2023 IEEE 8th European Symposium on Security and Privacy (EuroS&P) | 978-1-6654-6512-0/23/$31.00 ©2023 IEEE | DOI: 10.1109/EuroSP57164.2023.00020

When the Curious Abandon Honesty: Federated Learning Is Not Private

Franziska Boenisch*, Adam Dziedzic*†§, Roei Schuster*§,
Ali Shahin Shamsabadi*‡§, Ilia Shumailov*§, and Nicolas Papernot*†
*Vector Institute †University of Toronto ‡The Alan Turing Institute

***Abstract*—In federated learning (FL), data does not leave personal devices when they are jointly training a machine learning model. Instead, these devices share gradients, parameters, or other model updates, with a central party (*e.g.*, a company) coordinating the training. Because data never "leaves" personal devices, FL is often presented as privacy-preserving. Yet, recently it was shown that this protection is but a thin facade, as even a passive, honest-but-curious attacker observing gradients can reconstruct data of individual users contributing to the protocol.**

In this work, we show a novel data reconstruction attack which allows an active and dishonest central party to efficiently extract user data from the received gradients. While prior work on data reconstruction in FL relies on solving computationally expensive optimization problems or on making easily detectable modifications to the shared model's architecture or parameters, in our attack the central party makes inconspicuous changes to the shared model's weights before sending them out to the users. We call the modified weights of our attack *trap weights*.

Our active attacker is able to recover user data *perfectly*, *i.e.*, with zero error, even when this data stems from the same class. Recovery comes with near-zero costs: the attack requires no complex optimization objectives. Instead, our attacker exploits inherent data leakage from model gradients and simply amplifies this effect by maliciously altering the weights of the shared model through the trap weights. These specificities enable our attack to scale to fully-connected and convolutional deep neural networks trained with large mini-batches of data. For example, for the high-dimensional vision dataset ImageNet, we perfectly reconstruct more than 50% of the training data points from mini-batches as large as 100 data points. In textual tasks, such as IMDB sentiment analysis, more than 65% of data points from mini-batches containing 100 data points can be perfectly reconstructed.

1. Introduction

With machine learning (ML) being increasingly applied to sensitive data in critical use-cases such as health care [21], [39], smart metering [16], [49], or the internet of things [27], [36], there is a growing need for privacy-preserving training schemes that do not leak sensitive information. Federated learning (FL) is a widely popular distributed learning protocol [33] where user data can be utilized for jointly training an ML model without the data ever leaving the users' device. Instead, the device computes and sends model updates to a central party which aggregates them to produce a shared model. *Assuming* the model updates do not reveal the user data, FL would, thereby, preserve a notion of privacy.

Figure 1: **Original and Reconstructed Data.** Original data and data points extracted from model gradients with our trap weights attack. Extraction is **perfect** *i.e.* reconstruction error is zero.

This assumption has been repeatedly contested by prior work. It has been shown how the model updates sent to the central party not only leak training data membership [34] (*i.e.* allow the attacker to tell if a given data point was used in training) but also properties of the training data [14], [34]. Inspecting model updates allows attackers to even (partially) *reconstruct* [12], [15], [50], [53], [55], [56] users' training data. Ultimately, FL in its naive implementation offers little to no *guarantees* regarding potential leakage of user data to other users or to the central party.

Yet, existing data reconstruction attacks either are computationally expensive and yield low-fidelity extraction [55], [56], are limited to small mini-batch sizes [15], or require modifications of the model architecture that are trivially detected [12]. Another other line of concurrent work proposes modifications to the model parameters that are still easily noticeable: The attack introduced by Pasquini *et al.* [37] sets a noticeable portion of parameters to zero or negative values. Similarly, the attacks by Wen *et al.* [52] zero out many parameters of the last fully connected classification layer. In this work, we perform data extraction from large mini-batches of local data based on inconspicuous manipulations of the shared model weights. We start by showing scenarios where the gradients sent to the central party include full, memorized training data points. We then proceed to show that a malicious central party can significantly amplify this leakage by simply

§. Equal contribution.

DOI 10.1109/EuroSP57164.2023.00020

adversarially setting the model's weights with our *trap weights* method, prior to dispatching the weights to users.

Our trap weights mainly rely on re-scaling components in the model's weights matrix and can be applied to unmodified model architectures, which makes the attack more stealthy. By adversarially initializing the shared model with our trap weights, the central party can ensure they are able to *perfectly* extract a significant portion of the users' training data, as depicted in Figure 1. This even holds when the gradients are computed over large training data mini-batches containing only data from the same class, a scenario in which previous *optimization-based attacks* usually fail to obtain high-fidelity reconstructions [48]. Since in FL, the central party holds full control over the shared model weights that are sent out to users, our attack integrates naturally in the FL protocol. Furthermore, our attack is highly computationally efficient since it extracts individual training inputs by simply projecting the appropriate portions of the users' gradients onto the input domain. Finally, we show both in theory and in practice that our attack is equally successful when users perform multiple rounds of local training (Fed-Avg [33]) and send the model updates instead of the gradients to the central party.

In summary, we make the following contributions:

- We observe that in neural networks starting with a fully-connected layer, even gradients of large training data mini-batches contain *individual* training data points. In other words, in FL, mini-batch training data points are often directly sent from users to the central party, such that even an honest-but-curious central party has access to them.
- We show that a dishonest and active central party can amplify the leakage of individual training data points and extend it to other model architectures by adversarially initializing the weight of the shared model.
- In this setting, we perform data reconstruction on image and text data. Our attack is able to perform an extremely computationally efficient extraction of individual training data points in only a single-step computation over the received model updates with the attack setup depicted in Figure 2. For complex image datasets such as ImageNet [8], the attack yields perfect reconstruction of more than 50% of the training data points, even for large training data mini-batches that contain as many as 100 data points. For textual tasks such as IMDB sentiment analysis [32], it perfectly extracts more than 65% of the data points for mini-batches with 100 data points.

