controlling to adjust their local models toward the global one to achieve personalizing. Our framework is automatic and adaptive that servers/clients don't need to tune the parameters during the training. In addition, our framework leverages SMPC's operations, including multiplications, additions, and comparisons, where costly operations, like division and square root, are not required. Evaluations are carried out on some conventional datasets from the image classification field. The result shows that FLVoogd can effectively reject malicious uploads in most scenarios; meanwhile, it avoids data leakage from the server side.

Keywords: federated learning; secure-multi-party computation; differential privacy

1. Introduction

Unlike the centralized learning setting, where a server collects substantial users' data to build a model for predictions, Federated Learning (FL) requires the model parameters that clients train independently with their data and devices. In FL paradigm frameworks, such as FedAvg McMahan et al. (2016), the server iteratively aggregates local models trained by individuals and sends the global one back to clients for their local updates, to achieve collaborative training. Since no actual data is sent to the server, this paradigm was considered privacy-preserving. In the past half-decade, FL has been widely researched and applied in many fields such as image recognition Li et al. (2021a), natural language processing Liu et al. (2021), financial system Long et al. (2020), and medical care Dayan et al. (2021).

However, such a framework still faces two main challenges - privacy and security. On the one hand, the conventional FL setting cannot get rid of the disclosure of clients' information

and even enlarge the attacking surface Aono et al. (2017). Not only can the server be an adversary to infer the information from local models sent by clients in this scenario, but also every participant can perform the inference attack on the global model constructed by each individual. Therefore, some research employs SMPC to encrypt the uploads, such as Nguyen et al. (2021a). However, the cost of the design or operations is high, especially when dealing with the division and the square root. On the other hand, without countermeasures, adversaries can arbitrarily substitute the data with the poisoned one Wang et al. (2020) or even directly change the upload into a meaningless random number, leading their uploads to betray the regulation. To eliminate the attacking consequence, some research, like Li et al. (2021b) and Rieger et al. (2022), builds a practical defensive framework, but with too unintuitive hyper-parameters to tune for different situations.

To improve the efficiency and save expensive operations, we develop an updated framework that combines SMPC Knott et al. (2021); OpenMined (2021), DBSCAN Ester et al. (1996), differential privacy Geyer et al. (2017), and personalized local model Li et al. (2021b) to eliminate the malicious uploaded parameters without revealing any sensitive information and guarantee (ϵ, δ)-DP after the aggregation. Compared with the past research, our framework 1)filters the abnormal uploads without knowing their sensitive information; 2)performs the training process adaptively, requiring no parameter tuning; 3)uses SMPC operations efficiently supported by most protocols. We leverage the conventional image classification dataset to evaluate the framework. The results show that the filter can reject the Byzantine attacks under most situations without degrading the model performance, and the trade-off between predicting accuracy and DP strength can be customized for different scenarios.

2. Background and Problem Setting

2.1. Adversarial Attack

We use state-of-art attacks to test our FL's robustness and named from A1 to A6.

•**Random upload** (A1): As its name suggests, the adversary substitutes the factual update with a random noise chosen from $X \sim \mathcal{N}(0, 1)$. Consequently, the average of parameters can arbitrarily deviate from $w_{avg} = \frac{1}{n}\sum_{i=1}^{n} w_i$ to $w_{dev} = \frac{1}{m}\sum_{i=1}^{m} w_i + \frac{1}{n-m}\sum_{i=m+1}^{n}\mathcal{N}(0, 1)$, where m is the number of honest updates and $(n - m)$ is the number of malicious updates.

•**Krum attack** (A2): It is designed to crack the Krum aggregation rule. In a nutshell, Krum selects one vector from a set of n vectors that is the most comparable to the rest. Even if a compromised client gives the chosen vector, the impact is limited in this situation. However, when adversaries try to invalidate the Krum aggregation rule, they can conspire to elaborate a set of vectors to make $KR(w'_1, ..., w'_f, ..., w_n)$ output w'_1 such that w'_1 mostly inversely differs from the true selected one without being attacked Fang et al. (2020).

•**Trimmed-mean attack** (A3): Trimmed-mean sorts n updates for each j^{th} parameter $sort(w_{1j}, ..., w_{nj})$, eliminates the highest and smallest β amount from the sorted list, and averages the remaining $(n - 2\beta)$ parameters as the global model's j^{th} parameter Yin et al. (2018). To enervate this aggregation rule, adversaries collude to submit deviating models in the opposite direction that the global model would change in the absence of attacks.

•**Label flipping** (A4): Each adversary converts the label of a sample from l to $L - l - 1$, where l is the truth label of the sample, and L is the total number of classes Muñoz-González et al. (2017). For instance, adversaries label digit "0" as "9" and digit "9" as "0" to label-

flip the MNIST data.
•**Backdoor triggering** (A5): This kind of attack is also known as trojan attacks Gu et al. (2017). The adversary inserts a specific pattern into training samples or uses existing ones to render the corresponding testing samples with that pattern classified as the desired class. This pattern functions as a trigger. After the global model learns this pattern, it will be triggered and output the misled prediction. If the adversary uses the existing pattern in the sample, this backdoor attack is a semantic backdoor attack Rieger et al. (2022).
•**Edge-case attack**(A6): Under the edge-case attack setting, adversaries aim to attack the heavy-tail of the prediction Wang et al. (2020). They try to find or manufacture samples that the model predicts correctly but with a comparably low confidence value; then, they label those samples with a label they want. The intuition behind it is that the model cannot assure the correctness of predictions even if the result is correct, as the predicting score is not such high, so it can be easily misled by the attacker who feeds those edge-case samples with wrong labels.

2.2. DBSCAN

DBSCAN is initially designed for clustering and distinguishing the noise from the high dimensional database depending on the variance of density Ester et al. (1996). A non-negligible quantity of samples should be in the cluster if a cluster is formed, while the cluster can hardly be formed in areas where samples are located sparsely. These "depopulated zones" can be used as gaps to separate the different classes and to sift out noisy samples. We will consistently follow some of the concepts and symbols used in Ester et al. (1996). $N_{Eps}(p)$ represents neighbors of a point p within a range with radius Eps (Eps is a preset hyper-parameter). A point p is a *corepoint*, if $|N_{Eps}| \geq MinPts$ ($MinPts$ is a preset hyper-parameter). In addition, a *corepoint* is the centroid of a cluster, so in other words, a cluster is only formed when its centroid is a *corepoint*. A point p is a *borderpoint*, if its neighbours contain at least one *corepoint*. It should be noted that a point can be a *borderpoint* for different clusters, but it will be only assigned to a unique cluster eventually, and it depends on which cluster it assigns the point to first. If a point is neither a *corepoint* nor a *borderpoint*, it will be classified as noise.

