Robust Federated Learning with Differential Privacy

Research Internship

Tingxin Yang

July 2024

1 Introduction

In recent years, artificial intelligence has garnered increasing attention. At the core of AI lies machine learning, which has been widely applied across various aspects of our lives, including finance, healthcare, autonomous driving, online shopping, and job searching. While machine learning has brought significant convenience to our daily lives, its privacy implications are becoming a more pressing concern. Traditional machine learning models require servers to collect users' data for centralized training, and such data often contains sensitive personal information. For example, job application data might include details such as height, age, educational background, and address, while shopping data might contain account credentials, personal preferences, browsing history, and even medical information like health records. If these private details are leaked during data collection, they could threaten users' personal and financial security.

As societies become more diverse, the importance of protecting personal privacy has drawn increasing attention. People want their privacy safeguarded, and ideally, they would like to enjoy the benefits of machine learning without having to upload their private data. However, this contradicts the traditional centralized machine learning approach, where data is collected and aggregated on a central server for training. To address this contradiction, in 2017, Google introduced the concept of Federated Learning [1]. Federated learning is a decentralized machine learning framework where multiple data holders (such as smartphones and IoT devices) collaboratively train a model without sharing raw data. Instead, the training process involves exchanging only intermediate parameters. In an ideal scenario, the shared model produced by federated learning would have performance comparable to a model trained with centrally aggregated data[2].

Although federated learning prevents direct sharing of users' data with servers, there are still risks of privacy leakage. This is because federated learning requires a significant exchange of intermediate parameters during the training phase, which could expose raw data to all participating parties, thus leading to potential privacy breaches. For instance, research has shown that it is possible to reconstruct parts of the original data from gradients[3], or infer whether a specific participant's record is included in the data based on the exchanged intermediate parameters[4].

Moreover, unreliable participants exacerbate the risk of privacy leakage. In federated learning, the shared information between the server and participants is limited to model parameters. However, this setup is vulnerable to model poisoning attacks, as participants can modify the model parameters[5]. In Figure 1, participants can modify w_1 to a huge value to influence the global model w'.

Thus, it is evident that federated learning, without adequate protections, still carries risks of data leakage for participants, potentially causing them harm. In this paper, we first analyze the aggregation technique FedAvg [1]and then introduce the robust aggregation technique KRUM[6], and evaluate their robustness against Byzantine attacks[7]. We then introduce differential privacy[8] to enhance privacy protection further.

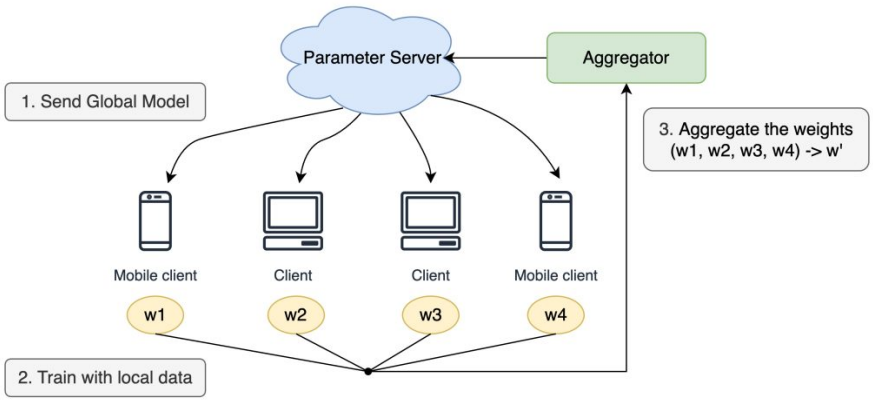

Figure 1: The framework of federated learning [9]

2 Optimization and Aggregation Algorithms

We often notice that smartphones can automatically categorize different people when we want to search for photos in our albums, helping us find pictures with friends. These features are powered by machine learning, which addresses an image classification problem. However, we do not want our private data uploaded to a server. So this paper focuses on using federated learning to achieve privacy protection in image classification tasks. Because federated learning is a distributed machine learning method, it shares a focus with machine learning on minimizing the objective function, such as:

$$\min_{w\in\mathbb{R}^d} f(w) \quad \text{where} \quad f(w) \triangleq \frac{1}{n}\sum_{i=1}^{n} f_i(w)$$

$f_i(w) = \ell(x_i, y_i; w)$ is the loss function for prediction on the sample (x_i, y_i), where x_i is the data, y_i is the label, and w is the model parameters. We assume a total of K clients (smartphones), where P_k is the index set of data points (image indices) on client k, and $n_k = |P_k|$ represents the number of data points (number of images). Therefore, we can rewrite the objective function as follows

$$f(w) = \sum_{k=1}^{K} \frac{n_k}{n} F_k(w) \quad \text{where} \quad F_k(w) = \frac{1}{n_k}\sum_{i\in P_k} f_i(w)$$

In addition, for convenience, we assume that our data is IID, meaning that P_k is formed by uniformly distributing the training samples randomly across the clients. Therefore, $E_{P_k}[F_k(w)] = f(w)$, where the expectation is taken over the set of examples assigned to the fixed client k.

2.1 The FedSGD Algorithm

Most current deep learning methods use stochastic gradient descent (SGD) for optimization, so we also adopt the SGD approach in our federated learning framework. Therefore, we can compute a single batch of gradients on randomly selected clients during each communication round, improving computational efficiency. At the same time, in federated learning, when involving more clients, we use large-batch synchronous SGD, as experiments by Chen et al[10]. have shown that this method performs excellently in data center environments, surpassing asynchronous methods. To apply this method in federated learning, we initialize the global model parameters w_0, and then in the t round, the server broadcasts w_t to all workers. In the Worker, after receiving w_0 from the server, a fraction F of clients is randomly selected from the K clients to compute the gradient of the loss function g_k and upload it to the server. When all workers are honest, the central server can average these gradients and update the global parameters to $w_{t+1} \leftarrow w_t - \eta \nabla f(w_t)$, then proceed to the next round until completion. $g_k = \nabla F_k(w_t)$ represents the gradient of the local data on the current model w_t, where t denotes the current iteration, and k denotes the client index. η is the learning rate. F is the proportion of clients participating in training in each round, where $F = 1$ corresponds to full-batch (non-stochastic) gradient descent. This algorithm is referred to as FedSGD [1].

