Approximate and Weighted Data Reconstruction Attack in Federated Learning

Yongcun Song, Ziqi Wang, and Enrique Zuazua

Abstract—Federated learning (FL) is a distributed learning paradigm that enables multiple clients to collaborate on building a machine learning model without sharing their private data. Although FL is considered privacy-preserved by design, recent data reconstruction attacks demonstrate that an attacker can recover clients' training data based on the parameters shared in FL. However, existing methods commonly fail to attack the most widely used federated averaging (FedAvg) scenario, where clients share model parameters after multiple local training steps. To tackle this issue, we propose an interpolation-based approximation method, which makes attacking FedAvg scenarios feasible by generating the intermediate model updates of the clients' local training processes. Then, we design a layer-wise weighted loss function to improve the quality of data reconstructions. We assign different weights to model updates in different layers based on the neural network architecture, with the weights tuned by Bayesian optimization. Finally, experimental results validate the superiority of our proposed approximate and weighted attack method over the other state-of-the-art methods, as demonstrated by the substantial improvement in different evaluation metrics for image data reconstructions.

Index Terms—Data reconstruction attack, federated learning, Bayesian optimization, gradient inversion

1 Introduction

With the growing amount of data generated by distributed personal electronic devices, conventional centralized approaches for training machine learning models face challenges in data collection and privacy protection. To address these challenges, federated learning (FL) [1], [2] has gained significant attention in recent years as a promising paradigm.

One prominent feature of FL is its ability to facilitate model training on distributed data sources owned by individual clients while keeping the data localized and exchanging only model updates. For example, in the most commonly used federated averaging (FedAvg) [2] algorithm, each client trains its local model with its private data and shares the updated model parameters to a server, where the model parameters are aggregated and used to update the global model. In other words, FL enables multiple participants to build a common and robust machine learning model without sharing data, thus addressing critical issues such as data privacy, data security, and data access rights. As a result, FL has gained significant attention in recent years to handle the growing amount of data and the increasing concerns about privacy in several applications, such as healthcare [3], [4], and learning a controller model across several autonomous vehicles without sharing their history trajectories [5].

Although it was widely believed that model updates in FL are safe to share, recent studies [6]–[9] have shown that clients' sensitive training data can be compromised through *data reconstruction attacks* [9]. In these attacks, the adversary first randomly initializes dummy samples and labels, and then executes forward and backward propagation to obtain dummy model updates. Through an iterative process of minimizing the discrepancy between the dummy model updates and the ground-truth ones, the dummy samples and labels are updated simultaneously. In the literature, some work has already been done to improve the reconstruction performance, we refer to the label inference techniques in [8], [10], the new distance functions and optimizers in [11], [12], and the regularization methods in [13], [14].

By inferring labels in advance, the joint optimization of both samples and labels can be avoided, thus reducing the complexity of the optimization problem. It was first discovered in [10] that the label information of a single sample can be analytically extracted based on the model updates of the last fully connected layer. Later, [8] extended the single sample label extraction to batch samples, under the limiting assumption of non-repeating labels in the batch. The above limitation is further addressed by the batch label inference approach proposed in [6].

To measure the discrepancy between the dummy and ground-truth model updates, the Euclidean distance is commonly used in the loss function for the attack, see [8]–[10]. Moreover, the angle-based cosine similarity was suggested in [7], [11] since the high-dimensional direction of the model updates carries more information than the magnitude. In [12], a Gaussian kernel of model updates differences was proposed to measure the discrepancy, allowing the scaling factor in the kernel to be adapted to the distribution of model updates in each attack.

- *Y.C. Song and Z.W. Wang (corresponding author) are with the Chair for Dynamics, Control, Machine Learning and Numerics – Alexander von Humboldt Professorship, Department of Mathematics, Friedrich-Alexander-Universität Erlangen-Nürnberg, Cauerstrasse 11, 91058 Erlangen, Germany. Email: yongcun.song@fau.de, ziqi.wang@fau.de.*
- *E. Zuazua is with the Chair for Dynamics, Control, Machine Learning and Numerics – Alexander von Humboldt Professorship, Department of Mathematics, Friedrich-Alexander-Universität Erlangen-Nürnberg, Cauerstrasse 11, 91058 Erlangen, Germany; the Chair of Computational Mathematics, Fundación Deusto Avda. de las Universidades 24, 48007 Bilbao, Basque Country, Spain; and also with the Departamento de Matemáticas, Universidad Autónoma de Madrid, 28049 Madrid, Spain. Email: enrique.zuazua@fau.de.*

As for the optimizers employed in the attacks, the L-BFGS [15] and the Adam [16] are the most commonly used ones, see e.g., [8]–[14]. In particular, the reconstruction performance of the above two optimizers was compared in [11], [12], [14]. It has been shown in [14] that the L-BFGS requires fewer attack iterations to achieve high reconstruction quality compared to the Adam when attacking the LFW dataset [17]. On the other hand, in [11], the L-BFGS performs worse than the Adam for the CIFAR-10 dataset [18]. Although there is no definitive analysis guiding the selection of cost functions and optimizers, it is evident that appropriate choices can enhance the effectiveness of attacks in specific scenarios.

Another important way to improve the reconstruction performance is to add auxiliary regularization terms to the loss function for the attack based on some prior knowledge of the data. In [14], a label regularizer was proposed to match the dummy samples and labels when both of them are optimized simultaneously. Moreover, some image prior information can be employed for image reconstruction attacks, such as the total variation regularization [11] to reduce the noise in images, and the group registration regularization [8] to center the position of the main object in images. In [8], a prior term is proposed based on the mean and variance of a mini-batch used at the batch normalization layers. A generative model pre-trained on the raw data distribution was used in [13] to improve the reconstruction. However, in normal distributed learning systems, batch normalization information and raw data distribution are not necessarily shared, which makes these methods less practical.

Despite remarkable progress, limited attention has been paid to attacking the FedAvg with multiple-step model updates, where clients share local model parameters after training for multiple epochs, each executed over multiple mini-batches. In this context, an approximate method named AGIC is proposed in [7]. The AGIC method first initializes a combined dummy batch whose size is the sum of all mini-batches used in the client's local training process. This combined dummy batch is then used to perform a single-step gradient descent and the computed dummy model update is used to approximate the ground-truth multi-step model update. Meanwhile, it employs a weighted cosine similarity loss function for the attack by assigning linearly increasing weights to different convolutional and fully connected layers. However, such an approximation method works only for scenarios with small local learning rates and the layer weights are chosen empirically rather than systematically.

To address the above issues, we propose a novel approximate and weighted attack (AWA) method in data reconstruction against FL systems utilizing the FedAvg algorithm. First, we present an interpolation-based approximation method that generates intermediate model updates of clients' local training processes. As a result, attacking against the FedAvg with multiple-step model updates becomes feasible. Then, we propose a layer-wise weighted loss function to enhance the reconstruction quality. Different weights are assigned to model updates at each layer based on the neural network architecture. The selection of the weights is optimized using the Bayesian optimization method [19].

Overall, our main contributions are as follows:

1) To attack the FedAvg using multiple-step model updates, we propose an interpolation-based approximation method. The model update corresponding to each epoch is approximated by interpolating the received multiple-step model updates. The proposed approximation method makes attacks against FedAvg scenarios feasible and effective, as demonstrated by numerical experiments.
2) To further improve the attack performance after approximation, we employ a layer-wise weighted loss function for the attack. Different weights are assigned to different layers, and these weights are determined by Bayesian optimization. Additionally, we enhance the weights of layers with large errors, improving the attack's adaptability and performance.
3) Our method demonstrates environment generality by being compatible with various neural network architectures, such as Convolutional Neural Networks (CNNs) and Residual Neural Networks (ResNets). Furthermore, it is capable of reconstructing training data based on the model updates leaked at different stages of the training process.

The rest of the paper is organized as follows. In Section 2, we provide a comprehensive background on FL and data reconstruction attacks. In Section 3, various attack scenarios are analyzed. Section 4 presents our proposed AWA method, including the approximation method and the layer-wise weighted loss function. The experimental setup and simulation results are presented in Section 5. Finally, we conclude the paper in Section 6.

2 Preliminaries

In this section, we first provide a detailed description of the mathematical formulation and training process of FL. Then, we introduce the formulation and setup of data reconstruction attacks.

2.1 Problem Statement of FL

FL aims to learn a model $h : \mathbb{R}^{d_x} \times \mathbb{R}^{d_\theta} \to \mathbb{R}^{d_y}$ parameterized by $\theta \in \mathbb{R}^{d_\theta}$ such that, given any data $x \in \mathbb{R}^{d_x}$, the value $h(x;\theta)$ offers an accurate prediction about the label $y \in \mathbb{R}^{d_y}$. A crucial constraint in FL is that the training data and labels are stored across C distributed clients, and each client's data and labels can only be accessed and processed locally during the training process.

