AGIC: Approximate Gradient Inversion Attack on Federated Learning

Jin Xu, Chi Hong, Jiayue Huang, Lydia Y. Chen, Jérémie Decouchant
Delft University of Technology, The Netherlands
j.xu-21@student.tudelft.nl, {c.hong, j.huang-4, y.chen-10, j.decouchant}@tudelft.nl

Abstract—Federated learning is a private-by-design distributed learning paradigm where clients train local models on their own data before a central server aggregates their local updates to compute a global model. Depending on the aggregation method used, the local updates are either the gradients or the weights of local learning models, e.g., FedAvg aggregates model weights. Unfortunately, recent reconstruction attacks apply a gradient inversion optimization on the gradient update of a global round, the server is able to update the global model by aggregating all the gradients or model updates it has received. At the end of a global round, the server first sends the most recent model to clients, which then train the local model on their private data and send gradients or model updates back to the server. At the end of a global round, the server is able to update the global model by aggregating all the gradients or model updates it has received.

An attacker that would compromise the server would observe model parameters and their updates, but would not have access to training samples. Therefore, FL is often assumed to safely protect the clients’ local data. However, recent works have shown that the observation of model parameters and gradient updates might allow attributes of local samples to be leaked [6], class representatives to be inferred [7], [8], and even real training samples to be reconstructed [9], [10].

Gradient inversion attacks that directly reconstruct training samples based on a model and gradient updates on it result in the most serious data leakages. Most gradient inversion attacks are optimization-based. In an optimization-based gradient inversion attack, the adversary randomly initializes dummy samples, and executes forward and backward propagation on them to obtain dummy gradients. The dummy samples are then optimized to minimize the sum of the distance between the observed real gradients and the dummy gradients, and regularization terms. To improve the reconstruction performance, some works define new distance functions and regularization terms [10], while others exploit prior knowledge, e.g., by using batch normalization data [11] or pre-trained generative models [12].

Gradient inversion attacks are designed to reconstruct samples from gradient updates. However, in practice, federated learning systems often use federated averaging (FedAvg) [4] where clients send model updates after conducting multiple local steps, each executed over a mini-batch, in order to further reduce the communication overhead. Few previous works have discussed how to attack model updates generated by FedAvg. These works either use simulation [10], which is slow and incompatible with label inference, a sub-task that significantly improves the reconstruction performance [13], or make an averaging approximation [14] to estimate the gradient of a full batch for FedAvg, which does not extend to the most general case where mini-batches are used.

In this paper, we describe AGIC, a novel Approximate Gradient Inversion Attack that specifically targets federated learning systems based on FedAvg’s model updates, and is also compatible with gradient updates. Overall, AGIC makes the following contributions.

First, to avoid the computational overhead of simulation attacks, AGIC leverages a one-batch approximation, which assimilates multiple local steps executed over multiple mini-batches as a single local step executed over a single aggregated mini-batch. The corresponding gradients of that larger mini-batch can also be approximated based on the received model update. Thanks to this one-batch approximation, AGIC is faster than previous simulation-based attacks on FedAvg and is compatible with label inference.

I. INTRODUCTION

Federated learning (FL) [1]–[5] is a popular collaborative learning paradigm that aims at providing accurate predictive models while preserving the clients’ data privacy and reducing communication costs. In a FL system, the training data of a client never leaves its initial premises. In each global round, the server first sends the most recent model to clients, which then train the local model on their private data and send gradients or model updates back to the server. At the end of a global round, the server is able to update the global model by aggregating all the gradients or model updates it has received.

An attacker that would compromise the server would observe model parameters and their updates, but would not have access to training samples. Therefore, FL is often assumed to safely protect the clients’ local data. However, recent works have shown that the observation of model parameters and gradient updates might allow attributes of local samples to be leaked [6], class representatives to be inferred [7], [8], and even real training samples to be reconstructed [9], [10].
Second, based on the observation that the attacker is able to observe updates across multiple epochs, AGIC better exploits the information contained in multiple updates that have been computed over common data samples. AGIC matches the collected updates with specific training samples and jointly optimizes with the updates to improve the reconstruction quality of the specific training samples.

Finally, AGIC assigns different weights to the gradients of different layers in the distance function to optimize, inspired by the results of recent works on model compression that have shown that model layers have unequal effects on model accuracy and other metrics [15], [16]. Interestingly, AGIC uses a layer weight modifier for convolutional neural networks (CNN) that use ReLU as activation function, to balance the contribution of each convolution layer, because ReLU imports zeros into gradients.

Our experiments with three datasets show that AGIC outperforms two representative state-of-the-art baselines in both reconstruction quality and efficiency. When attacking gradient updates, AGIC’s peak signal-to-noise ratio is up to 50% higher than with both baselines. When attacking FedAvg model updates configured with 8 local steps, AGIC reconstructs samples five times faster than the only applicable baseline, a state-of-the-art simulation-based attack.