Algorithm 1: Distributed Synchronous SGD[7]

Server
  Input: Initialization w_0
  for $t = 0, \ldots, T$ **do**
    Broadcast w_t to all the workers;
    Wait until all the gradients $\{g_k; k \in [K]\}$ arrive;
    Compute $\nabla f(w_t) = \sum_{k=1}^{K} \frac{n_k}{n} g_k$
    Update the parameter $w_{t+1} \leftarrow w_t - \eta \nabla f(w_t)$
Worker $i = 1, \ldots, K$
  for $t = 0, \ldots, T$ **do**
    Receive w_t from the server;
    Draw the samples, compute, and send the gradient $g_k = \nabla F_k(w_t)$ to the server;

2.2 The FedAvg Algorithm

The FedAvg algorithm is derived from a modification of the FedSGD algorithm. In the FedSGD algorithm, the model parameters on the server are updated according to the formula $w_{t+1} \leftarrow w_t - \eta \sum_{k=1}^{K} \frac{n_k}{n} g_k$, where the uploaded parameter g_k is the gradient of the loss function. To improve communication efficiency, we can rewrite the formula into a form that takes the weighted average of the local models w^k. For a client with n_k

local examples, the number of local updates per round is given by $u_k = E \cdot \frac{n_k}{B}$, where B denotes the local minibatch size, E represent the number of local epochs. For all clients k, each client updates its local model parameters as $w_{t+1}^k \leftarrow w_t - \eta g_k$. The central server updates the global model by performing a weighted average of the obtained local models, $w_{t+1} \leftarrow \sum_{k=1}^{K} \frac{n_k}{n} w_{t+1}^k$. We can increase the computation for each client by iterating the local updates $w^k \leftarrow w^k - \eta \nabla F_k(w^k)$ multiple times before the aggregation step. We call this method FedAvg.

In FedAvg, clients perform E rounds of gradient descent training on their local data and upload the model parameters w^k to the server. The server updates the global model w_{t+1} by calculating the average of the model parameters from each client[1].

Algorithm 2: FederatedAveraging [1]

Server executes:
  initialize w_0;
  for each round $t = 1, 2, \ldots$ **do**
    $m \leftarrow \max(F \cdot K, 1)$;
    $S_t \leftarrow$ (random set of m clients);
    for each client $k \in S_t$ **in parallel do**
      $w_{t+1}^k \leftarrow \text{ClientUpdate}(k, w_t)$;
    $m_t \leftarrow \sum_{k \in S_t} n_k$;
    $w_{t+1} \leftarrow \sum_{k \in S_t} \frac{n_k}{m_t} w_{t+1}^k$
ClientUpdate(k, w): // **Run on client** k
  for each local epoch i from 1 to E **do**
    $w^k \leftarrow w^k - \eta \nabla F_k(w^k)$
  return w^k to server;

3 Attack

To protect the confidentiality of training data, the information shared between the server and participants in federated learning is limited to model parameters. However, this setup is vulnerable to model poisoning attacks, as participants can modify the model parameters. Model attacks alter the global model by tampering with or replacing the client's model parameters. A diagram of a model attack is shown in Figure 2, where the adversary uploads malicious (red) model parameters to the server, altering the aggregated global model. The server then broadcasts the incorrect global model to all local clients, achieving the attacker's goal. In federated learning, the information typically related to model updates is the model gradients. Specifically, the attacker influences the global model by attacking certain members in federated learning, altering the gradients of the attacked members or sending incorrect information during the model update process. This causes the direction of the global model to deviate as much as possible from

its original direction, achieving the intended attack effect.

In most modern SGD-based learning algorithms, a common aggregation rule is to compute the average of the vectors uploaded by the clients. However, Lemma 1 [6] below points out that a linear combination of vectors cannot tolerate attacks from a single Byzantine user. Therefore, if there is a vector that is significantly larger compared to the others, it can adversely affect the aggregation result. Similarly, in the FedAvg algorithm, if one of the client's model parameters w becomes excessively large, it can also lead to a failure in aggregation. To achieve Byzantine tolerance, the KRUM [6]algorithm was proposed.

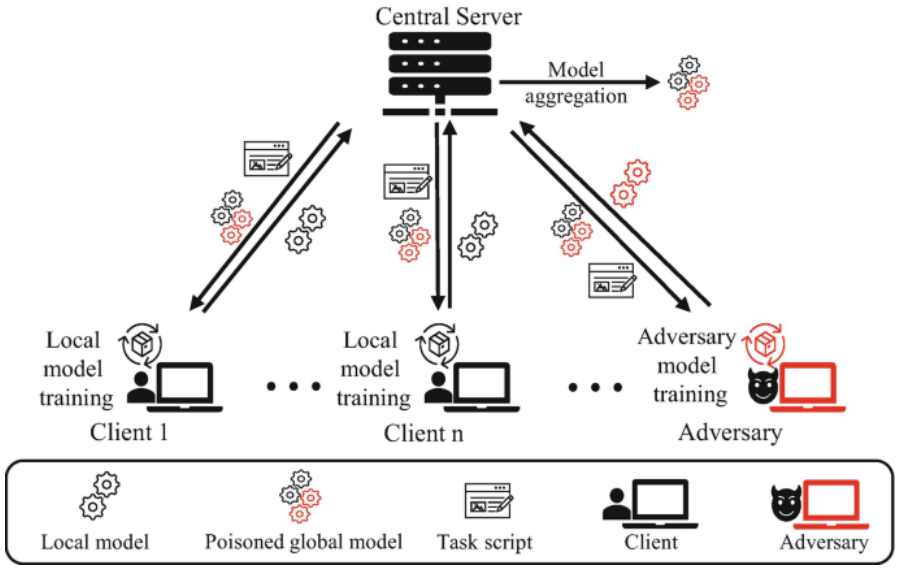

Figure 2: The architecture of model poising attack in federated learning [11]

Lemma 1. *[6] Consider an aggregation rule F_{lin} of the form*

$$F_{lin}(w_1, \ldots, w_K) = \sum_{k=1}^{K} \alpha_i w_i$$

where the α_i are non-zero scalars. Let U be any vector in $\mathbb{R}^d$. A single Byzantine worker can make F_{lin} always select U. In particular, a single Byzantine worker can prevent convergence.