Mathematically, the training process of FL can be formulated as the following minimization problem [1]:

$$\min_{\theta} \sum_{k=1}^{C} p_k F_k(\theta), \tag{1}$$

where $F_k : \mathbb{R}^{d_\theta} \to \mathbb{R}$ represents the local loss function for client k, and $p_k \geq 0$ with $\sum_{k=1}^{C} p_k = 1$ specifies the relative impact of client k. In practice, F_k is typically defined as the empirical risk over client k's local dataset $\{(x_i^{(k)}, y_i^{(k)})\}_{i=1}^{N^{(k)}}$ of size $N^{(k)}$, i.e., $F_k(\theta) = 1/N^{(k)} \sum_{i=1}^{N^{(k)}} \ell(h(x_i^{(k)};\theta), y_i^{(k)})$,

where $\ell: \mathbb{R}^{d_y} \times \mathbb{R}^{d_y} \to \mathbb{R}$ is a prescribed loss function. Common choices of ℓ include the ℓ_2 and the cross-entropy loss function, see [20] for more options. The relative impact p_k is often chosen as $p_k = N^{(k)}/N_C$, where $N_C = \sum_{k=1}^{C} N^{(k)}$ is the total size of all the clients' datasets.

2.2 FedAvg Algorithm

For solving (1), FL algorithms normally combine local model update processes performed by each client with model aggregation steps performed by a central server. To fix ideas, we focus on the FedAvg [2], which is the most commonly used algorithm in FL. The FedAvg presented in Algorithm 1 involves a series of training rounds, in which the server first dispatches the latest global model parameters to a subset of clients. Then, the selected clients compute model updates to the current global model with their private data and send the updated model parameters back to the server. Finally, the server aggregates the received model parameters to update the global model parameters, which serve as the initializer for the next training round.

Algorithm 1 Federated Averaging.

1: Server initializes the global model parameters θ_1.
2: **for** each training round $t = 1, 2, \ldots, T$ **do**
3: Server selects a subset $\mathcal{K} \subseteq \{k\}_{k=1}^{C}$ of clients and sends them θ_t.
4: **for** each client $k \in \mathcal{K}$ in parallel **do**
5: $\theta_{t+1}^{(k)} \leftarrow$ **ClientUpdate**(k, t, θ_t).
6: **end for**
7: $N_K = \sum_{k\in\mathcal{K}} N^{(k)}$,
8: $\theta_{t+1} = \sum_{k\in\mathcal{K}} \frac{N^{(k)}}{N_K} \theta_{t+1}^{(k)}$.
9: **end for**

1: **ClientUpdate**(k, t, θ_t) :
2: Client k sets its local model parameters $\theta_{t,1,1}^{(k)} := \theta_t$.
3: **for** each epoch $e = 1, 2, \ldots, E$ **do**
4: Randomly split dataset $\mathcal{D}^{(k)} = \{(X_{t,e,b}^{(k)}, Y_{t,e,b}^{(k)})\}_{b=1}^{B^{(k)}}$.
5: **for** each mini-batch $b = 1, 2, \ldots, B^{(k)}$ **do**
6: $\theta_{t,e,b}^{(k)} := \theta_{t,e,b}^{(k)} - \eta \nabla_{\theta_{t,e,b}^{(k)}} \ell(X_{t,e,b}^{(k)}, Y_{t,e,b}^{(k)})$.
7: **end for**
8: **end for**
9: Set $\theta_{t+1}^{(k)} := \theta_{t,e,b}^{(k)}$.
10: **return** $\theta_{t+1}^{(k)}$ back to the server.

Next, we elaborate on the implementation of Algorithm 1. At each round $t = 1, 2, \ldots, T$, the server selects a set of clients $\mathcal{K} \subseteq \{k\}_{k=1}^{C}$ to participate in the training and sends them the current global model parameters θ_t. Then, each selected client $k \in \mathcal{K}$ sets its local model parameters $\theta_t^{(k)} = \theta_t$ and updates $\theta_t^{(k)}$ for E epochs, each consisting of $B^{(k)}$ mini-batches.

In each epoch $e = 1, 2, \ldots, E$, client k first shuffles its dataset $\mathcal{D}^{(k)}$ and partitions it into $B^{(k)} = N^{(k)}/M$ mini-batches (without loss of generality, we assume that the dataset is divisible into $B^{(k)}$ mini-batches of size M):

$$\mathcal{D}^{(k)} = \{(X^{(k)}, Y^{(k)})\} = \{(X_{t,e,b}^{(k)}, Y_{t,e,b}^{(k)})\}_{b=1}^{B^{(k)}}, \tag{2}$$

where $X^{(k)} = \{x_i^{(k)}\}_{i=1}^{N^{(k)}}$ is the set of the training data, $Y^{(k)} = \{y_i^{(k)}\}_{i=1}^{N^{(k)}}$ is the set of the labels, and $\{(X_{t,e,b}^{(k)}, Y_{t,e,b}^{(k)})\}$ represents the b-th mini-batch at round t, epoch e. If not otherwise stated, the subscripts t, e, and b in the following discussions indicate the index of the round, the epoch, and the mini-batch, respectively.

For each mini-batch $b = 1, 2, \ldots, B^{(k)}$, client k updates its local model parameters $\theta_{t,e,b}^{(k)} := \theta_t^{(k)}$ using the mini-batch gradient descent:

$$\theta_{t,e,b}^{(k)} := \theta_{t,e,b}^{(k)} - \eta \nabla_{\theta_{t,e,b}^{(k)}} \ell\left(X_{t,e,b}^{(k)}, Y_{t,e,b}^{(k)}\right), \tag{3}$$

where $\eta > 0$ is the learning rate (step size), and we use $\ell(X, Y) := 1/M \sum_{i=1}^{M} \ell(h(x_i; \theta), y_i)$ to represent the average loss of a mini-batch $\{(X, Y)\} := \{(x_i, y_i)\}_{i=1}^{M}$ of size M for simplicity.

After training for E epochs (each epoch consists of $B^{(k)}$ mini-batches), client k's model update $\Delta\theta_t^{(k)}$ can be obtained as

$$\Delta\theta_t^{(k)} = -\eta \sum_{e=1}^{E} \sum_{b=1}^{B^{(k)}} \nabla_{\theta_{t,e,b}^{(k)}} \ell\left(X_{t,e,b}^{(k)}, Y_{t,e,b}^{(k)}\right). \tag{4}$$

As a result, client k's local model parameters $\theta_{t+1}^{(k)}$ become

$$\theta_{t+1}^{(k)} = \theta_t^{(k)} + \Delta\theta_t^{(k)}. \tag{5}$$

Then, client k sends its updated local model parameter $\theta_{t+1}^{(k)}$ back to the server for averaging.

After receiving updated local model parameters $\{\theta_{t+1}^{(k)}\}_{k\in\mathcal{K}}$ from the clients, the server performs the weighted model parameters averaging as follows:

$$\theta_{t+1} = \sum_{k\in\mathcal{K}} \frac{N^{(k)}}{N_K} \theta_{t+1}^{(k)}, \tag{6}$$

where $N_K = \sum_{k\in\mathcal{K}} N^{(k)}$ is the total size of K participated clients' datasets. Finally, the aggregated global model parameters θ_{t+1} are used as the initializer for the next round.

2.3 Data Reconstruction Attack

Despite the fact that the clients only share the updated model parameters with the server, their private training data are still vulnerable to data reconstruction attacks [8], [9], [11]. In this subsection, we introduce the formulation and the general procedure of a data reconstruction attack.

As shown in (4), during the local training process at round t, client k's model update $\Delta\theta_t^{(k)}$ consists of the gradients computed over $E \times B^{(k)}$ mini-batches. Let $G_t^{(k)}$ be a mapping from the training data $\{(X^{(k)}, Y^{(k)})\}$ defined in (2) to the model update $\Delta\theta_t^{(k)}$, then we can rewrite (4) in a compact manner as

$$\Delta\theta_t^{(k)} = G_t^{(k)}\left(X^{(k)}, Y^{(k)}\right). \tag{7}$$

For an attacker with access to $\Delta\theta_t^{(k)}$, reconstructing $\{(X^{(k)}, Y^{(k)})\}$ is essentially an inverse problem. In particular, if $[G_t^{(k)}]^{-1}$ exists and is known analytically, the attacker can recover $\{(X^{(k)}, Y^{(k)})\}$ directly as follows:

$$\left(X^{(k)}, Y^{(k)}\right) = [G_t^{(k)}]^{-1}\left(\Delta\theta_t^{(k)}\right). \tag{8}$$

However, since neural networks are highly nonlinear and the model updates are aggregated over multiple mini-batches, it is generally difficult to identify $[G_t^{(k)}]^{-1}$. To address this issue, we introduce a numerical approach for solving (7).