2.3. SMPC

As mentioned, uploading weights instead of the raw data to the server is not privacy-preserving. As shown in Zhang and Luo (2020), model parameters can disclose some information about individual data. For example, adversaries can use generative adversary networks to reconstruct the class representatives from the aggregated parameters. This powerful reconstruction is more harmful if it happens on the server side because the server can steal the class representatives from each individual uploading. To avoid revealing the uploads to the server, SMPC can be used for private aggregation, and the result will only be revealed eventually. Following the structures in Nguyen et al. (2021b,a); Rieger et al. (2022), we will use Secure 2-party Computation (S2PC), a ramification of SMPC, to guarantee that the individual upload will not be plain-text to the server. Under the S2PC setting, each client will not directly send the model parameters to the server but separate the upload into two parts and share one with the server for aggregation and another with

the external server. As both servers hold merely one piece of the secret, the secret cannot be known if they do not collude, because it is computationally infeasible for a single server to reconstruct the actual data from decrypting the individual upload. Based on the secret sharing scheme, each server can do arithmetic operations relying on its own share and through some communication. To achieve this, two libraries CrypTen[1] Knott et al. (2021) and SyMPC[2] OpenMined (2021) derived from PySyft are used for the experiment. Both of them use secret sharing but with different protocols to achieve S2PC. CrypTen is currently designed only for semi-honest parties, while SyMPC can tolerate minor malicious parties.

2.4. Security Assumption

We primarily consider possible server-side and client-side risks. There are two types of servers in our setting. Firstly, servers can be honest-but-curious who infer the actual data or relevant information from uploads while heeding the regulation. Secondly, if FLVoodg runs under SyMPC-Falcon Wagh et al. (2020), servers can be malicious (minority) who betray the secure aggregation rule and send an incorrect model back to participants. In terms of participants, in each round, less than half of them can be malicious and perform byzantine attacks 2.1 to deteriorate the performance of the global model. In addition, any client can be curious about information from others and performs client-level inference attacks Geyer et al. (2017), inferring whether a particular client participates in the training, given a specific dataset of that client.

3. FLVoogd Overview and Design

3.1. FLVoogd Server

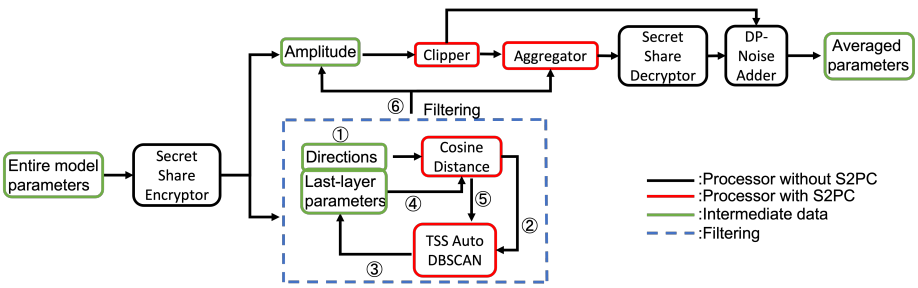


Figure 1: FLVoogd server framework.

The overview of our framework, FLVoogd with SMPC, is shown in Fig. 1. Initially, each client computes the $l2$-norm of its uploaded parameters w_i as the amplitude $||w||_i \leftarrow \sqrt{\sum_j w_{ij}^2}$ and unitizes the parameters to $\overline{w}_i \leftarrow \frac{w_i}{||w||_i}$ as the direction. The unitizing can simplify the later SMPC computations, e.g., cosine distance where costly division and square root operations Nguyen et al. (2021a) are saved. Besides, the client performs the same for the parameters from the last layer and obtains the unitized last layer parameters $\overline{v}_i$. The

1. https://github.com/facebookresearch/CrypTen
2. https://github.com/OpenMined/SyMPC

reason for extracting the last layer is that parameters in the final layer reveal more explicit information relevant to the dataset's distribution Rieger et al. (2022) which can be used for distinguishing backdoor uploads. Then, the client secretly shares the direction $\overline{w}_i$, the amplitude $||w||_i$, and last-layer direction $\overline{v}_i$ with two parties, the server for aggregation and external server. If the SMPC protocol uses Falcon, one more server is added and receives the third share from the client.

Once servers receive the uploads from the selected clients, they can perform secure aggregation. It is supposed that the number of received uploads is n, and the clipping boundary for the current round is $c^{(t)}$. The server first carries out the filtering process, the block with a light yellow background in Fig. 1, and the procedure can be divided into six steps. ①: In the first step, the server uses the directional vector $\overline{w}$ to compute the cosine distance matrix M_{cos} by Eq. (1), where $M_{cosij} = cos - dist(i,j)$ for $i \neq j$, $M_{cosij} = 0$ for $i = j$, i or j denotes the client's index, and u denotes the parameter's index. Since the vectors are unit vectors, the cosine distance between two vectors can be simplified into a dot production. The computation is collaboratively completed by two/three servers, involving the addition and multiplication among secret shares. ②: In the second step, the server feeds the Total-Sum-of-Square (TSS)-based DBSCAN with the distance matrix from the former step. DBSCAN calculates the TSS by Eq. (2) for each pair of rows to obtain a new distance matrix M_{tss}. M_{cos} provides the directional similarity, while M_{tss} enlarges the variance of counter-directions and narrows the variance of identical directions such that the filter can capture the difference more easily. There are two hyper-parameters for DBSCAN, Eps and $MinPts$. As honest-majority is the basic assumption, $MinPts$ is set to $\lfloor n/2 \rfloor + 1$ and Eps is the average of the median ($\lfloor n/2 \rfloor$) in each row of the distance matrix $M_{tss} \in \mathbb{R}^{n \times n}$. Unlkie Nguyen et al. (2021a) where a binary iterative search is applied for finding the appropriate parameter in every round, our setting for Eps and $MinPts$ guarantees the DBSCAN can automatically adjust its radius accordingly through the whole process without any manual involvement and iteration. This step includes the addition, multiplication, and comparison of shares. ③: In the third step, DBSCAN filters out the noise and minority group and returns indices of the majority group. The server selects the corresponding clients' parameters from the last layer according to the indices. ④: For the next step, similar to step 1, the server again computes the cosine distance matrix but now uses the last-layer parameters. Then, the server obtains a cosine distance matrix M'_{cos}. ⑤: The fifth step is identical to the second step. ⑥: In the final steps, DBSCAN outputs the indices that the server will consider as benign.

$$cos - dist(i,j) = \frac{\sum_u \overline{w}_{i,u}\overline{w}_{j,u}}{\sqrt{\sum_u \overline{w}_{i,u}^2}\sqrt{\sum_u \overline{w}_{j,u}^2}} = \sum_u \overline{w}_{i,u}\overline{w}_{j,u} = \overline{w}_i \cdot \overline{w}_j \tag{1}$$

$$tss - dist(i,j) = \sum_u (M_{cosi,u} - M_{cosj,u})^2 \tag{2}$$

After knowing which clients are considered as benign ones, the server clips their amplitudes before performs the aggregation by $||w||_i = \min(||w||_i, c^{(t)})$. On the one hand, it restrains the abnormally large amplitude and controls the next descent step size. On the other hand, it provides the $l2$-sensitivity for the DP budget tracer. During the clipping, the clipper records the ratio of clients not being clipped as $\hat{\gamma}$. The expected clipping ratio is set

as γ. The next round clipping boundary $c^{(t+1)}$ is updated by $c^{(t+1)} = c^{(t)} \cdot \exp(-\eta_c(\hat{\gamma} - \gamma))$, where η_c is the learning rate of the clipper. If the actual non-clipping number of clients is larger than expected, the clipping boundary will decrease to cut more clients in the next round; otherwise, it will increase to be looser. The exponential base guarantees that any adjustment is a positive number. As adaptive clipping provides a public clipping boundary based on rough counting, it cannot reveal sensitive information. SMPC's operation in this step requires comparing a share with a public number.