Proof. Immediate: if the Byzantine worker proposes

$$w_K = \frac{1}{\alpha_K} U - \frac{1}{\alpha_K} \sum_{k=1}^{K-1} \alpha_i w_i$$

then $F = U$. □

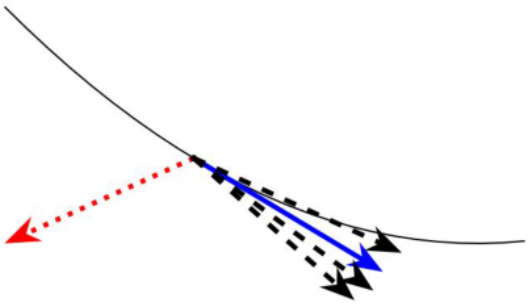

Figure 3: The gradient estimates computed by correct workers (black dashed arrows) are distributed around the actual gradient (solid arrow) of the cost function (thin black curve). A Byzantine worker can propose an arbitrary vector (red dotted arrow). [6]

3.1 KRUM aggregation rule

We consider a distributed system model consisting of a parameter server and K workers($C = 1$), b of which are Byzantine (behaving arbitrarily). Computation is divided into (many rounds of) synchronous rounds. During round t, the parameter server broadcasts its parameter vector $\mathbf{w}_t \in \mathbb{R}^d$ to all workers. Each correct worker h computes an estimate $\mathbf{g}_k^t = \nabla f(\mathbf{w}_t)$,where $f(\mathbf{w}_t)$ is the cost function. A Byzantine worker b proposes a vector $\mathbf{g}_b^t$, which can arbitrarily deviate from the vector it is supposed to send. The parameter server updates the parameter vector using the following stochastic gradient descent (SGD) equation [6]:

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \eta KR(\mathbf{g}_1^t, \ldots, \mathbf{g}_K^t)$$

where $KR(\mathbf{g}_1^t, \ldots, \mathbf{g}_K^t)$ denotes the Krum aggregation rule of the parameter vectors. For any $i \neq j$, we denote by $i \to j$ the fact that $\mathbf{g}_j$ belongs to the $K - b - 2$ closest vectors to $\mathbf{g}_i$. Then, we define for each worker i, the score

$$s(i) = \sum_{j \to i} \|\mathbf{g}_i - \mathbf{g}_j\|^2$$

where the sum runs over the $K - b - 2$ closest vectors to $\mathbf{g}_i$. Finally,

$$\mathrm{KR}(\mathbf{g}_1, \ldots, \mathbf{g}_K) = \mathbf{g}_i^*$$

where i^* refers to the worker minimizing the score, i.e., $s(i^*) \leq s(j)$ for all j.

3.2 Model poisoning attack on Krum

According to the definition of Krum, we know that Krum has Byzantine tolerance in distributed synchronous SGD. However, we consider an attack in a specific scenario where this Byzantine tolerance may be compromised [7].

We consider the worst case where $K = 2b + 3$. The server receives $(K - b)$ correct gradients $\mathbf{g} = \{\mathbf{g}_1, \ldots, \mathbf{g}_{K-b}\}$ and b Byzantine gradients $\mathbf{u} = \{\mathbf{u}_1, \ldots, \mathbf{u}_b\}$. We assume that the stochastic gradients have identical expectation $E[\mathbf{g}_i] = \mathbf{v},\ \forall i \in [K - b]$. We define the mean of the correct gradients as:

$$\bar{\mathbf{g}} = \frac{1}{K-b} \sum_{i \in [K-b]} \mathbf{g}_i$$

We assume that the correct gradients are bounded by

$$\|\mathbf{g}_i - \bar{\mathbf{g}}\|^2 \leq \|\bar{\mathbf{g}}\|^2,\ \forall i \in [K - b]$$

Furthermore, we assume that

$$\exists \beta \text{ such that } \|\mathbf{g}_i - \mathbf{g}_j\|^2 \geq \beta^2\ ,\ \forall i \neq j \in [K - b]$$

We take $\mathbf{u}_1 = \mathbf{u}_2 = \ldots = \mathbf{u}_b = -\epsilon\ \bar{\mathbf{g}}$, where ϵ is a small positive constant value such that $\epsilon^2 \|\bar{\mathbf{g}}^2\| \leq \beta^2$. When $(K - b)$ is large enough:

$$K - b > \frac{2(\epsilon + 2)^2}{\epsilon^2}$$

After derivation, we obtain:

$$\mathrm{KR}(\mathbf{u}) \leq 2(\epsilon + 2)^2 \|\bar{\mathbf{g}}\|^2 < (K - b - 2)\epsilon^2 \|\bar{\mathbf{g}}\|^2 \leq \mathrm{KR}(\mathbf{g})$$

When ϵ is small, we have:

$$\langle \mathbf{g}, E[\mathrm{KR}(\mathbf{g} \cup \mathbf{u})] \rangle < 0$$

Note that, in practice, the attackers only need to assign $\bar{\mathbf{g}} = \frac{1}{K-b} \sum_{i \in [K-b]} \mathbf{g}_i$ to all the Byzantine gradients, with $\epsilon > 0$ being small enough. Figure 4 illustrates the working principle of KRUM. It computes the sum of distances from each gradient g to the other gradients g and selects the g that is closest to the others. As shown in the figure, the red gradient g_5 is the one selected by KRUM to update the global parameters.

Figure 5 presents a simplified model of Byzantine model attacks, where the blue color represents the correct gradient after aggregation $\mathbf{g}_k$, and the red color represents the gradient from Xie's attack[7]. When we use the Krum aggregation, it is easy to observe that $KR(\mathbf{u})$ has the minimum score, meaning it has the smallest sum of distances to all $\mathbf{g}_k$. This situation allows the attack to be successful.

4 Differential Privacy in Federated Learning

In this section, we will introduce the definition of differential privacy, provide an overview of the differential privacy SGD algorithm in deep learning [8], and discuss its application in federated learning.