Notice from (8) that the attacker can independently attack any client $k \in \mathcal{K}$ that participated in the training at round t. Therefore, in the sequel we omit the superscript k to simplify the notation.

Assuming an attacker can get access to the client's training process and hence knows G_t, then problem (7) can be solved by the numerical approach elaborated below. First, to launch the attack, the attacker randomly initializes a dummy dataset $(\hat{X}, \hat{Y})$ with the same dimension as that of the client's ground-truth dataset (X, Y). The attacker uses G_t to calculate the dummy model update $\Delta\hat{\theta}_t$ given by

$$\Delta\hat{\theta}_t = G_t(\hat{X}, \hat{Y}). \tag{9}$$

Proceeding as in [9], the attacker can reconstruct the client's dataset by matching the dummy model update $\Delta\hat{\theta}_t$ with the ground-truth model update $\Delta\theta_t$, minimizing a model update matching loss function ℓ_m, for instance, of the type

$$\ell_m(\hat{X}, \hat{Y}) = \|\Delta\hat{\theta}_t - \Delta\theta_t\|^2. \tag{10}$$

In practice, one can also use other loss functions like the cosine similarity loss [11] to evaluate the distance between $\Delta\hat{\theta}_t$ and $\Delta\theta_t$. Finally, the reconstructed data $(\hat{X}^*, \hat{Y}^*)$ can be obtained by solving the following optimization problem:

$$(\hat{X}^*, \hat{Y}^*) = \arg\min_{\hat{X}, \hat{Y}} \ell_m(\hat{X}, \hat{Y}). \tag{11}$$

This can be done using the gradient descent method with a learning rate $\hat{\eta}$:

$$\hat{X} := \hat{X} - \hat{\eta}\nabla_{\hat{X}}\ell_m(\hat{X}, \hat{Y}), \quad \hat{Y} := \hat{Y} - \hat{\eta}\nabla_{\hat{Y}}\ell_m(\hat{X}, \hat{Y}).$$

Remark 2.1. *In FL, the central server holds substantial information about the training process. Data reconstruction attacks can typically be developed by an honest-but-curious server [11], acting as an attacker, who has access to the following information, and in particular G_t in* (9)*.*

1) *Model architecture of h: Normally the server decides the architecture of the neural network that is shared among all the clients.*
2) *Initial model parameters θ_t: For each client, its initial local model parameter θ_t is dispatched from the server.*
3) *Model update $\Delta\theta_t$: Each client sends the updated model parameters θ_{t+1} obtained in* (5) *back to the server. Thus, $\Delta\theta_t = \theta_{t+1} - \theta_t$ can be obtained by the server easily.*
4) *Loss function ℓ: Similar to h, it is common for the server to know the form of the loss function that is shared among all the clients. The choice made is kept unchanged during the training process.*
5) *Dataset size N: This information is shared with the server for weighted aggregation as shown in* (6)*.*
6) *Client's local training hyperparameters: As shown in* (4) *and* (7)*, the knowledge of G_t depends on the hyperparameters listed below. In general cases, the server can assign these hyperparameters to each client:*
    a) *Number of epochs E;*
    b) *Number of mini-batches B;*
    c) *Learning rate η.*

3 Analysis of Attack Scenarios: Different E and B

As shown in (9), a data reconstruction attack requires the knowledge of G_t to calculate the dummy model update $\Delta\hat{\theta}_t$. In this section, we discuss the difficulty of knowing G_t for four FedAvg scenarios in terms of different values of E and B, especially when $E > 1$ and $B > 1$.

Scenario 1: $E = 1, B = 1$**.** As we shall see, the attacker can get access to G_t and replicate the client's training process. Indeed, since $E = 1$ and $B = 1$, the client uses the full-batch gradient descent for one epoch as follows:

$$\begin{aligned} G_t(X, Y) &= -\eta\nabla_{\theta_t}\ell(X, Y) \\ &= -\eta\frac{1}{N}\sum_{i=1}^{N}\nabla_{\theta_t}\ell(x_i, y_i). \end{aligned} \tag{12}$$

Thus, the attacker can replicate the client's training process by replacing (X, Y) with the dummy dataset $(\hat{X}, \hat{Y}) = \{(\hat{x}_i, \hat{y}_i)\}_{i=1}^N$ in the following way:

$$\begin{aligned} G_t(\hat{X}, \hat{Y}) &= -\eta\nabla_{\theta_t}\ell\left(\hat{X}, \hat{Y}\right) \\ &= -\eta\frac{1}{N}\sum_{i=1}^{N}\nabla_{\theta_t}\ell(\hat{x}_i, \hat{y}_i). \end{aligned} \tag{13}$$

Scenario 2: $E > 1, B = 1$**.** The attacker can also obtain the knowledge of G_t in this scenario. To be concrete, the client uses the full-batch gradient descent for E epochs:

$$G_t(X, Y) = -\eta\sum_{e=1}^{E}\nabla_{\theta_{t,e}}\ell(X, Y). \tag{14}$$

In each epoch, since $B = 1$, the gradients are computed on the N samples, as in Scenario 1. In this case, the attacker can replicate the client's training process by replacing (X, Y) with the dummy dataset $(\hat{X}, \hat{Y})$, and train the model for E epochs as follows:

$$G_t(\hat{X}, \hat{Y}) = -\eta\sum_{e=1}^{E}\nabla_{\theta_{t,e}}\ell\left(\hat{X}, \hat{Y}\right). \tag{15}$$

Scenario 3: $E = 1, B > 1$**.** In this scenario, since $B > 1$, the client uses the mini-batch gradient descent for one epoch in the following way:

$$G_t(X, Y) = -\eta\sum_{b=1}^{B}\nabla_{\theta_{t,b}}\ell(X_{t,b}, Y_{t,b}). \tag{16}$$

When using the mini-batch gradient descent, at the beginning of each epoch, the client randomly shuffles the dataset and separates it into B mini-batches. In this scenario, since $E = 1$, the dataset is only shuffled once.

The attacker can first separate its dummy dataset $(\hat{X}, \hat{Y})$ into B mini-batches $\{(\hat{X}_{t,b}, \hat{Y}_{t,b})\}_{b=1}^B$. Then, by replacing $(X_{t,b}, Y_{t,b})$ in (16) with $(\hat{X}_{t,b}, \hat{Y}_{t,b})$, it can replicate the client's training process as below:

$$G_t(\hat{X}, \hat{Y}) = -\eta\sum_{b=1}^{B}\nabla_{\theta_{t,b}}\ell\left(\hat{X}_{t,b}, \hat{Y}_{t,b}\right). \tag{17}$$

The only difference is that the reconstructed $(\hat{X}, \hat{Y})$ are in the same order as that of the shuffled client's dataset.

Hence, the attacker can still obtain G_t and replicate the local training process in this scenario.

Scenario 4: $E > 1, B > 1$**.** In this case, the attacker cannot gain the needed knowledge of G_t to calculate the dummy model update $\Delta\hat{\theta}_t$ defined in (9).

Indeed, the client uses the mini-batch gradient descent for E epochs:

$$G_t(X, Y) = -\eta \sum_{e=1}^{E} \sum_{b=1}^{B} \nabla_{\theta_{t,e,b}} \ell\left(X_{t,e,b}, Y_{t,e,b}\right). \tag{18}$$

In each epoch, the client first shuffles its dataset and then separates it into B mini-batches. As a result, the attacker cannot replicate the client's mini-batch separation when $E > 1$ due to the randomness of the shuffling process.

The existing attack methods are normally applicable to Scenarios 1-3 and limited attention has been given to the more challenging Scenario 4. To address it, we propose an interpolation-based approximation method. By interpolating the intercepted model updates, the attacker can approximate the intermediate model updates corresponding to each epoch, thereby reducing the problem from Scenario 4 to Scenario 3. The details of the proposed method are presented in Section 4.1.

4 Approximate and Weighted Attack in Data Reconstruction

In this section, we propose an approximate and weighted data reconstruction attack method for solving (7).