The aggregation rule is simply averaging benign clients' uploads by $w_{global} = \frac{1}{m}\sum_{i=1}^{m} \overline{w}_i \cdot \min(||w||_i, c^{(t)})$. It is supposed that the number of clients after filtering is $m(\leq n)$. After obtaining the merged model w_{global}, servers collaboratively reveal and announce the plain text of w_{global}. The aggregation contains the addition and multiplication of shares, and the multiplication of shares and public numbers.

The server eventually knows the global parameter till finishing aggregation, and local parameters are already merged into one; thus, the server has no idea of individual local updates. Before sending the global update back to clients, the server adds Gaussian noise to provide a differential privacy guarantee to defend against client-level inference from the client side. The mean is 0, and the standard deviation is the maximum $l2$-sensitivity multiplied by a coefficient σ that represents the strength of DP. Thanks to the clipping, all uploads are bounded into a sphere whose radius is exactly the clipping boundary $c^{(t)}$. Therefore, the noise is added following Eq. (3). Notably, the noise is added to the sum of updates not after averaging. DP-Noise Adder also tracks the DP budget for the server because it knows the number of clients used in this round and the amplitude of Gaussian noise. Finally, $||\tilde{w}_{global}||$ is compared with $||c^{(t)}||$. If $||\tilde{w}_{global}|| > ||c^{(t)}||$, from which the server deduces that the amount of noise is added too much, the server will scale down $||\tilde{w}_{global}||$ to a smaller value by Eq. (4). This operation follows the post-processing property of differential privacy so that (ϵ, δ) cannot be influenced. This post-processing is equivalent to adjusting the model learning rate lower after knowing the noise influences too much on the result. The algorithm of the FLVoogd server is manifested in Alg. 1.

$$\tilde{w}_{global} = \frac{1}{m}\{\sum_{i=1}^{m} \overline{w}_i \cdot \min(||w||_i, c^{(t)}) + \mathcal{N}(0, \sigma^2 \cdot (c^{(t)})^2)\} \tag{3}$$

$$\tilde{w}_{global} := \tilde{w}_{global} \cdot \min(1, \frac{c^{(t)}}{||\tilde{w}_{global}||}) \tag{4}$$

3.2. FLVoogd Client

Referencing the idea from Ditto Li et al. (2021b), each FLVoogd's client builds two identical models, namely, the global model and the local model. Varying from Cao et al. (2020); Nguyen et al. (2021a,b); Rieger et al. (2022) where the server entirely takes the responsibility of a robust model, clients can share that responsibility locally. This defensive scheme builds a gap between the self-used local and global models to enhance the robustness and offers personalization to users. In each round, the client receives the averaged aggregated weight difference $\tilde{w}_{global}$ from the server and updates the weight of the global model accordingly by

Algorithm 1 FLVoogd server algorithm

Input:
$\mathcal{C}$, N ▷ $\mathcal{C}$ is the set of clients, $N = |\mathcal{C}|$
T, q ▷ T is the number of training iteration, q is the sampling ratio
$c^{(0)}$, γ, η_c ▷ $c^{(0)}$ is the initial clipping boundary, γ is the expected clipping ratio, η_c is Clipper's learning rate
σ, δ ▷ σ is the coefficient to control the noise strength, δ is for DP
for round t: $1, 2, ..., T$ **do**
  $\mathcal{C}^{(t)}$, $n \leftarrow$ subsample($\mathcal{C}$, N, q) ▷ $n = |\mathcal{C}^{(t)}|$
  for $client_i \in \mathcal{C}^{(t)}$ **do**
    $\overline{w}_i^{(t)}$, $||w||_i^{(t)}$, $\overline{v}_i^{(t)} \leftarrow client_i(t, send)$ ▷ $\overline{w}$ is the unit vector of weight difference, $||w||$ is the norm of weight difference, $\overline{v}$ is the unit vector of last layer's weight difference
  $idx_{f1}^{(t)}$, $n^{(t)'} \leftarrow$ Auto_DBSCAN($\{\overline{w}_1^{(t)}, \overline{w}_2^{(t)}, ..., \overline{w}_n^{(t)}\}$) by Alg. 2 ▷ $n^{(t)'} = |idx_{f1}^{(t)}|$
  $idx_{f2}^{(t)}$, $n^{(t)''} \leftarrow$ Auto_DBSCAN($\{\overline{v}_i^{(t)} : i \in idx_{f1}\}$) by Alg. 2 ▷ $n^{(t)''} = |idx_{f2}^{(t)}|$
  $w_{global}^{(t)} \leftarrow 0$, $\hat{\gamma}^{(t)} \leftarrow 0$
  for index $i \in idx_{f2}^{(t)}$ **do**
    if $||w||_i^{(t)} > c^{(t)}$ **then**
      $||w||_i^{(t)} \leftarrow c^{(t)}$
    else
      $\hat{\gamma}^{(t)} \leftarrow \hat{\gamma}^{(t)} + 1$
    $w_{global}^{(t)} \leftarrow w_{global}^{(t)} + ||w||_i^{(t)} \cdot \overline{w}_i^{(t)}$
  $w_{global}^{(t)} \leftarrow \frac{w_{global}^{(t)}}{n^{(t)''}}$, $\hat{\gamma}^{(t)} \leftarrow \frac{\hat{\gamma}^{(t)}}{n^{(t)''}}$
  $c^{(t+1)} \leftarrow c^{(t)} \cdot \exp(-\eta_c(\hat{\gamma}^{(t)} - \gamma))$, $\epsilon^{(t)} \leftarrow$ DP_budget($\frac{n^{(t)''}}{N}$, σ, δ)
  $\tilde{w}_{global}^{(t)} \leftarrow w_{global}^{(t)} + \frac{1}{n^{(t)''}}\mathcal{N}(0, \sigma^2(c^{(t)})^2)$ ▷ satisfying (ϵ, δ)-differential privacy
  $\tilde{w}_{global}^{(t)} \leftarrow \tilde{w}_{global} \cdot \min(1, \frac{c^{(t)}}{||\tilde{w}_{global}||})$ ▷ satisfying post-processing
  $client_i(t, receive) \leftarrow \tilde{w}_{global}^{(t)}$