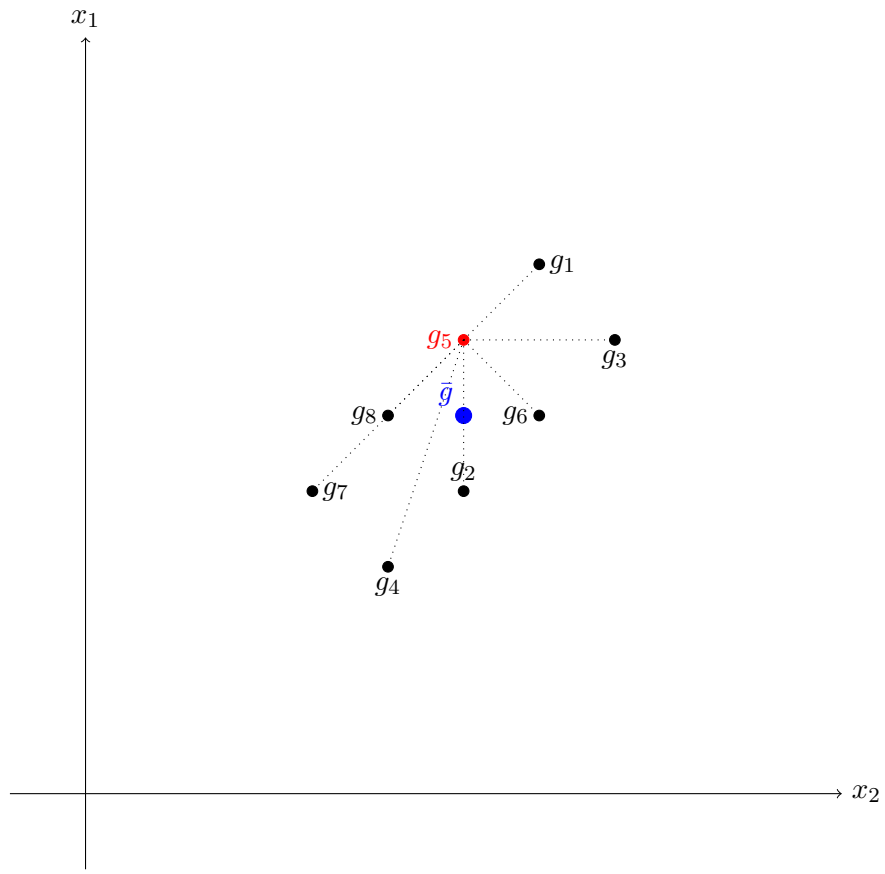

Figure 4: Simplified model of KRUM algorithm, $\bar{g}$ is the mean of all g_i

4.1 Differential Privacy

Today, mobile payments, electronic prescriptions, and electronic tickets have become integral to our lives. However, the vast amounts of personal privacy data pose a greater risk of privacy breaches. To address this issue, many countries and regions have enacted laws and regulations to protect personal privacy information. However, simply anonymizing personal information does not effectively safeguard privacy. In 2008, two researchers from the University of Texas detailed how privacy attacks could be carried out on data provided by Netflix[12]. They employed a linkage attack method: by comparing information from other public databases (such as IMDB) with the information provided by Netflix (such as favorite movies and ratings), they could partially infer the identity of a specific anonymous Netflix user on IMDB.

Similarly, in federated learning, while this approach can protect local data from being uploaded to the server, workers still need to upload gradient information to the server, and it has been shown that gradient information can also leak sensitive information[13]. Therefore, finding ways to reduce the risk of data leakage while enhancing data usability

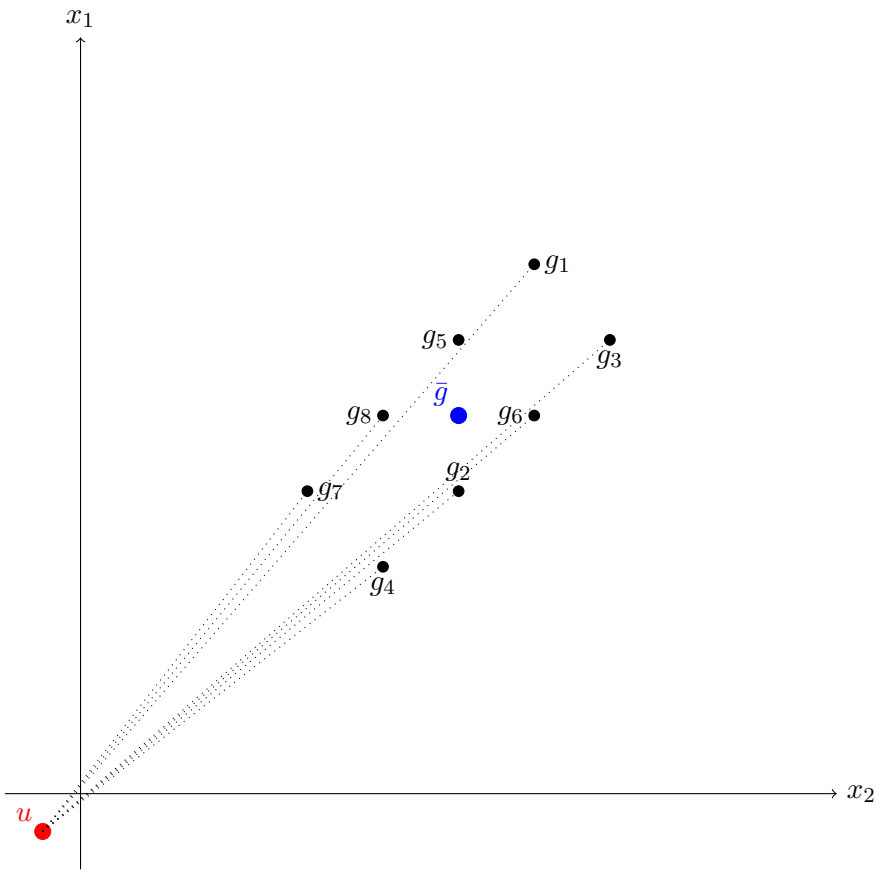

Figure 5: simplified model of Byzantine attacks, $\bar{g}$ is the mean of all g_i, u is the attacker

has become an important issue.

Differential privacy provides a solution to this problem by introducing a certain amount of random noise into the query results, making it impossible for an attacker, even if they possess external information, to determine whether a specific individual is present in the dataset. For example, suppose there are two databases, x and y: database x contains the weight data of 40 students in a class, while database y contains the same 40 students' weight data but also includes the teacher's weight. If we know the actual average weight from both databases, it would be easy to calculate the teacher's weight. This is an example of a differential attack. However, if we introduce another person's weight into database x, then even if we know the average weight from both databases, it would be impossible to deduce the teacher's weight. This is the purpose of the differential privacy algorithm.

Definition 1. A randomized mechanism $M : D \to R$ with domain D and range R satisfies (ϵ, δ)-differential privacy if for any two adjacent inputs $d, d' \in D$ and for any subset of outputs $S \subseteq R$, it holds that

$$\Pr[M(d) \in S] \leq e^{\epsilon} \Pr[M(d') \in S] + \delta.$$

In our experiment, each training dataset is a set of image-label pairs, where D represents the image data, S represents the labels, and M is the algorithm we use.

4.2 Differentially Private SGD Algorithm

Since differential privacy can better protect user data, we incorporated the differential privacy SGD algorithm from deep learning into federated learning. We applied the original algorithm to the gradient model g_k uploaded by each user, as shown in Algorithm 3.