The remainder of this paper is organized as follows. Section II provides some background on federated learning and gradient inversion attacks. Section III introduces our system and threat models, and an overview of AGIC. Section IV describes AGIC’s three key features. Section V evaluates AGIC’s performance. Finally, Section VII concludes this paper.

II. BACKGROUND

A. Federated learning

Federated learning (FL) [2], [5] is a collaborative learning paradigm that allows distributed data owners (called clients) to jointly learn machine learning models without centrally pooling their data. Clients train the model using their own local data and rely on a central server to aggregate their learning results. The FL system progresses over multiple global rounds. At the beginning of a global round, the server sends the latest global model to clients. Clients then train the latest model on their own data, and send their training results, i.e., gradients or model updates, back to the server.

The local training process typically follows the stochastic gradient descent (SGD) algorithm, which iterates through the data using batches. A batch might contain the full local dataset, but in the more general case a batch is a subset of the local dataset and is also called a mini-batch. During an epoch, clients iterate over their complete training data once. A client may use its full dataset in a single batch during one round, which is then regarded as an epoch. We focus on the more general mini-batch case, where the full dataset is split into multiple mini-batches, and the client iterates over these mini-batches in multiple global rounds during an epoch.

Figure 1 depicts two typical local training procedures of FL learning systems that respectively send gradient updates after one local step (top), or send model updates after multiple local steps (bottom), which is the case with federated averaging (FedAvg) [4]. With the first method, the clients execute a single local training step on a batch of local samples and their labels \((X, y)\) with the latest received global model \(W\), and send back the corresponding gradient update \(\nabla W\). With FedAvg, the clients run \(T\) local steps on \(T\) batches, which means that they will update their local model parameters \(T\) times, and then send the trained model weights \(W_T\) back to the server. In practical scenarios, FedAvg is more commonly used because it reduces the amount of data transmitted over the network, in particular because a client can process several batches during a global round, i.e., before sending its results to the server. In the rest of this paper, we will note \(\Delta W = W_T - W\), the difference between the updated model sent by the client at the end of a round and the global model it received at its beginning.

B. Gradient inversion attacks

Even though the clients and the server only exchange intermediate learning results such as models or gradient updates, it has been shown that sensitive information from local training datasets can be inferred in FL systems [6], [8]. Furthermore, it has been shown that an attacker that obtains gradient updates can launch a gradient inversion attack to recover the training samples that were used by clients to generate the gradients [9]. Gradient inversion attacks generally assume the attacker is an honest-but-curious server [10].

Most existing gradient inversion attacks solve an optimization problem, as illustrated in Figure 2a. After retrieving gradient updates \(\nabla W\), the attacker generates dummy samples \((\hat{X}, \hat{y})\) and minimizes the distance between the received gradients \(\nabla W\) and its dummy gradients \(\nabla \hat{W}\), which are retrieved by feeding dummy samples through the obtained model in one forward-backward pass. During the optimization process, the values of the dummy samples are optimized, so that at the
Gradient inversion attacks based on gradient updates and model updates.

End of the attack the dummy samples approximate the training samples.

An optimization-based gradient inversion attack typically relies on an optimization objective such as:

$$\min \text{Dist}(\frac{\partial L(F(W, \hat{X}), \hat{y})}{\partial W}, \nabla W) + \text{Reg}(\hat{X}, \hat{y})$$

While most gradient inversion attacks can only be applied to gradient updates, a recent attack on model updates [10] simulates the execution of $T$ local steps in each optimization iteration, retrieves the dummy model update $W'_1$ after the last local step, and optimizes the dummy samples in all mini-batches $\hat{X}_1, \ldots, \hat{X}_T$ based on the difference between the dummy model and the observed model update $\Delta W'_1 = W'_1 - W$ and the real observed model difference $\Delta W_T$. Figure 2b illustrates a simulation-based gradient inversion attack based on FedAvg model updates.

**III. OVERVIEW OF AGIC**

This section defines our system and threat models, and provides an overview of AGIC, our gradient inversion attack.

**A. System and adversarial models**

We consider a FL system with one server that aggregates updates from multiple clients. We assume all clients to be correct and that the server is honest-but-curious. Therefore, our adversary observes the communications between the server and the clients and can retrieve both models and model updates. The attack can occur at any phase of the training process to target untrained, in-training, or well-trained networks. The attacker may observe updates from different epochs during training. During an epoch a client iterates over its whole dataset, possibly using different mini-batches.

We focus on attacking convolutional neural networks (CNN), which are used for image classification, one of the most popular uses of FL systems. For example, it can be used by medical institutes to collaboratively train a diagnosis model [17] without sharing patient-related information.