2. Background: Neural Networks, Federated Learning, and Differential Privacy

Neural Networks. Let $f_{\mathcal{W}} : \mathbb{R}^m \rightarrow \{1, \cdots, k\}$ be a k-class classifier defined as a set of l layers parameterized by trainable *weights* $\mathcal{W}$. Each layer consists of a linear operation paired with a non-linear activation function (e.g. ReLU). In this work, we consider two popular layer types: fully-connected and convolutional layers.

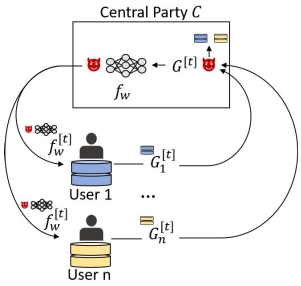


Figure 2: **Course of our Attack.** Our attack (😈) targets two points in the FL protocol: At iteration t, the central party actively manipulates the weights $\mathcal{W}$ of the shared model $f_{\mathcal{W}}$ before the model is sent out to the users. This causes the gradients $G_i^{[t]}$ of user i to contain individual training data points which the central party can then extract before calculating the averaged gradients $G^{[t]}$ and applying them to $f_{\mathcal{W}}$.

The goal of the model $f_{\mathcal{W}}$ is to map an input $x_i \in X$ to its desired ground-truth $y_i \in Y$. Therefore, the model weights $\mathcal{W}$ are adapted in a training process, most commonly with the mini-batch *Stochastic Gradient Descent* (SGD). To adjust the initial $\mathcal{W}$, mini-batch SGD repeats the following sequence of steps: (1) sample a mini-batch of size B from the training data $\{(X,Y)_b\}_{b=1}^{B}$, (2) take a forward pass through the model to obtain its predictions on the mini-batch, (3) compute the difference between predictions and ground-truth labels, called the *loss* $\mathcal{L}$, (4) compute the gradient of $\mathcal{L}$ w.r.t. the weights, called the *weight gradient* G, and update the weights accordingly.

To bootstrap mini-batch SGD, weights need to be initialized by sampling from a random distribution; popular distributions include the zero-mean Gaussian [17], *Xavier* [18] or *He* [22] distributions. The choice of distribution has a large effect on learning success [10]. In fact, when weights are maliciously initialized, the final model's utility might be degraded [20].

Federated Learning. FL [33] is a communication protocol for training a shared ML model $f_{\mathcal{W}}(\cdot)$ on decentralized data $\{(X_i,Y_i)\}_{i=1}^{\mathsf{N}}$ owned by N different users $\{u_i\}_{i=1}^{\mathsf{N}}$. Since collecting and managing all the data centrally might be costly, time consuming, and stand in conflict with the confidentiality of these respective users' data, FL enables each user to keep their data locally. A central party coordinates the training of the shared model by iteratively aggregating gradients computed locally by users.

More formally, let $t \in \{1, \cdots T\}$ be the current iteration of the FL protocol. At iteration $t = 0$, the model $f(\cdot)$ is initialized (at random) by the central party denoted as C. Let $f_{\mathcal{W}}^{[t]}(\cdot)$ be the model with its weights $\mathcal{W}^{[t]}$ at iteration t. At every iteration t, M out of the N ($\mathsf{M} \ll \mathsf{N}$) users are selected to contribute to the learning. Then, each of the selected M users u_i obtains $f_{\mathcal{W}}^{[t]}(\cdot)$ from C and calculates the gradients $G_i^{[t]}$ for $f_{\mathcal{W}}^{[t]}(\cdot)$ based on one mini-batch b sampled from their local dataset $(X_i,Y_i)_b$. In other words, the user computes the gradient $G_i^{[t]} = \nabla_{\mathcal{W}}\mathcal{L}((X_i,Y_i)_b)$. Each u_i uploads their gradients to C, who then averages all of these gradients to update the shared model's param-

eters:

$$G^{[t]} = \frac{1}{\mathsf{M}} \sum_{i=1}^{\mathsf{M}} G_i^{[t]}, \quad \mathcal{W}^{[t+1]} = \mathcal{W}^{[t]} - \eta G^{[t]}. \quad (1)$$

FL, thereby, represents a decentralization of the mini-batch SGD (i.e. distributed training from mini-batches of user data).

3. Existing Data Reconstruction Attacks

This section introduces prior work on passive and active data reconstruction attacks in FL and discusses the limitations of attacks based on iterative optimization.

3.1. Passive Attackers

Passive attackers performing data reconstruction attacks in FL can simply observe the received gradients but not maliciously manipulate the protocol. Phong *et al*. [38] were the first to show how gradients leak information that can be used to recover training data from single neurons or linear layers. Recent work [15], [23], [38], [45], [50], [53], [55], [56] proposed that the central party or users involved in FL training launch data reconstruction attacks based on either training a Generative Adversarial Network [19] (GAN) or solving a second order optimization problem.

Optimization-based Instance Reconstruction Attacks. Several attacks aim to reconstruct individual user data points while also relaxing the assumption that data labels are available to the attacker. Zhu *et al*. [56] proposed Deep Leakage from Gradients (DLG), where a data reconstruction attack is formulated as a joint optimization problem on the labels and input data, see Algorithm 2 in Appendix A. iDLG [55] sped up the convergence rate of DLG [56] by analytically computing the labels based on the users' gradients of the last layer. These works, and other optimization-based ones [15], are limited to a setting where mini-batches only contain a single example, i.e., $B = 1$. GradInversion [53] regularizes DLG's objective to improve the extraction fidelity, attaining some success in extraction for mini-batches of size $B > 1$. In Section 7.4 we compare performance of our approach against a state-of-the-art optimization-based attack. Our attack is superior in extracting individual training data even for large mini-batch sizes of $B \geq 100$, and being far more computationally efficient (even for passive adversaries in the honest-but-curious model). We present a more thorough overview on passive data reconstruction attacks in Appendix A.