4.1 Approximation of the Intermediate Model Updates

As discussed in Section 3, when $E > 1$ and $B > 1$, the shuffle of the dataset in each epoch makes it difficult for the attacker to know G_t. However, if the attacker knows the intermediate model update $\Delta\theta_{t,e}$ of any epoch $e \in \{1, 2, \ldots, E\}$, the problem can be reduced from Scenario 4 ($E > 1$ and $B > 1$) to Scenario 3 ($E = 1$ and $B > 1$) by attacking each epoch separately.

To this end, we interpolate between θ_{t+1} and θ_t to approximate the intermediate model parameters $\{\theta_{t,e}\}_{e=1}^{E}$ corresponding to each epoch. Particularly, consider a client who trains its model for E epochs, the approximate intermediate model parameters $\{\tilde{\theta}_{t,e}\}_{e=1}^{E}$ can be obtained as

$$\tilde{\theta}_{t,e} = \frac{\theta_{t+1} - \theta_t}{E} e + \theta_t, \quad e = 1, 2, \ldots, E. \tag{19}$$

Then, the approximate model updates $\{\Delta\tilde{\theta}_{t,e}\}_{e=1}^{E}$ corresponding to each epoch can be obtained as

$$\Delta\tilde{\theta}_{t,e} = \tilde{\theta}_{t,e} - \tilde{\theta}_{t,e-1}, \quad e = 1, 2, \ldots, E, \tag{20}$$

where $\tilde{\theta}_{t,0} = \theta_t$ is the initial model parameter at round t.

After obtaining the approximate $\{\Delta\tilde{\theta}_{t,e}\}_{e=1}^{E}$, the attacker can use it to represent the unknown intermediate model update $\Delta\theta_{t,e}$ for any epoch $e \in \{1, 2, \ldots, E\}$. In other words, the attack problem is reduced from Scenario 4 ($E > 1$ and $B > 1$) to Scenario 3 ($E = 1$ and $B > 1$).

We denote by $G_{t,e}$ the client's training process at round t, epoch e. Then, the dummy model update $\Delta\hat{\theta}_{t,e}$ can be obtained based on (17), given as

$$\Delta\hat{\theta}_{t,e} = G_{t,e}(\hat{X}, \hat{Y}) = -\eta \sum_{b=1}^{B} \nabla_{\theta_{t,e,b}} \ell\left(\hat{X}_{t,e,b}, \hat{Y}_{t,e,b}\right). \tag{21}$$

Finally, the attack can be conducted following the procedures of Scenario 3.

4.2 Improved Weighted Loss Function for the Data Reconstruction Attack

The commonly used loss function (10) for the data reconstruction attack treats different components of the model update $\Delta\theta_t$ equally. However, as observed in [21], different layers in a neural network provide different contributions to boosting the performance. Inspired by this fact, we propose to assign different weights to the model updates at different layers to facilitate the reconstruction. The implementation of our method is elaborated below

Consider a neural network with L layers. Then the model update $\Delta\theta_t$ consists of superposition of the updates $\Delta\theta_t^{(l)}$ of each layer:

$$\Delta\theta_t = \{\Delta\theta_t^{(l)}\}_{l=1}^{L}. \tag{22}$$

The same applies to the dummy model updates $\Delta\hat{\theta}_t = \{\Delta\hat{\theta}_t^{(l)}\}_{l=1}^{L}$. Then, by assigning the weight $q^{(l)} > 0$ to the model updates at layer l, the loss function (10) for the attack becomes

$$\ell_Q(\hat{X}, \hat{Y}) = \sum_{l=1}^{L} q^{(l)} \left\| \Delta\hat{\theta}_t^{(l)} - \Delta\theta_t^{(l)} \right\|^2. \tag{23}$$

The weighted loss function (23) leverages the distinct characteristics of different layers in the model update. In the next subsection, we introduce a systematic method to design the layer weights $q^{(l)}$ to enhance the reconstruction performance.

4.2.1 Design of Layer Weights

Increasing weights layer by layer. We consider linearly increasing weight functions for different types of layers. To expose our ideas clearly, we focus on the commonly used ResNet architecture [22], which contains convolutional, batch normalization, and fully connected layers. Specifically, we design the following weight functions for each kind of layer:

$$q_{cv}^{(l)} = \begin{cases} \frac{q_{cv}-1}{L_{cv}-1}(l-1)+1, & \text{if } l = 1, 2, ..., L_{cv} > 1, \\ q_{cv}, & \text{if } l = L_{cv} = 1, \end{cases} \tag{24a}$$

$$q_{bn}^{(l)} = \begin{cases} \frac{q_{bn}-1}{L_{bn}-1}(l-1)+1, & \text{if } l = 1, 2, ..., L_{bn} > 1, \\ q_{bn}, & \text{if } l = L_{bn} = 1, \end{cases} \tag{24b}$$

$$q_{fc}^{(l)} = \begin{cases} \frac{q_{fc}-1}{L_{fc}-1}(l-1)+1, & \text{if } l = 1, 2, ..., L_{fc} > 1, \\ q_{fc}, & \text{if } l = L_{fc} = 1, \end{cases} \tag{24c}$$

where L_{cv}, L_{bn}, and L_{fc} are the numbers of the convolutional, batch normalization, and fully connected layers, respectively, and $L = L_{cv} + L_{bn} + L_{fc}$. The values $q_{cv} > 1$, $q_{bn} > 1$, and $q_{fc} > 1$ are the largest weights assigned to

the last layer of each respective kind. For a given neural network with a fixed number of layers (L_{cv}, L_{bn} and L_{fc}), the values of q_{cv}, q_{bn} and q_{fc} determine the slope of the linearly increasing weight functions in (24).

Enhancing the weights of layers with larger errors. Adding linearly increasing weights determined by (24) to the loss function (23) may overly emphasize the importance of some layers in the neural network and lead to a biased reconstruction. To strike a balance between adding linearly increasing weights and avoiding biased reconstructions, we propose to modify the weights of layers with larger errors by exploiting the statistical information including the mean $\mu(\cdot)$ and the variance $\sigma^2(\cdot)$ of the layer-wise model updates $\{\Delta\theta_t^{(l)}\}_{l=1}^L$ in (22). The procedure of our enhancing method is elaborated below.

First, we calculate the relative error $e_{mean}^{(l)}$ and $e_{var}^{(l)}$ of the dummy model update $\Delta\hat{\theta}_t^{(l)}$ and the ground-truth model update $\Delta\theta_t^{(l)}$ at each layer as follows:

$$e_{mean}^{(l)} = \frac{|\mu(\Delta\hat{\theta}_t^{(l)}) - \mu(\Delta\theta_t^{(l)})|}{|\mu(\Delta\theta_t^{(l)})|}, \quad l = 1, 2, ..., L, \tag{25}$$

$$e_{var}^{(l)} = \frac{|\sigma^2(\Delta\hat{\theta}_t^{(l)}) - \sigma^2(\Delta\theta_t^{(l)})|}{|\sigma^2(\Delta\theta_t^{(l)})|}, \quad l = 1, 2, ..., L. \tag{26}$$

Next, we select a subset $\mathcal{P} \subseteq \mathcal{L} = \{l\}_{l=1}^L$ of layers with the largest relative errors in terms of $\{e_{mean}^{(l)}\}_{l=1}^L$ and $\{e_{var}^{(l)}\}_{l=1}^L$, and set their layer weights $q^{(l)}$ to $q_{en} \in \mathbb{R}$, i.e.,

$$q^{(l)} = q_{en}, \quad l \in \mathcal{P}. \tag{27}$$

The choice of the subset $\mathcal{P}$ can be decided by the proportional parameters $p_{mean} \in [0, 1]$ and $p_{var} \in [0, 1]$ in the following way: For a given p_{mean}, we first select $N_{mean} = \lceil p_{mean} \cdot L \rceil$ layers with the largest relative error in terms of $\{e_{mean}^{(l)}\}_{l=1}^L$. Let the set of indices corresponding to the N_{mean} layers be denoted as $\mathcal{P}_{mean}$, that is

$$\mathcal{P}_{mean} = \{i_1, i_2, \ldots, i_{N_{mean}}\}, \tag{28}$$

where $i_1, i_2, \ldots, i_{N_{mean}} \in \{1, 2, \ldots, L\}$ are the indices selected such that $e_{mean}^{(i_1)} \geq e_{mean}^{(i_2)} \geq \ldots \geq e_{mean}^{(i_{N_{mean}})} \geq e_{mean}^{(l)}, l \in \mathcal{L} \setminus \mathcal{P}_{mean}$.