**B. Attack overview**

To successfully run gradient inversion attacks on FedAvg, AGIC relies on three key features: the one-batch approximation for federated averaging, the leveraging of model updates from multiple epochs, and the assignment of layer weights in the distance function.
The base loss function used during optimization. The base loss function that AGIC uses to reconstruct images from an update is the negative cosine distance between the dummy gradients \( \nabla W' = \frac{\partial L(W,X,y)}{\partial W} \) and the received real gradients \( \nabla W \), which is added to a total variation (TV) regularization term that reduces image noises with weight \( \zeta_{TV} \) [18]:

\[
\min_{\hat{X}} 1 - \cos(\nabla W', \nabla W) + \zeta_{TV} TV(\hat{X})
\] (2)

\[
\cos(\nabla W', \nabla W) = \frac{\nabla W' \cdot \nabla W}{\|\nabla W'\|_2 \|\nabla W\|_2}
\] (3)

The dummy samples are optimized based on the gradient values from back propagation of the loss. AGIC modifies this base loss function to adapt to different attack scenarios. AGIC is a novel approximate gradient inversion attack that is applicable to both model updates and gradient updates. We focus on the more challenging model update scenario, but our methods (namely, leveraging multiple epochs and choosing layer weights) also apply to gradient updates.

IV. KEY IMPLEMENTATION FEATURES OF AGIC

This section details AGIC’s three key features that enable its use with FedAvg and increase its accuracy: one-batch approximation, use of multiple updates, and layer weights.

A. One-batch approximation for FedAvg

As previously explained, accurately reconstructing samples from FedAvg’s model updates is more difficult than with gradient updates because a model update is computed using more samples. When attacking FedAvg with possibly multiple local steps, AGIC is faster and more accurate than simulation-based gradient inversion attacks thanks to its one-batch approximation. This approximation allows AGIC not to use computationally expensive simulation and to be compatible with label inference, as there is no need to assign inferred labels to different local steps.

The one-batch approximation is equivalent to assuming that the model remains mostly identical after each local step. This assumption has been discussed in large mini-batch training [19]. A similar first-order approximation has been used in meta learning [20]. It should be noted that if model parameters change drastically in each local step, e.g. when the learning rate of local SGD is \( 1 \times 10^{-2} \), the one-batch approximation should not be used. In our experiments, with a learning rate of \( 1 \times 10^{-4} \), the approximation brings good performance (cf. V-B).

Figure 4 illustrates AGIC’s one-batch approximation on FedAvg’s updates. Compared to the multi-step simulation of [10], which is presented in Figure 2b, AGIC’s approximation only runs one local step with the aggregated batch, and its loss function is based on approximated gradients instead of models difference \( \Delta W \). More precisely, given all the mini-batches used in one FedAvg update \( [X_1, \ldots, X_n] \) and their inferred labels \( y_1, \ldots, y_n \), and local learning rate \( \mu \), AGIC makes the approximation that all the mini-batches compose a large batch \( X_A = [X_1, \ldots, X_n] \) to produce the received model update:

\[
\Delta W = \sum_{j=1}^n \mu \frac{\partial \mathcal{L}(F(W, X_j), y_j)}{\partial W} = -\mu \frac{\partial \mathcal{L}(F(W, X_A), y_A)}{\partial W}
\] (4)

Based on Formula 4, AGIC computes the approximated gradients \( \nabla W = \frac{\Delta W}{\Delta W} \) for aggregated batch \( X_A \) from the model updates given then local learning rate. Then AGIC initializes dummy samples \( \hat{X}_A \) as the size of \( X_A \), and optimizes on the distance between dummy gradients from \( \hat{X}_A \) and the computed gradients from received model updates. Because the objective function is the distance between gradients, any improvement method designed for gradient updates can...
with more pairs of model parameters it observes. Better reconstruct the samples by making use of the updates specific samples. Can therefore observe several updates that are based on some

B. Leveraging updates from multiple epochs

AGIC be directly applied on FedAvg model updates with AGIC’s approximation.

Fig. 5: Reconstruction from updates from multiple epochs. We use updates with batch size 1 for illustration purposes.

be directly applied on FedAvg model updates with AGIC’s approximation.

B. Leveraging updates from multiple epochs

A training sample is used in many global rounds, each round producing an update, and thus is involved in multiple updates that are communicated to the server. The attacker can therefore observe several updates that are based on some specific samples. AGIC demonstrates that the adversary can better reconstruct the samples by making use of the updates it observes.

Prior work by Geng et al. has shown that reconstruction quality of a specific batch is improved if jointly optimizing with more pairs of model parameters $W_F$ and their corresponding gradient update from that batch $W_F'$ [14]. However, this method only works on full batch gradient descent and does not provide a solution to the general case where the client uses multiple mini-batches to go over its data as in an epoch. In contrast, AGIC is able to reconstruct training samples using multiple model updates or gradient updates from different mini-batches across epochs during training.

