

# LOKI: Large-scale Data Reconstruction Attack against Federated Learning through Model Manipulation

Joshua C. Zhao<sup>1</sup>, Atul Sharma<sup>1</sup>, Ahmed Roushyd Elkordy<sup>2</sup>, Yahya H. Ezzeldin<sup>2</sup>

Salman Avestimehr<sup>2</sup>, Saurabh Bagchi<sup>1</sup>

<sup>1</sup>Purdue University, <sup>2</sup>University of Southern California

{zhao1207, sharm438, sbagchi}@purdue.edu, {aelkordy, yessa, avestime}@usc.edu

**Abstract**—Federated learning was introduced to enable machine learning over large decentralized datasets while promising privacy by eliminating the need for data sharing. Despite this, prior work has shown that shared gradients often contain private information and attackers can gain knowledge either through malicious modification of the architecture and parameters or by using optimization to approximate user data from the shared gradients.

However, prior data reconstruction attacks have been limited in setting and scale, as most works target FedSGD and limit the attack to single-client gradients. Many of these attacks fail in the more practical setting of FedAVG or if updates are aggregated together using secure aggregation. Data reconstruction becomes significantly more difficult, resulting in limited attack scale and/or decreased reconstruction quality. When both FedAVG and secure aggregation are used, there is no current method that is able to attack multiple clients concurrently in a federated learning setting.

In this work we introduce LOKI, an attack that overcomes previous limitations and also breaks the anonymity of aggregation as the leaked data is identifiable and directly tied back to the clients they come from. Our design sends clients customized convolutional parameters, and the weight gradients of data points between clients remain separate even through aggregation. With FedAVG and aggregation across 100 clients, prior work can leak less than 1% of images on MNIST, CIFAR-100, and Tiny ImageNet. Using only a single training round, LOKI is able to leak 76-86% of all data samples.

## 1. Introduction

Federated learning (FL) [1] is a machine learning paradigm introduced to address growing user data privacy concerns where clients participate in large-scale model training without sending their private data to servers for centralized training. A general training round in FL consists of a server sending the model out to the clients and clients training the received model with their private data before sending their updates back to the server. The server then aggregates the updates from the clients and uses the aggregate to update the global model before sending it out to repeat another iteration of the training process. Two primary methods of training are FedSGD and FedAVG. FedSGD

involves a client training a model for a single local iteration before sending the gradients to the server. FedAVG, on the other hand, involves several local iterations of client training before sending the updated model parameters to the server. Due to communication efficiency, FedAVG is preferred in many settings.

While the promise of FL is that user data should be private and secure from a malicious server, this only holds true if gradients from the clients cannot be used by a malicious server to infer properties of the training data. Prior work has shown that property inference [2], [3], membership inference [4], [5], [6] or GAN-based attacks [7], [8] are all effective in inferring information about client data from their shared gradients. However, the stronger class of data reconstruction attacks, when a malicious server wants to directly steal private client training data, has been demonstrated against even the strictest defenses. These attacks fall into two categories.

Optimization attacks [9], [10], [11], [12] generally work on image data, starting with a dummy randomly initialized data sample and optimizing over the difference between the true gradient and the one generated through the dummy sample. Iteratively, the dummy data sample is updated and gets closer to the ground truth data that was used in computing the true gradient. However reconstructions following this method degrade in quality as the batch size and image resolution increase, failing with a batch size of greater than 48 as demonstrated in [12] on the ImageNet dataset. Most optimization attacks work only on FedSGD, but [13] demonstrates an optimization attack on FedAVG that is applicable to single client attacks with a relatively small client dataset size and image size (50 images in total, FEMNIST/CIFAR-100). This is promising, as FedAVG is inherently more difficult to attack since the malicious server cannot see the intermediate model updates coming from local iterations.

However, secure aggregation has been shown to be an effective defense against all optimization-based privacy attacks in FL. Secure aggregation guarantees that the server will not gain access to individual model updates of other clients, only the aggregate of all client updates [14], [15], [16]. This poses a large problem for optimization as aggregation introduces a massive number of total images in training while these attacks only have a high image reconstruction

rate in the small batch size regime.

The second class of attacks, analytic reconstruction attacks [17], [18], involve customizing the model parameters or the model architecture to directly retrieve the training data from the gradients of a fully-connected (FC) layer — this is referred to as *linear layer leakage*. These approaches do not have issues with quality as they reconstruct the inputs near exactly. These methods work well when individual client updates are visible. Under secure aggregation and FedSGD, [17] also has the ability to maintain a high leakage rate by increasing the size of the injected FC layer, although this often results in very large models. The introduced sparse variant of their attack functions to attack individual updates in FedAVG, but fails when secure aggregation is applied.

Some recent works have targeted FL with secure aggregation by magnifying gradients [19], making the aggregate update the same as an individual update of a target user [20], or looking to solve a blind source separation problem [21]. However, these attacks still have limited power in the FL setting. The first approach can only steal a single user image for each training round while additionally requiring multiple iterations prior to setup the attack. The second approach can use either analytic attacks or optimization attacks as a backbone but suffers from a similar scale limitation, only reaching a single client each round. The third approach can work with up to 1024 images in aggregation for FedSGD (# images/client  $\times$  # clients), but fails for more images. **Thus, no prior method is able to scale to attacking multiple clients in FedAVG with secure aggregation.**

**Our work.** We introduce LOKI<sup>1</sup>, an attack whereby a malicious server can directly reconstruct the training data of multiple users in a single iteration using only the aggregate update. The attack works for both FedAVG and FedSGD, and even when secure aggregation is applied with hundreds of clients participating in the training round. To achieve this, the malicious server modifies the model that it sends to each target client. *The key insight behind our attack is that the server sends customized convolutional kernels to each client, so that the gradients of the input data of each client is separable in the aggregated update and is thus recoverable at the server.* With this separated weight gradient, an FC layer can then be used to leak the data of each individual client. Furthermore, the attack can increase the leakage rate by observing and adjusting the neurons in the FC layer that client images activate in each training round. This ability to leak client data regardless of aggregation breaks the previous scaling limitations of reconstruction attacks. While previous linear layer leakage methods [17], [18] must scale the FC layer to address an increase in the the number of samples coming from an increasing batch size or increasing number of clients, we instead introduce a *split scaling* for this through our design. LOKI can scale to larger client dataset sizes by increasing the FC layer size, but it can also scale to a larger number of clients by increasing the number of

<sup>1</sup>In Norse mythology, LOKI is a cunning trickster who had the ability to change his shape. Our attack can figuratively change its shape for different clients and trick them into giving up their private data.

convolutional kernels. This prevents a diminishing return in the leakage rate when higher numbers of clients are aggregated [18]. With this property, we work especially well in large-scale aggregation such as cross-device FL, being able to leak 76%-86% of all images through only a single training round of FedAVG. On top of being able to break aggregation, gradients used for reconstruction also directly trace images back to the client that owns them. As a result, information on data ownership is also obtained and allows the attacking server another degree of freedom: the ability to target only high-value clients and identify their reconstructed data afterwards.

Our main contributions are:

- (1) We introduce LOKI, an attack that allows data leakage even with secure aggregation. Using a single FedAVG training round with 100 clients and a local dataset size of 64, we leak 82.66% (5290 of 6400) training images from the aggregated update on CIFAR-100. Further, LOKI can pinpoint which client each training sample comes from.
- (2) The attack works regardless of the number of clients in FedAVG and FedSGD. By increasing the size of the network and our split scaling technique, we can continue to scale our attack to increasing batch sizes *and* number of clients.
- (3) Using the convolutional scaling factor, LOKI is able to prevent images between separate local iterations from activating the same neuron, a fundamental problem in linear layer leakage for FedAVG. This allows the attack to achieve a higher leakage rate for FedAVG than any prior linear layer attacks.
- (4) LOKI is able to handle non-IID clients by learning the distribution of the dataset and individual clients over multiple rounds. Through learning over just a single training round, we achieve up to a 62.3% increase in leakage rate from the no-learning case on OrganAMNIST.

## 2. Background and related work

While many different attacks on FL have been proposed, we focus on data reconstruction attacks, the strongest attacks on privacy in FL. Reconstruction attacks aim to break the fundamental notion of privacy of FL by obtaining client data directly through their updates. Prior work has done this through optimization [9], [10], [11], [12], [13], analytic attacks [19], [20], linear layer leakage [17], [18], or other approaches including GANs [7], [8]. These attacks typically aim to attack either individual client gradients or aggregated gradients. The following subsections will discuss the details and limitations of prior attacks.

### 2.1. Optimization-based attacks

Optimization approaches have shown great success in leaking data from individual updates, especially with smaller batch sizes. These attacks typically operate under the threat model of an honest-but-curious server or an external attacker that has access to the model and individual gradients from each client. With only this information, the attacker initial-

	Attack type	Uses one training round	FedSGD individual update	FedSGD aggregation	FedAVG individual update	FedAVG aggregation	Non-IID support
Inverting Gradients (NeurIPS '20)	Optimization	✓	✓	✗	✗	✗	
GradInversion (CVPR '21)	Optimization	✓	✓	✗	✗	✗	
Eluding Secure Agg. (ACM CCS '22)	Analytic	✓	✓	🟡	✓	🟡	N/A
Fishing for User Data (ICML '23)	Analytic	✗	🟡	🟡	✗	✗	
When the Curious (Euros&P '23)	Linear layer leakage	✓	✓	✗	✗	✗	✗
Robbing the Fed (ICLR '22)	Linear layer leakage	✓	✓	✓	✓	✗	✗
<b>LOKI</b>	Linear layer leakage	✓	✓	✓	✓	✓	✓

Yes      
 No      
 Partial

TABLE 1. COMPARISON OF DATA RECONSTRUCTION ATTACK FEATURES. PARTIAL SUPPORT INDICATES THE ATTACK HAS LIMITED REACH/SCALE.

izes some dummy data and computes the gradient of that data on the model.

$$x^* = \arg \min_x \|\nabla L(x, y, \theta) - \nabla W\|_2 \quad (1)$$

An optimizer minimizes the difference between the generated gradient  $\nabla L(x, y, \theta)$  and the ground truth gradient  $\nabla W$  (benign client update). Here  $x$  is the dummy data,  $x^*$  is the reconstructed data,  $L$  is the loss function,  $y$  is the label, and  $\theta$  is the model parameters.

More recent optimization approaches [11], [12] work under the assumption that user labels are known prior to optimization. Typically, these labels are directly retrieved through a zero-shot solution without using optimization approaches [10], [12]. Furthermore, regularizers and strong image priors specific to image data are often used to guide optimization results [11], [12]. These can also result in image artifacts typical of an image class, but not in the actual training image. These approaches have shown surprising success with image data on smaller batch sizes. However, as batch sizes increase, the fraction of images recovered decreases along with the reconstruction quality and the number of iterations required for the optimization also increases. One reason stated by [9] was that regardless of the order of images in the batch, the gradient will remain the same. Having multiple possible permutations then makes the optimization more difficult. Another fundamental reason is that a larger batch size means more images and more variables for optimization.

## 2.2. Aggregate attack methods

There have been several attacks aimed specifically for aggregate gradients, however, they are currently limited in the attack scalability.

In [20], attackers send different models to clients such that the resulting aggregated gradient is only a targeted

client’s individual gradient. This is done by sending model parameters such that ReLU activated layers would have fully dead units (and an update with zero gradients) for any non-targeted client. The targeted client would get an attacked model, which would then be the only one to return non-zero gradients. On the other hand, [19] focuses on attacking a single data point through gradient magnification. The server sends weights to magnify the gradients of a targeted class by decreasing the model confidence on that class’ prediction. Within the targeted class, the server will also focus on a specific feature in order to target a single sample. The resulting gradient will be similar to the gradient for a single image, allowing optimization-based approaches to retrieve the input. However, this process requires multiple training iterations. The attack can also only target a single image each time, and even in this case it does not succeed every time. Another method treats the inputs to a fully-connected layer as a blind source separation problem [21] where the weight gradients for the neuron make up a set of weighted combinations of the inputs. While the approach is able to attack aggregated gradients in FedSGD, the number of inputs is limited to be 1024 or fewer.

Another work [22] has looked to enable prior individual gradient methods through gradient disaggregation, separating out individual updates over time. However, along with requiring additional side-channel information about client participation, the method also requires a large number of training iterations to accomplish the goal. Similarly, due to partial user selection [23], [24], the server can reconstruct the individual models of some users using the aggregated models from previous rounds [25], [26]. These approaches also require multiple training rounds and can be prevented by proper client selection so that no individual updates can be singled out.