Similarly, we can get a set $\mathcal{P}_{var}$ with $N_{var} = \lceil p_{var} \cdot L \rceil$ elements as

$$\mathcal{P}_{var} = \{j_1, j_2, \ldots, j_{N_{var}}\}, \tag{29}$$

where $j_1, j_2, \ldots, j_{N_{var}} \in \{1, 2, \ldots, L\}$ are the indices that satisfy $e_{var}^{(j_1)} \geq e_{var}^{(j_2)} \geq \ldots \geq e_{var}^{(j_{N_{var}})} \geq e_{var}^{(l)}, l \in \mathcal{L} \setminus \mathcal{P}_{var}$.

Finally, the subset $\mathcal{P}$ can be obtained as the intersection of $\mathcal{P}_{mean}$ and $\mathcal{P}_{var}$:

$$\mathcal{P} = \mathcal{P}_{mean} \cap \mathcal{P}_{var}. \tag{30}$$

Hyperparameters to tune. Following (24) and (27), the layer weights $\{q^{(l)}\}_{l=1}^L$ in (23) are determined by the parameter vector $Q \in \mathbb{R}^6$, which is defined as

$$Q = (q_{cv}, q_{bn}, q_{fc}, q_{en}, p_{mean}, p_{var}). \tag{31}$$

Given a Q, by using the weighted loss function (23), one can obtain the reconstructed data $(\hat{X}^*, \hat{Y}^*)$ by solving the following optimization problem:

$$(\hat{X}^*, \hat{Y}^*) = \arg\min_{\hat{X}, \hat{Y}} \ell_Q(\hat{X}, \hat{Y}). \tag{32}$$

We then use Bayesian Optimization to choose a proper Q for a better reconstruction.

4.2.2 Choice of Q by Bayesian Optimization

Objective function. As shown in (32), one can obtain the reconstructed data $(\hat{X}^*, \hat{Y}^*)$ with a given Q. Then, the corresponding reconstructed model update $\Delta\hat{\theta}_t^*$ can be calculated as $\Delta\hat{\theta}_t^* = G_t(\hat{X}^*, \hat{Y}^*)$.

Let $f : \mathbb{R}^6 \to \mathbb{R}$ be the objective function that measures the distance between the reconstructed model update $\Delta\hat{\theta}^*$ and the ground-truth model update $\Delta\theta$ in the following form:

$$f(Q) = \left\| \Delta\hat{\theta}_t^* - \Delta\theta_t \right\|^2. \tag{33}$$

Finding the optimal Q^* is equivalent to solving the following optimization problem:

$$Q^* = \arg\min_Q f(Q). \tag{34}$$

For the above optimization problem, f is a black box function that does not have an analytic expression. Meanwhile, calculating f is computationally expensive since one has to complete a data reconstruction attack to obtain $\Delta\hat{\theta}_t^*$. As a result, traditional parameter determination methods like grid search are not feasible. To overcome the above difficulties, we employ the Bayesian optimization [19] to solve (34).

A Bayesian optimization algorithm for (34)**.** Bayesian optimization is a powerful technique for optimizing black-box functions that are expensive to evaluate and may have noise or other sources of uncertainty. In general, Bayesian optimization iteratively uses a surrogate model to approximate the black-box function and then employs an acquisition function to determine the next set of parameters to evaluate.

As for the surrogate model, it works to approximate the black-box objective function f, which is commonly chosen to be a Gaussian process (GP) [23]. Formally, a GP is a collection of random variables, any finite number of which have a joint Gaussian distribution.

Given an initial set $\mathcal{O} = \{(Q_i, f(Q_i))\}_{i=1}^n$ that contains n pairs of the sampling points and their function values, the resulting prior distribution can be given as

$$\boldsymbol{f} \sim \mathcal{N}(\boldsymbol{\mu}, \Sigma_{\boldsymbol{QQ}}), \tag{35}$$

where $\boldsymbol{f} = (f(Q_1), \ldots, f(Q_n))^\top$, $\boldsymbol{Q} = (Q_1, \ldots, Q_n)^\top$, $\boldsymbol{\mu} = (\mu(Q_1), \ldots, \mu(Q_n))^\top$ with $\mu(\cdot)$ being the mean function and is commonly set to zero, $\Sigma_{\boldsymbol{QQ}} \in \mathbb{R}^{n \times n}$ is the matrix of the covariances with its (i, j) entry being $\kappa(Q_i, Q_j)$, and $\kappa(\cdot, \cdot)$ is a positive definite kernel function, typically set to the Gaussian kernel.

Then, we can infer the value of $f(Q)$ at a new point Q by computing the posterior distribution of $f(Q)$ given prior observations [23] as follows:

$$\begin{aligned} f(Q) \mid \boldsymbol{f} &\sim \mathcal{N}(\mu_Q, \sigma_Q^2), \\ \mu_Q &= \Sigma_{Q\boldsymbol{Q}} \Sigma_{\boldsymbol{QQ}}^{-1} \boldsymbol{f}, \\ \sigma_Q^2 &= \Sigma_{QQ} - \Sigma_{Q\boldsymbol{Q}} \Sigma_{\boldsymbol{QQ}}^{-1} \Sigma_{\boldsymbol{Q}Q}. \end{aligned} \tag{36}$$

As for the acquisition function, it is used to propose the parameters for the next trial by trading off exploitation

and exploration. Exploitation means sampling at locations where the surrogate model predicts a high objective value and exploration means sampling at locations where the prediction uncertainty is high.

One of the most popular acquisition functions is Expected Improvement (EI) [24]. Let $f_{\min}$ denote the best function value obtained so far. Then, the improvement over $f_{\min}$ at point Q can be defined as

$$I(Q) = \max(f_{\min} - f(Q), 0). \tag{37}$$

The improvement $I(Q)$ is a random variable since $f(Q) \sim \mathcal{N}\left(\mu_Q, \sigma_Q^2\right)$ as shown in (36). To obtain the expected improvement, we can take the expected value as follows:

$$EI(Q) = \mathbb{E}[\max(f_{\min} - f(Q), 0)]. \tag{38}$$

The expected improvement can be evaluated analytically under the GP [25], given as:

$$\begin{aligned} EI(Q) = &(f_{\min} - \mu_Q)\,\Phi\left(\frac{f_{\min} - \mu_Q}{\sigma_Q}\right) \\ &+ \sigma_Q \varphi\left(\frac{f_{\min} - \mu_Q}{\sigma_Q}\right), \end{aligned} \tag{39}$$

where φ and Φ are the probability density and cumulative distribution functions of the standard normal distribution, respectively. It can be seen that $EI(Q)$ is higher for a point Q predicted to have a smaller μ_Q and a larger σ_Q, indicating the trade-off between exploitation and exploration.

Given the EI, the parameter Q_{n+1} for the next trial is chosen to be the point with the largest expected improvement:

$$Q_{n+1} = \arg\max_{Q} EI(Q). \tag{40}$$

The evaluation of $EI(Q)$ is much easier than that of the function f in (34). The optimization problem (40) can be solved by some classic optimization techniques such as Newton's method.

4.3 Approximate and Weighted Data Reconstruction Attack Method

Based on the discussions in Sections 4.1-4.2, we propose an approximate and weighted data reconstruction attack method and summarize it in Algorithm 2.

Algorithm 2 Approximate and Weighted Attack (AWA).

Intercept a client's model update $\Delta\theta_t$ at round t.
Calculate approximate $\Delta\tilde{\theta}_{t,e}$ and $G_{t,e}$. ▷ (20)
Initialize an empty set $\mathcal{O}$.
for each trial $i = 1, 2, ..., n$ **do**
  Generate a random test point Q_i.
  Obtain $\hat{X}, \hat{Y}, f(Q_i) \leftarrow$ **RecAttack**$(Q_i, \Delta\tilde{\theta}_{t,e}, G_{t,e})$.
  Update $\mathcal{O} := \mathcal{O} \cup \{(Q_i, f(Q_i))\}$.
end for
for each trial $i = n+1, n+2, ..., N_{BO}$ **do**
  Fit GP over f with samples in $\mathcal{O}$. ▷ (35)
  Choose Q_i with the largest EI. ▷ (40)
  Obtain $\hat{X}, \hat{Y}, f(Q_i) \leftarrow$ **RecAttack**$(Q_i, \Delta\tilde{\theta}_{t,e}, G_{t,e})$.
  Update $\mathcal{O} := \mathcal{O} \cup \{(Q_i, f(Q_i))\}$.
end for
Get $Q^* = \arg\min_{Q_i}\{f(Q_i)\}_{i=1}^{N_{BO}}$.
Obtain $\hat{X}^*, \hat{Y}^*, f(Q^*) \leftarrow$ **RecAttack**$(Q^*, \Delta\tilde{\theta}_{t,e}, G_{t,e})$.
return $\hat{X}^*, \hat{Y}^*$.