We can add noise to the parameters uploaded by clients before aggregation to protect the privacy of the training data. However, we may not be able to provide a useful, tight characterization of the dependence between these parameters and the training data, and excessive noise could undermine the utility of the learned model. Therefore, we prefer a more sophisticated approach, where we aim to control the influence of the training data during the training process, particularly in the context of stochastic gradient descent (SGD) computations.

The differential privacy SGD algorithm trains a model by minimizing the empirical loss function. In each step of stochastic gradient descent (SGD), we compute the gradient $g_k = \nabla F_k(w_t)$ for a random subset of samples, clip the ℓ_2 norm of each gradient with parameter C, compute the average of these gradients, add gaussian noise with standard deviation σ to protect privacy, and finally update the model parameters [8]. We can extend the steps before the parameter update process to each client in federated learning, and then perform parameter updates through aggregation on the server. This approach achieves differential privacy on the client side.

Algorithm 3: Distributed Synchronous SGD [7] with Differentially private SGD[8]

Server
  Input: Initialization w_0
  for $t = 0, \ldots, T$ **do**
    Broadcast w_t to all the workers;
    Wait until all the gradients $\{\bar{g}_k; k \in [K]\}$ arrive
    Compute aggregation $\nabla f(w_t) = \sum_{k=1}^{K} \frac{n_k}{n} \bar{g}_k$
    Update the parameter $w_{t+1} \leftarrow w_t - \eta \nabla f(w_t)$
Worker $i = 1, \ldots, K$
  for $t = 0, \ldots, T$ **do**
    Receive w_t from the server;
    Draw the samples, compute $g_k = \nabla F_k(w_t)$
    Clip gradient $\bar{g}_k \leftarrow g_k / \max(1, \frac{\|g_k\|_2}{C})$
    Add noise $\bar{g}_k \leftarrow \sum_i (\bar{g}_k + \mathcal{N}(0, \sigma^2 C^2 \mathbb{I})$
    send $\bar{g}_k$ to the server

5 Results

This chapter primarily introduces the parameter settings for the FedAvg and FedSGD algorithms, the selection of the epsilon parameter in Xie's attack and the impact of different numbers of Byzantine workers on the results, as well as the choice and comparison of the sigma parameter in differential privacy algorithms. Finally, it discusses several issues encountered during the research process.

5.1 Implementation

We conducted experiments on the benchmark CIFAR-10 image classification dataset [7], which consists of 50,000 training images and 10,000 test images. We used a convolutional neural network (CNN) with 4 convolutional layers and 1 fully connected layer. In each experiment, the data was shuffled and divided into 100 clients (K = 100), with the data for each client being independently and identically distributed (IID). We launched 25 worker processes, with $C = 0.25$. We used top-1 accuracy on the test set and the cross-entropy loss function on the training set as evaluation metrics. For each worker process, the mini-batch size for SGD was set to 10.

When studying Byzantine tolerance, our focus was on exploring the performance of federated learning in image classification tasks and its robustness against attacks. We introduced Byzantine attacks at the 10th iteration, with the attack parameter ϵ typically set to 0.1, 1, and 10 for comparison. The goal was to investigate how the choice of different epsilon values impacts test accuracy and loss, in order to find an epsilon parameter that can withstand attacks and enhance the model's robustness.

When studying differential privacy, we set the clipping parameter C to 3, 5, and 10 to explore how different clipping parameters affect test accuracy and loss, aiming to identify the relationship between parameter C and accuracy and loss. Additionally, we compared the standard deviation σ of the added noise, ranging from 0 to 5, to investigate how the magnitude of the introduced noise affects model accuracy and loss, aiming to find the correlation between parameter σ and model accuracy and loss.

5.2 Experimental Results

In our study of the impact of the number of attackers on the KRUM algorithm, we chose to run $K = 25$ clients, with a maximum number of malicious clients set to $b = 11$, determined by $K - 2b = 3$. When we compared different numbers of Byzantine attackers in Figure 6, we observed that when the number of malicious workers $b = 1$, the accuracy gradually decreased while showing slight fluctuations. This indicates that even a small number of attackers can affect the model's accuracy, and the fluctuations are likely due to the limited impact of a single attacker on the overall model parameters.

As the number of attackers increased, we found that the accuracy declined more rapidly; for example, the curves for $b = 10$ and $b = 11$ dropped to around 10 percent accuracy by the time they reached 50 epochs. This demonstrates that the attackers have completely compromised the predictive capability of the original model.

When we compared Figure 7, we observed that the loss changes after the attacks were quite consistent across different numbers of attackers. This suggests that the number of attackers has little impact on the training loss. A possible reason for this is that the attackers initially exert a minimal influence on the overall model parameters. By epoch equal 10, the model had already achieved a high training accuracy, putting the loss in a convergent state. As more training rounds occurred, the attackers' influence gradually increased, leading to a rise in training loss and fluctuations in the loss curve. Furthermore, a greater number of attackers corresponded to a larger increase in training loss, indicating a proportional relationship.

In summary, introducing attackers once the training accuracy has begun to converge can be managed by stopping training early to maintain model accuracy. The number of attackers is proportional to their ability to degrade accuracy.

In our experiments, we used Xie's attack[7], where the choice of the parameter ϵ affects both accuracy and loss. As shown in Figure 8, when $\epsilon = 0.1$, the accuracy decreases the fastest, and the accuracy curve fluctuates around 10 percent. This indicates that the KRUM algorithm, when the model accuracy is low, is unable to consistently select the attackers' updates, leading to some honest users being selected for updates. When $\epsilon = 1$, as the epochs progress, the attack steadily degrades the model's accuracy, but the decline is slower, and it does not converge until epoch 30. This shows that a smaller ϵ ensures that the KRUM algorithm always selects the attackers' model parameters to update the global model.

However, when a larger $\epsilon = 10$ is chosen, the attack becomes completely ineffective and has no impact on the model. This is because the KRUM algorithm selects the gradient vector with the smallest sum of distances to other gradient vectors. When ϵ is large enough, the attacker's gradient vector is too far from the others, making it impossible for KRUM to select it. The loss curves in Figure 9 also support this conclusion. When $\epsilon = 1$, the training loss increases rapidly, and when the loss reaches around 8, the KRUM algorithm becomes unstable, fluctuating between selecting attackers and honest users.

Therefore, if we want the KRUM algorithm to always select the attackers, we should choose ϵ less than 1, which ensures the stability of the attack.

When we applied differential privacy SGD to protect the gradients uploaded by users, we found that the standard deviation σ of the added noise had a more noticeable impact on accuracy and training loss. As shown in Figure 10, as σ increases, the accuracy decreases, indicating that introducing noise leads to reduced accuracy. Compared to the curve without LDP (Local Differential Privacy), the accuracy when using differential privacy is lower, further confirming the impact of noise on accuracy. Therefore, we must balance privacy and model performance when introducing noise, selecting the largest possible σ to ensure maximum privacy while maintaining high accuracy.