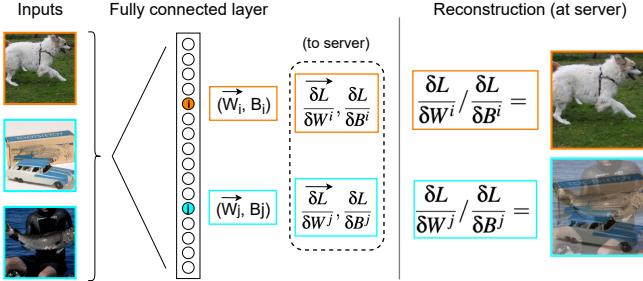


Figure 1. Using the weight gradient  $\frac{\delta L}{\delta W}$  and bias gradient  $\frac{\delta L}{\delta B}$  of a fully connected layer to reconstruct the inputs. Neuron  $i$  is only activated by a single image, while  $j$  is activated by two. As a result, the reconstruction of neuron  $i$  is correct while  $j$  is a combination of images.

### 2.3. Linear layer leakage attacks

Linear layer leakage attacks are a sub-class of analytic attacks that modify FC layers to leak inputs. Using the weight and bias gradients of an FC layer to leak inputs was discussed in [27], [28]. When only a single image activates a neuron in a fully connected layer, the input to that layer can be directly computed using the resulting gradients as

$$x^i = \frac{\delta L}{\delta W^i} / \frac{\delta L}{\delta B^i} \quad (2)$$

where  $i$  is the activated neuron,  $x^i$  is the input that activates neuron  $i$ , and  $\frac{\delta L}{\delta W^i}$ ,  $\frac{\delta L}{\delta B^i}$  are the weight gradient and bias gradient of the neuron respectively. This idea forms the basis for several reconstruction attacks [17], [18]. Figure 1 shows the basic process of leaking images through an FC layer.

When the fully-connected layer is placed at the start of a network, the data reconstructed from the layer would be the input data. This reconstruction is exact, as opposed to the optimization approaches which function as estimations. However, inputs are only reconstructed exactly when a single data sample activates that neuron. If more than one input activates the neuron, the weight and bias gradients of these inputs will contribute to the batch gradient. When the gradient division of Equation 2 is done to retrieve the input, the resulting reconstruction would be a combination of all contributing images, a case of failed attack.

To alleviate this problem, [17], [18] use malicious modification of the parameters in the FC layer. For [18], trap weights were introduced, initializing the weights randomly to be half positive, half negative. In order to ensure that neuron activation is less common, the negative weights come from a larger negative magnitude range than the positive weights. They also discuss the use of convolutional layers to push the input image forward, allowing the attack to function on models starting with convolutional layers followed by fully-connected layers. However, one of the main problems of the method lies with scalability. Even if the size of the FC layer increases proportionately with an increasing total number of images, the leakage rate decreases. On the other hand, Robbing the Fed (RtF) [17] introduced another approach with higher leakage rate called “binning”, where the weights of the FC layer would measure some known continuous CDF of the input data such as image brightness.

The bias for each neuron then serves as a different cutoff, allowing only inputs with a high enough value to activate it. The goal of this method would be that only one input activates each “bin”, where the bin is defined as the activated neuron with the largest cutoff (for ReLU, the largest negative bias)<sup>2</sup>. For any case where only one input activates a bin, it can then be reconstructed as

$$x^i = (\frac{\delta L}{\delta W^i} - \frac{\delta L}{\delta W^{i+1}}) / (\frac{\delta L}{\delta B^i} - \frac{\delta L}{\delta B^{i+1}}) \quad (3)$$

where  $i$  is the activated bin and  $i + 1$  is the bin with the next higher cutoff bias.

For Equation 3 to hold true, the attack requires the use of two consecutive FC layers. The first layer is used to leak the inputs using Equation 3 and the second FC layer maintains the requirement that  $\frac{\delta L}{\delta B^i}$  and  $\frac{\delta L}{\delta W^i}$  are the same for any neuron that the same input activates. This is achieved by having the same weight parameters connecting each neuron of the first FC layer to the second FC layer. For example, if the first FC layer has 1024 units and the second has 256, the weights connecting them would have a dimension of  $1024 \times 256$ . The above property indicates that every row of the weight matrix is equivalent, e.g.  $[0, :] = [1, :] = \dots = [1023, :]$ .

**FedAVG.** While the previous method works for FedSGD, for FedAVG, the model changes during local iterations and this prevents the reconstruction attack. As a result, [17] proposed the sparse variant of the attack which uses an activation function with a double-sided threshold (e.g., Hardtanh) such as:

$$f(x) = \begin{cases} 0 & x \leq 0 \\ x & 0 \leq x \leq 1 \\ 1 & 1 \leq x \end{cases} \quad (4)$$

With this activation function, only when the input is between 0 and 1 will there be a non-zero gradient.

Using this activation function, neuron activation will be sparse (i.e., images will only activate a single neuron). However, this range between 0 and 1 for the non-zero gradient is fixed for all neurons. Since RtF’s approach sets up neuron biases following a distribution of the images, the weights and biases of the FC layer will need to be adjusted to follow the new non-zero gradient range. This requires scaling the magnitude of these parameters based on the distance between the subsequent neuron biases. Consider that the weights originally measure average pixel brightness. In this case, all the weights would originally be set to  $\frac{1}{N}$ , where  $N$  is the total number of pixels. Then, the weights and biases are rescaled as:

$$W_i^* = \frac{W_i}{b_{i+1} - b_i}, \quad b_i^* = \frac{b_i}{b_{i+1} - b_i} \quad (5)$$

where  $W_i^*$  and  $b_i^*$  are the scaled weights and biases of neuron  $i$  respectively and  $b_{i+1} - b_i$  is the distance between adjacent biases in the original distribution. This process uses the same distribution as the FedSGD case to setup the

<sup>2</sup>The bin biases are set as negative. The weights are positive and so the negative bins are used to prevent ReLU activation.

initial biases, while incorporating the fixed range of the new activation function by scaling the parameters. In FedAVG, after the clients send the updated model parameters, the server computes a “gradient” as:

$$\nabla W_{FedAVG} = \Theta_{t+1} - \Theta_t \quad (6)$$

where  $\Theta$  is the model parameters for the securely aggregated model and  $\nabla W_{FedAVG}$  the computed gradient.

### 3. Methodology

As discussed previously, optimization becomes much more difficult with larger batch sizes and aggregation. The main problem is with more data to approximate, reconstructions end up having much lower quality and ultimately fails at larger batch sizes, even before aggregation is applied. On the other hand, the linear layer leakage methods [17], [18], [27] provide a powerful way of directly reconstructing training data without having reconstruction quality issues. Increasing the FC layer proportional to the number of total images in aggregation also allows [17] to scale in FedSGD without losing effectiveness. However, the attack fails at scale for the much more difficult, and much more common, aggregation strategy of FedAVG.

#### 3.1. Model assumptions and attack scope

**Threat model.** We operate under the same threat model as [17], [18], a malicious federated learning server with the ability to manipulate the model architecture and parameters before sending the model to clients. The server launches the attack by inserting a malicious module into the architecture, where the goal of the malicious server is to recover private training data.

**System model.** We operate in cross-device FL, a setting where hundreds of clients participate in a single round of aggregation. The updates from the clients are aggregated through a secure aggregation mechanism before being made available to the server. Clients in this setting do not have the power to do a thorough verification of the models sent by the server. However, due to the application of secure aggregation, outside of the aggregate update, the server cannot receive any information about the clients or about individual updates.

**Attack scope.** In this work, we target both FedAVG and FedSGD. Under the threat model, LOKI can leak the inputs to that model regardless of the data type (e.g. image, video, audio, text), similar to [17]. We experimentally demonstrate this attack on image data.

#### 3.2. Scalability problem

**FedAVG.** Since Robbing the Fed (RtF) [17] can scale to aggregation in FedSGD by increasing the size of the FC layer, it is not immediately apparent why the same cannot be done in FedAVG with the sparse variant. In theory, the images will activate the same set of neurons between FedAVG and FedSGD (given a bit of difference coming

from parameters changing slightly during local iterations), so reconstruction rate will be the same. However, there is a critical problem with scaling the size of the FC layer in FedAVG. The biases will follow the same distribution regardless of the size of the layer. If the distribution is the average pixel brightness, the range of values will always be between 0 and 1. Having a larger FC layer size results in more neurons in the same range, and the distance between adjacent biases decreases. This means the value of  $b_{i+1} - b_i$  gets smaller and as a result, the scaled weights  $W_i^*$  and biases  $b_i^*$  become larger (Equation 5).

$$\begin{aligned} \|W_t, b_t\| &\gg \|\nabla W_t, \nabla b_t\| \\ W_{t+1} &= W_t + \alpha \cdot \nabla W_t \approx W_t \\ b_{t+1} &= b_t + \alpha \cdot \nabla b_t \approx b_t \end{aligned} \quad (7)$$

The increasing size of the FC layer results in increasing  $W_i^*$  and  $b_i^*$  parameter magnitude, but the magnitude of each client’s gradient update  $(\nabla W_t, \nabla b_t)$  will remain relatively unchanged. As a result, the relative parameter shift in the model between different local iterations becomes much smaller. This same property is the Achilles heel, as once the magnitude of the parameters increase beyond a certain point, the update computed for the FC layer essentially becomes zero, i.e.,  $W_{t+1} - W_t = 0$  for the FC layer (Equation 7). Even before the updates become zero, the values lose precision which reflects in poor reconstruction quality. This becomes a large problem, resulting in a heavily decreasing leakage rate and reconstruction quality as the number of clients in aggregation increases.

The default precision used for deep learning models in most packages is floating-point 32 (FP32). If the original value of the parameters is much larger than the update, adding the gradient to the parameters results in a loss of precision or in the extreme case, no update at all. This is typically caused by increasing the FC layer size, but also can be affected by other factors such as the learning rate or local mini-batch size. RtF [17] mentions a way to increase the effectiveness of the method by scaling the linear distribution  $h_{new}(x) = c \cdot h_{original}(x)$ ,  $c > 1$  to minimize the change in the parameters over the local iterations. However, what we find is that this does not address the problem in FedAVG, as scaling the distribution not only scales the weights of the FC layer, but also the biases. Consider a case where the image brightness distribution is scaled by a factor of 10. Since the weights used to measure it increase by  $10 \times$  to achieve this, the biases used as cutoffs also have to be scaled by the same factor. These layer parameters are then scaled by the distance between the biases  $b_{i+1} - b_i$ , which in turn is also  $10 \times$  larger. This ultimately results in no change in the values of  $W_i^*$  and  $b_i^*$  of Equation 5. Thus, scaling the linear distribution does *not* fix the precision problem.

**FedSGD.** For FedSGD on a single client, if multiple images from that client’s batch activate the same neuron, reconstruction fails. This problem is directly exacerbated with aggregation. If multiple images *across any of the clients* activate the same neuron, reconstruction at the server also fails. For a successful reconstruction, only one image across

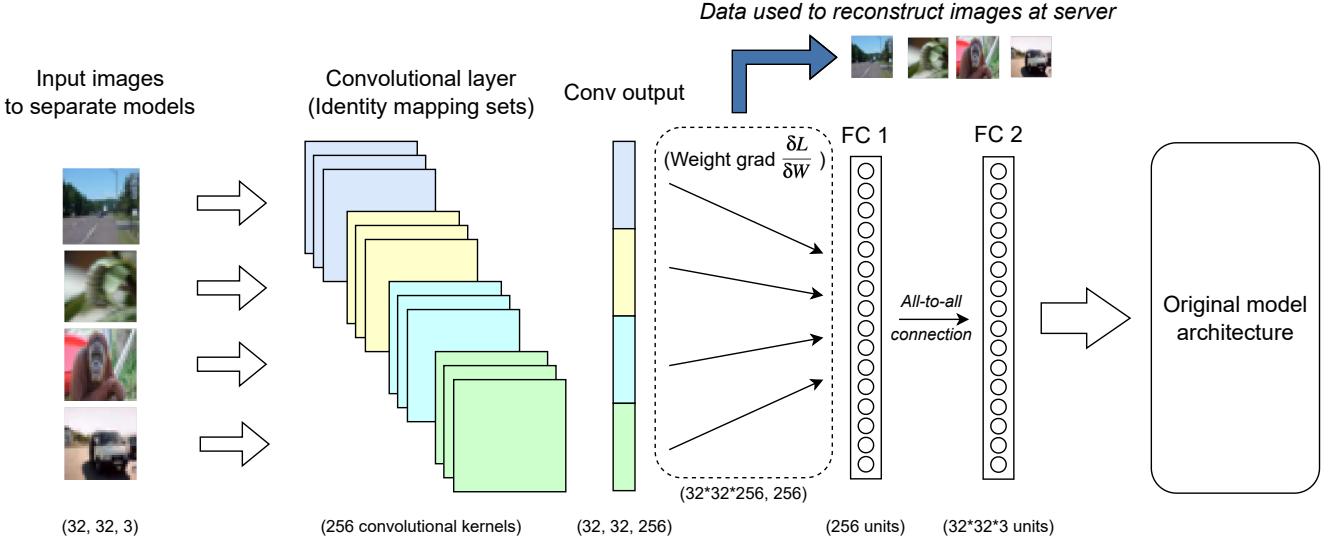


Figure 2. The inserted attack module of a convolutional layer and two FC layers. Images are leaked using the gradients of the weight parameters connecting the convolutional output to the first FC layer. Weight gradient de-aggregation is done through separate identity mapping sets, indicated by the different color values in the convolutional kernels. In the aggregate update, the weight gradients of different clients are separated and used to leak the images.

all client batches can activate the same neuron. Consider a case where we have 100 clients each with a batch size of 64. If we have an FC layer of 256 units, this means  $64 \cdot 100 = 6400$  images are shared, resulting in an average of 25 images per neuron. For linear layer leakage, the generalization to these larger-scale attacks is done by simply increasing the size of the FC layer. In the case of RTF [17], a proportional increase in FC layer size with the number of images will maintain the same leakage rate. However, for trap weights [18] the leakage rate will still decrease.