AGIC processes updates from multiple epochs in two steps: (i) update matching, which matches the collected updates whose corresponding samples are overlapping; and (ii) joint reconstruction, which reconstructs the samples from the matching updates. We discuss those two steps in the following, and Figure 5 illustrates them with a batch size equal to 1.

Step 1: Update matching. AGIC first matches updates that have overlapping samples so that they can be jointly reconstructed afterwards. If the training process does not shuffle the samples used between epochs, or if the entire local dataset is used in a full batch, then the matching process is trivial. In the general case, a more complex algorithm is required to match gradient updates of single mini-batches or FedAvg model updates based on multiple mini-batches. AGIC’s matching mechanism first conducts a reconstruction process for each update to obtain a first reconstruction of images, and then identifies the most similar pairs of reconstructed images.

As the adversary observes updates from two complete epochs, the pre-reconstructed samples from the first epoch’s updates can be mapped one-to-one with the samples from the updates of the second epoch. Equivalently said, for any sample $X_s$, two updates from the two epochs have been trained on it.

To match pairs of reconstructed images, we evaluate the similarity of pairs of images by processing the images with an average pooling layer that takes average value of each small part of an image in order to mitigate noises, and computing their mean squared error. We refer the reader to §V-D, which demonstrates the performance improvement that using a pooling layer brings.

We use a simple greedy algorithm to find a one-to-one mapping between reconstructed images from two epochs. First, we measure the similarity of each image pair and sort them. From the highest similarity to the lowest one, we check each pair of still unmatched images and associate them to each other. Using this method, all pre-reconstructed images from one epoch can be matched to those from another epoch. For any matched image pairs, which are then considered to reconstruct the same sample, AGIC executes a joint reconstruction using the all updates they participated in (Step 2).

It is easy to extend this method to leverage more than two epochs, by successively computing matching update pairs from consecutive epoch pairs. A specific sample is then associated to the image pairs reconstructed from updates from consecutive epochs, and to one reconstructed image per epoch. A sample is therefore associated to exactly one update per epoch.

Since label inference can be applied before optimizing for each update, AGIC uses a label-based filter. Based on their inferred label, AGIC computes the similarity between image pairs that have the same inferred label to reduce the search space and limit the possibility of incorrect matching, thereby increasing accuracy. For a multi-sample batch, generated either from gradient update directly or from the one-batch approximation of model update, label inference returns a list of labels for all samples. Therefore, whether two samples can be matched depends on whether their inferred label lists share at least a common label.

Step 2: Joint reconstruction with multiple updates. AGIC jointly optimizes updates that have been identified as having been trained over a common sample, in order to improve reconstruction quality. We use gradients during the optimization process, obtained either from the one-batch approximation of model update or directly from received gradient update. Our distance function uses the summation of distance of all pairs of input gradients and dummy gradients, with weights $\gamma_i$ assigned. A decreasing weight is assigned to the gradients generated later during the training process, because the reconstruction quality of gradient inversion attacks is higher in the earliest training phases. These gradient weights are set as experimental hyperparameters ($\S$V details the values
do the reconstructed gradients $\nabla W'$. AGIC’s negative cosine similarity objective with layer weights $\alpha_i$ is indicated in Formula 6.

$$\min_x \frac{1}{N_{\text{conv}}} \sum_i \alpha_i (\nabla L_i' \cdot \nabla L_i)$$  \hspace{1cm} (6)

**Linear assignment.** It is impractical to separately select a weight for each layer using tedious methods like grid search. Therefore, we use a function to assign the weights to convolutional layers, and set weights of fully connected layers to the average of convolutional layer weights.

Both the shape and the range of the weight function affect performance. For the range, we first determine that the function passes through points $(1, 1)$ and $(N_{\text{conv}}, \beta)$. Parameter $\beta$ controls the range of the function and $\beta = 1$ implies homogeneous weights. We also look into various function shapes, such as linear functions $y=ax+b$, exponential-like functions $y=ae^x+b$ and logarithmic-like functions $y=\log x+b$ to cover various function shapes (e.g., concave or convex). The two points $(1, 1)$ and $(N_{\text{conv}}, \beta)$ determine values of coefficients $a, b$ in the various functions.

**Table I** presents some results obtained with different function shapes. We experimentally observe that an increasing linear weight function maximizes the reconstruction performance, and therefore determines a layer’s weight using Formula 7.

$$l_i = 1 + \frac{\beta - 1}{N_{\text{conv}}} (i - 1)$$  \hspace{1cm} (7)

In practice, the properties of the attacked network change during the training process. For example, the magnitude of gradient values becomes smaller and the reconstruction quality deteriorates. In consequence, the best value for parameter $\beta$ changes according to the attacked training phase (as illustrated in Figure 9).