Algorithm 2 Auto DBSCAN

Input: W ▷ $W \in \mathbb{R}^{n \times m}$ represents $n \times m$ matrix where each row is a client's unit vector from n clients and the dimension of the vector is m
Output: idx, $|idx|$ ▷ idx is a list of indices of benign clients
$M_{cos} \leftarrow$ CosDist(W) by Eq. (1)
$M_{tss} \leftarrow$ TSSDist(M_{cos}) by Eq. (2)
for row i: $1, 2..., n$ **do**
  $median_i \leftarrow$ quickMedian($M_{tss,i}$)
$median \leftarrow \frac{1}{n}\sum_{i=1}^{n} median_i$
$Eps \leftarrow median$, $MinPts \leftarrow n//2 + 1$
$idx \leftarrow DBSCAN(M_{tss}, Eps, MinPts, precomputed)$
return idx, $|idx|$

$W_{global} := W_{global} + \tilde{w}_{global}$. In contrast, the local model is not updated in this step. After updating the locally global model, the client tests the model accuracy using evaluation data and obtains the testing accuracy acc_{ref}.

The client feeds the partial training data to the global and local models in each mini-batch iteration. It is supposed that there is a coefficient λ_{ditto} to control the distance of the local model from the global model. The objective function of the local model becomes like Eq. (5), where $F(\cdot)$ is the objective function for the global model and originally for the local

model. The change in the local model's objective function now is that the client adds an additional $l2$-regularization term to force the local model to approximate the global model. Consequently, the local model can learn from the global model, and the gap between them is constrained by λ_{ditto}.

$$\min_{W_{local}} F'(W_{local}; W_{global}) = F(W_{local}) + \frac{\lambda_{ditto}}{2}||W_{local} - W_{global}||^2 \tag{5}$$

Furthermore, Eq. (5) can be converted into a gradient decent format shown in Eq. (6), where η_{local} is the client's local learning rate. The formula shown in Eq. (6) can be easily implemented by PyTorch where the client extracts the gradient and adds the $\lambda_{ditto}(W_{local} - W_{global})$ term to it before running *optimizer.step()*.

$$g := g - \eta_{local}(\triangledown F(W_{local}) + \lambda_{ditto}(W_{local} - W_{global})) \tag{6}$$

Till now, the client has λ_{ditto} as a controller to adjust the learning distance between the local and global models, but how to set an appropriate value λ_{ditto} for the local model? Intuitively, if the global model is admirable and exemplary, we expect the local model to learn as much helpful information as possible from the global model; otherwise, we desire the local model to learn less or even not learn from the global model. Then, the client can use the testing accuracy acc_{ref} as a reference to flexibly adjust λ_{ditto} by Eq. (7). In the formula, λ_{max} and λ_{min} are the maximum and minimum values for λ_{ditto}, η_{ditto} is the learning rate, acc_{local} is the testing accuracy of the local model, and acc_{thres} is the minimum threshold to increase λ_{ditto}. λ_{max} and λ_{min} restrain the coefficient of the $l2$-regularization in a reasonable interval. η_{ditto} controls each mini-batch iteration's growing/decaying speed for λ_{ditto}. acc_{thres} is the threshold to control whether the current global model is worth being learned. In other words, the local model will absorb from the global model, only if acc_{ref} is higher than $acc_{local} + acc_{thres}$. The client secretly shares his/her update with servers after completing the training. The algorithm of FLVoogd's client is manifested in Alg. 3.

$$\lambda_{ditto} := \min(\lambda_{max}, \max(\lambda_{min}, \lambda_{ditto} + \eta_{ditto}(acc_{ref} - acc_{local} - acc_{thres}))) \tag{7}$$

4. Experiment

4.1. Experimental Setup

We conducted all the experiments using PyTorch, and the source code was available on https://github.com/Timo9Madrid7/maliciousfl.

Datasets and Neural Network. We followed the recent research on Byzantine attacks Fang et al. (2020); Wang et al. (2020) on FL and chose a typical application scenario - image classification. The datasets in our experiments included MNIST, CIFAR-10, and EMNIST. The non-IID data splitting for MNIST and CIFAR-10 in our experiment followed the method carried out in FLTrust Cao et al. (2020), where Deg_{nIID} (q in Cao et al. (2020)) controlled the level of non-IID. In terms of EMNIST, we applied the method from He et al. (2020) where the smaller α_{sim} was, the more tasks were dissimilar. CNNs were used as our global and local models, where CIFAR-10 was trained by ResNet-20 He et al. (2016)

Algorithm 3 FLVoogd client algorithm

Input:
$\mathcal{D}_{train}$, $\mathcal{D}_{eval}$ ▷ $\mathcal{D}_{train}$ is the training set, $\mathcal{D}_{eval}$ is the testing set
W_{local}, W_{global}, F ▷ W_{local}/W_{global} are the local/global parameters, F is the objective function
η_{local}, η_{global}, E ▷ $\eta_{local}/\eta_{global}$ is the learning rate for the local/global model, E is the number of local training epochs
$\lambda^{(0)}_{ditto}$, η_{ditto}, acc_{thres}, λ_{min}, λ_{max} ▷ $\lambda^{(0)}_{ditto}$ is the initial value for λ_{ditto}, η_{ditto} is the learning rate for λ_{ditto}, acc_{thres} is the threshold to start learning, $\lambda_{min}/\lambda_{max}$ is the minimum/maximum learning rate of λ_{ditto}
$\tilde{w}_{global} \leftarrow client(receive)$ ▷ receive the update from the server
$W_{global} \leftarrow W_{global} + \tilde{w}_{global}$
$W_{temp} \leftarrow$deepCopy(W_{global})
$acc_{ref} \leftarrow$Eval$(W_{global}, \mathcal{D}_{eval})$
for local epoch e: $1, 2, ..., E$ **do**
    for batch iteration $\mathcal{B} \in \mathcal{D}_{train}$ **do**
        $W_{global} \leftarrow W_{global} - \eta_{global} \nabla F(W_{global}, \mathcal{B})$ ▷ global train
        $W_{local} \leftarrow W_{local} - \eta_{local}(\nabla F(W_{local}, \mathcal{B}) + \lambda_{ditto}(W_{local} - W_{global}))$ ▷ local train
        $acc_{local} \leftarrow$Eval$(W_{local}, \mathcal{D}_{eval})$
        $\lambda_{ditto} \leftarrow \lambda_{ditto} + \eta_{ditto}(acc_{ref} - acc_{local} - acc_{thres})$ by Eq. (7)
$w \leftarrow W_{global} - W_{temp}$
$||w|| \leftarrow \sqrt{\sum_j w_j^2}$, $\overline{w} \leftarrow \frac{w}{||w||}$, $\overline{v} \leftarrow \frac{\{w_j : j=k,...,m\}}{\sqrt{\sum_{j=k}^{m} w_j^2}}$ ▷ k is the starting index of the last layer
$client(send) \leftarrow \overline{w}, ||w||, \overline{v}$

(269,772 parameters in total), and MNIST & EMNIST were trained by a 2×convolutional layers' NN. ResNet-20 was a pre-trained version[3] to accelerate the training process.