Therefore, while the underlying reasoning is different for the sparse variant of RTF [17] in FedAVG and for trap weights [18] in FedSGD, the fundamental scalability problem is the same. *Current linear layer leakage methods only scale the size of the FC layer for an increasing batch size or number of clients, which results in scalability problems.* LOKI breaks this scaling problem for both settings and separates the scaling of the batch size and the number of clients. We increase the size of the FC layer for larger dataset sizes (batch sizes) and the number of convolutional kernels for more clients. Furthermore, we introduce a convolutional scaling factor that prevents multiple activations of a neuron across epochs and helps mitigate precision problems coming from factors such as local mini-batch size or learning rate.

### 3.3. Attack architecture

We insert an attack module at the start of a model that consists of a convolutional layer followed by two FC layers. This module is shown in Figure 2. We leak images using the computed gradients of the weight parameters connecting the output of the convolutional layer to the first FC layer. The dimension of the output of the convolutional layer depends on the image size and number of kernels. For a  $32 \times 32$  image and 100 kernels, the output would have dimension  $32 \times 32 \times 100$ .

The size of the first FC layer depends on the local dataset size for FedAVG or batch size for FedSGD. Generally, we add 4 units in the layer for each image. With a batch size of 64, this would be 256 units. Every unit in the first FC layer is connected to the second FC layer, which is the input to the rest of the model architecture and will have the same dimensions as the input image. As an inserted module, it is important that the input and output dimensions are the same, so that the dimensionality expected by the benign model is not altered.

### 3.4. Convolutional parameters

We start our discussion by describing the attack in FedSGD before discussing the (significant) sophistication for FedAVG. Previous works [17], [18] have discussed the idea of leaking with convolutional layers followed by FC layers. The standard way to achieve this would be the use of convolutional kernels to push the image forward and have FC layers further in the model leak the inputs. We will further explore the use of convolutional parameters.

For a 3-channel input image (RGB), pushing the image forward can be done with three kernels in a convolutional layer. If we have  $3 \times 3$  kernels, the dimension of each kernel would be  $3 \times 3 \times 3$  with the final dimension corresponding to the input channels. For any given client, we will need a minimum of three kernels for a 3-channel image. Within each kernel, there is only one key channel, which will have a value of 1 in the center and all other elements will be zero. The three kernels will have this in a different channel. We call the single non-zero value the key value  $kv$ . When the convolutional kernels are applied to the RGB input image, each kernel will push a separate channel forward, resulting in the image in its entirety being pushed through. This setup is shown in Figure 3. We will call these three convolutional kernels an *identity mapping set*. This operation only requires

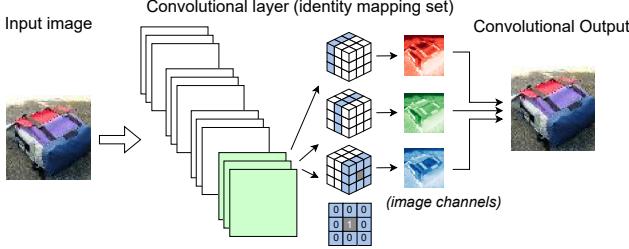


Figure 3. Identity mapping set for a 3-channel input. The first three kernels ( $3 \times 3 \times 3$  cubes) of the convolutional layer push a different input channel forward. All parameters are zero except for a single element. The 2D slice with a single non-zero value is shown in the figure, and the locations of the slice for each kernel allows them to push different input channels forward.

the use of three convolutional kernels, a small number in the context of typical models. The other kernels are not needed to propagate any information, so the outputs are set to zero. There are multiple ways to set the kernel parameters such that the output is zero, but generally a large negative bias or zero/negative weights can cause the output to be zero.

This simple approach still severely under-utilizes the number of convolutional kernels. Consider a situation where we have 256 convolutional kernels. To push one 3-channel image forward, we only use three kernels. As a result, the other 253 kernels do not contribute. With this in mind, we propose the use of multiple, separate identity mapping sets. Each set requires three kernels (number of input channels), and following this, we have  $\lfloor \frac{N}{3} \rfloor$  separate identity mapping sets, where  $N$  is the total number of convolutional kernels. These separate identity mapping sets, each corresponding to a different set of three convolutional kernels, are then used in different models and are sent to different clients.

### 3.5. De-aggregated update

By sending the carefully crafted separate convolutional kernels to each client, the weight gradient for each set of identity mapping sets is non-zero only for a different set of convolutional output channels. Therefore, when updates are aggregated together, the weight gradients remain separate. During the reconstruction phase, if inputs from different clients activate the same bin, the computed weight gradient of that bin will not be shared between clients. The only inputs that can share the same set of weight gradients would be images within that single client’s batch. This essentially allows the size of the FC layer to scale only based on the client batch size, and *not with the number of clients*.

When reconstructing images, this allows us to work with different sets of weight gradients, each corresponding to separate identity mapping sets (and client model). Instead of Equation 3, we would then reconstruct images as:

$$x_k^i = \left( \frac{\delta L}{\delta W^i k} - \frac{\delta L}{\delta W^{i+1} k} \right) / \left( \frac{\delta L}{\delta B^i k} - \frac{\delta L}{\delta B^{i+1} k} \right) \quad (8)$$

where  $k$  indicates the client and the corresponding weight and bias gradients respective to their identity mapping set.  $x_k^i$  is the input from client  $k$  that activates bin  $i$ . Figure 2 shows the process of using separate identity mapping sets to

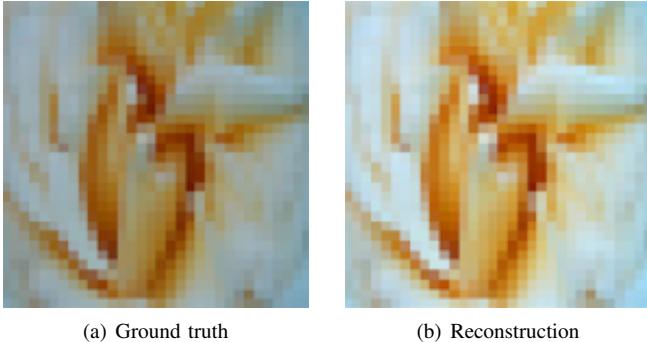
split the aggregate weight gradient. This allows the attacker to leak images from each client separately after aggregation.

This decoupled weight gradient partially solves the scaling problem of reconstructing images as the number of clients participating in aggregation increases. However, Equation 8 brings up another problem for reconstruction, as the bias gradient is needed for the computation. While the weight gradients are separated through the identity mapping sets, the bias gradients are not. Secure aggregation aggregates the bias gradients of each neuron (of the first FC layer). Thus the neuron  $i$ ’s bias values from the FC layers of all clients are aggregated. Consider that an image  $j$  activates neuron  $i$  in client  $k$  and another image  $j'$  activates the neuron  $i$  in client  $k'$  (note that the neurons are physically separate before aggregation as they are in the local FCs of clients  $k$  and  $k'$ ). After secure aggregation, the bias update of neuron  $i$  from clients  $k$  and  $k'$  are coupled and therefore the server is not able to use Equation 8 to decouple images  $j$  and  $j'$ . To solve this problem, we look at the purpose of the bias gradient in reconstruction.

Observing Equation 8, we note that for reconstruction of each neuron we subtract the weight gradients, and the resulting value is divided by the subtraction of bias gradients. Previously we mentioned how each neuron has a single bias, so after subtracting bias gradients, the resulting value remains a scalar. The purpose of the bias gradient, then, is to *scale the value produced by subtracting the weight gradients such that it becomes the same as the input*. Therefore, as long as we know what the weight gradient needs to be scaled to, we will not need to know the exact bias gradient. Rather, knowledge of the input range is all that is required to reconstruct the input. For image datasets such as MNIST, CIFAR-10, or Imagenet, the training data will be between 0 and 1. Using this, the images can be reconstructed by scaling the gradient such that the maximum value is 1, without requiring knowledge of the bias gradient.

$$x_k^i = \frac{\text{abs}(\frac{\delta L}{\delta W^i k} - \frac{\delta L}{\delta W^{i+1} k})}{\max(\text{abs}(\frac{\delta L}{\delta W^i k} - \frac{\delta L}{\delta W^{i+1} k}))} \quad (9)$$

The numerator is the absolute value of the subtracted weight gradient for client  $k$  between neuron  $i$  and  $i + 1$ , while the denominator is the maximum of that value across the set of three slices in the identity mapping set corresponding to client  $k$ . The technical question now is much simpler — estimating the maximum value of the denominator. The input range can be estimated by the server, or may be known through other public sources, such as through the standardized normalization prior to training. This estimation could be imprecise, but we see empirically that inaccuracies in this estimation do not hurt the reconstruction performance much, as the image remains the same structurally. Figure 4 shows a reconstructed image with a shifted brightness. The maximum pixel intensity of the ground truth was 0.7804 but was scaled to be 1.0 during reconstruction. Importantly, the error in reconstruction is contained to just that image and does not affect the reconstruction of successive images in that batch. Thus, our key result is that in linear layer leakage



(a) Ground truth

(b) Reconstruction

Figure 4. Image reconstructed using Equation 9 to scale the maximum to 1. Ground truth image (a) has a maximum intensity of 0.7804, resulting in a brightness shift in the reconstructed image (b).

methods, the bias gradient is *not* required for reconstruction of data.

### 3.6. FedAVG and convolutional scaling factor

By itself, the trap weights attack [18] suffers from scalability problems in FedSGD. However, utilizing the convolutional attack structure of LOKI, the scalability problems are fixed through the split scaling between the number of clients and batch size (Figure 7). The methodology is generalizable to FedAVG by using a double-bounded activation function as in Equation 4 for the FC layer and scaling the parameters using the distance between adjacent biases using Equation 5. The split scaling design mitigates the scalability problems in FedAVG coming from precision when having large parameters and small gradients. Since the FC layer size does not scale based on the number of clients, the distance between the biases in the layer does not change. As a result, the magnitude of the parameters  $W_i^*$  and  $b_i^*$  will not increase, helping alleviate the problem.

However, factors such as the learning rate or local mini-batch size still affect the update (gradient) size. The learning rate directly affects the size of the gradient update and the local mini-batch size also affects the magnitude of the individual gradient contributions from each image. For example, for a mini-batch of 10 images the gradients are averaged over the 10 images. For 20 images the gradients are averaged over 20 images, so the individual contributions are smaller. Furthermore, the FC layer size still needs to increase linearly with the local dataset size (the total number of images used in training) in order to maintain the same leakage rate. These factors still impact the precision problem.