**ReLU modifier.** We can directly set $\alpha_i=l_i$ in the general case, but we additionally propose a layer weight modifier for CNNs that apply ReLU after convolution layers, as $\text{ReLU}(x) = \max(0, x)$ is the most commonly used activation function in current CNN designs such as ResNet [22]. ReLU is more popular than sigmoid functions, because it simplifies gradient computation in backward propagation and avoids gradient vanishing and gradient inversion.

When the input of ReLU is negative, its output is set to zero, and as a result, after back propagation, the gradients of input are also equal to zero. Therefore, a proportion of zeros exists in each convolution layer’s gradients. These zeros limit the performance of gradient inversion attacks, because it is then easier for the optimization process to get its dummy gradients to achieve these zeros, which appear as soon as the corresponding output of a convolutional layer is negative. Thus, layers that have more zero gradients contribute less in the reconstruction.

To remove the effect of zeros imported by ReLU and balance the contribution of each convolutional layer, AGIC
multiplied by a zero-based weight \( z \).

The linearly assigned weights of convolutional layers:

\[ \text{EPYC 7542 32-core CPU, 256GB of RAM and an NVIDIA GeForce RTX 2080 Ti GPU for our experiments.} \]

**Datasets.** We conduct experiments on three image datasets: CIFAR-10 (size \( 32 \times 32 \), 10 classes) [23], CIFAR-100 (size \( 32 \times 32 \), 100 classes) [23], and ImageNet (size \( 224 \times 224 \), 1000 classes) [24]. All three datasets contain 3-channel colored images. The datasets are normalized to mean 0 and standard deviation 1. For the experiments, we randomly select 100 minibatches from the validation set of each dataset. We attack ResNet trained on the datasets [22]: ResNet20-4 on CIFAR-10 and CIFAR-100, and ResNet50 on ImageNet.

**Baselines.** We compare our attack with two baselines. The first is an improved version of DLG [9], namely DLG-Adam, which uses the Adam optimizer instead of L-BFGS in the original version, because L-BFGS often fails to converge while attacking large networks like ResNet [10]. The other baseline is Inverting Gradients [10], which we call InvG. The two baselines cover the two main distance functions applied in gradient inversion attacks.

We do not compare our attack to those based on pretrained models, e.g., an image generative model [12], because we do not assume the attacker to have access to additional knowledge. We also do not consider a batch normalization (BN) regularization term [11], because it requires that each synchronous BN layer has cross-node communication during forward propagation, which is very time-consuming and impractical in FL. Our methods are however orthogonal to these methods that leverage extra information and could be combined with them.

**Hyperparameters.** The Adam optimizer with learning rate 0.1 is used based on the gradient values on dummy samples during optimization. \( \gamma_{TV} \) is equal to \( 10^{-4} \) for untrained ResNet20-4, \( 10^{-2} \) for trained ResNet20-4, \( 10^{-3} \) for untrained ResNet50, and \( 10^{-1} \) for trained ResNet50. For FedAvg, the learning rate of local SGD is \( \mu = 1 \times 10^{-4} \). The kernel size of the averaging pooling layer used in update matching is \( 2 \times 2 \) and the stride of it is \( 2 \). Regarding epoch weights \( \gamma_{e} \) in multi-epoch reconstruction, \( \gamma = 1 \) for a batch from the first epoch, and \( \gamma = 0.1 \) for the other batches. For layer weights, \( \beta = 50 \) for untrained networks, and \( \beta = 2 \) for trained networks. The optimization has 10,000 iterations.

Because label inference is an independent subtask that has already been well explored and for which very accurate methods already exist, we assume that labels are correctly inferred in experiments of gradient updates, if not specified in other ways. For our experiments with FedAvg, we apply the label inference method from [11]. Labels in a mini-batch are assumed not to be duplicated, following the experiment setup of previous works [10], [11].

**Metrics.** We use three metrics to measure the similarity between the reconstructed images and the real training samples: (i) peak signal-to-noise ratio (PSNR), (ii) structural similarity index measure (SSIM) [25], and (iii) LPIPS, a perceptual image similarity score [26]. Higher PSNR, higher SSIM, and lower LPIPS all indicate better reconstruction quality.

![Fig. 6: Reconstructed CIFAR-10 images from untrained ResNet20-4 with batch size 1.](image)

![Fig. 7: Reconstructed ImageNet images from untrained ResNet50 with batch size 1.](image)

relied on a layer weight modifier based on the proportion of zeros of each layer. Assuming that the proportion of zeros in a convolutional layer’s gradients is \( p_i \), its modifier zero-based coefficient is computed as \( z_i = \frac{1}{1 - p_i} \).