Figure 11, which depicts the training loss, shows that larger values of σ lead to higher training losses, while smaller σ values result in losses closer to those without noise. This might be because the model loss had already converged before the attack was applied, so the attack's impact in a converged state is less noticeable.

Finally, we also compared the use of differential privacy algorithms before and after the attack to investigate the robustness of differential privacy against attacks. As shown

in Figure 12, we observed that the accuracy curve converged faster when differential privacy was applied after the attack. This may be because the attack disrupts the model parameters, and the noise introduced by the LDP algorithm further disturbs the parameters, leading to lower model accuracy. Figure 13 also demonstrates that applying differential privacy after the attack increases the model's loss, suggesting that we should introduce differential privacy earlier at the local level.

6 Conclusions

In this paper, we found that as the number of attackers b increases, the damage caused to the model parameters becomes stronger. Even when the accuracy of the trained model is close to convergence, applying an attack remains effective. This indicates that the KRUM algorithm cannot resist effective attacks. However, stopping the training early when the model accuracy is high can prevent further model degradation.

We should also choose a smaller attack parameter ϵ to ensure the stability and effectiveness of the attack, as a larger ϵ may lead to attack failure. Introducing a differential privacy algorithm before the attack can slow down the damage to the model parameters to some extent, but it cannot change the outcome where the attack reduces the test accuracy of the model.

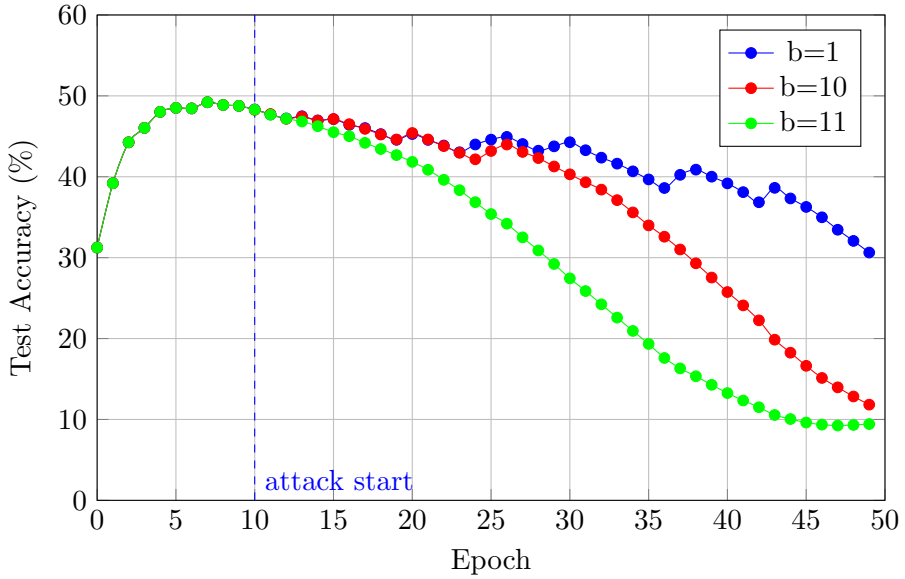

Figure 6: The accuracy comparison on the CIFAR dataset under the KRUM algorithm, with Byzantine attacks introduced at Epoch 10, where epsilon is set to 0.1, b represents the number of Byzantine attacks.

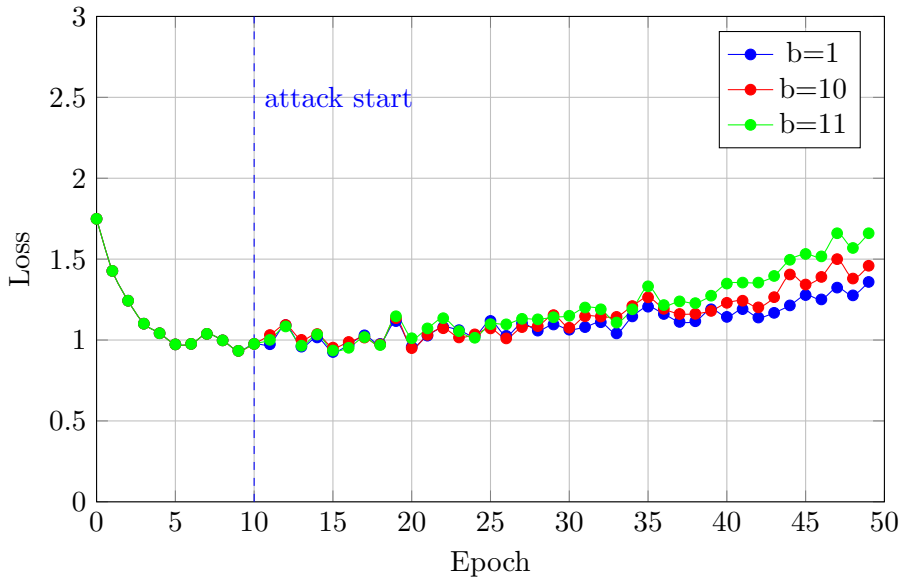

Figure 7: The Loss comparison on the CIFAR dataset under the KRUM algorithm, with Byzantine attacks introduced at Epoch 10, where epsilon is set to 0.1

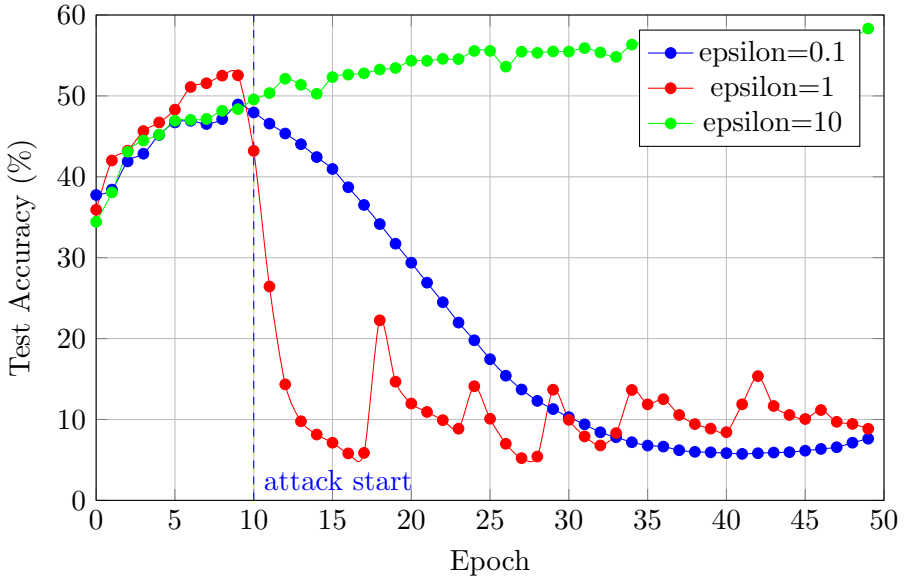

Figure 8: The accuracy comparison on the CIFAR dataset under the KRUM algorithm, with Byzantine attacks introduced at Epoch 10, where epsilon is set to 0.1,1,10.