Limitations of Optimization-Based Attacks. We hereby provide a brief exposition to Zhu *et al*. [56]'s DLG, as a representative case study of an optimization-based attack. Their approach, characteristic of optimization-based data reconstruction attacks, is given in Algorithm 2 in Appendix A. It firstly randomly initializes a "dummy data point and corresponding label" $(\hat{\mathbf{x}}, \hat{y})$ and computes the resulting "dummy gradients" as $\hat{G} = \nabla_{\mathcal{W}^t} \mathcal{L}(f_{\mathcal{W}^t}(\hat{\mathbf{x}}), \hat{y})$. Then, they iteratively optimize the dummy data to produce gradients that are close to the original gradients G_i^t by solving:

$$\mathbf{x}_i^* = \arg\min_{\hat{\mathbf{x}}} \|G_i^{[t]} - \hat{G}\|^2 \quad (2)$$

$$y_i^* = \arg\min_{\hat{y}} \|G_i^{[t]} - \hat{G}\|^2. \quad (3)$$

DLG often fails to reconstruct high-fidelity data points and discover the ground-truth labels consistently because of a lack of convergence in the optimization. While other methods offer improvements (*e.g*. iDLG [55] sped up the convergence by simplifying the objectives in Equations 2 and 3 from both data and label reconstruction to only data reconstruction; and GradInversion [53] adds useful regularization), they suffer from the same pathology.

We identify several reasons for this. First, the gradient of the loss is non-injective *i.e*. is not invertible everywhere: different mini-batches may yield nearly identical gradients [43]. This holds whether the user samples mini-batches that contain multiple data points or a single data point only, *i.e*. $B = 1$. Second, optimization-based attacks converge to different minima due to the underlying randomness (see step 1 in Algorithm 2 in the Appendix). These minima correspond to different possible reconstructions of the input that often differ from the original training points [53]. Third, optimization-based attacks are computationally expensive: they either need to train a GAN or solve a second-order gradient optimization problem. Instead, our attack extracts *exact* data points from the gradients without any optimization or GAN training.

3.2. Active Attackers

In the work most similar to ours, [12] considers a threat model with an active and dishonest central party, similar to our setup. This attack relies on the existence of a fully-connected layer early within the network (otherwise, the attack adds it). Since this layer's weights have to contain many weight rows with the exact same weight values, this layer is inherently detectable.[1] Additionally, they do not discuss passive analytical-extraction attacks. Finally, our work generalizes their setup and performs successful extraction also for textual data.

In follow-up work, [52] proposes an attack that requires modifications to the model parameters (specifically, to the last fully-connected classification layer) sent to a user but without changing the model architecture. The attack extracts single data points by increasing the gradient contribution of a target data point and decreasing the gradient contribution of other data points. The final goal of an attacker is to reduce an aggregated gradient to an update calculated on a single sample. The attack is easily detectable since it requires many parameters in the last layer to be zeroed out. Moreover, our attack extracts individual data points in a single training round while

1. This is inherent to the attack because the method relies on each row computing the exact same function on the data and binning its result by varying only the bias term, such that it becomes likely that a bin contains only one input. Conversely, our trap weights are initialized such that it is likely that an output neuron is only activated for a single input in a mini-batch while avoiding imposing a highly regular structure on the weight matrix.

their approach requires a collection of many updates from an individual user. In the cross-device FL setting where participants get randomly sampled from millions of users, it is possible that single users participate fewer times than required by the attack.

In another active attack proposed by [37], a server sends distinct malicious parameters to individual users. The main purpose of the attack is to circumvent the protection of Secure Aggregation (SA) in FL and enable the central party to learn individual model updates from a target user. However, the work does not propose individual user-data point extraction, as enabled by out trap weights. We argue that by including our trap weights into their attack and sending our trap weights to the target user, they could efficiently extract this target user's private data.

4. Threat Model and Assumptions

This section presents our threat model in terms of the assumed attacker, the FL deployment, and the assumptions required for our attack to succeed.

4.1. The Attacker

Our attacker aims at extracting individual training data points from a chosen subset of the participating users. Therefore, the attacker's primary vantage point is the central party who is in charge of orchestrating the FL protocol. The assumption here is that, for example, the company orchestrating the FL protocol or potential rogue employees, are untrusted. This is the same attacker that FL is meant to defend against by leaving data on the users' devices. For brevity, in the following, we will refer to the central party as the attacker, even though the attacker can be a third party controlling the central party to deploy our trap weights-attack.

In FL, the central party initiates the FL protocol and chooses the task to train the shared ML model for. Therefore, the central party is aware of the type, domain, and dimensionality of data held by the users. It instantiates the shared model appropriately to learn from this data. Furthermore, in the standard FL scenario considered in this work, the central party holds full control over the shared model weights and can read users' gradient updates that are sent back. Finally, our central party is in charge of sampling the users who contribute their gradients in a given round—following standard deployments of the protocol [5]. This allows the central party to even run targeted attacks against specific users.

4.2. Assumptions and FL Setup

Following prior work [5], we consider an FL protocol where users calculate the model gradients locally on one (potentially large) mini-batch of their training data and share the resulting gradients directly with the central party. We assume that the data features are scaled in the range $[0, 1]$, which is a standard pre-processing step in ML. When users have abundant amounts of data, they can perform local gradient calculation and averaging over more than one mini-batch, see evaluation in Section 7.3.

We, furthermore, assume that the attacker is in possession of a small amount (*e.g.*, one mini-batch) of data from the users' private data domain. This is no strong assumption given that the central party chooses the ML task and has to instantiate the ML model appropriately.