Evaluation Metrics. Main Task Accuracy (MA) represents the accuracy of a model tested by its benign task. It indicates the fraction of correct predictions. If the model is under targeted attacks, Backdoor Accuracy (BA) is the metric to reflect how successful the adversaries are. It denotes the fraction of correct predictions for backdoor samples.

FL Configuration. The total number of clients N was set to 100. Each client received unique training samples and testing samples from the split. The learning rates of global model η_{global} and local model η_{local} were 0.01. The local training epoch E was 1 since clients did not hold an adequate number of samples. The coefficient of $l2$-regularization λ_{ditto} was initialized as 0 and its min-max interval was $[0.0, 2.0]$, where the maximum was suggested by Li et al. (2021b). The threshold acc_{thres} for the local model starting learning from the global was 0.05. The learning rate η_{ditto} for λ_{ditto} was 1. The expected clipping ratio γ, the initial clipping boundary $c^{(0)}$, the clipping learning rate η_c were 0.5, 10, 0.3. Other settings varied for different experiments.

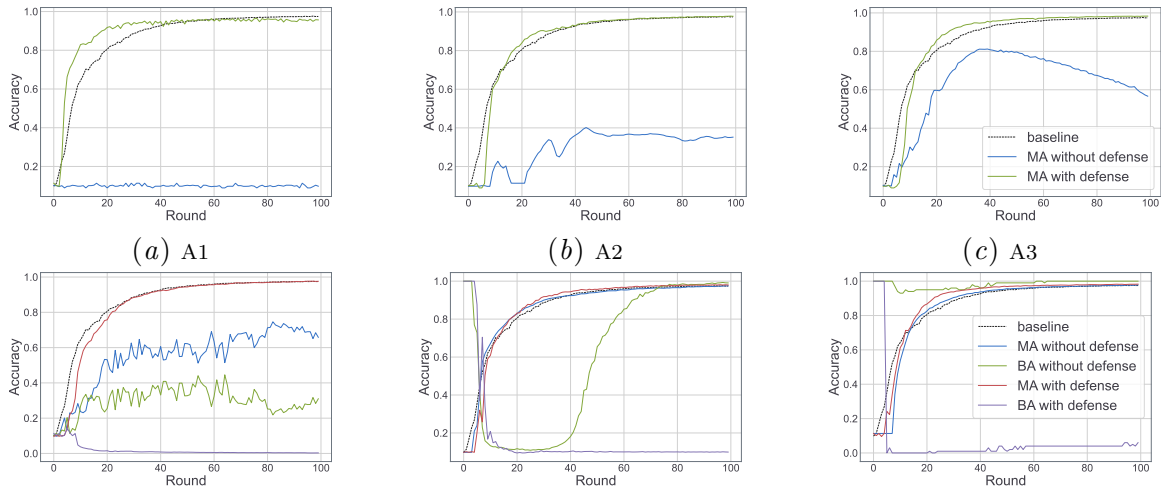
Malicious Configuration. Both model poisoning attacks and data poisoning attacks share a parameter Poisoned Model Rate (PMR), indicating the fraction of poisoned models for each round. If the attacking type is data poisoning, it has one more parameter Poisoned Data Rate (PDR), representing the poisoned ratio of the data. Common attacking settings are shown in table 1, where settings are nearly marginal thresholds, below which the attacking effect is non-significant even if it escapes from being detected.

3. https://github.com/chenyaofo/pytorch-cifar-models

Table 1: Attack settings

Attacks	(E)MNIST	CIFAR-10
A1	mean = 0, std = 1	
A2	Krum's $\epsilon = 10^{-3}$, threshold = 2×10^{-2}	
A5	a 5x5 white square is inserted into the targeted data and labeling it as "0"	the semantic pattern is a car with stripes and labeling "car" to "bird"
A6	by adding Ardis_IV to training and labeling "7" as "1"	by adding Southwest Airline images to training and labeling "airplane" as "truck"

4.2. Fending-off Byzantine Attacks



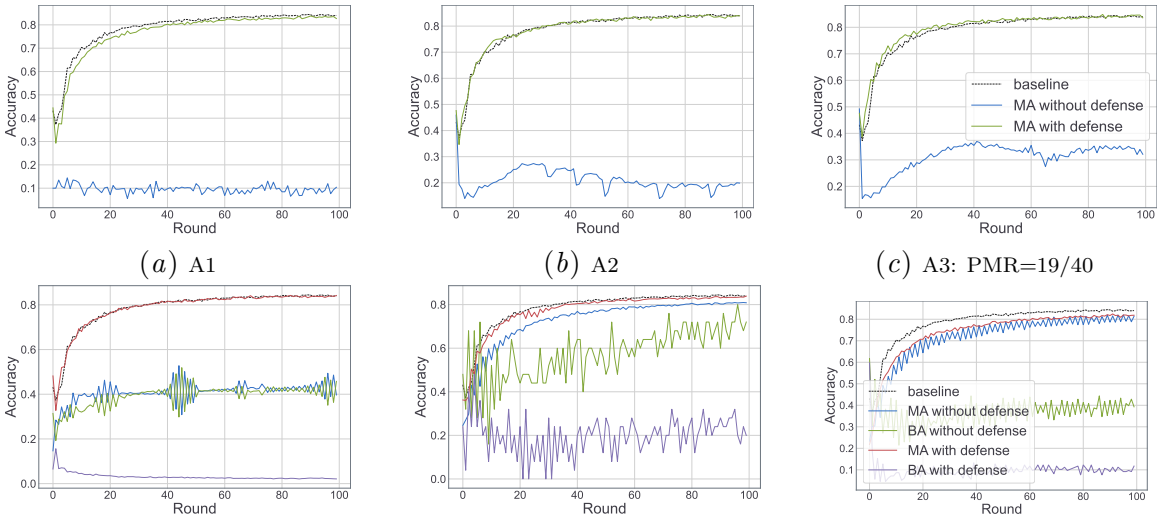
(a) A1 (b) A2 (c) A3

(d) A4: PDR=1.0 (e) A5: PDR=0.2 (f) A6: PDR=0.33

Figure 2: Byzantine attacks on MNIST

Fig. 2 shows MNIST under different attacks from A1 to A6. "Baseline" or "without defense" results from the server running FedAvg. 21 clients are uniformly randomly selected from 100 participants for the baseline, while 40 clients are for other situations (i.e. PMR=19/40). In Fig. 2(*a*), due to malicious random uploads, the aggregated updates are meaningless, leading to the blue curve with extremely low accuracy; however, the filtering process conducts so effectively that the learning curve - the green one - can behave normally under this attack. In Fig. 2(*b*), the Krum attack tries its best to upload counter-directional updates to devalue global accuracy. Consequently, the global accuracy is even worse than a random guess (50%). However, the global accuracy can reach an original level using the defense of FLVoogd. In Fig. 2(*c*), the trimmed-mean attack starts to degrade the model accuracy approximately at midterm and reduces accuracy from higher than 80% to less than 60% within 50 rounds. FLVoogd prevents this malicious reduction of accuracy effectively. In Fig. 2(*d*), Byzantine clients flip the labels of all training samples, rendering the final prediction like a random guess. One notable point is that the filter cannot correctly recognize flipping uploads in several initial rounds, but it can acknowledge and expel malicious updates once the global model learns a little from those majorities. In Fig. 2(*e*), the backdoor triggering attack shows its supremacy at the initial stage, and BA can easily approximate to 100% in the first several rounds. After the model learns sufficient benign