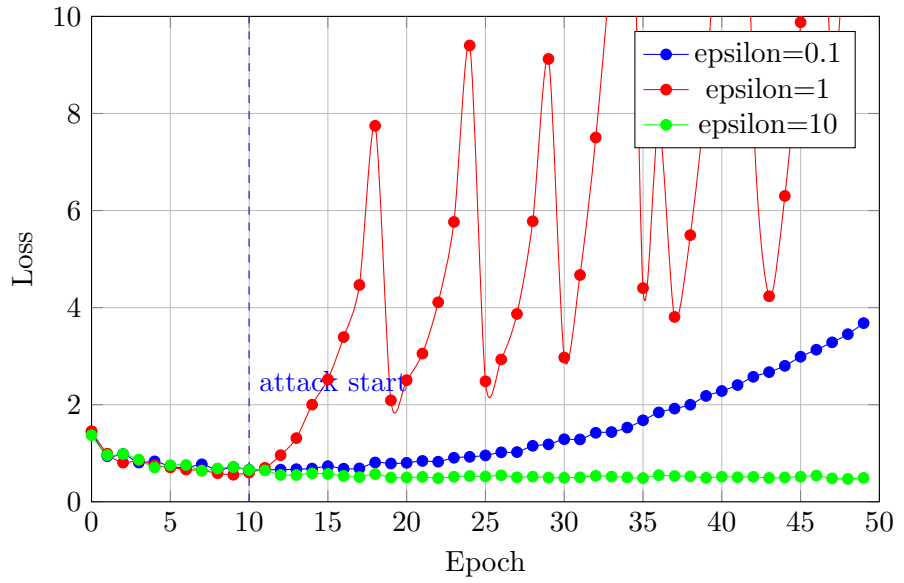

Figure 9: The Loss comparison on the CIFAR dataset under the KRUM algorithm, with Byzantine attacks introduced at Epoch 10, where epsilon is set to 0.1,1,10.

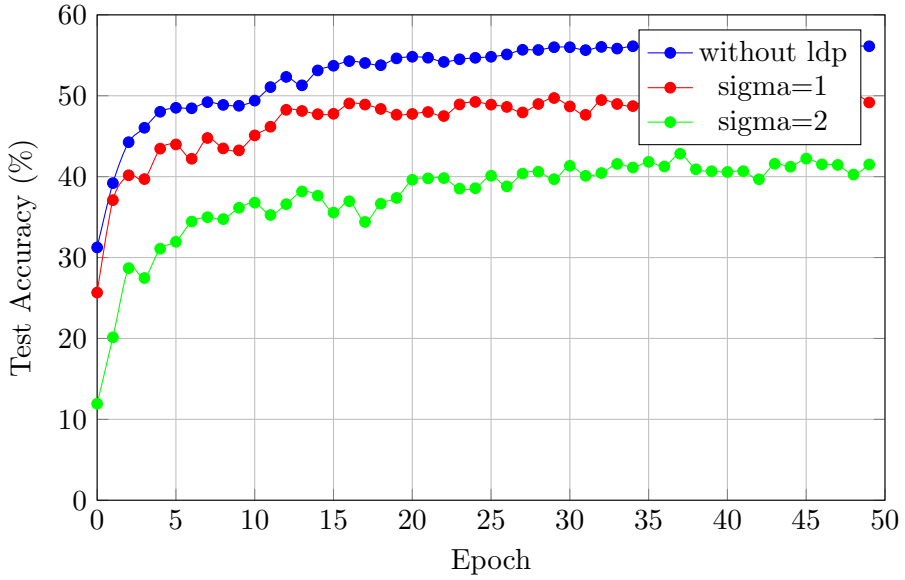

Figure 10: The accuracy comparison on the CIFAR dataset under the KRUM algorithm, with Byzantine attacks introduced at Epoch 10, where epsilon is set to 0.1, The blue line represents the results without applying the differential privacy algorithm.

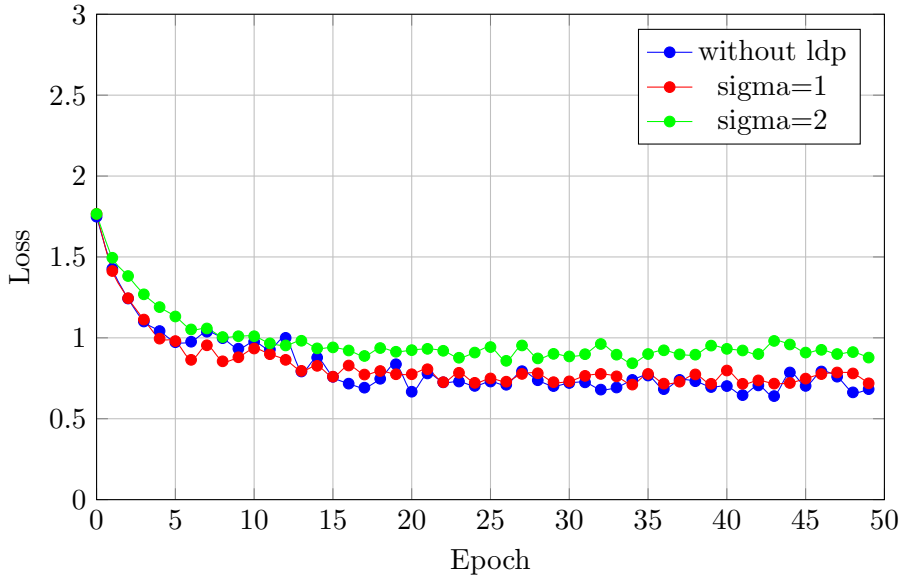

Figure 11: The Loss comparison on the CIFAR dataset under the KRUM algorithm, with Byzantine attacks introduced at Epoch 10, where epsilon is set to 0.1, The blue line represents the results without applying the differential privacy algorithm.