In conclusion, the layer weight for the \( i \)-th convolutional layer is equal to a linearly assigned weight \( l_i \), which is multiplied by a zero-based weight \( z_i \) if ReLU is used, and the layer weight of a fully connected layer is the average of the linearly assigned weights of convolutional layers:

\[
\alpha_{i} = l_{i} \cdot \frac{1}{1 - p_{i}} \quad \text{for the } i\text{-th convolutional layer} \\
\alpha_{fc} = \frac{\sum_{i=1}^{N_{\text{conv}}} l_{i}}{N_{\text{conv}}} \quad \text{for fully connected layers}
\]

**V. PERFORMANCE EVALUATION**

**A. Setup**

**Hardware.** We use a machine equipped with an AMD EPYC 7542 32-core CPU, 256GB of RAM and an NVIDIA Hardware. We use a machine equipped with an AMD EPYC 7542 32-core CPU, 256GB of RAM and an NVIDIA
TABLE II: Reconstruction quality of gradient inversion attacks on untrained ResNet20-4 on CIFAR-10 and CIFAR-100, and trained ResNet50 on ImageNet, on FedAvg model updates. Here, FedAvg runs on 4 size-1 mini-batches and 4 local steps.

| Method                   | Simulation | CIFAR-10, untrained | CIFAR-100, untrained | ImageNet, trained |
|--------------------------|------------|---------------------|----------------------|------------------|
|                          |            | PSNR↑ | SSIM↑ | LPIPS↓ | PSNR↑ | SSIM↑ | LPIPS↓ | PSNR↑ | SSIM↑ | LPIPS↓ |
| DLG-Adam                 | ✓          | 13.611 | 0.391 | 0.123 | 15.114 | 0.463 | 0.096 | 8.044 | 0.028 | 1.314 |
| InvG                     | ✓          | 14.108 | 0.414 | 0.117 | 15.465 | 0.480 | 0.092 | 10.593 | 0.167 | 0.779 |
| AGIC (this work, one-batch) | ×          | 14.250 | 0.414 | 0.115 | 15.437 | 0.475 | 0.093 | 10.522 | 0.161 | 0.791 |
| AGIC (this work, one-batch + layers) | ×          | 16.182 | 0.529 | 0.112 | 19.133 | 0.672 | 0.050 | 10.699 | 0.177 | 0.780 |

TABLE III: Reconstruction quality of gradient inversion attacks on FedAvg, with local step 4 and batch size 2, on ResNet20-4 on CIFAR-100. The matching batches from two epochs are assumed to have the same samples. A ‘1’ in the Epochs column means that the updates are collected after one epoch, while ‘1, 2’ means that the matching updates are collected after one epoch and two epochs.

| Method                   | Simulation | Epochs | PSNR↑ | SSIM↑ | LPIPS↓ |
|--------------------------|------------|--------|-------|-------|--------|
| DLG-Adam                 | ✓          | 1      | 12.196 | 0.392 | 0.087  |
| InvG                     | ✓          | 1      | 14.398 | 0.419 | 0.065  |
| AGIC (this work)         | ×          | 1      | 15.653 | 0.486 | 0.055  |
| AGIC (this work)         | ×          | 1.2    | 16.664 | 0.533 | 0.040  |

TABLE IV: Reconstruction quality of gradient inversion attacks on single gradient updates, reconstructing CIFAR-10 images from untrained ResNet20-4 and ImageNet images from untrained ResNet50.

| Dataset | Batch size | Method             | PSNR↑ | SSIM↑ | LPIPS↓ |
|---------|------------|--------------------|-------|-------|--------|
| CIFAR-10 | 1          | DLG-Adam           | 19.908 | 0.727 | 0.034  |
|         |            | InvG               | 20.671 | 0.753 | 0.031  |
|         |            | AGIC (this work)   | 31.341 | 0.963 | 0.004  |
| CIFAR-10 | 4          | DLG-Adam           | 14.034 | 0.412 | 0.110  |
|         |            | InvG               | 14.421 | 0.433 | 0.103  |
|         |            | AGIC (this work)   | 17.183 | 0.586 | 0.092  |
| ImageNet | 1          | DLG-Adam           | 11.311 | 0.094 | 0.751  |
|         |            | InvG               | 12.832 | 0.214 | 0.691  |
|         |            | AGIC (this work)   | 15.573 | 0.366 | 0.670  |
| ImageNet | 4          | DLG-Adam           | 9.879  | 0.070 | 0.801  |
|         |            | InvG               | 11.640 | 0.216 | 0.720  |
|         |            | AGIC (this work)   | 12.836 | 0.302 | 0.721  |

B. Overall performance of AGIC

We first report the overall performance of AGIC on FedAvg model updates and Gradient updates. Table II shows the attack results on single FedAvg model updates. The one-batch approximation and layer weights are used with this FL setting. AGIC with the one-batch approximation outperforms the other two baselines that are based on simulation of FedAvg. Table III further shows the attack results on FedAvg with updates from multiple epochs. One can also observe that AGIC further increases its reconstruction quality when reconstructing from matching updates from multiple epochs.

AGIC can also attack gradient updates, the most common setup in gradient inversion attacks, as shown in Table IV.