samples, BA tends to decrease while MA tends to increase. Without the defense, BA again grows sharply at midterm and finally attains relatively high accuracy. In contrast, under the protection, BA is restrained at low accuracy and can impossibly revive after the filter starts to work. The filter does not work at the initial stage because the model is chaotic, and the updates produced from the model reveal little information about the data distribution. In Fig. 2(f), under no defense, BA manifests likewise A5 at the initial stage but does not decline after MA rises. Since the malicious data is sampled from another dataset without intersection with MNIST, learning from the benign samples cannot benefit the model, so BA persists at high accuracy. However, FLVoogd can successfully detect and block these uploads according to the abnormal data distribution, preventing the intrusion of edge-case.



(a) A1 (b) A2 (c) A3: PMR=19/40

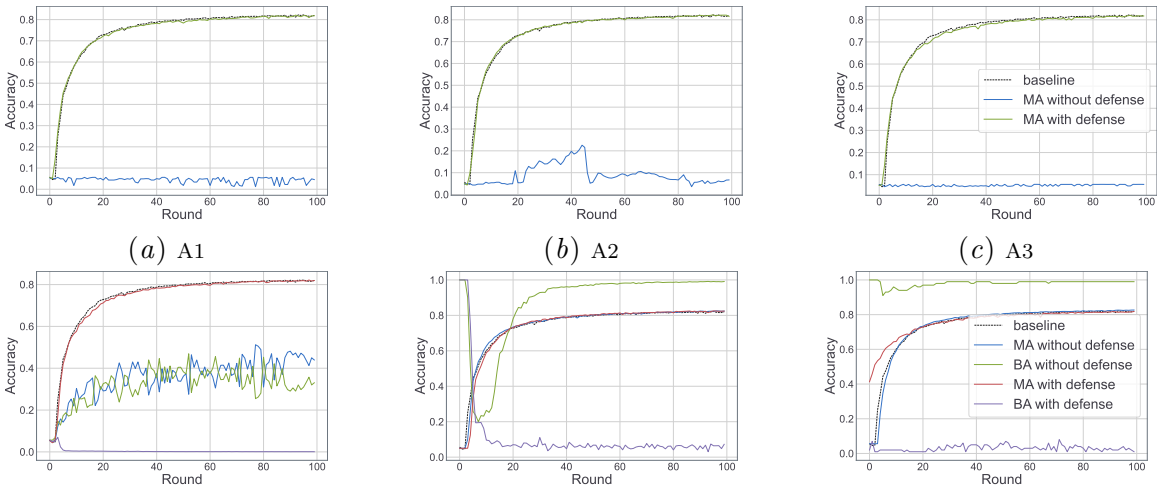
(d) A4: PDR=1.0 (e) A5: PDR=0.33 (f) A6: PDR=0.33

Figure 3: Byzantine attacks on CIFAR-10

Fig. 3 shows CIFAR-10 under different attacks from A1 to A6, where all the attacks have been eliminated or restrained below comparably low accuracy. Fig. 4 shows EMNIST under various attacks from A1 to A6. Similar to MNIST and CIFAR-10, all the attacks have been eliminated or restrained below comparably low accuracy. However, we initially found that FLVoogd could not prevent EMNIST from A6 productively since 1) we did not use the whole dataset for the training but followed the advice from He et al. (2020), where 20% was suggested for 100 clients; 2) there were 62 classes to be classified. Consequently, the model initially required several rounds to figure out what correct "7" and "1" roughly looked like. The model would not rebound those edge cases if edge-case clients instructed the model incorrectly with the mislabelled pictures at the beginning. Therefore, in the first five rounds, we put the model under training with benign-only samples to compensate for this unfairness. After that, the model could successfully filter out those malicious uploads.

4.3. Adding and Tracking DP

Experiments that test the DP effect and monitor the ϵ budget will be independently studied in this subsection. The experiments consist of different combinations of subsampling ratio

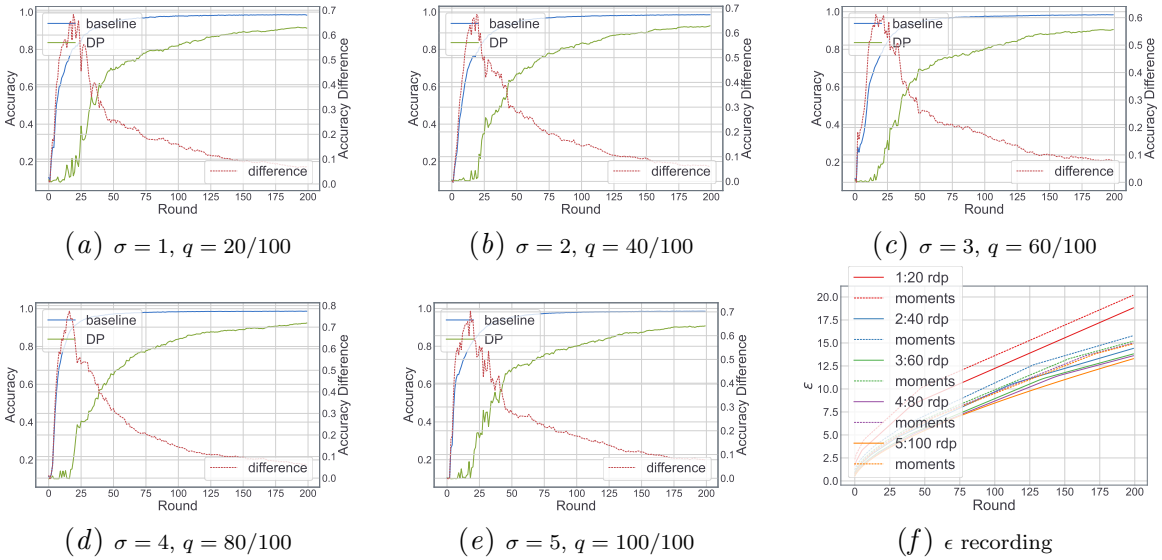


(a) A1 (b) A2 (c) A3

(d) A4: PDR=1.0 (e) A5: PDR=0.33 (f) A6: PDR=0.33

Figure 4: Byzantine attacks on EMNIST

q and noise strength coefficient σ. The DP noise, to some extent, will adversely affect the convergence and accuracy of the model. In return, this kind of sacrifice gains a differential privacy guarantee. The growth of ϵ after each iteration is estimated by Moments accountant or Rényi-DP (RDP) Wang et al. (2019), where δ is set as a constant ($= 10^{-3}$ Geyer et al. (2017)) considering 100 is the total number of clients.