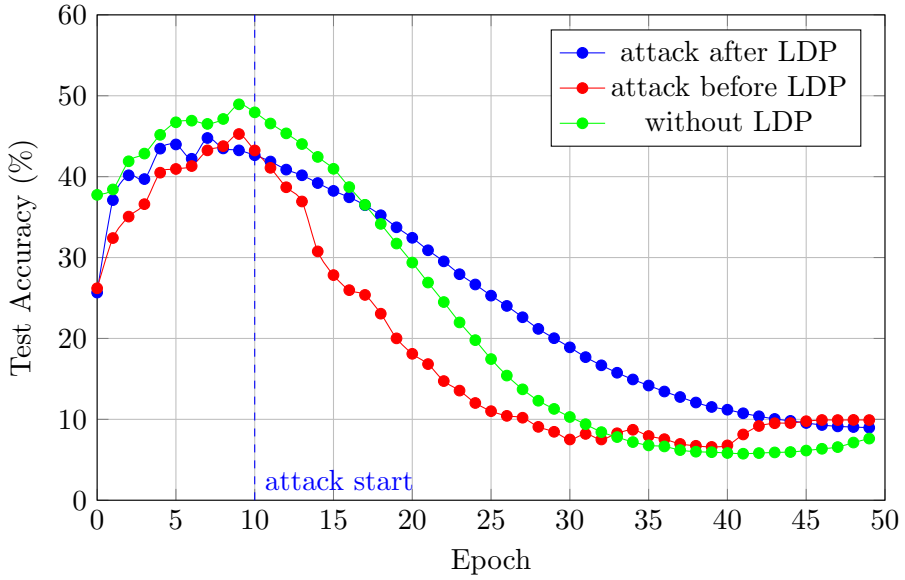

Figure 12: The accuracy comparison with Byzantine attacks before and after applying the differential privacy algorithm, using the CIFAR dataset and the KRUM aggregation algorithm, with Byzantine attacks introduced at Epoch 10, where epsilon is set to 0.1. In the differential privacy implementation, the standard deviation of the Gaussian noise is set to $\sigma = 1$ and the clipping coefficient C is set to 5.

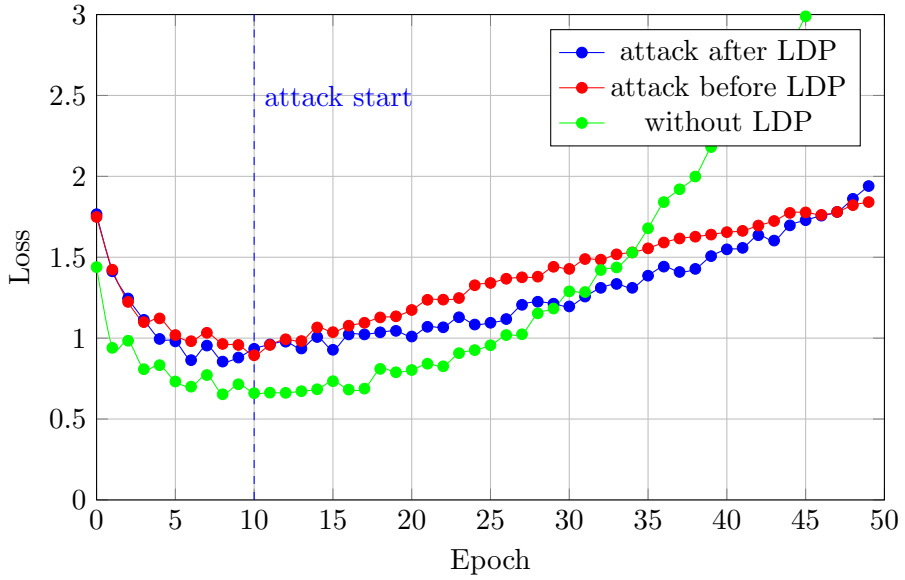

Figure 13: The Loss comparison with Byzantine attacks before and after applying the differential privacy algorithm

References

[1] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas, "Communication-efficient learning of deep networks from decentralized data," in Artificial intelligence and statistics, pp. 1273–1282, PMLR, 2017.

[2] T. Li, A. K. Sahu, A. Talwalkar, and V. Smith, "Federated learning: Challenges, methods, and future directions," IEEE signal processing magazine, vol. 37, no. 3, pp. 50–60, 2020.

[3] L. Zhu, Z. Liu, and S. Han, "Deep leakage from gradients," Advances in neural information processing systems, vol. 32, 2019.

[4] C. Song, T. Ristenpart, and V. Shmatikov, "Machine learning models that remember too much," in Proceedings of the 2017 ACM SIGSAC Conference on computer and communications security, pp. 587–601, 2017.

[5] X. Zhou, M. Xu, Y. Wu, and N. Zheng, "Deep model poisoning attack on federated learning," Future Internet, vol. 13, no. 3, 2021.

[6] P. Blanchard, E. M. El Mhamdi, R. Guerraoui, and J. Stainer, "Machine learning with adversaries: Byzantine tolerant gradient descent," Advances in neural information processing systems, vol. 30, 2017.

[7] C. Xie, O. Koyejo, and I. Gupta, "Fall of empires: Breaking byzantine-tolerant sgd by inner product manipulation," in Uncertainty in Artificial Intelligence, pp. 261–270, PMLR, 2020.

[8] M. Abadi, A. Chu, I. Goodfellow, H. B. McMahan, I. Mironov, K. Talwar, and L. Zhang, "Deep learning with differential privacy," in Proceedings of the 2016 ACM SIGSAC conference on computer and communications security, pp. 308–318, 2016.

[9] RoyC30ne, "Fed image," 2023. Accessed: 2024-10-10.

[10] J. Chen, X. Pan, R. Monga, S. Bengio, and R. Jozefowicz, "Revisiting distributed synchronous sgd," arXiv preprint arXiv:1604.00981, 2016.

[11] S. Yu and L. Cui, Poisoning Attacks and Counterattacks in Federated Learning, pp. 37–54. Singapore: Springer Nature Singapore, 2023.

[12] A. Narayanan and V. Shmatikov, "Robust de-anonymization of large sparse datasets," in 2008 IEEE Symposium on Security and Privacy (sp 2008), pp. 111–125, IEEE, 2008.

[13] L. Melis, C. Song, E. De Cristofaro, and V. Shmatikov, "Exploiting unintended feature leakage in collaborative learning," in 2019 IEEE symposium on security and privacy (SP), pp. 691–706, IEEE, 2019.