Fig. 8: Reconstruction qualities and computation times for the simulation-based attack InvG and for AGIC on CIFAR-10 images from untrained ResNet20-4. Each model update is generated with 8 images, e.g., for the 8-step scenario each update is generated from 8 size-1 mini-batches.

These results are interesting because only layer weights are used in this experiment, and AGIC significantly outperforms baseline methods, with different datasets and batch sizes. AGIC improves the PSNR over CIFAR-10’s by up to almost 50%. The reconstructed CIFAR-10 samples in Figure 6 are visually almost identical to the ground truth. Figure 7 shows two examples of reconstructed ImageNet samples, where AGIC also clearly outperforms the baselines.

C. Computation time reduction with the one-batch approximation

We further compare the one-batch approximation of AGIC with the simulation approach of InvG, for attacking FL systems using FedAvg. We look into the case where a FedAvg update is generated from 8 images, but they are divided into mini-batches with different sizes in different experiments, which implies that the number of local steps also changes. Figure 8 show that AGIC outperforms InvG in both quality and speed with all settings. It can also be noticed that both methods have worse performance when the number of local steps increases, if the total number of samples are fixed.

When the number of local steps increases, the time cost of attacks based on simulation also increases, while the one-batch approximation implies that AGIC has a constant time cost. When FedAvg has 8 local steps, AGIC is 5 times faster than InvG: AGIC requires 6.7 minutes on an NVIDIA GeForce RTX 2080 Ti GPU instead of 33.2 minutes with InvG.
TABLE V: Matching success rate of 1000 CIFAR-100 images from epochs 1 and 2, with different update matching methods and batch sizes.

| Labels | Pooling | Batch size 1 | Batch size 4 |
|--------|---------|--------------|--------------|
| ×      | ×       | 0.819        | 0.509        |
| ✓      | ×       | 0.873        | 0.547        |
| ✓      | ✓       | **0.988**    | 0.672        |
| ✓      | ✓       | 0.981        | **0.705**    |

TABLE VI: Reconstruction quality with 200 CIFAR-100 images using gradients from epochs 1 and 2, using label inference and update matching, and equal layer weights. The model is ResNet20-4 and the batch size equals 1.

| Epochs | PSNR↑ | SSIM↑ | LPIPS↓ |
|--------|-------|-------|-------|
| 1      | 20.379| 0.665 | 0.026 |
| 2      | 19.085| 0.626 | 0.022 |
| 1,2    | **21.464** | **0.697** | **0.016** |

D. Reconstruction quality improvement with update matching and joint reconstruction

We now focus on reconstructing samples from multiple epochs. We first conduct experiments on our pre-reconstruction matching method with 2,000 iterations. We consider matching with and without using an average pooling layer, and with and without restriction on inferred labels. Table V presents the matching rate with pre-reconstructed CIFAR-100 images. It shows that with pre-reconstruction, the matching rate is very high (0.988) when the batch size is 1. When the batch size is 4, the matching rate decreases (0.672), because reconstruction quality decreases when the batch size increases, but it is still high. Using inferred labels provides strong prior knowledge for gradient matching, even when the batch size is larger than 1, which implies that labels may be incorrectly inferred. A pooling layer successfully mitigates noise, except in the case where the batch size equals 1 and labels are used, which already has an almost perfect accuracy.

We further evaluate on a full setup that includes label inference that ranges from ReLU weight modifier \( \beta > 0 \) and without ReLU weight modifier \( \beta \). The zero-proportion-based weight modifier consistently improves the attack performance, bringing up to 2.0% improvement on PSNR.

VI. RELATED WORK

A. Gradient inversion attack on gradient updates

Gradient inversion attacks on federated learning systems reconstruct training samples from gradient updates. Deep Leakage from Gradients (DLG) [9] is the first gradient inversion attack. DLG jointly optimizes on the samples and on the labels, which complicates the optimization objective and impairs the reconstruction quality if the labels are inaccurately optimized. Improved DLG (iDLG) [13] proposes to conduct label inference before DLG’s optimization and shows that labels can be inferred with perfect accuracy using some analytical method when the batch size is equal to 1. Following iDLG, several label inference methods explained how to infer labels in larger batches with higher accuracy [11], [14], [29]. Label inference is now an essential part of current gradient inversion attacks since it significantly improves the reconstruction quality.

Most existing gradient inversion attacks are optimization-based; they formulate an optimization problem and minimize the distance between the real observed gradient update and a dummy gradient update in order to optimize the reconstructed samples. Therefore, one way to improve the performance of reconstructed images of one CIFAR-10 sample on untrained ResNet20-4, with increasing \( \beta \) values. One can notice that the reconstruction quality improves when \( \beta \) increases.

Figure 9 also compares attacking untrained ResNet20-4 with and without ReLU weight modifier \( \beta \). The zero-proportion-based weight modifier consistently improves the attack performance, bringing up to 2.0% improvement on PSNR.