(a) $\sigma = 1$, $q = 20/100$ (b) $\sigma = 2$, $q = 40/100$ (c) $\sigma = 3$, $q = 60/100$

(d) $\sigma = 4$, $q = 80/100$ (e) $\sigma = 5$, $q = 100/100$ (f) ϵ recording

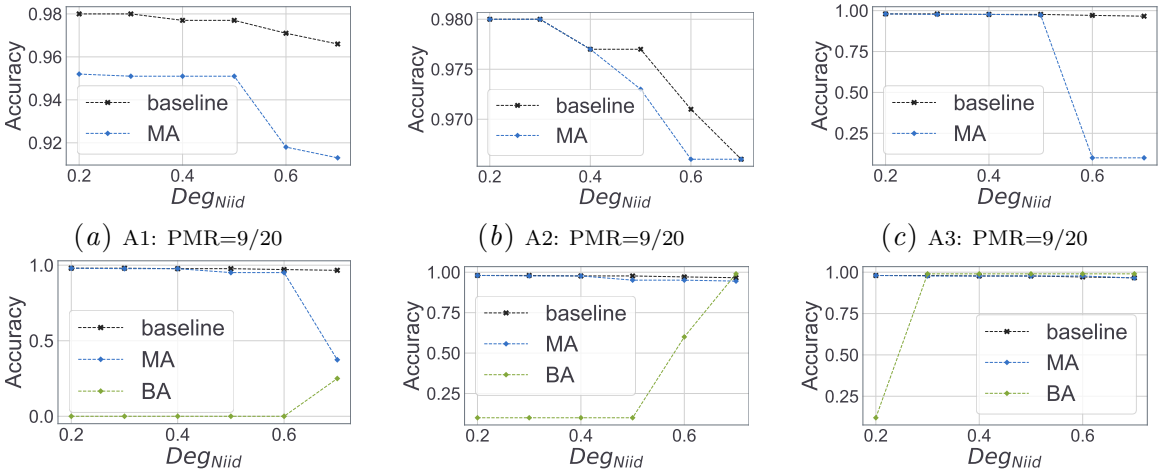
Figure 5: DP's effect on MNIST

The consequence of DP Gaussian noise is tested by MNIST and shown in Fig. 5. We kept the ratio of noise strength σ and the chosen number of clients for each round n constant, $\frac{\sigma}{n} = 1/20$. Each experiment used a different subsampling rate q from 0.2 to 1.0. The total number of training rounds extended to 200, as the noise postponed the convergence. Figures

nearly exhibit a similar trend. In general, the added noise decreased the final accuracy by 7.0%. In return, almost in all the experimental settings, (ϵ, δ) was better than $(20, 10^{-3})$, and the best one can achieve $(13.29, 10^{-3})$ estimated by RDP. Solid lines and dash lines in Fig. 5(f) are ϵ estimated by Moment's accountant and RDP, respectively, where RDP always provides a lower ϵ's boundary. The FLVoogd framework provides an adaptive supervisor for ϵ, which the server can customize - the values (ϵ, δ) are widely acceptable in reality - so the server can stop the training or adjust the sampling ratio and the amount of adding noise in time once ϵ is undesirable.

4.4. Non-IID Interference

Experiments will test Deg_{nIID} from 0.2 to 0.7. The number of clients per round is reduced from 40 to 20 since DBSCAN is unstable with noisy points where the non-IID noise may connect clusters. The defense may collapse if too many clients are selected per round under the non-IID setting.



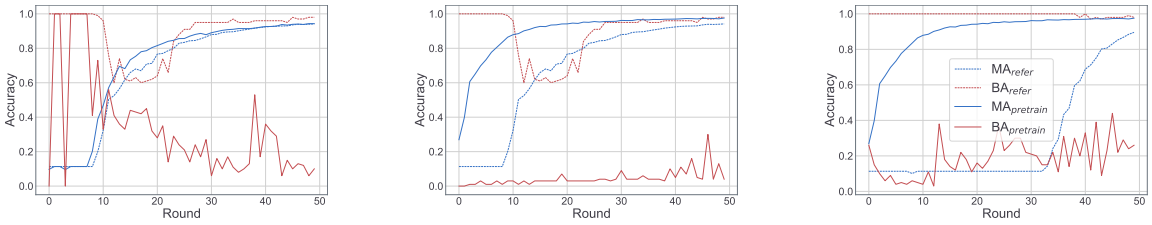
(a) A1: PMR=9/20 (b) A2: PMR=9/20 (c) A3: PMR=9/20

(d) A4: PMR=9/20, PDR=1.0 (e) A5: PMR=9/20, PDR=0.2 (f) A6: PMR=9/20, PDR=0.33

Figure 6: Non-IID effect on MNIST

The defensive effect under the non-IID setting has been tested on the MNIST dataset, shown in Fig. 6. In general, FLVoogd cannot ultimately tolerate that clients hold extreme non-IID samples. In Fig. 6(a), the filter rejects all these uploads until the model converges. After convergence, the filter hardly distinguishes between the malicious and model uploads because both appear somewhat random behaviors, resulting in fluctuations in the convergence state. However, the accuracy is still above 90% since benign uploads again become meaningful once below this threshold, and the filter once more rejects malicious random uploads. Krum attack shows ineffectual regardless of the non-IID degree in Fig. 6(b). The filter is so sensitive to the direction of uploads that uploads with reverse directions can scarcely pass the filtering. In Fig. 6(c), the filter relinquishes its duty after the non-IID degree is more extensive than 0.5. Compared to A1, A3 is a more advanced scheme, which chooses the randomness adaptively, so the attacking effect is more significant. In Fig. 6(d), random flipping works after the non-IID ratio is higher than 0.6. After $Deg_{nIID} > 0.5$, one

class completely dominates a dataset, leading to each mini-batch iteration containing over 50% samples from the same class. Consequently, the learning process is tampered with by the flipping of one class intermittently once the non-flipped samples of this class miss the training round. In Fig. 6(e), backdoor accuracy cannot be constrained if increasing the non-IID degree to more than 0.5, as the filter cannot discriminate whether the non-IID or the backdoor targets cause the directional difference. This situation similarly happens in Fig. 6(f) where the result is even worse because the defense collapses when the non-IID ratio is just higher than 0.2. Contradicting A5, where the backdoor targets are still the samples in the dataset, A6 introduces the backdoor targets from another dataset and aims to compromise the weakness of the model prediction. The filter performs ineptly if the model digest cannot reflect normal/abnormal directions. Since the model can never learn those edge cases with true labels, the model cannot provide evidence of deviant behaviors. When the non-IID ratio is lower than 0.3, the filter can detect those edge cases mainly because of the distribution of uploads. However, after non-IID increases, the upload lacks this kind of information.