The weight-manipulation attacks we study in this paper are not agnostic of the model architecture. In designing the attack, we focus on victim models that contain a ReLU-based fully-connected layer, and we experiment with several different types of such networks. This is not a material limitation: the approach of manipulating shared model weights to promote leakage is very flexible, and can be extended to cover many more architectures as needed using similar techniques. For example in Section 6.3 and Appendix B, we show how to extend the attack to work on networks that contain convolutional layers and in Section 7, also experimentally evaluate extraction under the presence of a token-embedding layer.

4.3. Course of Attack

The course of our attack is illustrated in Figure 2. In a given round of the protocol, the central party maliciously manipulates the shared model with our trap weights. Note that the central party does not necessarily attack all users in every round of the protocol. Instead, it can target one or several specific users in one or more chosen round(s). To target a subset of the M users at iteration t, the central party can send out different models to users under attack and other users [37]: while the targeted users receive a model initialized with our trap weights, all other users receive the shared model used to train the ML task. Attacking only a few users in a few rounds makes the attack more stealthy and allows the central party to train a performant shared model based on the gradient updates received in the benign rounds or from non-targeted users.

After receiving the gradients from users under attack, the central party simply projects the appropriate portions these gradients onto the input domain. In the following section, we will show how this approach can yield perfect extraction of the users' data points.

5. Passive Analytical Extraction for FC-NNs

Here, we show how the gradients of an FC-NN directly leak the individual training data points they are computed on, even to a passive attacker who just observes said gradients. In Section 5.1, we formally show that for a single training data point, *i.e.* a mini-batch size of $B = 1$, perfect extraction from the network gradients is possible. Then, in Section 5.2, we motivate why it is also possible to perfectly extract a small number of individual data points from gradients, even when working with larger mini-batches of size $B > 1$. However, the success of this passive extraction attack drops as the mini-batch sizes increase. This limitation motivates our active adversarial weight initialization attack, which we introduce in Section 6.

5.1. Single-Input Gradients Directly Leak Input

It has been shown by Geiping *et al*. [15] that a single input data point $\mathbf{x}$ can be reconstructed from the gradients of any fully-connected layer which is preceded

only by fully-connected layers and contains a bias $\mathbf{b}$. This holds if the gradient of the loss w.r.t. the layer's output $\mathbf{y} = \mathrm{ReLU}(W\mathbf{x}+\mathbf{b}) = \max(0, W\mathbf{x}+\mathbf{b})$ contains at least one non-zero entry. For detailed proof of the above see Proposition D.1 in [15]. In particular, when considering the first model layer, reconstructing its input data *directly* corresponds to obtaining the original input data point $\mathbf{x}$. Let y_i denote the output of the i^{th} neuron of the first and fully-connected layer of a model, and let $\mathbf{w}_i^T$ be the corresponding row in the weight matrix and b_i the corresponding component in the bias vector. Assume $\mathbf{w}_i^T\mathbf{x} + b_i > 0$, and therefore, $\mathrm{ReLU}(\mathbf{w}_i^T\mathbf{x} + b_i) = \mathbf{w}_i^T\mathbf{x} + b_i$. The reconstruction of the input $\mathbf{x}$ is done by calculating the gradients of the loss w.r.t. the bias and the weights as follows:

$$\frac{\partial \mathcal{L}}{\partial b_i} = \frac{\partial \mathcal{L}}{\partial y_i}\frac{\partial y_i}{\partial b_i} = \frac{\partial \mathcal{L}}{\partial y_i} \tag{4}$$

since $\frac{\partial y_i}{\partial b_i} = 1$, where $y_i = \mathbf{w}_i^T\mathbf{x} + b_i$.

$$\frac{\partial \mathcal{L}}{\partial \mathbf{w}_i^T} = \frac{\partial \mathcal{L}}{\partial y_i}\frac{\partial y_i}{\partial \mathbf{w}_i^T} = \frac{\partial \mathcal{L}}{\partial b_i}\mathbf{x}^T \tag{5}$$

Thus, if any $\frac{\partial \mathcal{L}}{\partial b_i} \neq 0$, perfect reconstruction is given by:

$$\mathbf{x}^T = (\frac{\partial \mathcal{L}}{\partial b_i})^{-1}\frac{\partial \mathcal{L}}{\partial \mathbf{w}_i^T} \tag{6}$$

According to Equation (5), the gradient of the loss w.r.t. the weights directly contains a scaled version of the input data. The exact scaling factor is $(\frac{\partial \mathcal{L}}{\partial b_i})$, which is the gradient of the loss w.r.t. the bias. This gradient is computed in the regular backward pass together with the gradient of the weights. Therefore, obtaining the scaling factor by just reading it from the gradients of the bias and inverting it to $(\frac{\partial \mathcal{L}}{\partial b_i})^{-1}$ comes at zero costs and the factor can be directly applied to rescale the gradient of the weights and obtain the input data point $\mathbf{x}$, see Equation (6). Intuitively, the reason why there is a rescaled version of the input data in the gradients and why this would be beneficial for learning can be motivated by revisiting the simple perceptron algorithm [13]. When an input is misclassified, the weight update in the perceptron algorithm consists simply in adding this input to the weights, which makes the algorithm learn.

5.2. Mini-batch Gradients Directly Leak Some Individual Inputs

It turns out that individual data point leakage is not limited to gradients computed over a mini-batch of size $B = 1$: we observe that gradients computed over larger mini-batches also sometimes leak individual training points. To forge an intuition for this phenomenon, Figure 8 in Appendix D visualizes the gradients of the first fully-connected layer's weight matrix of the FC-NN described in Table 9. We see that we are able to clearly distinguish some of the training data points within the rescaled gradients. This is despite the fact that these gradients were computed over a mini-batch of $B = 100$ inputs sampled from the CIFAR10 dataset.