To address this, we introduce a convolutional scaling factor (*CSF*) for the FedAVG attack. We can scale the image coming out of the convolutional layer by using a different key value (*kv*) for the identity mapping sets in the convolutional kernels. This can be used to offset the increase in the magnitude of the weight parameters  $W_i^*$  coming from the increasing FC layer size or the small gradients coming from the learning rate or local mini-batch size. (Note that since we do not use the bias parameters  $b_i^*$  to reconstruct images, we do not need to worry about update precision for

them.) Using the *CSF*, we can maintain the distribution and scale the parameters as:

$$kv_{new} = CSF \cdot kv, W_{i,new}^* = \frac{1}{CSF} \cdot W_i^* \quad (10)$$

where *kv* is the non-zero value (typically 1) in the identity mapping set. The *CSF* allows us to prevent precision problems during reconstruction, but also helps minimize the changes in the convolutional kernel parameters to preserve the original purpose of pushing the inputs forward. The value produced after the image passes through the convolutional layer and the weights of the FC layer remains the same as in the baseline (*CSF* = 1) in order to fit the distribution for the biases correctly.

Crucially, the *CSF* is also used to help address the fundamental problem in linear layer leakage of having multiple images activate the same neuron. After an image activates a neuron during a local iteration, a smaller value of  $W_{i,new}^*$  results in a larger relative change in the parameters. However, after an image activates the neuron, we do not want any additional activations in subsequent local iterations. Therefore, when the *CSF* is large, the changes in the weights after each local iteration become large enough so additional images do not activate the same neuron in subsequent local iterations (if images are orthogonal, e.g., their dot product is zero, there can still be an activation of the same neuron. However this is a very rare case in practice). When multiple images in separate local iterations would have activated the same neuron, only the first image ends up actually activating it due to parameter shift. This method allows for a correct reconstruction of the first image, which was previously impossible. As a result, LOKI achieves higher leakage rate in the FedAVG attack compared to FedSGD. This same property of the *CSF* also prevents images from different local epochs from activating the same neurons.

## 4. Additional attack details

### 4.1. Setting up FC layer biases

While leaking images from a linear layer is superior in reconstruction speed and quality compared to optimization, the initialization of neuron biases can pose a challenging problem in practice. The weights of the FC layer measure some aspect of an image such as the average pixel intensity and the biases must be used as cut-offs for image activations. If not set properly, the number of recovered inputs is lower. For example, if we initialize completely randomly on CIFAR-100, the leakage rate can drop by over 20% as shown in Appendix Table 3. If the server knows the dataset distribution for average pixel intensity, as assumed in [17], setting the biases of the FC layer is simple. However, in practice client datasets are private, so the server is unlikely to have such knowledge.

We therefore incrementally learn the distributions of the dataset. If the FC layer weights were used to measure the average pixel intensity, this value must be between 0 and 1. An initialization for the biases following this could be so that the biases have a mean of -0.5 and standard deviation

of 0.25. This setting is progressively improved through training iterations by observing the neuron activations. After receiving the aggregated gradient, the server observes which neuron activations result in a successful reconstruction and which do not (same neuron activated multiple times). For each observation, it also notes the bias values of the neuron and using these observed biases, it computes a new mean and variance over them to use for the initialization of the biases for the next iteration. Over several training rounds, the server arrives at a close estimate of the dataset distribution without any prior knowledge. This determination is done by the server separately for each client in the case of non-IID data where clients may have vastly different distributions. Additionally, the server can observe the type of distribution (i.e. normal, multi-Gaussian etc.) and set up biases to fit them. For this work we setup biases following a normal distribution.

## 4.2. Identifying client data

Even if the aggregated gradient is able to leak training data, it may appear to be of some comfort that the attacker cannot identify which data belongs to which client. However, LOKI gives the attacker the ability to identify the owner of the reconstructed data even through aggregation. The identity mapping sets keep the weight gradients for each client separate after aggregation. As a result, when reconstructing inputs, the set of weight gradients used allows the server to identify which model it originates from and hence, which client it originates from. With this information, the server may subsequently focus on specific high-value clients.

## 4.3. Parameter comparison

For linear layer leakage, attacks where the server modifies the model, a key question is how many parameters the attack adds. A small addition is desirable as we are operating in the cross-device setting where communication and storage are at a premium. The fundamental premise behind linear layer leakage is using the gradients of a model update to store the information of the input data. For example, with a  $32 \times 32 \times 3 = 3072$  image, the model must have at least 3072 weight parameters to store this. To store 1000 images, we would then need  $3072 \cdot 1000 = 3,072,000$  parameters. However, exactly 1000 units for 1000 images assumes that every image activates a separate neuron, an extremely optimistic case (for the attack). The FC layer size will typically need to be larger than the number of images ( $4 \times$  is typical).

For comparison between LOKI and RtF for number of added parameters, consider we have 100 clients with a local dataset size of 64. Each image is given 4 units, resulting in 256 units for each client. For simplicity, we will ignore the number of bias parameters, as the amount is much smaller than the weight parameters (e.g. for LOKI, 0.005% of the total parameters are biases). For RtF [17], it ends up needing 157M parameters while LOKI needs roughly half (50.7%). The basic reason for this reduction is that RtF needs a large

first FC layer ( $25,600 = \text{batch size} \times \text{multiplicative factor } (4) \times \# \text{ clients}$ ), while we only need 256. The number of connections from the first to the second FC layer are determined by dimensionality considerations and cannot be reduced. This improvement in LOKI is taking into account the additional parameters due to the convolutional kernels.

The added parameters are also extremely sparse, as only the weight parameters connecting the identity mapping set kernel outputs to the first FC layer need to be non-zero (3 out of 300). When attacking 100 clients, 98% of our total added parameters are zero. As the number of clients increases, the absolute number of non-zero parameters at each client stays constant. Sparsity can allow for additional compression of the model/update and more efficient optimization to reduce the computational and communication cost overhead significantly [29], [30], [31].

## 4.4. Model inconsistency

We have so far discussed the attack in the context of using separate identity mapping sets for each client. This method inherently creates model inconsistency among clients. However, this is not a requirement in LOKI. Some clients can be sent the same convolutional kernels. For any clients that are sent the same identity mapping set, if images between clients activate the same neuron, there will be failure in reconstruction, so the size of the FC layer will need to be increased accordingly. The two extremes of model inconsistency are: full model inconsistency where all clients get different models, and no model inconsistency where all clients get the same model.

In the latter case of no model inconsistency, the original scalability problem seems to exist. As the size of the FC layer increases with the total number of clients or local dataset size, the magnitude of the scaled parameters increases. However, the convolutional scaling factor of LOKI offsets this change and prevents reconstruction problems. The  $CSF > 1$  setting allows us to avoid precision problems and thus keep leakage rate high. There are some additional downsides to not using model inconsistency for LOKI, such as a slightly lower leakage rate (Figure 6), larger model size overhead (Figure 9), or losing the ability to identify client data ownership. However, with no model inconsistency LOKI still achieves scalability to an arbitrary local dataset size or number of clients in FedAVG.

## 5. Experiments

In this section, we provide experimental results for LOKI on aggregated updates in FedAVG. We assess our baseline attack on the CIFAR-100 [32], Tiny ImageNet [33], and MNIST [34] datasets. The leakage module is used with a ResNet-50 [35], but the benign model itself does not affect reconstruction. The FC layer weights measure the average pixel intensity and the biases of the FC layer are set up according to the dataset distribution (known to the server or learned through the first few training iterations). Unless otherwise specified, we have 4 units in the FC layer per image for LOKI and Robbing the Fed (RtF) [17].

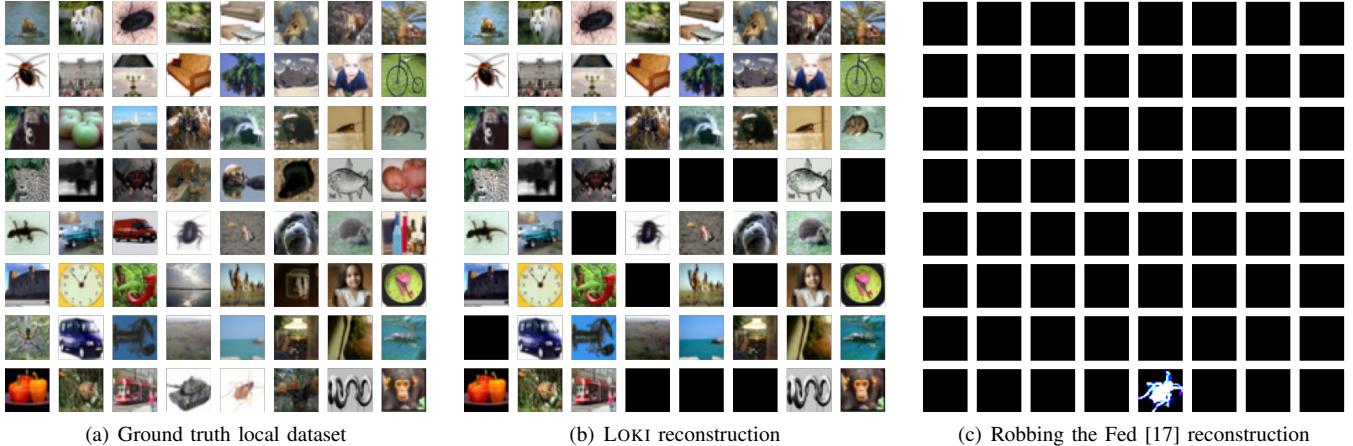


Figure 5. LOKI and Robbing the Fed (RtF) reconstructions on CIFAR-100 for 1 client out of 100 in FedAVG aggregation. Clients use 5 epochs with 8 local iterations and a local mini-batch size 8. Empty boxes indicate images were not reconstructed because multiple images activated a neuron. In the case of RtF, nearly all empty boxes occur due to precision problems causing the computed gradient to be zero rather than multiple activations.

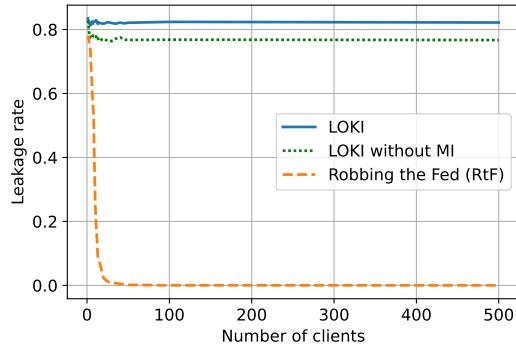


Figure 6. Leakage rate of LOKI with and without model inconsistency (MI) and Robbing the Fed (RtF) for a number of clients between 1-500 on CIFAR-100. Clients train with 8 local iterations of mini-batch size 8.

Dataset	Metrics	LOKI	RtF [17] + MI [20]
CIFAR-100	Leaked imgs	5290	50
	Total imgs	6400	6400
	<b>Leakage rate</b>	<b>82.66%</b>	<b>0.78%</b>
Tiny ImageNet	Leaked imgs	5202	49
	Total imgs	6400	6400
	<b>Leakage rate</b>	<b>81.28%</b>	<b>0.77%</b>
MNIST	Leaked imgs	4907	49
	Total imgs	6400	6400
	<b>Leakage rate</b>	<b>76.67%</b>	<b>0.77%</b>

TABLE 2. LEAKAGE RATE FOR FEDAVG AGGREGATED UPDATE WITH 100 PARTICIPATING CLIENTS.  $\alpha = 1e - 4$ ,  $CSF = 100$  AND 5 LOCAL EPOCHS OF 8 ITERATIONS OF MINI BATCH SIZE 8 USED.

We first show the leakage rate that LOKI achieves for each dataset in FedAVG before also showing the scalability in FedSGD. Further experiments show various FedAVG training settings, the effects of the CSF, a non-IID attack which we evaluate using the OrganAMNIST [36] dataset (which has larger distribution differences when using a class-based non-IID skew compared to CIFAR-100), and differential privacy. Some additional experimental results are shown in Appendix A.

### 5.1. FedAVG aggregation attack

Figure 6 shows the leakage rate for RtF with a varying numbers of clients between 1-500 averaged over 10 runs. We define an image as leaked only if a single image activates the neuron and has a reasonable reconstruction quality with SSIM above 0.5. If two or more different images (not same image across multiple epochs) activate the neuron, even with a high SSIM, we do not count it as leaked. Clients are trained with with 8 local iterations with mini-batch size 8 on CIFAR-100, and learning rate  $\alpha = 1e - 4$ . We use LOKI with  $CSF = 100$  ( $CSF = 100 \cdot num. of clients$  when not using model inconsistency). While RtF is applicable to a smaller numbers of clients, the leakage rate very quickly decreases as the number of clients increases. With only 15 clients, the leakage rate of RtF drops to 7.57%. With 30 clients, the leakage rate drops to 0.95%. As the FC layer size increases with the number of clients, the leakage rate decreases due to precision problems. We use a relatively small number of local iterations and mini-batch size, but if either increase, RtF will function on even fewer clients. LOKI is unaffected by training parameters (learning rate, local iterations, mini-batch size) and achieves a high leakage rate regardless of the number of clients. Without model inconsistency, LOKI has a slightly lower leakage rate but still achieves scalability with no impact on reconstruction quality.