Fig. 9: Effect of different layer weights on PSNR with different experimental settings, including untrained and trained networks.

Fig. 10: Reconstructions of a CIFAR-10 image of a horse from untrained ResNet20-4 with a weight ratio \( \beta \) that ranges from 1 (left) to 20 (right).
Table VII: Comparison of AGIC and other gradient inversion attacks on (horizontal) federated learning systems.

| Method                          | Main loss term | Other loss terms | Optimizer | Label inference capability | Extra knowledge | Trained network | Multiple epochs’ updates | FedAvg |
|---------------------------------|----------------|------------------|-----------|---------------------------|-----------------|-----------------|----------------------|--------|
| DLG [9]                         | L2             | None             | L-BFGS    | No use                    | No              | ×               | ×                    | ×      |
| iDLG [13]                       | L2             | None             | L-BFGS    | One                       | No              | ×               | ×                    | ×      |
| Inverting Gradients (InvG in this paper) [10] | Cosine | Total variation | Adam      | One                       | No              | ✓               | ✓                    | ✓      |
| CPL [27]                        | L2             | Label-based      | L-BFGS    | One                       | No              | ×               | ×                    | ×      |
| R-GAP [28]                      | Recursively    | N/A              | N/A       | One                       | No              | ×               | ×                    | ×      |
| Grad-Inversion [11]             | L2             | Total variation, L2 of input, BN, group consistency | Adam | Multiple | BN statistics, pretrained image alignment model | ✓ | × | × | |
| Geng et al. [14]                | L2             | Clipping, scaling | L-BFGS    | Multiple | No | × | × | Only full batch |
| GIAS [12]                       | Cosine         | Total variation | Adam      | One | Pretrained generative model | × | × | × | |
| AGIC (this work)                | Cosine         | Total variation | Adam      | Multiple | No | ✓ | ✓ | ✓ | |

A gradient inversion attack is to design a better objective function. Geiping et al. uses the cosine distance instead of L2 distance in DLG [10]. CPL adds a label-based regularization term to improve the optimization stability [27]. Another way to improve the reconstruction performance is to leverage prior knowledge in specific scenarios. GradInversion adds a batch normalization regularizer, because the attacker is assumed to know batch normalization statistics, and it designs a group consistency regularization term leveraging multiple reconstruction processes with different initialization seeds to find an enhanced reconstruction result [11]. GIAS assumes a known data distribution and improves the reconstruction quality using a pre-trained generative model [12].

R-GAP is a gradient inversion attack that is not based on optimization. Instead, R-GAP recursively reconstructs each layer’s input from the last layer to first by solving linear equations, which is limited to a batch size equal to 1 [28]. The works we have discussed so far consider horizontal FL systems [3], [30] where client datasets share a given feature space. Differently, CAFE is a gradient inversion attack [31] for vertical FL systems where client datasets have different feature spaces [32], [33].

Table VII compares our attack, AGIC, with existing gradient inversion attacks, and shows that it is applicable in more general federated learning scenarios.

B. Gradient inversion attack on FedAvg model updates

Although using FedAvg model updates is more practical than using gradient updates, only a handful of gradient inversion attacks on FedAvg’s model updates have been described.

To be compatible with the multiple local steps of FedAvg, Geiping et al. simulate the local training process of multiple local steps to obtain a dummy model update, and then minimize the distance between the dummy model update and the real model update [10]. However, this method is incompatible with label inference because it is unable to associate correct labels to each mini-batch, as label inference returns a set of labels for all mini-batches. In addition, the computation time of simulation-based attacks also increases with the number of local steps FedAvg uses, because the computation graph for a model update then also gets larger [10]. Geng et al. present an attack that targets FedAvg systems that use full batch gradient descent [14]. Their attack divides the received model update by the number of local steps to approximate the model update of each local step, which is then used to approximate the full batch’s gradient update. However, this attack cannot cover the common case where FedAvg uses multiple mini-batches.

AGIC addresses the limitations of previous attacks on FedAvg. First, its one-batch approximation is compatible with label inference. Second, AGIC can be used in common FedAvg scenarios that use several mini-batches per round, and it is faster than simulation-based attacks.

VII. CONCLUSION

In this paper, we presented AGIC, an accurate and fast gradient inversion attack that can leverage both model updates and gradient updates across multiple epochs. AGIC uses a one-batch approximation to convert model updates into approximate gradient updates of a larger mini-batch. AGIC also leverages updates from multiple epochs with update matching to jointly reconstruct specific training samples accurately. Finally, AGIC assigns different weights to layers in the objective function, which significantly improves the reconstruction quality. Compared to previous works, AGIC can be used with more general federating learning system settings. Our experiments demonstrate that AGIC reconstructs samples with up to 50% PSNR improvement compared to state-of-the-art baselines, and is up to 5x faster than simulation-based attacks, which is the only baseline that can attack FedAvg.