Why do some gradients contain individual training data points? We denote a training data mini-batch by $X = \{x_1, x_2, \cdots, x_B\} \in \mathbb{R}^{(m\times B)}$ with $B > 1$. The gradient of this mini-batch X is equal to the average of all gradients computed for each of the data points $\{x_1, x_2, \cdots, x_B\}$ that make up the mini-batch. Let y_i denote again the output of the i^{th} neuron of the fully-connected layer, and let $\mathbf{w_i}$ and b_i be the corresponding row in the weight matrix and the component in the bias vector, respectively. Then the gradient $\mathbf{G}_{\mathbf{w}_i^T}$ and G_{b_i} of $\mathbf{w_i}$ and b_i can be computed as follows:

$$\begin{aligned}\mathbf{G}_{\mathbf{w}_i^T} &= \frac{1}{B}\sum_{j=1}^{B}\frac{\partial \mathcal{L}}{\partial y_{(i,j)}}\frac{\partial y_{(i,j)}}{\partial \mathbf{w}_i^T}\\ G_{b_i} &= \frac{1}{B}\sum_{j=1}^{B}\frac{\partial \mathcal{L}}{\partial y_{(i,j)}}\frac{\partial y_{(i,j)}}{\partial b_i}\end{aligned} \tag{7}$$

with $y_{(i,j)} = \mathrm{ReLU}(\mathbf{w}_i^T\mathbf{x}_j + b_i)$. These equations illustrate that the gradient $\mathbf{G}_{\mathbf{w}_i^T}$ over the data mini-batch X contains a weighted overlay of all the input data points $\mathbf{x}_j$ from the mini-batch. The weighting, therein, depends on the contribution of each data point to the model loss $\mathcal{L}$.

We observe that, in some cases, all but one training data point $\mathbf{x}^*$ from the data mini-batch have zero gradients. This is due to the $\max$ operation in $\mathrm{ReLU}(\mathbf{w}_i^T\mathbf{x} + b_i) := \max(\mathbf{w}_i^T\mathbf{x} + b_i, 0)$. When $\mathbf{w}_i^T\mathbf{x} + b_i$ is negative, the ReLU outputs zero, which results in zero gradients for the corresponding data point. When the gradients are zero for all data points but for the one data point $\mathbf{x}^*$, the weight gradient $\mathbf{G}_{\mathbf{w}_i^T}$ from Equation (7) becomes $\mathbf{G}_{\mathbf{w}_i^T} = \frac{1}{B}\frac{\partial \mathcal{L}}{\partial y_{(i,*)}}\frac{\partial y_{(i,*)}}{\partial \mathbf{w_i^T}}$ with $y_{(i,*)} = \mathrm{ReLU}(\mathbf{w}_i^T\mathbf{x}^* + b_i)$. This reduces the data extraction from the case of $B > 1$ to the case of $B = 1$, for which we saw in Section 5.1 that the data point $\mathbf{x}^*$ can be perfectly extracted. In other words, $\mathbf{w}_i^T\mathbf{x} + b_i$ being negative for all data points but one results in accidental leakage of that data point—enabling its exact reconstruction by a passive adversary.

5.3. Individual Inputs still Leak from Mini-batch Gradients computed in FedAvg

FedAvg is another popular protocol for FL [33] where users do not send their gradients after each local iteration of training. Instead, they calculate T' many local epochs over l mini-batches of their data. After each iteration t' of the total $T' \cdot l$ many local iteration, they update the model according to the respective gradients of the weights and biases, and a learning rate η as $f_{\mathcal{W}}^{t'+1}(\cdot) = f_{\mathcal{W}}^{t'}(\cdot) - \eta(\frac{\partial \mathcal{L}}{\partial \mathbf{w}^{t'}}, \frac{\partial \mathcal{L}}{\partial b^{t'}})$. Once the local training is completed, the users send the updated shared model $f_{\mathcal{W}}^{T'}(\cdot)$ to the central party. By calculating the difference between the shared model $f_{\mathcal{W}}^{0}(\cdot)$ sent to the user and the obtained model, and by re-scaling according to η, the central party obtains the value of the user's local model update as $\sum_{t'=1}^{T'}\frac{\partial \mathcal{L}}{\partial \mathbf{w}^{t'}}, \sum_{t'=1}^{T'}\frac{\partial \mathcal{L}}{\partial b^{t'}} = \frac{f_{\mathcal{W}}^{0}(\cdot) - f_{\mathcal{W}}^{T'}(\cdot)}{\eta}$. According to Equation (5), after every local iteration t' the gradient of the local weights $\frac{\partial \mathcal{L}}{\partial \mathbf{w}^{t'}} = \mathbf{x}\frac{\partial \mathcal{L}}{\partial b^{t'}}$. Therefore, $\sum_{t'=1}^{T'}\mathbf{w}^{t'} = \sum_{t'=1}^{T'}\mathbf{x}b^{t'} = \mathbf{x}\sum_{t'=1}^{T'}b^{t'}$. Since the server knows $\sum_{t'=1}^{T'}b^{t'}$ from the model update, it can multiply $\left(\sum_{t'=1}^{T'}b^{t'}\right)^{-1} \cdot \sum_{t'=1}^{T'}\mathbf{w}^{t'} = \mathbf{x}$ and extract the user

data perfectly. We experimentally validate this theoretical insight on large mini-batches at the end of Section 7.3.

6. Active Adversarial Initialization of the First Fully-Connected Layer

Section 5 illustrates under which conditions model gradients leak data points to a passive attacker capable of observing these gradients. In the following, we show how an active attacker can amplify previously-accidental leakage during the passive attack by controlling the weights $\mathbf{w}_i$ and biases b_i. For example, while a passive attacker can extract roughly 20% of arbitrary data points from a batch size $B = 100$ for 1000 neurons (*i.e.* weight rows in the fully-connected layer) on ImageNet, the active attack can more than double the number of extracted data points to 45%. Next, we show how to make such malicious choices to extract a larger number of individual training data points from model gradients.