Figure 5 shows the reconstruction of a client for LOKI and RtF [17] for the FedAVG attack on 100 clients. We do not use the bias for any reconstructions as discussed in Section 3.5. The images reconstructed by LOKI are not affected by the aggregation of many clients and are very clearly identifiable. However, RtF is unable to reconstruct images correctly, with only 1 image having a non-zero computed weight gradient out of the FedAVG update. However, even after proper normalization, the image is visually incorrect due to precision errors. We find that a large majority of clients do not have even a single image with a non-zero gradient. Appendix Figure 13 shows the top-8 SSIM reconstructed images across all clients in RtF for 10, 25, and 50

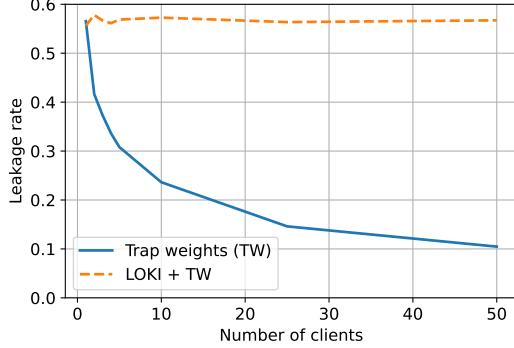


Figure 7. Leakage rate on CIFAR-100 for the trap weights (TW) attack [18] and LOKI + TW for a varying number of clients in FedSGD with a batch size of 64. Leakage rate given as an average over 10 runs. The TW attack leakage rate decreases as the number of clients increases, but using the convolutional attack fixes the scalability problem through split scaling.

clients. As the number of clients increases, both the quantity and the quality of reconstructed images decrease.

We use the same FedAVG settings as above and additionally add 5 local epochs and compare the leakage rate for 100 clients. Table 2 shows the leakage rate of LOKI and RtF + model inconsistency (MI) [20] on CIFAR-100, Tiny ImageNet, and MNIST. We do not compare to RtF alone, as it is unable to leak images in this setting. As discussed previously, increasing the FC layer size results in more precision problems and decreasing the size also does not help due to increased overlapping neuron activations. Using RtF with MI achieves the current SOTA for a large scale FedAVG secure aggregation attack and allows the attack to reach a single client. LOKI achieves a significantly higher leakage rate as it reaches all clients and scales to aggregation. Increasing the size of the FC layer will also allow the leakage rate to increase (for LOKI, as RtF runs into precision problems beyond a point as discussed in Section 3.6).

## 5.2. FedSGD aggregation attack

We also show the applicability of LOKI to helping the trap weights (TW) [18] attack in FedSGD aggregation. Figure 7 shows the leakage rate of the TW attack and LOKI + TW for a varying number of clients between 1-50. Clients train with a batch size of 64 on CIFAR-100 and we use an FC layer size of  $10 \times$  the total number of images for the TW attack. For each setting of the number of clients, we use the TW scaling factor that achieves the highest leakage rate tested by 0.01 increments. For 1-50 clients, we find that scaling factors between 0.91-0.96 achieve the highest leakage rates, with lower values needed for higher numbers of clients. Despite tuning, the TW attack still suffers from a decreasing leakage rate as the number of clients increases. However, using the convolutional attack of LOKI in addition to TW allows the scaling to split between the batch size and number of clients and maintains a constant leakage rate regardless of the number of clients.

## 5.3. Local dataset size and FC layer size

The leakage rate is affected by the local dataset size and the FC layer size. In this section we experiment with

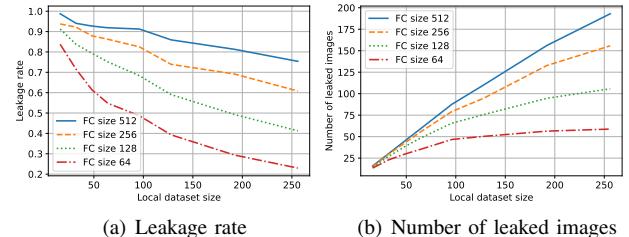


Figure 8. For LOKI, (a) leakage rate and (b) number of leaked images as a function of the local dataset size and FC layer size averaged over 10 clients. While the overall leakage rate decreases with a larger local dataset size, the total number of leaked images continues to increase.

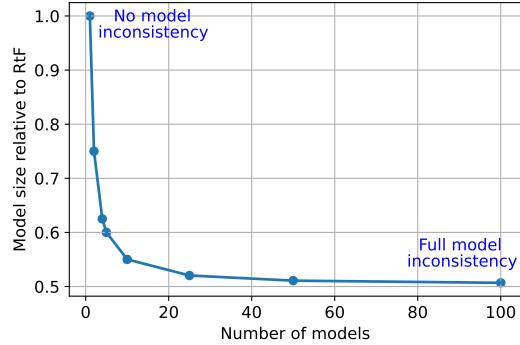


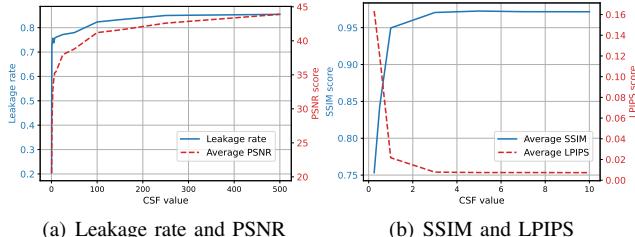
Figure 9. Model size of LOKI relative to the size added by Robbing the Fed (RtF) for a 100 client attack. The number of clients sharing each separate model is equal across all models. LOKI's size is  $50.69\% - 100\%$  of RtF's.

how the changes to both affect overall leakage rate. We fix the local mini-batch size to 8 and vary the number of local iterations as the dataset size increases. We use  $\alpha = 1e-4$  and  $CSF = 500$  for all settings and train on CIFAR-100. For each local dataset size, we compute the leakage rate with an FC layer size of 64, 128, 256, and 512 as an average over 10 clients. Figure 8(a) shows the average leakage rate for each setting as the local dataset size increases from 32-256.

Figure 8(b) shows the average number of leaked images per client with the varying local dataset size. While the overall leakage rate decreases as the local dataset size increases, the total number of leaked images continues to increase. The local iteration parameter changes prevent multiple images between separate local iterations from activating the same neuron. As a result, even as the ratio of images to neurons increases, the number of leaked images can also continue to increase without having issues with multiple activations on the same neuron preventing reconstruction, up until the point where nearly each neuron leaks a separate image. Appendix Figure 15 shows the effect of the number of local iterations on the leakage rate.

## 5.4. Convolutional scaling factor

While the convolutional scaling factor ( $CSF$ ) allows us to have attacks without model inconsistency, there is a trade-off in model size. Figure 9 shows the model size for varying levels of model inconsistency (number of clients sharing the same model) relative to the size added by RtF when they both achieve the same leakage rate in FedSGD attacking 100 clients in aggregation (FedSGD in-



(a) Leakage rate and PSNR↑

(b) SSIM↑ and LPIPS↓

Figure 10. For LOKI (a) leakage rate and  $\text{PSNR}^\uparrow$ , (b)  $\text{SSIM}^\uparrow$  and  $\text{LPIPS}^\downarrow$  as a function of the  $\text{CSF}$ . The leakage rate and  $\text{PSNR}^\uparrow$  increase over a larger range of  $\text{CSF}$  values while the  $\text{SSIM}^\uparrow$  and  $\text{LPIPS}^\downarrow$  scores stop improving much quicker.

stead of FedAVG, since RtF cannot scale in FedAVG due to precision). This happens when the effective number of bins, *number of identity mapping sets*  $\times$  *FC layer size*, is the same as the total FC layer size in RtF. With full model inconsistency, LOKI adds only 50.69% of the size of the RtF attack. With no model inconsistency, LOKI is the same size at 100.00%. LOKI maintains the same leakage rate for all points. Recall that RtF has no model inconsistency.

The other purpose of the  $\text{CSF}$  is to mitigate changes in the identity mapping kernels and prevent images from activating the same neurons. Figure 10 show the leakage rate and average  $\text{PSNR}^\uparrow$ ,  $\text{SSIM}^\uparrow$ , and  $\text{LPIPS}^\downarrow$  scores of reconstructed images when using various  $\text{CSF}$  values (we report the average of reconstructed images without activation overlap, even when the  $\text{SSIM}$  is below 0.5 in order to properly show the impact of the  $\text{CSF}$  on the metrics). We average the values over 10 clients in FedAVG aggregation training on CIFAR-100 using 8 local iterations of mini-batch size 8 and  $\alpha = 1e - 4$ . With very small  $\text{CSF}$  values, the leakage rate is much lower because the convolutional kernel parameter changes prevent images from being pushed through correctly. Similarly, we see lower metric scores for reconstructed images. High  $\text{CSF}$  values result in larger changes to the FC layer weights and a higher leakage rate by preventing images from activating the same neurons. Due to smaller changes in the identity mapping sets, the reconstruction metrics also improve. The  $\text{PSNR}$  score improves over a large range of  $\text{CSF}$  values (continuing beyond  $\text{CSF} = 500$ ) while the  $\text{SSIM}$  and  $\text{LPIPS}$  score improvement stops much quicker. For example, the attack achieves a 72.59% leakage rate when  $\text{CSF} = 1$  and increases until  $\text{CSF} = 500$  where it peaks at 85.47% and no longer increases with larger  $\text{CSF}$  values.

## 5.5. Non-IID federated learning

Previous experiments worked under the setting that client data was IID and the biases were initialized the same for everyone. In these experiments, the clients contain non-IID distributions. We use OrganAMNIST instead of the CIFAR dataset, as it has a less uniform average pixel intensity distribution which is shown in Appendix Figure 19. For OrganAMNIST, label-based separation also results in individual distributions with larger differences in mean and SD. For this separation, each client has data from a single class. Since OrganAMNIST only has 11 classes, several

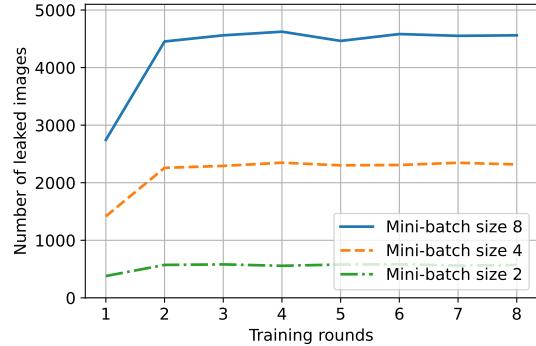


Figure 11. [Effect of non-IID clients] Number of images leaked for various local mini-batch sizes training on the OrganAMNIST dataset with 100 non-IID clients over several training rounds. The first training iteration starts with a dataset agnostic bias initialization and subsequent training rounds improve through observing activations. The attacks improve significantly after a single round.

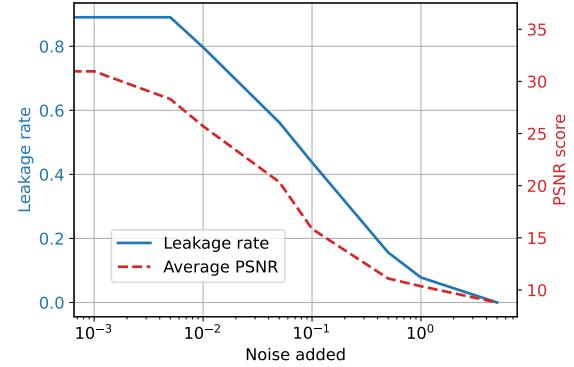


Figure 12. Leakage rate and average  $\text{PSNR}$  for client reconstruction in CIFAR-100 under  $\sigma$  Gaussian noise between  $1e-3$  and 5. Images with an  $\text{SSIM} > 0.5$  are counted as leaked. At  $\sigma = 5$ , no images are leaked.

clients share data from the same label. However, this is not known to the server so the distributions of clients are learned independently. We use a dataset agnostic bias initialization for the first round, where all clients have initial biases with mean -0.5 and SD 0.25 following a normal distribution. This value is adjusted after each training round separately for each client based on observing the activated biases.

We use 100 clients for each experiment each training over 8 local iterations, but we vary the local mini-batch size as 8, 4, and 2 (total of 64, 32, and 16 images). The FC layer size is setup equal to  $4 \times$  the total number of images and we use  $\text{CSF} = 100$ . Figure 11 shows the number of images leaked over several training rounds for the different local mini-batch sizes. The total number of images for each local mini-batch size is 6400, 3200, and 1600 respectively. After just the first training round of observing neuron activations, a jump in leaked images occurs for all mini-batch sizes. An increase of 62.32%, 59.53%, and 50.92% leaked images is observed for local mini-batch sizes 8, 4, and 2 respectively. After the initial jump, the total leakage fluctuates slightly between rounds due to randomness of the client mini-batch.