RecAttack$(Q_i, \Delta\tilde{\theta}_{t,e}, G_{t,e})$:
Initialize the dummy data $(\hat{X}, \hat{Y})$ and set $Q := Q_i$.
for each attack iteration from 1 to N_{AT} **do**
  Compute $\Delta\hat{\theta}_{t,e} = G_{t,e}(\hat{X}, \hat{Y})$.
  Calculate $\{q^{(l)}\}_{l=1}^{L}$ based on Q. ▷ (24)
  Select layers $l \in \mathcal{P}$ with the largest errors. ▷ (30)
  Set $q^{(l)} = q_{en}, \forall l \in \mathcal{P}$ based on Q. ▷ (27)
  Calculate $\ell_Q(\hat{X}, \hat{Y}) = \sum_{l=1}^{L} q^{(l)} \|\Delta\hat{\theta}_{t,e}^{(l)} - \Delta\tilde{\theta}_{t,e}^{(l)}\|^2$,
  Update $\hat{X} := \hat{X} - \hat{\eta}\nabla_{\hat{X}}\ell_Q(\hat{X}, \hat{Y})$,
  Update $\hat{Y} := \hat{Y} - \hat{\eta}\nabla_{\hat{Y}}\ell_Q(\hat{X}, \hat{Y})$.
end for
Calculate $f(Q) = \|G_{t,e}(\hat{X}, \hat{Y}) - \Delta\tilde{\theta}_{t,e}\|^2$.
return $\hat{X}, \hat{Y}, f(Q)$.

5 Experimental Tests

In this section, we conduct numerical experiments to validate the effectiveness of our proposed AWA method given in Algorithm 2. We first introduce experimental environments and implementation details used in our experiments. We then explain the choice of hyperparameters and describe the evaluation metrics. Finally, we test our proposed AWA method for image data reconstructions and compare its performance with two state-of-the-art methods, namely AGIC [7] and DLG [9]), in attacking various FedAvg scenarios.

5.1 Setups

Hardware. For all the experiments, we use a computer equipped with an Xeon E5-2680 v4 CPU, 32GB of RAM, and an NVIDIA GeForce RTX 1080 Ti GPU.

Implementation details. To implement the FedAvg, we select images from the training set of CIFAR-10 [18] (color images of 10 categories, size 32×32) as the clients' ground-truth data. The model used by each client is the ResNet18 [22]. The client's local training process employs stochastic gradient descent with a learning rate of 0.001. For the attack optimizations, the Adam optimizer [16] with a learning rate of 0.1 is used and each attack runs for 1,000 iterations. Following the label inference method in [6], [8], we proceed with the assumption that the label information is known. Table 1 lists the simulation scenarios of the data reconstruction attack.

TABLE 1: Simulation settings of the data reconstruction attack.

Client's dataset	CIFAR-10
Client's neural network	ResNet18
Attack optimizer	Adam
Attack learning rate	0.1
Attack iterations	1,000

In our AWA method, the parameter Q defined by (31) is selected by using Bayesian optimization, where we use the Gaussian process (36) as the surrogate model and the expected improvement (39) as the acquisition function. The objective function of Bayesian optimization is the squared Euclidean norm of the dummy model update $\Delta\hat{\theta}^*$ and the ground-truth model update $\Delta\theta$ as defined in (33). We run Bayesian optimization for $N_{BO} = 50$ iterations, with the first $n = 12$ iterations initiating the dataset $\mathcal{O}$. The search ranges of Q are listed in Table 2.

TABLE 2: Search ranges of Q in Bayesian optimization.

Parameters	Search ranges
q_{cv}	[1, 1000]
q_{bn}	[1, 1000]
q_{fc}	[1, 1000]
q_{en}	[1, 1000]
p_{mean}	[0, 0.5]
p_{var}	[0, 0.5]

Evaluation metrics. To evaluate the efficiency of the attack and the quality of the data reconstruction, we use three different metrics to measure the dissimilarity between reconstructed data and the ground-truth data: pixel-wise Mean Square Error (MSE) [26], Peak Signal-to-Noise Ratio (PSNR) [26], and Structural Similarity Index Measure (SSIM) [27]. The above three metrics are prevalent and appropriate indicators in evaluating the effect of image reconstruction [26], and they have been widely used in evaluating the existing data reconstruction attacks, see e.g., [8], [11], [14].

- MSE quantifies the discrepancy between the ground-truth image $\boldsymbol{D}$ and the reconstructed image $\boldsymbol{D'}$, and it is defined as

$$\mathrm{MSE}(\boldsymbol{D}, \boldsymbol{D'}) = \frac{1}{d_m d_n} \sum_{i=1}^{d_m} \sum_{j=1}^{d_n} [\boldsymbol{D}(i,j) - \boldsymbol{D'}(i,j)]^2,$$

  where $d_m \times d_n$ is the image size, $\boldsymbol{D}(i,j)$ and $\boldsymbol{D'}(i,j)$ represent the pixel values at coordinates (i,j) of $\boldsymbol{D}$ and $\boldsymbol{D'}$. A smaller MSE indicates a better reconstruction quality.
- PSNR represents the rate of the maximum possible signal power to the distortion noise power. PSNR can be calculated as

$$\mathrm{PSNR}(\boldsymbol{D}, \boldsymbol{D'}) = 10 \log_{10} \frac{\max_{\boldsymbol{D}}^2}{\mathrm{MSE}(\boldsymbol{D}, \boldsymbol{D'})},$$

  where $\max_{\boldsymbol{D}}$ is the maximum pixel value in the ground-truth image $\boldsymbol{D}$. It is easy to see that the larger the PSNR value, the better the image reconstruction quality.
- SSIM measures the structural similarity between the ground-truth and reconstructed images. The value of SSIM ranges between zero and one, and a higher value indicates a better reconstruction. To be concrete, SSIM is calculated as

$$\mathrm{SSIM}(\boldsymbol{D}, \boldsymbol{D'}) = \frac{(2\mu_{\boldsymbol{D}}\mu_{\boldsymbol{D'}} + c_1)(2\sigma_{\boldsymbol{D}\boldsymbol{D'}} + c_2)}{(\mu_{\boldsymbol{D}}^2 + \mu_{\boldsymbol{D'}}^2 + c_1)(\sigma_{\boldsymbol{D}}^2 + \sigma_{\boldsymbol{D'}}^2 + c_2)}.$$

  Here, $\mu_{\boldsymbol{D}}$ and $\mu_{\boldsymbol{D'}}$ are the average pixel values of $\boldsymbol{D}$ and $\boldsymbol{D'}$, $\sigma_{\boldsymbol{D}}$ and $\sigma_{\boldsymbol{D'}}$ are the standard deviations of pixel values of $\boldsymbol{D}$ and $\boldsymbol{D'}$, $\sigma_{\boldsymbol{D}\boldsymbol{D'}}$ is the covariance of $\boldsymbol{D}$ and $\boldsymbol{D'}$, $c_1 = (k_1 H)^2$ and $c_2 = (k_2 H)^2$ are constants with $k_1 = 0.01$, $k_2 = 0.03$, and H being the range of the pixel values.

5.2 Results for Image Data Reconstruction Attacks

To verify the feasibility and effectiveness of our proposed AWA method, we test it in image data reconstruction attacks and compared the results with two state-of-the-art attack methods (AGIC [7] and DLG [9]) under the following four different FedAvg scenarios:

- Case 1: $N = 4$, $E = 1$, and $B = 1$;
- Case 2: $N = 4$, $E = 4$, and $B = 1$;
- Case 3: $N = 4$, $E = 1$, and $B = 4$;
- Case 4: $N = 4$, $E = 2$, and $B = 2$.

Recall the analysis in Section 3, when $E > 1$ and $B > 1$, the attacker has to use an approximate method for the attack. Therefore, only AWA and AGIC can conduct the attack for Case 4. The approximate strategy of each method is given below. Our proposed AWA method uses (20) to get the approximate intermediate model updates $\Delta\tilde{\theta}_{t,e}$ in each epoch $e = 1, 2, \ldots, E$. Then, the attacker can reconstruct the client's dataset by using any of the $\Delta\tilde{\theta}_{t,e}$ and the corresponding $G_{t,e}$. In Case 4 with $E = 2$, we have two options: $\Delta\tilde{\theta}_{t,1}$ and $\Delta\tilde{\theta}_{t,2}$. In the following test, we choose $\Delta\tilde{\theta}_{t,1}$ as the target model update for the reconstruction. On the other hand, AGIC assumes that a combined batch consisting of all the mini-batches used in the client's local training process can approximate the received model update in a single local update step. For this purpose, it assumes all the mini-batches used in the client's local training form a combined batch. Then, the dummy model update is calculated by doing a full-batch gradient descent using the combined batch.