6.1. Intuition of our Trap Weights

Without loss of generality, we will suppress the bias term in the following considerations. The multiplication of a single weight row $\mathbf{w}_i$ corresponding to the i^{th} neuron at the fully-connected layer with some input data point $\mathbf{x}$ can be expressed as a weighted sum of all of the features in $\mathbf{x}$ as follows

$$\mathbf{y}_i = \mathbf{w}_i^T \mathbf{x} = \sum_{j=1}^{m} w_i^{(j)} x_j. \tag{8}$$

In weight row $\mathbf{w}_i$, let $\mathtt{N}$ and $\mathtt{P}$ denote the sets of indices that hold the negative and positive weight components, respectively. Given ReLU activation, the i^{th} neuron is only activated on $\mathbf{x}$ if the sum of the features weighted by the negative components is smaller than the sum of the features weighted by the positive components:

$$\sum_{n \in \mathtt{N}} w_i^{(n)} x_n < \sum_{p \in \mathtt{P}} w_i^{(p)} x_p. \tag{9}$$

Therefore, $\mathbf{x}$ will yield non-zero gradients at the i^{th} neuron if and only Equation (9), holds for its features.

When the inequality holds only for a single data point in a mini-batch, this data point can be individually extracted from the gradients, as described in Section 5.2. The idea behind our trap weights is to set the components within each weight row corresponding to the neurons of the first fully-connected layer, such that Equation (9) only holds relatively rarely in inputs, and is therefore likely to only hold for a single data point within a mini-batch.

6.2. Adversarial Weight Initialization

Intuitively, our approach adversarially initializes each row of the weight matrix to increase the likelihood that only one data point in a given mini-batch will activate the neuron corresponding to that row. To achieve this, we initialize a randomly chosen half of the components of the weight row to negative values, and the other half to the corresponding positive values, by sampling from a Gaussian normal distribution. The positive components of the weight row are scaled down with a small factor $s < 1$ in comparison to the negative components. This increases the impact of the negative components on the weighted input sum to the corresponding neuron. This causes most input data points to produce non-positive input to the neuron, such that only a few (in the best case only one) input data point activates the neuron. See Algorithm 1 for a formalization of our initialization.

Algorithm 1: Adversarial Initialization of a Weight Row.

Input: Weight row $\mathbf{w_i}$ of length L, Gaussian distribution $\mathcal{N}(\mu, \sigma)$ with mean μ and std σ, Scaling factor $s < 1$, Discrete uniform distribution $\mathcal{U}(\cdot, \cdot)$

Output: Adversarially initialised weight row $\mathbf{w_i}$

$\mathtt{N} = \{i | i \sim \mathcal{U}(1, L)\}$ where $|\mathtt{N}| = \frac{1}{2}L$ ▷ Select randomly indices for negative weights

$\mathtt{P} \leftarrow \{i \notin \mathtt{N} | i \in [L]\}$ ▷ Select indices for positive weights

$\mathbf{z}_- \sim \mathcal{N}(\mu, \sigma) | \mathbf{z}_- \in \mathbb{R}_-^{\frac{1}{2}L}$ ▷ Negative samples

$\mathbf{z}_+ = -s \cdot \mathbf{z}_-$ ▷ Positive samples

$\mathbf{w_i}[\mathtt{N}] \leftarrow \mathrm{Shuffle}(\mathbf{z}_-)$ ▷ Initialize negative weights

$\mathbf{w_i}[\mathtt{P}] \leftarrow \mathrm{Shuffle}(\mathbf{z}_+)$ ▷ Initialize positive weights

We use the scaling factor s to specify how much larger the absolute values of the negative weight components should be than the positive values. This determines how "aggressively" our activation causes weighted inputs to individual neurons to be negative, thereby to be filtered out by the ReLU function and to have zero gradients for most input data points. The ideal value of s when it comes to attack effectiveness is dataset-dependent. The attacker can to fine-tune s either on a small amount of data from the users' input domain it holds before sending the trap weights to the users. Alternatively, they can fine-tune s without any data from the users' input domain and solely by exploiting the passive data leakage, or using data with the same dimensionality as the users' data as we show in Section 7.3.

Our adversarial initialization causes the ReLU function for many neurons at the fully-connected layer to activate only for one input data point per mini-batch. Due to the randomness in the initialization of each weight row corresponding to a neuron, different neurons are likely to be activated by different input data points. Thereby, the gradients of different weight rows allow for the extraction of different individual data points. We demonstrate the success of our trap weights for data extraction in Section 7.3 by showing that they increase the proportion of neurons that only activate on one random individual data point in a mini-batch by more than factor 10, and thus we are able to extract more than double the number of individual training data points. *E.g.* our trap weights cause 51.4% of active neurons out of 1000 to by activated by individual data points from the ImageNet dataset while random model weights with a Gaussian normal initialization with $\sigma = 0.5$ only yield 4.4%. This allows for an individual extraction of 45.7% of the data points in a

mini-batch of size $B = 100$ for our trap weights versus 21.8% with random model weights.

6.3. Trap Weights for Other Architectures

To enable perfect extraction, our attack relies on the presence of a ReLU-based fully-connected layer at the beginning of the model architecture. Since in FL, the central party is in charge of instantiating the model architecture, this does not represent a practical limitation.

For some application domains, the central party might want to train ML models beyond pure FC-NNs though, *i.e.*, models where the first layer is not fully-connected. In Appendix B, we show how an attacker can apply malicious manipulations to shared model's weights to extend our attack to CNN-based architectures. These architectures consist of several convolution layers and some fully-connected layers which the attacker can leverage for extraction. The intuition of the attack-extension is to convert the convolution layers to identity functions which transfer the user's input data to the first fully-connected layer in the model. The attacker initializes this layer with our trap weights and can then extract user data. In Appendix B.3, we discuss how to also make the manipulations of the convolutional layers most stealthy.