## 5.6. Adding noise

Applying noise locally is used as a common form of defense against privacy attacks. We explore the effects of

adding different levels of Gaussian noise to the attack on a single client. We quantify the effects of noise through the average reconstructed image PSNR score and the leakage rate (reconstructions with SSIM > 0.5). We use  $CSF = 500$  in order to achieve the maximum default leakage rate,  $\alpha = 1e - 1$ , and train with 4 local iterations of mini-batch size 16. Compared to FedSGD, the learning rate  $\alpha$  in FedAVG directly impacts the size of the computed gradient magnitude. Figure 12 shows the PSNR and leakage rate for various standard deviation  $\sigma$  amounts of noise between  $1e - 3$  and 5 added to the update trained on CIFAR-100. Only when the amount of noise added is  $\sigma = 5$ , is all image leakage prevented. However, while adding  $\sigma = 5$  noise to the model can prevent leakage, it will also destroy a model’s training accuracy. Appendix Figure 14 shows reconstruction results with added noise.

Increasing the number of clients in secure aggregation also increases the amount of noise added to the reconstructions. The noise in the aggregation works as adding multiple gaussian distributions together. The mean of the cumulative gaussian is the sum of the means of the individual distributions (in this case it is 0) and the variance is the sum of individual variances. In the above figure, the attacks achieves 0 leakage with added noise of STD  $\sigma = 5$ . If each client uses  $\sigma = 5e - 2$ , having  $N = \frac{(5)^2}{(5e-2)^2} = 10,000$  clients in aggregation will achieve the same amount of noise as a single client adding  $\sigma = 5$ .

## 6. Defenses and mitigation

As a major outcome of our work, we find that FedAVG using secure aggregation for defense does *not* prevent a malicious server from recovering large amounts of private user data regardless of the number of clients participating. The same is true for secure shuffling, discussed in the context of federated learning by [37]. Secure shuffling prevents a server from receiving any additional information (outside of the update itself) which would allow identification of which client sent each individual message. However, secure shuffling cannot nullify our attack as the update itself carries the fingerprint of the client. While reconstructing the image, the server can identify the client it came from.

One possibility of defense is through identification of a modified model architecture or the parameters. While LOKI uses a certain order of layers (convolutional layer followed by two FC layers), this module can be difficult to identify as the layers used can be further in the model architecture. Models can use multiple convolutional layers followed by FC layers. In this case, LOKI can use the final convolutional layer and subsequent FC layers as the leakage module. Figure 18 in the appendix shows reconstructions after max-pooling. While the reconstructions lose resolution, the content is still clear. LOKI is similar to Robbing the Fed [17] in stealthiness. Although it requires an additional convolutional layer, identification is just as difficult since it is easy to place this in multiple locations of the benign architecture of models such as ResNet or VGG by having prior layers act to push the input forward. A client could also attempt to

identify malicious tampering of parameters prior to training on local data and sending an update. However, this could also be difficult due to the server’s ability to mask the weight gradient in different ways to have the non-identity mapping sets output zero. Furthermore, the no model inconsistency attack does not use zero parameters. There is also still a fundamental problem with identification in that clients have limited computational abilities in the setting of cross-device FL, which may preclude such verification. They typically only follow a standard federated learning protocol given to them that consists of training the model and sending updates back to the server.

Model inconsistency defense has also been proposed in [20]. Having clients utilize a pseudorandom generator for SA using a condition that the received models are the same will prevent the server from being able to unencrypt the SA updates if it sends different models to each client. This method incurs no additional communication overhead and is applicable to SA algorithms such as [14], [38]. However, the practical scenario of asynchronous FL also uses SA [16], [39] but inherently has model inconsistency due to client staleness. Methods for on-device efficient training also send clients models with different architectures [40]. Furthermore, the server does not absolutely require model inconsistency for LOKI. By choosing to not use model inconsistency, the leakage rate of the attack is slightly lower, but the server can increase the stealth of the attack with no impact on the reconstruction quality or scalability (Figure 6).

One standard mitigation strategy in federated learning would be to use differential privacy [41], [42], [43] to add noise to updates prior to sending them to the server. While this method can be effective in preventing a server from fully reconstructing private data, it comes with the hefty downside in a decrease of model performance, especially with large vision models. Particularly, the large dimensionality of many vision models causes the impact of noise to be greater when achieving the same privacy guarantees [44]. The modification of the model due to the attack does not need to occur during every step of the training process. Without knowledge of when the attack will occur, differential privacy must then be applied during every step of the training process, thus seriously impacting accuracy.

## 7. Conclusions

In this paper, we have demonstrated how to break the privacy of secure aggregation in FedAVG federated learning through a malicious server that sends customized models to clients. Our key design idea is to send customized convolutional kernels to each client, an identity mapping set, that separates the weight gradients from the clients despite the use of secure aggregation. The server then uses these weight gradients to reconstruct the original data points. Using our proposed convolutional scaling factor, the attack can avoid model inconsistency and achieve a higher leakage rate in FedAVG than FedSGD attacks. We are the first to achieve a privacy attack in FL with FedAVG that scales well with the size of the local dataset and the number of

clients. For us to handle an increasing local dataset size, the fully connected layer size increases linearly. To handle an increasing number of clients, the size of the convolutional layer increases linearly and so the total number of parameters grows linearly, while the number of non-zero parameters stays constant. We achieve high reconstruction quality and a leakage rate between 76–86% for CIFAR-100, Tiny ImageNet, and MNIST with 100 clients, while that of the state-of-the-art is less than 1%.

**Acknowledgements.** This work was supported by Army Research Lab under Contract No. W911NF-2020-221, National Science Foundation CNS-2038986, Defense Advanced Research Projects Agency (DARPA) under Contract No. HR001120C0156, ARO award W911NF1810400, and ONR Award No. N00014-16-1-2189. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the sponsors.

## References

- [1] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas, “Communication-efficient learning of deep networks from decentralized data,” in *Artificial intelligence and statistics*. PMLR, 2017, pp. 1273–1282.
- [2] L. Melis, C. Song, E. De Cristofaro, and V. Shmatikov, “Exploiting unintended feature leakage in collaborative learning,” in *2019 IEEE symposium on security and privacy (SP)*. IEEE, 2019, pp. 691–706.
- [3] X. Luo, Y. Wu, X. Xiao, and B. C. Ooi, “Feature inference attack on model predictions in vertical federated learning,” in *2021 IEEE 37th International Conference on Data Engineering (ICDE)*. IEEE, 2021, pp. 181–192.
- [4] R. Shokri, M. Stronati, C. Song, and V. Shmatikov, “Membership inference attacks against machine learning models,” in *2017 IEEE symposium on security and privacy (SP)*. IEEE, 2017, pp. 3–18.
- [5] C. A. Choquette-Choo, F. Tramer, N. Carlini, and N. Papernot, “Label-only membership inference attacks,” in *International conference on machine learning*. PMLR, 2021, pp. 1964–1974.
- [6] M. Nasr, R. Shokri, and A. Houmansadr, “Comprehensive privacy analysis of deep learning: Passive and active white-box inference attacks against centralized and federated learning,” in *2019 IEEE symposium on security and privacy (SP)*. IEEE, 2019, pp. 739–753.
- [7] B. Hitaj, G. Ateniese, and F. Perez-Cruz, “Deep models under the gan: information leakage from collaborative deep learning,” in *Proceedings of the 2017 ACM SIGSAC conference on computer and communications security*, 2017, pp. 603–618.
- [8] Z. Wang, M. Song, Z. Zhang, Y. Song, Q. Wang, and H. Qi, “Beyond inferring class representatives: User-level privacy leakage from federated learning,” in *IEEE INFOCOM 2019-IEEE Conference on Computer Communications*. IEEE, 2019, pp. 2512–2520.
- [9] L. Zhu, Z. Liu, and S. Han, “Deep leakage from gradients,” *Advances in neural information processing systems*, vol. 32, 2019.
- [10] B. Zhao, K. R. Mopuri, and H. Bilen, “idlq: 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?” *Advances in Neural Information Processing Systems*, vol. 33, pp. 16 937–16 947, 2020.
- [12] H. Yin, A. Mallya, A. Vahdat, J. M. Alvarez, J. Kautz, and P. Molchanov, “See through gradients: Image batch recovery via gradient-inversion,” in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2021, pp. 16 337–16 346.
- [13] D. I. Dimitrov, M. Balunovic, N. Konstantinov, and M. Vechev, “Data leakage in federated averaging,” *Transactions on Machine Learning Research*, 2022.
- [14] K. Bonawitz, V. Ivanov, B. Kreuter, A. Marcedone, H. B. McMahan, S. Patel, D. Ramage, A. Segal, and K. Seth, “Practical secure aggregation for privacy-preserving machine learning,” in *proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, 2017, pp. 1175–1191.
- [15] H. Fereidooni, S. Marchal, M. Miettinen, A. Mirhoseini, H. Möllering, T. D. Nguyen, P. Rieger, A.-R. Sadeghi, T. Schneider, H. Yalame *et al.*, “Safelearn: secure aggregation for private federated learning,” in *2021 IEEE Security and Privacy Workshops (SPW)*. IEEE, 2021, pp. 56–62.
- [16] J. So, C. J. Nolet, C.-S. Yang, S. Li, Q. Yu, R. E Ali, B. Guler, and S. Avestimehr, “Lightsecagg: a lightweight and versatile design for secure aggregation in federated learning,” *Proceedings of Machine Learning and Systems*, vol. 4, pp. 694–720, 2022.
- [17] L. H. Fowl, J. Geiping, W. Czaja, M. Goldblum, and T. Goldstein, “Robbing the fed: Directly obtaining private data in federated learning with modified models,” in *International Conference on Learning Representations*, 2022.
- [18] F. Boenisch, A. Dziedzic, R. Schuster, A. S. Shamsabadi, I. Shumailov, and N. Papernot, “When the curious abandon honesty: Federated learning is not private,” *8th IEEE European Symposium on Security and Privacy (IEEE Euro S&P)*, 2023.
- [19] Y. Wen, J. Geiping, L. Fowl, M. Goldblum, and T. Goldstein, “Fishing for user data in large-batch federated learning via gradient magnification,” *International Conference on Machine Learning*, 2022.
- [20] D. Pasquini, D. Francatì, and G. Ateniese, “Eluding secure aggregation in federated learning via model inconsistency,” in *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, 2022, pp. 2429–2443.
- [21] S. Kariyappa, C. Guo, K. Maeng, W. Xiong, G. E. Suh, M. K. Qureshi, and H.-H. S. Lee, “Cocktail party attack: Breaking aggregation-based privacy in federated learning using independent component analysis,” in *International Conference on Machine Learning*. PMLR, 2023, pp. 15 884–15 899.
- [22] M. Lam, G.-Y. Wei, D. Brooks, V. J. Reddi, and M. Mitzenmacher, “Gradient disaggregation: Breaking privacy in federated learning by reconstructing the user participant matrix,” in *International Conference on Machine Learning*. PMLR, 2021, pp. 5959–5968.
- [23] Y. J. Cho, J. Wang, and G. Joshi, “Client selection in federated learning: Convergence analysis and power-of-choice selection strategies,” *arXiv preprint arXiv:2010.01243*, 2020.
- [24] W. Chen, S. Horváth, and P. Richtárik, “Optimal client sampling for federated learning,” *Transactions on Machine Learning Research*, 2022.
- [25] B. Pejó and G. Biczók, “Quality inference in federated learning with secure aggregation,” *arXiv preprint arXiv:2007.06236*, 2020.
- [26] J. So, R. E. Ali, B. Guler, J. Jiao, and S. Avestimehr, “Securing secure aggregation: Mitigating multi-round privacy leakage in federated learning,” *arXiv preprint arXiv:2106.03328*, 2021.
- [27] L. T. Phong, Y. Aono, T. Hayashi, L. Wang, and S. Moriai, “Privacy-preserving deep learning: Revisited and enhanced,” in *International Conference on Applications and Techniques in Information Security*. Springer, 2017, pp. 100–110.
- [28] L. Fan, K. W. Ng, C. Ju, T. Zhang, C. Liu, C. S. Chan, and Q. Yang, “Rethinking privacy preserving deep learning: How to evaluate and thwart privacy attacks,” in *Federated Learning*. Springer, 2020, pp. 32–50.