In all the tests, the DLG method utilizes the unweighted loss function (10) for the attack, while the AGIC method employs a weighted cosine similarity loss function by assigning linearly increasing weights to convolutional and fully connected layers. However, these weights are assigned empirically and not tailored case by case. In contrast, the AWA method employs the weighted and enhanced loss function (23) for the attack, with the layer weights determined by (24) and (27). The values of Q are selected by utilizing Bayesian optimization. The cumulative minimum loss $f(Q)$ of Bayesian optimization in 50 trials is presented in Figure 1. It can be seen that the Bayesian optimization approach successfully finds a trend of smaller $f(Q)$ values as the trial count increases. Finally, the parameter settings refined by Bayesian optimization for our AWA method in each case are listed in Table 3.

TABLE 3: Parameter settings tuned by Bayesian optimization for our AWA method in four cases.

Parameters	Case 1	Case 2	Case 3	Case 4
q_{cv}	519.19	236.31	547.17	655.98
q_{bn}	802.55	552.31	222.48	692.94
q_{fc}	42.83	54.28	394.39	283.42
q_{en}	946.44	837.80	899.47	665.28
p_{mean}	0.24	0.11	0.31	0.40
p_{var}	0.07	0.13	0.01	0.33

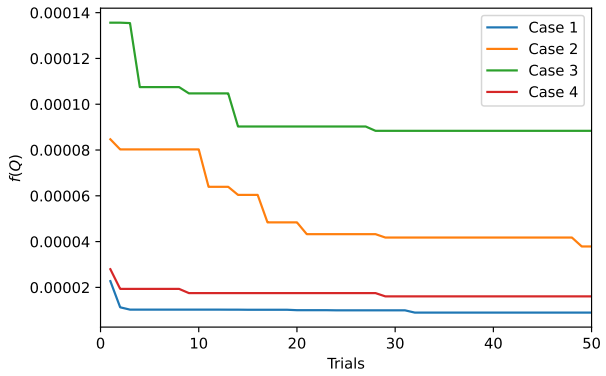


Figure 1: Cumulative minimum loss $f(Q)$ of Bayesian optimization in four cases.

The numerical comparisons of the above three data reconstruction attack methods for Cases 1-4 are presented in Figure 2 and Table 4. We observe that in all four cases, our proposed AWA method consistently yields images with substantially enhanced resolution compared to those obtained by DLG and AGIC. This is further validated by the consistently highest values of PSNR and SSIM achieved by AWA in each case. Another noteworthy finding is the consistency in the reconstruction qualities of images produced by our AWA method across the diverse FedAvg cases, which demonstrates the robustness of our proposed approach.

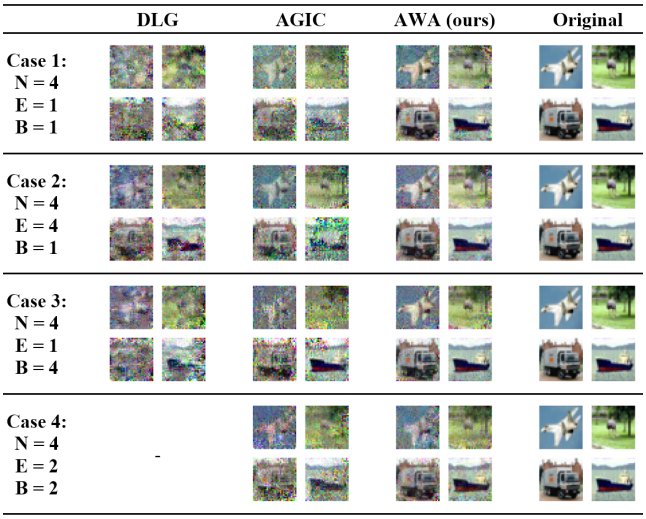


Figure 2: Comparison of the reconstruction results achieved by three data reconstruction methods in four FedAvg scenarios after 1,000 attack iterations. FedAvg parameters: batch size N, number of epochs E, number of mini-batches B. Methods: AWA (ours): reconstruction with the method in Algorithm 2, AGIC: reconstruction with the method in [7], DLG: reconstruction with the method in [9].

In Case 4, where $E > 1$ and $B > 1$, DLG encounters difficulties in performing the attack due to the absence of an approximate strategy. On the other hand, although AGIC incorporates an approximate strategy, it fails to reconstruct images with identifiable objects effectively. In sharp contrast, the proposed AWA method demonstrates its capability

TABLE 4: Detailed evaluation metrics of the data reconstructed by three data reconstruction methods in four cases.

	Metrics	DLG	AGIC	AWA(ours)
Case 1	PSNR	14.06	15.89	**22.40**
	SSIM	0.534	0.689	**0.935**
	MSE	0.042	0.028	**0.006**
Case 2	PSNR	16.43	16.07	**20.40**
	SSIM	0.739	0.631	**0.901**
	MSE	0.024	0.035	**0.009**
Case 3	PSNR	14.05	15.63	**22.47**
	SSIM	0.505	0.718	**0.936**
	MSE	0.044	0.029	**0.006**
Case 4	PSNR	-	16.41	**19.68**
	SSIM	-	0.734	**0.882**
	MSE	-	0.025	**0.011**

to successfully reconstruct images with reasonable resolutions, overcoming the limitations faced by DLG and AGIC. Furthermore, the PSNR of images reconstructed in Case 4 is found to be comparable to that achieved in the first three cases, which strongly suggests the effectiveness of our proposed approximate strategy.

Furthermore, we present the reconstruction results of our AWA method after 3,000 attack iterations in four cases in Figure 3 and Figure 4. All the other parameters maintain the same as listed in Table 1. The resulting reconstructed images and the corresponding SSIM metrics for each case are represented in Figure 3. Evidently, the attained SSIM values surpass those presented in Table 4 for the 1000-iteration attack. The results clearly show a significant visual proximity between the reconstructed images and their ground-truth counterparts. Additionally, Figure 4 presents the evaluation metrics PSNR and SSIM, as well as the cumulative minimum value of the weighted loss during the attack across all four cases. It is evident that our method achieves a satisfactory level of reconstruction, with SSIM values exceeding 0.9 within 1500 attack iterations for all four cases. The outcomes of Case 4 exhibit a marginally inferior performance compared to Cases 1-3 due to the need for approximation. However, the reconstructed images in Case 4 remain sufficiently clear to facilitate object identification. These results further validate the effectiveness of our proposed AWA method.

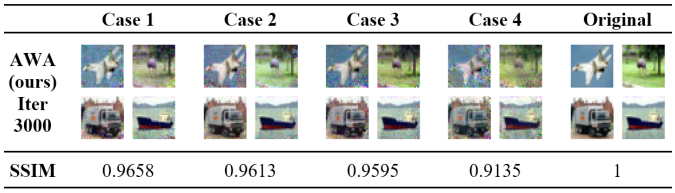
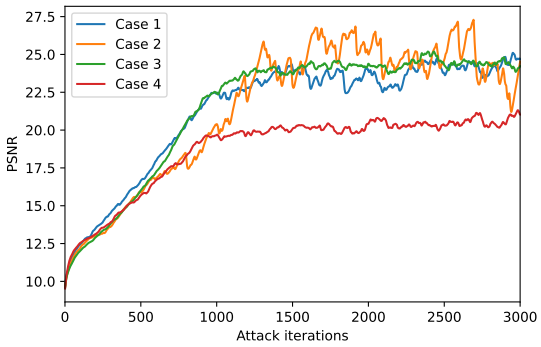
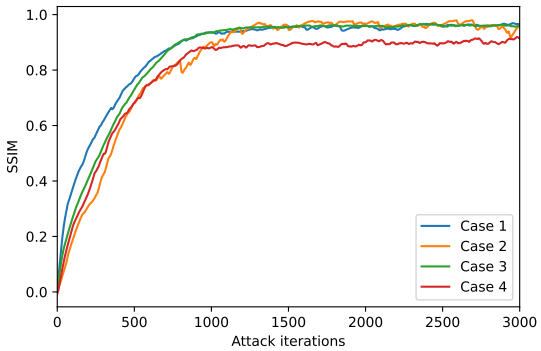


Figure 3: Reconstruction results of our AWA method in four cases after 3,000 attack iterations.