In the following section, we also evaluate extraction for text-data in model architectures that contain an embedding layer before the first fully-connected layer. Extraction is done from the fully-connected layer whose input consists of the embedding layer's output. To reconstruct the original text tokens from a sequence of extracted embeddings, the attacker creates a lookup dictionary, mapping its initialized embeddings back to their corresponding tokens (this is the inverse mapping to the embedding layer). To avoid vector-comparisons for each lookup, the attacker uses hash values for vector embeddings as keys.

7. Experimental Evaluation

In this section, we validate that our adversarial weight initialization attack allows a central party to reconstruct individual training data points from gradients shared by users. We use three different image datasets, namely MNIST [31], CIFAR10 [29], and ImageNet [8] and the text-based IMDB [32] dataset for sentiment analysis. Because our approach is applicable to FC-NNs and CNNs, we test it against both of these architectures. We instantiate our attack against an FC-NN for the MNIST dataset, and against a CNN for CIFAR10 and ImageNet. For the IMDB dataset, we use a model whose input is 250-token sentences, and consists of an *embedding layer*, which maps each token in a 10,000-word vocabulary to a 250-dimensional floating-point vectors, and inputs these to a fully-connected layer. The specifics of our model architectures for image and text data are described in Table 9 and Table 10 in Appendix C, respectively. We implemented our trap weights, and the experiments in TensorFlow [3] version 2.4. The code will be open sourced after the peer review process.

Attack Instantiation. Since the central party has access to the gradients of *all* model layers uploaded by users, it is able to choose which layer to instantiate the attack on.

For the FC-NN, we adversarially initialize the first layer with our trap weights and extract training data points from its gradients. In the CNNs, we first initialize the convolutional layers to transmit the input data to the first fully-connected layer of the architecture. Then, we adversarially initialize this layer's weights with our trap weights for extraction.

For the text classifier, we initialize the weights of the embedding layer with a random uniform distribution (min=0., max=1.) to create the inputs for the fully-connected layer. We then adversarially initialize this fully-connected-layer's weights with our trap weights to perform extraction of the embeddings there.

7.1. Extraction Success Metrics

We introduce three novel metrics to measure the success of individual data point extraction.

Active Neurons. By measuring the number of *active neurons* (A) we can determine for how many neurons their respective weighted inputs are positive. This is important because data extraction for both overlaying and individual data points is only possible with activated neurons. If a neuron is not activated by any data point, no information can be transmitted over this neuron and, hence, the gradients will all be zero.

Extraction-Precision. Our second metric, which we call *extraction-precision*, captures the percentage of non-zero gradient rows at the given layer's weight matrix from which we can extract any input data point *individually*. This metric enables us to quantify how well the adversarial weight initialization manages to generate weights that cause activation for exactly one single data point. *Extraction-Precision* can be calculated as follows:

$$\mathrm{P} = \frac{G_1}{A}, \tag{10}$$

with A denoting the *active neurons*, and G_1 denoting the number of gradient rows from which we can extract a data point individually and with an ℓ_2-distance of zero to any of the input data points.

However, the *extraction-precision* metric alone would not be expressive enough since a high *extraction-precision* could be achieved despite the exact same individual training input being reconstructed from all gradient rows. Therefore, we defined another metric that we call *extraction-recall*.

Extraction-Recall. The *extraction-recall* measures the percentage of input data points that can be perfectly extracted from any gradient row. We define it by

$$\mathrm{R} = \frac{B_0}{B}, \tag{11}$$

where B is the number of data points in the given mini-batch and B_0 is the number of these data points that we can extract with an ℓ_2-error of zero from the rescaled gradients.

Interpretation of Success Metrics. Note that our attack seeks to find an adversarial initialization that balances setting enough neurons' outputs to zero (such that a gradient is more likely to isolate individual points from large mini-batches) with, at the same time, having enough neuron

Weights Initializer	MNIST P	MNIST R	CIFAR10 P	CIFAR10 R	ImageNet P	ImageNet R
Xavier Normal	.004	.037	.048	.203	.046	.213
Xavier Uniform	.005	.048	.053	.229	.040	.201
Gaussian (σ=0.01)	.005	.048	.051	.226	.041	.203
Gaussian (σ=0.1)	.005	.049	.053	.238	.043	.209
Gaussian (σ=0.5)	.006	.050	.058	.255	.044	**.218**
Gaussian (σ=1)	.006	.059	.058	.256	.045	**.218**
Gaussian (σ=2)	.007	**.061**	.058	**.259**	.047	.217

TABLE 1: **Extractability with Random Initializations.** Impact of random initialization functions on the *extraction-precision* (P) and *extraction-recall* (R) of individual training data points from the model gradients. The displayed numbers refer to a mini-batch of 100 data points and 1000 neurons for extraction in the first model layer (FC-NN architecture from Table 10). Results are averaged over 10 runs with different random initializations.

B	Passive Attack A	Passive Attack P	Passive Attack R	Our Active Attack A	Our Active Attack P	Our Active Attack R
20	.842	.072	**.900**	.519	.610	**1.000**
50	.885	.050	**.552**	.776	.376	**.962**
100	.909	.036	**.254**	.910	.192	**.654**
200	.927	.030	**.128**	.978	.070	**.255**

TABLE 2: **Data Extraction on IMDb Dataset.** The extraction success depends on the size **B** of the mini-batches for passive attack and active attack with adversarial initialization. The results depict the percentage of *active neurons* (A), *extraction-precision* (P), and *extraction-recall* (R). All numbers are averaged over 10 runs with different random and adversarial initialization of the model from Table 10, respectively.

outputs' that are non-zero (otherwise, in the limit, no points would be extracted). Thus, *active neurons* provide additional context for the *extraction-precision*: with few *active neurons*, even a high *extraction-precision* might not be able to extract many individual training data points, simply because there are very few gradients to perform data extraction from. However, with many *active neurons*, the *extraction-recall* might become small, due to each neuron being most likely activated by several input data points, preventing individual extraction.