- [29] I. S. Duff, R. G. Grimes, and J. G. Lewis, “Sparse matrix test problems,” *ACM Transactions on Mathematical Software (TOMS)*, vol. 15, no. 1, pp. 1–14, 1989.
- [30] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga *et al.*, “Pytorch: An imperative style, high-performance deep learning library,” *Advances in neural information processing systems*, vol. 32, 2019.
- [31] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard *et al.*, “{TensorFlow}: a system for {Large-Scale} machine learning,” in *12th USENIX symposium on operating systems design and implementation (OSDI 16)*, 2016, pp. 265–283.
- [32] A. Krizhevsky, G. Hinton *et al.*, “Learning multiple layers of features from tiny images,” Master’s thesis, University of Toronto, 2009.
- [33] Y. Le and X. Yang, “Tiny imagenet visual recognition challenge,” *CS 231N*, vol. 7, no. 7, p. 3, 2015.
- [34] Y. LeCun, “The mnist database of handwritten digits,” <http://yann.lecun.com/exdb/mnist/>, 1998.
- [35] 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.
- [36] J. Yang, R. Shi, D. Wei, Z. Liu, L. Zhao, B. Ke, H. Pfister, and B. Ni, “Medmnist v2-a large-scale lightweight benchmark for 2d and 3d biomedical image classification,” *Scientific Data*, vol. 10, no. 1, p. 41, 2023.
- [37] P. Kairouz, H. B. McMahan, B. Avent, A. Bellet, M. Bennis, A. N. Bhagoji, K. Bonawitz, Z. Charles, G. Cormode, R. Cummings *et al.*, “Advances and open problems in federated learning,” *Foundations and Trends® in Machine Learning*, vol. 14, no. 1–2, pp. 1–210, 2021.
- [38] J. H. Bell, K. A. Bonawitz, A. Gascón, T. Lepoint, and M. Raykova, “Secure single-server aggregation with (poly) logarithmic overhead,” in *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, 2020, pp. 1253–1269.
- [39] J. Nguyen, K. Malik, H. Zhan, A. Yousefpour, M. Rabbat, M. Malek, and D. Huba, “Federated learning with buffered asynchronous aggregation,” in *International Conference on Artificial Intelligence and Statistics*. PMLR, 2022, pp. 3581–3607.
- [40] E. Diao, J. Ding, and V. Tarokh, “Hetero{fl}: Computation and communication efficient federated learning for heterogeneous clients,” in *International Conference on Learning Representations*, 2021.
- [41] C. Dwork, A. Roth *et al.*, “The algorithmic foundations of differential privacy,” *Foundations and Trends® in Theoretical Computer Science*, vol. 9, no. 3–4, pp. 211–407, 2014.
- [42] B. Jayaraman and D. Evans, “Evaluating differentially private machine learning in practice,” in *28th USENIX Security Symposium (USENIX Security 19)*, 2019, pp. 1895–1912.
- [43] K. Wei, J. Li, M. Ding, C. Ma, H. H. Yang, F. Farokhi, S. Jin, T. Q. Quek, and H. V. Poor, “Federated learning with differential privacy: Algorithms and performance analysis,” *IEEE Transactions on Information Forensics and Security*, vol. 15, pp. 3454–3469, 2020.
- [44] X. Zhang, X. Chen, M. Hong, Z. S. Wu, and J. Yi, “Understanding clipping for federated learning: Convergence and client-level differential privacy,” in *International Conference on Machine Learning, ICML 2022*, 2022.
- [45] S. Kadhe, N. Rajaraman, O. O. Koyleoglu, and K. Ramchandran, “Fastsecagg: Scalable secure aggregation for privacy-preserving federated learning,” *arXiv preprint arXiv:2009.11248*, 2020.
- [46] Y. Zhao and H. Sun, “Information theoretic secure aggregation with user dropouts,” *IEEE Transactions on Information Theory*, vol. 68, no. 11, pp. 7471–7484, 2022.
- [47] J. So, B. Güler, and A. S. Avestimehr, “Turbo-aggregate: Breaking the quadratic aggregation barrier in secure federated learning,” *IEEE Journal on Selected Areas in Information Theory*, vol. 2, no. 1, pp. 479–489, 2021.
- [48] A. R. Elkordy and A. S. Avestimehr, “Heterosag: Secure aggregation with heterogeneous quantization in federated learning,” *IEEE Transactions on Communications*, vol. 70, no. 4, pp. 2372–2386, 2022.
- [49] Z. Wang, A. C. Bovik, H. R. Sheikh, and E. P. Simoncelli, “Image quality assessment: from error visibility to structural similarity,” *IEEE transactions on image processing*, vol. 13, no. 4, pp. 600–612, 2004.
- [50] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein *et al.*, “Imagenet large scale visual recognition challenge,” *International journal of computer vision*, vol. 115, no. 3, pp. 211–252, 2015.

## Appendix A.

	<b>Leakage rate (images)</b>
<b>True initialization</b>	85.8% (5492)
<b>Dataset agnostic</b>	82.9% (5305)
<b>Random</b>	62.1% (3977)

TABLE 3. LEAKAGE RATE OF LOKI USING DIFFERENT BIAS INITIALIZATION METHODS. 100 CLIENTS ARE TRAINED ON CIFAR-100 IN FEDAVG AGGREGATION. THE LEAKAGE RATE ONLY DROPS BY 2.9% FOR THE DATASET AGNOSTIC INITIALIZATION.

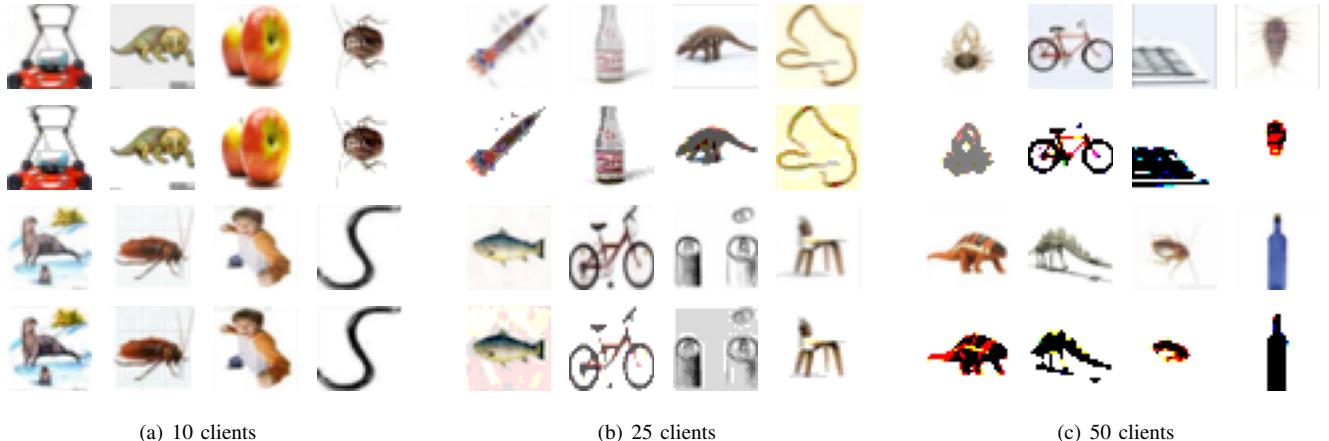
	<b>LOKI</b> <b>FedSGD</b>	<b>RtF</b> <b>FedSGD</b>
<b>CIFAR-100</b>	77.1% (4936)	77.1% (4931)
<b>Tiny ImageNet</b>	77.2% (4939)	77.7% (4970)
<b>MNIST</b>	72.0% (4610)	75.1% (4803)

TABLE 4. LEAKAGE RATE OF LOKI AND ROBBING THE FED (RtF) [17] ATTACK ON SEVERAL DATASETS IN FEDSGD. 100 CLIENTS IN AGGREGATION WITH A BATCH SIZE OF 64 ARE USED.

### A.1. Additional experiments

We show the leakage rate for LOKI under different bias initialization methods in Table 3. 100 clients are trained on the CIFAR-100 dataset with 8 local iterations and local mini-batch size 8, FC layer size 256,  $CSF = 500$ , and  $\alpha = 1e-4$  are used. For average pixel intensity, the actual dataset distribution for CIFAR-100 is  $\mu = 0.4782$  and  $\sigma = 0.1470$ . For the dataset agnostic initialization, we assume the server has no prior knowledge and the biases are initialized with  $\mu = 0.5$  and  $\sigma = 0.25$  as discussed in Section 4.1. The random method initializes all biases randomly between 0 and 1. The random initialization drops the leakage rate by 23.7% while the dataset agnostic initialization only drops the overall leakage rate by 2.9%.

Table 4 shows the leakage rate for LOKI and Robbing the Fed (RtF) [17] in the FedSGD case on the CIFAR-100, Tiny ImageNet, and MNIST datasets. We use the same settings as in Section 5.1 to allow for comparison of the leakage rate in FedAVG vs. FedSGD. For the FedSGD attacks, there are 100 clients in aggregation each with a batch size of 64. The  $CSF$  value does not impact the FedSGD attacks, so we use  $CSF = 1$ . We use the same FC size/convolutional kernels as in the FedAVG attacks in Section 5.1 for both methods. This also leads to a model size roughly  $2\times$  larger for RtF in FedSGD. Despite this, both methods achieve very similar leakage rates. However, compared to the leakage rate of LOKI in FedAVG, both FedSGD attacks achieve a lower leakage rate on all datasets.



(a) 10 clients

(b) 25 clients

(c) 50 clients

Figure 13. Top-8 SSIM reconstructed images for Robbing the Fed (RtF) [17] for (a) 10, (b) 25, and (c) 50 clients in FedAVG. Clients train with 8 local iterations of mini-batch size 8 with  $\alpha = 1e - 4$  on CIFAR-100. The 1st and 3rd rows are ground truth and the 2nd and 4th rows are the corresponding reconstructions. The leakage rate along with the quality of reconstructed images decreases with an increasing number of clients.

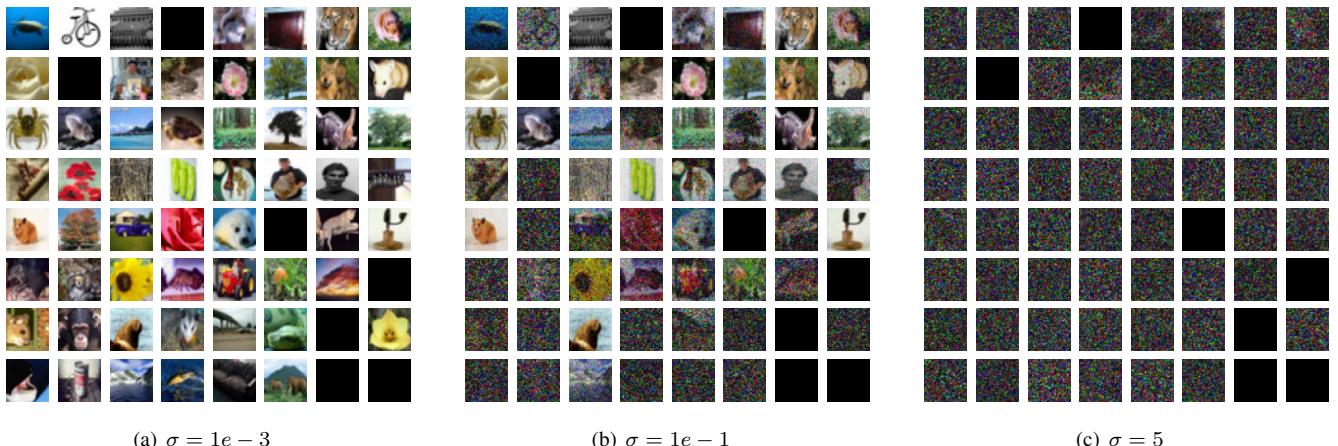
(a)  $\sigma = 1e - 3$ (b)  $\sigma = 1e - 1$ (c)  $\sigma = 5$ 

Figure 14. Reconstruction examples for a client with varying  $\sigma$  noise added to the update. Client training with  $\alpha = 1e - 1$ , 4 local iterations of mini-batch size 16 and  $CSF = 500$ . When (a)  $\sigma = 1e - 3$  the attack has the maximum leakage rate and the images are clearly identifiable. When (b)  $\sigma = 1e - 1$  the average SSIM is lower, but some images are visually identifiable. With (c)  $\sigma = 5$  all images are unidentifiable.