(a) Deg_{nIID}=0.3: lowering PDR to 6/20=30%

(b) Deg_{nIID}=0.3: pretraining the model to accuracy around 25%

(c) Deg_{nIID}=0.5: pretraining the model to accuracy around 25%

Figure 7: Pretraining and a lower PDR help the defense

Since the FLVoogd performed worse when it suffered from A6's attack, we selected this situation for further study. We wanted to assist FLVoogd somewhat - training the model ahead or decreasing the PMR. As mentioned, pre-training is doable for some application scenarios. In addition, according to Shejwalkar et al. (2021), PMR$\approx$ 50% is a very pessimistic assumption, so we tried to lower it a little bit to see how our framework would react. In Fig. 7(a), PDR is reduced from 45% to 30%, and the BA learning curve declines once the model has learned the correct direction from the benign uploads. The poisoning effect is weakened because of the lower PDR. In Fig. 7(b) and Fig. 7(c), the model accuracy is trained approximately to 25% before the attacks deploy. The updating directions of models become consistent after the pre-train. Thus, the filter can sift those malicious uploads once it first time meets the upload in an abnormal direction. The results also verify that defending against targeted attacks depends on the performance of models on the dataset. If the model can separately recognize the poisoned and normal samples, it can output distinguishable model updates. Then, after the filter captures this variance, the defense effectively works.

5. Conclusion

We introduce FLVoogd, a robust and privacy-preserving federated learning framework that restrains the adverse impact of Byzantine attacks within an acceptable level while maintaining the performance of model predictions on the main task. There are two critical differences between our design and prior works. Firstly, most procedures are executed under privacy preservation, where operations are doable for mostly popular SMPC protocols. Secondly, we provide adaptive adjustments such that the whole process can run automatically. Future works could include: merging the transfer learning into the current framework to tackle GAN inference and combine it with other efficiently communicative schemes, e.g., sketch, to reduce the communication bandwidth and enhance differential privacy.

Acknowledgments

This research is supported by European Union's Horizon 2020 research and innovation programme under grant agreement No. 952697 (ASSURED), No. 101021727 (IRIS), and No. 101070052 (TANGO).

References

Yoshinori Aono, Takuya Hayashi, et al. Privacy-preserving deep learning via additively homomorphic encryption. *IEEE TIFS*, pages 1333–1345, 2017.

Xiaoyu Cao, Minghong Fang, et al. Fltrust: Byzantine-robust federated learning via trust bootstrapping. *arXiv preprint arXiv:2012.13995*, 2020.

Ittai Dayan, Holger R Roth, et al. Federated learning for predicting clinical outcomes in patients with covid-19. *Nature medicine*, pages 1735–1743, 2021.

Martin Ester, Hans-Peter Kriegel, et al. A density-based algorithm for discovering clusters in large spatial databases with noise. In *kdd*, pages 226–231, 1996.

Minghong Fang, Xiaoyu Cao, et al. Local model poisoning attacks to {Byzantine-Robust} federated learning. In *USENIX Security*, pages 1605–1622, 2020.

Robin C Geyer, Tassilo Klein, et al. Differentially private federated learning: A client level perspective. *arXiv preprint arXiv:1712.07557*, 2017.

Tianyu Gu, Brendan Dolan-Gavitt, et al. Badnets: Identifying vulnerabilities in the machine learning model supply chain. *arXiv preprint arXiv:1708.06733*, 2017.

Chaoyang He, Songze Li, et al. Fedml: A research library and benchmark for federated machine learning. *arXiv preprint arXiv:2007.13518*, 2020.

Kaiming He, Xiangyu Zhang, et al. Deep residual learning for image recognition. In *CVPR*, pages 770–778, 2016.

Brian Knott, Shobha Venkataraman, et al. Crypten: Secure multi-party computation meets machine learning. *NIPS*, pages 4961–4973, 2021.

Qinbin Li, Bingsheng He, et al. Model-contrastive federated learning. In *CVPR*, pages 10713–10722, 2021a.

Tian Li, Shengyuan Hu, et al. Ditto: Fair and robust federated learning through personalization. In *ICML*, pages 6357–6368, 2021b.

Ming Liu, Stella Ho, Mengqi Wang, et al. Federated learning meets natural language processing: A survey. *arXiv preprint arXiv:2107.12603*, 2021.

Guodong Long, Yue Tan, et al. Federated learning for open banking. In *Federated learning*, pages 240–254. 2020.

H Brendan McMahan, Eider Moore, et al. Federated learning of deep networks using model averaging. *arXiv preprint arXiv:1602.05629*, 2016.

Luis Muñoz-González, Battista Biggio, et al. Towards poisoning of deep learning algorithms with back-gradient optimization. In *AISec*, pages 27–38, 2017.

Thien Duc Nguyen, Phillip Rieger, et al. Flguard: secure and private federated learning. *arXiv preprint arXiv:2101.02281*, 2021a.

Thien Duc Nguyen, Phillip Rieger, et al. Flame: Taming backdoors in federated learning. *Cryptology ePrint Archive*, 2021b.

OpenMined. Sympc: A smpc companion library for syft. *GitHub repository*, 2021. URL https://github.com/OpenMined/SyMPC.

Phillip Rieger, Thien Duc Nguyen, et al. Deepsight: Mitigating backdoor attacks in federated learning through deep model inspection. *arXiv preprint arXiv:2201.00763*, 2022.

Virat Shejwalkar, Amir Houmansadr, et al. Back to the drawing board: A critical evaluation of poisoning attacks on production federated learning. *arXiv preprint arXiv:2108.10241*, 2021.

Sameer Wagh, Shruti Tople, et al. Falcon: Honest-majority maliciously secure framework for private deep learning. *arXiv preprint arXiv:2004.02229*, 2020.

Hongyi Wang, Kartik Sreenivasan, et al. Attack of the tails: Yes, you really can backdoor federated learning. *NIPS*, pages 16070–16084, 2020.

Yu-Xiang Wang, Borja Balle, et al. Subsampled rényi differential privacy and analytical moments accountant. In *AISTATS*, pages 1226–1235, 2019.

Dong Yin, Yudong Chen, et al. Byzantine-robust distributed learning: Towards optimal statistical rates. In *ICML*, pages 5650–5659, 2018.

Xianglong Zhang and Xinjian Luo. Exploiting defenses against gan-based feature inference attacks in federated learning. *arXiv preprint arXiv:2004.12571*, 2020.