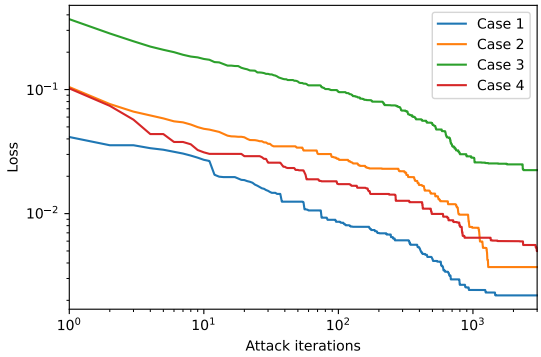
Overall, the aforementioned results demonstrate a substantial improvement in reconstruction achieved by our AWA method. The comprehensive evaluation strongly supports the superiority of our method AWA in data reconstruction attacks against various FedAvg scenarios.



(a) PSNR.



(b) SSIM.



(c) Weighted Loss.

Figure 4: Evaluation metrics of our AWA method in four cases after 3,000 attack iterations.

6 Conclusions

The privacy benefits of federated learning (FL) are compromised by recently developed data reconstruction attacks. In this paper, we first formulate the attack as an inverse problem, allowing us to reconstruct the client's training data iteratively by solving an optimization problem. To attack the widely used federated averaging (FedAvg) scenario, we propose an interpolation-based approximation method, where the intermediate model updates corresponding to each epoch are approximated by interpolating the model parameters. Furthermore, we propose a layer-wise weighted and enhanced loss function for the attack to improve the quality of reconstructed data. By assigning appropriate weights to model updates in different layers by using the Bayesian optimization method, we achieve superior reconstruction results compared to the existing state-of-the-art methods. Moreover, our method is compatible with various neural network architectures like Convolutional Neural Networks and Residual Neural Networks. Numerical results validate that our proposed approximate and weighted data reconstruction attack method is effective for adversaries to exploit the vulnerabilities of FL systems utilizing the FedAvg algorithm. The ability to reconstruct data from intermediate model updates highlights the need for robust defense mechanisms. Future research could focus on developing countermeasures and enhancing the security of FL frameworks to mitigate the risks associated with such attacks.

Acknowledgments

This work has been funded by the Humboldt Research Fellowship for postdoctoral researchers, the Alexander von Humboldt-Professorship program, the European Union's Horizon Europe MSCA project ModConFlex (grant number 101073558), the COST Action MAT-DYN-NET, the Transregio 154 Project of the DFG, grants PID2020-112617GB-C22 and TED2021-131390B-I00 of MINECO (Spain). Madrid Government - UAM Agreement for the Excellence of the University Research Staff in the context of the V PRICIT (Regional Programme of Research and Technological Innovation).

References

[1] 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, ISSN: 1558-0792.

[2] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas, "Communication-Efficient Learning of Deep Networks from Decentralized Data," in *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics*, PMLR, 2017, pp. 1273–1282.

[3] T. S. Brisimi, R. Chen, T. Mela, A. Olshevsky, I. C. Paschalidis, and W. Shi, "Federated learning of predictive models from federated Electronic Health Records," *International Journal of Medical Informatics*, vol. 112, pp. 59–67, 2018, ISSN: 1386-5056.

[4] J. Xu, B. S. Glicksberg, C. Su, P. Walker, J. Bian, and F. Wang, "Federated Learning for Healthcare Informatics," *Journal of Healthcare Informatics Research*, vol. 5, no. 1, pp. 1–19, 2021, ISSN: 2509-498X.

[5] T. Zeng, O. Semiari, M. Chen, W. Saad, and M. Bennis, "Federated Learning on the Road Autonomous Controller Design for Connected and Autonomous Vehicles," *IEEE Transactions on Wireless Communications*, vol. 21, no. 12, pp. 10 407–10 423, 2022, ISSN: 1558-2248.

[6] J. Geng, Y. Mou, Q. Li, *et al.*, "Improved Gradient Inversion Attacks and Defenses in Federated Learning," *IEEE Transactions on Big Data*, pp. 1–13, 2023, ISSN: 2332-7790.

[7] J. Xu, C. Hong, J. Huang, L. Y. Chen, and J. Decouchant, "AGIC: Approximate Gradient Inversion Attack on Federated Learning," in *2022 41st International Symposium on Reliable Distributed Systems (SRDS)*, 2022, pp. 12–22.

[8] H. Yin, A. Mallya, A. Vahdat, J. M. Alvarez, J. Kautz, and P. Molchanov, "See through gradients: Image batch recovery via gradinversion," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2021, pp. 16 337–16 346.

[9] L. Zhu, Z. Liu, and S. Han, "Deep Leakage from Gradients," in *Advances in Neural Information Processing Systems*, vol. 32, Curran Associates, Inc., 2019.

[10] B. Zhao, K. R. Mopuri, and H. Bilen, "iDLG: Improved Deep Leakage from Gradients," *arXiv preprint arXiv:2001.02610*, 2020.

[11] J. Geiping, H. Bauermeister, H. Dröge, and M. Moeller, "Inverting Gradients - How easy is it to break privacy in federated learning?" In *Advances in Neural Information Processing Systems*, vol. 33, Curran Associates, Inc., 2020, pp. 16 937–16 947.

[12] Y. Wang, J. Deng, D. Guo, *et al.*, "SAPAG: A Self-Adaptive Privacy Attack From Gradients," *arXiv preprint arXiv:2009.06228*, 2020.

[13] J. Jeon, J. Kim, K. Lee, S. Oh, and J. Ok, "Gradient Inversion with Generative Image Prior," in *Advances in Neural Information Processing Systems*, vol. 34, Curran Associates, Inc., 2021, pp. 29 898–29 908.

[14] W. Wei, L. Liu, M. Loper, *et al.*, "A Framework for Evaluating Client Privacy Leakages in Federated Learning," in *Computer Security – ESORICS 2020*, L. Chen, N. Li, K. Liang, and S. Schneider, Eds., ser. Lecture Notes in Computer Science, Cham: Springer International Publishing, 2020, pp. 545–566, ISBN: 978-3-030-58951-6.

[15] D. C. Liu and J. Nocedal, "On the limited memory BFGS method for large scale optimization," *Mathematical Programming*, vol. 45, no. 1, pp. 503–528, 1989, ISSN: 1436-4646.

[16] D. P. Kingma and J. Ba, "Adam: A Method for Stochastic Optimization," *arXiv preprint arXiv:1412.6980*, 2017.

[17] G. B. Huang, M. Mattar, T. Berg, and E. Learned-Miller, "Labeled Faces in the Wild: A Database forStudying Face Recognition in Unconstrained Environments," in *Workshop on Faces in 'Real-Life' Images: Detection, Alignment, and Recognition*, 2008.

[18] A. Krizhevsky and G. Hinton, "Learning multiple layers of features from tiny images," 2009.

[19] P. I. Frazier, "A Tutorial on Bayesian Optimization," *arXiv preprint arXiv:1807.02811*, 2018.

[20] O. Calin, *Deep Learning Architectures: A Mathematical Approach* (Springer Series in the Data Sciences). Cham: Springer International Publishing, 2020, ISBN: 978-3-030-36720-6 978-3-030-36721-3.

[21] S. Chen and Q. Zhao, "Shallowing Deep Networks: Layer-Wise Pruning Based on Feature Representations," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 41, no. 12, pp. 3048–3056, 2019, ISSN: 1939-3539.

[22] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 770–778.

[23] C. E. Rasmussen and C. K. I. Williams, *Gaussian Processes for Machine Learning* (Adaptive Computation and Machine Learning). Cambridge, Mass: MIT Press, 2006, ISBN: 978-0-262-18253-9.

[24] J. Močkus, "On Bayesian methods for seeking the extremum," in *Optimization Techniques IFIP Technical Conference: Novosibirsk, July 1–7, 1974*, Springer, 1975, pp. 400–404, ISBN: 3-662-37713-6.

[25] D. R. Jones, M. Schonlau, and W. J. Welch, "Efficient Global Optimization of Expensive Black-Box Functions," *Journal of Global Optimization*, vol. 13, no. 4, pp. 455–492, 1998, ISSN: 1573-2916.

[26] U. Sara, M. Akter, and M. S. Uddin, "Image Quality Assessment through FSIM, SSIM, MSE and PSNR—A Comparative Study," *Journal of Computer and Communications*, vol. 7, no. 3, pp. 8–18, 2019.

[27] Z. Wang, A. Bovik, H. Sheikh, and E. Simoncelli, "Image quality assessment: From error visibility to structural similarity," *IEEE Transactions on Image Processing*, vol. 13, no. 4, pp. 600–612, 2004, ISSN: 1941-0042.