Figure 15 shows the leakage rate for several FC layer sizes with a local dataset size of 256 when varying the number of local iterations averaged over 10 clients. Clients train on CIFAR-100 with  $CSF = 500$  and  $\alpha = 1e - 4$ . Leakage rate is sampled with local iterations between 1-64, sampled by powers of 2. With a single local iteration, the attack leakage in FedAVG is the same as FedSGD with a batch size of 256.

## A.2. Secure aggregation in FL

Secure Aggregation [14] is one of the core privacy-preserving techniques in FL, that enables the server to aggregate local model updates from a number of clients, without observing any of their individual model updates in the clear. At their core, state-of-the-art secure aggregation protocols [14], [16], [26], [38], [45], [46], [47], [48] in FL rely on using additive masking to protect the privacy of individual models. In particular, in a secure aggregation protocol, each user  $i \in [N]$  encrypts its own model update  $\mathbf{y}_i^{(t)} = \text{Enc}(\mathbf{g}_i^{(t)})$  before sending it to the server in the  $t$ -

th communication round. This encryption is done such that secure aggregation guarantees the following:

**Correct decoding.** The encryption guarantees correct decoding for the aggregated model such that the server should be able to decode

$$\text{Dec} \left( \sum_{i \in [N]} \mathbf{y}_i^{(t)} \right) = \sum_{i \in [N]} \mathbf{g}_i^{(t)}, \quad (11)$$

**Privacy guarantee.** The encrypted model updates  $\{\mathbf{y}_i^{(t)}\}_{i \in [N]}$  leak no information about the model updates  $\{\mathbf{g}_i^{(t)}\}_{i \in [N]}$  beyond the aggregated model  $\sum_{i=1}^N \mathbf{g}_i^{(t)}$ . This is formally given as the following

$$I \left( \{\mathbf{y}_i^{(t)}\}_{i \in [N]}, \{\mathbf{g}_i^{(t)}\}_{i \in [N]} \middle| \sum_{i=1}^N \mathbf{g}_i^{(t)} \right) = 0, \quad (12)$$

where  $I(\cdot)$  represents the mutual information metric.

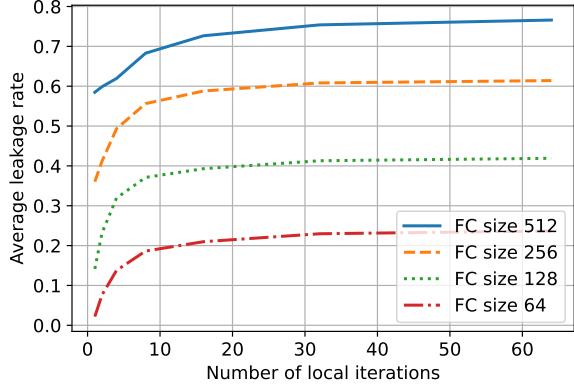


Figure 15. Leakage rate based on the number of local iterations and FC layer size averaged over 10 clients. The local dataset size is fixed at 256 for CIFAR-100. Increasing the local iterations increases the leakage rate.

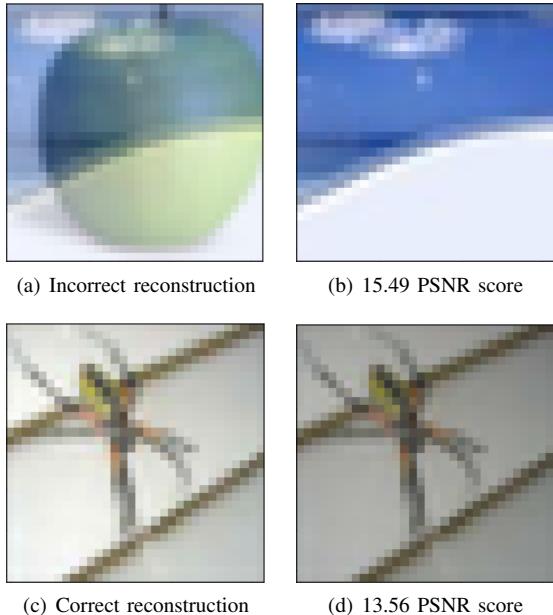


Figure 16. Reconstructed images from CIFAR-100 compared to the highest PSNR ground truth image. The incorrect reconstruction (a) has an overlap in the neuron activation while the correct reconstruction (c) has no overlap. The incorrectly reconstructed pair (a, b) has a higher PSNR score than the correct pair (c, d).

### A.3. Sample reconstructed images for other datasets

We show reconstructions of a randomly sampled client training on several datasets. There are 100 clients in aggregation training with 8 local iterations of mini-batch size 8,  $\alpha = 1e - 4$ , and  $CSF = 100$ . Figure 20 shows the MNIST dataset, Figure 21 shows the OrganAMNIST dataset, Figure 22 shows Tiny ImageNet, and Figure 23 shows ImageNet.

### A.4. Image metrics

For image reconstruction, the ability to identify an image and quantify the reconstruction quality are important for a metric to capture. We show from our attack results that the PSNR score achieves poor results in both categories.



Figure 17. Matching a reconstructed image from CIFAR-100 to the ground truth image in the batch using the highest (b) PSNR and (c) SSIM scores. SSIM chooses the correct image while PSNR chooses an incorrect image.

Previous works have also discussed that the use of mean squared error, which PSNR is based upon, is a poor reflection of image similarity. We use the SSIM [49] metric as a baseline for comparison and show that it serves as a better metric for perceptual similarity. Our empirical results on the reconstructed images also support the idea that PSNR is not the best choice for a reconstruction quality metric, especially for linear layer leakage attacks. Particularly, when reconstructing an image involves a large shift in the pixel value range, PSNR functions very poorly. This section shows several cases to demonstrate this.

When observing the PSNR score of reconstructions, even if an image is reconstructed incorrectly, the PSNR score can be higher than the score for a correct reconstruction. Figure 16 shows one such example, with two reconstruction results: the first being a failed reconstruction due to image overlap and the second being correct. Here the incorrectly reconstructed image has a higher PSNR score of 15.49 compared to the correct image which has a score of 13.56. The SSIM metric has the desired result, with a score of 0.91 for the correct image and 0.67 for the incorrect one.

A more extreme case occurs if we use the highest PSNR score to match reconstructed images to their ground truth counterparts. Figure 17 shows the result of choosing the corresponding ground truth image from the batch using the highest PSNR and SSIM scores. Comparing to the top ground truth image, the reconstruction in Figure 17(b) has a PSNR score of 16.45 and an SSIM score of 0.17. The reconstruction in Figure 17(c) has a PSNR score of 12.04 and a SSIM score of 0.88. As shown, using the maximum PSNR for image matching will often result in mismatches. In a batch of 100 images, there will usually be between 2-5 samples with an incorrect match. Comparatively, SSIM has no problems matching reconstructions to the ground truth.

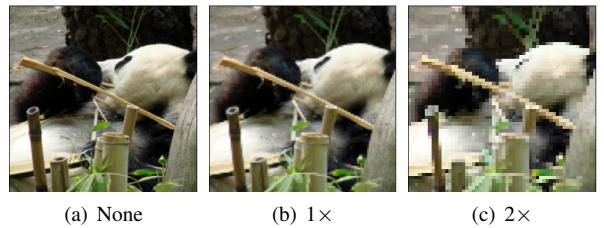
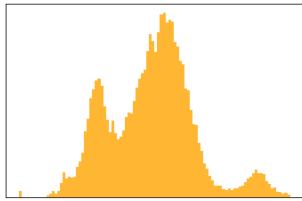
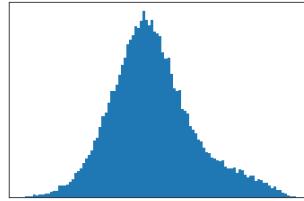


Figure 18. Sample from ImageNet dataset (a) without downsampling and after being downsampled (b) once and (c) twice. The number of max-pooling layers prior to reconstruction with an FC layer changes the amount of downsampling.



(a) OrganAMNIST



(b) CIFAR-100

Figure 19. Average pixel intensity dataset distribution for (a) OrganAMNIST and (b) CIFAR-100.

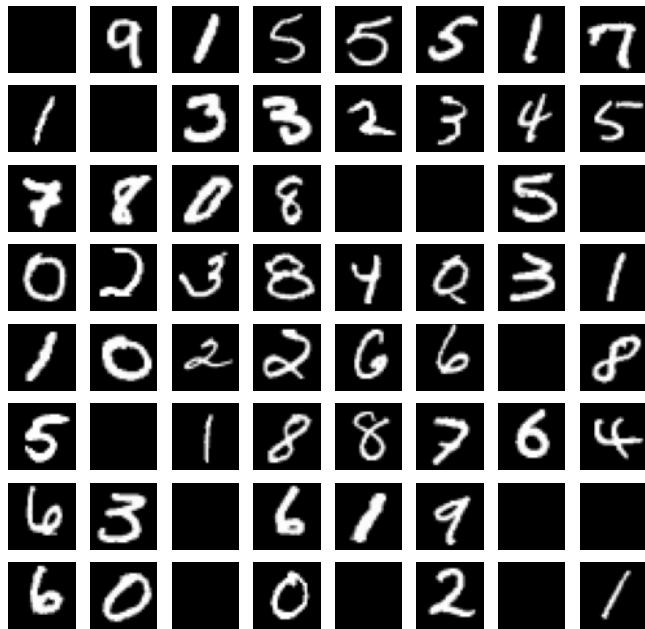


Figure 20. Leaked images for a randomly chosen client training on the MNIST dataset. Out of the 64 total images, 51 images are leaked.



Figure 21. Leaked images for a randomly chosen client training on the OrganAMNIST dataset. Out of the 64 total images, 51 images are leaked.

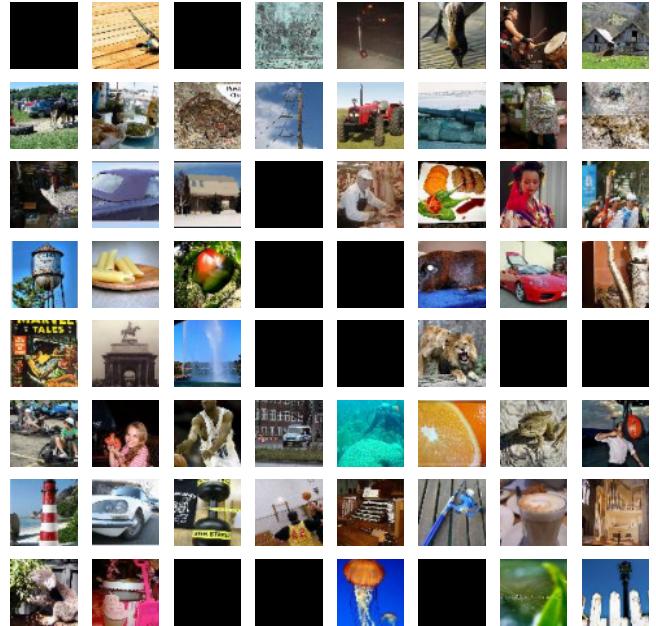


Figure 22. Leaked images for a randomly chosen client training on the Tiny ImageNet dataset. Out of the 64 total images, 52 images are leaked.

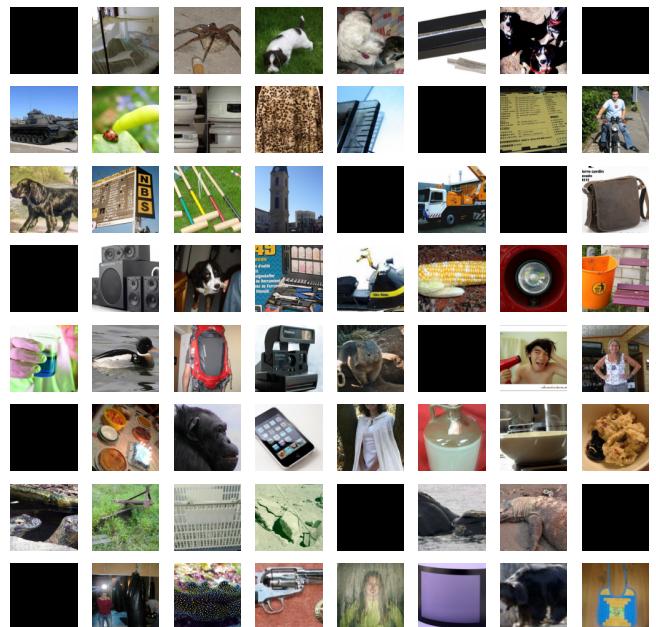


Figure 23. Leaked images for a single client training on the ImageNet [50] dataset. Out of the 64 total images, 53 images are leaked