7.2. Evaluating the Passive Attack

Recall from Section 5 that extraction of training data from gradients is possible even when model weights are initialized randomly. We evaluate this passive attack to obtain a baseline for our adversarial weight initialization strategies. To evaluate the passive attack, we measure the extraction success of individual training data points from the gradients of randomly initialized models.

Table 1 reports the *extraction-precision* and *extraction-recall* of training data point extraction from the gradients of randomly initialized models. These gradients are computed over a mini-batch of 100 data points for 1000 neurons (*i.e.* 1000 weight rows' gradients for extraction) in the first fully-connected layer. We later study the impact of these two parameters on the success of reconstruction attacks. Even if this attack is passive, and the central party has not modified any of the weights adversarially, training data extraction is often successful: for the MNIST dataset, around 6% of individual training data points can

Epoch	MNIST Loss $\mathcal{L}$	MNIST A	MNIST P	MNIST R	CIFAR10 Loss $\mathcal{L}$	CIFAR10 A	CIFAR10 P	CIFAR10 R
0	.526	.998	.005	.050	1.857	.907	.053	.232
5	.067	.997	.044	.137	1.352	.900	.044	.195
10	.021	.997	.116	.154	1.088	.913	.041	.196
15	.006	.997	.131	.165	.768	.923	.043	.206
20	.002	.997	.136	.167	.472	.931	.050	.232
25	.001	.997	.140	**.169**	.282	.935	.058	.241
30	.001	.997	.142	.168	.200	.936	.062	**.267**

TABLE 3: **Data Extractability from Converging Models.** Results depict the success of passive data extraction based on the training stage of the corresponding models. We show the percentage of *active neurons* (A), *extraction-precision* (P), and *extraction-recall* (R) for extraction with a mini-batch size of 100 data points from the first layer of the fully-connected network from Table 9. All numbers are averaged over 10 runs with different *random* initializations.

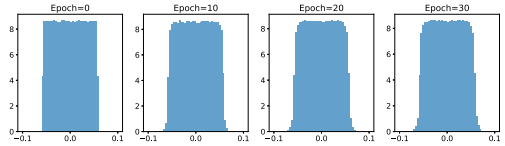


Figure 3: **Evolution of Model Weights over Training.** Distribution of the first layer's weights of the FC-NN from Table 9 over training on the MNIST dataset. Weights at epoch zero were initialized with a random uniform distribution.

be directly extracted from the model gradients, whereas for CIFAR10 and ImageNet, roughly 26% and 22% of the training data points can be perfectly extracted. The passive attack for extracting embeddings from the IMDB dataset yields roughly 25% *extraction-recall* for 1000 neurons and mini-batches of 100 data points, see Table 2.

These results also suggest that setting higher spread, in form of standard deviations to random weight distributions alone can already significantly increase the *extraction-recall* of individual data points from the model gradients, see Table 1. This is, most likely, due to the larger span within the weight values.

Additionally, we also set out to investigate how as training progresses, and the model's weights converge, the extraction' success evolves. We initialized the FC-NN from Table 9 with a Xavier Uniform distribution and trained the model on MNIST and CIFAR10 for 30 epochs. Table 3 depicts the results. We observe that the *extraction-recall* increases slightly over the training epochs. Analyzing the distribution of the model weights in Figure 3 shows that over training, the uniformly initialized weight values resemble more a normal distribution and obtain a wider spread, which might be the reason for the increased extraction success.

7.3. Evaluating Active Manipulations

We now turn to our active attack, which implements our trap weights to amplify the vulnerability exploited by passive attacks. This amplification is controlled by the scaling factor s in the trap weights. We first evaluate the impact of this scaling factor on the reconstruction quality

s	MNIST A	MNIST P	MNIST R	CIFAR10 A	CIFAR10 P	CIFAR10 R	ImageNet A	ImageNet P	ImageNet R
.400	.022	.803	.114	0.	0.	0.	.0.	0.	0.
.500	.149	.636	.354	0.	0.	0.	0.	0.	0.
.600	.462	.408	.526	0.	0.	0.	0.	0.	0.
.700	.796	.203	**.540**	0.	0.	0.	0.	0.	0.
.800	.959	.062	.334	0.	0.	0.	0.	0.	0.
.900	.996	.010	.089	.034	.946	.077	0.	0.	0.
.950	.999	.003	.029	.729	.412	**.540**	0.	0.	0.
.960	.999	.003	.027	.925	.175	.522	0.	0.	0.
.970	1.	.002	.020	.993	.025	.198	.002	.900	.013
.980	1.	.002	.021	1.	.001	.008	.043	.986	.049
.990	1.	.002	.020	1.	0.	0.	.655	.514	**.457**
.995	1.	.002	.018	1.	0.	0.	.999	.007	.055
.999	1.	.002	.017	1.	0.	0.	1.	0.	0.

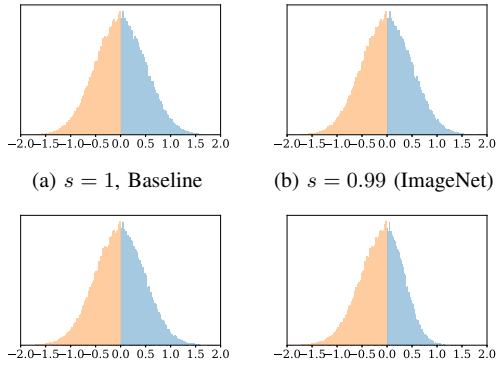
TABLE 4: **Impact of Hyperparameter s.** Success of our adversarial weight initialization dependent on the hyperparameter s, which downscales the positive weights. The results depict the percentage of *active neurons* (A), *extraction-precision* (P), and *extraction-recall* (R) with a mini-batch size of 100 data points from the first fully-connected layer of the respective architectures from Table 9. All numbers are averaged over 10 runs with different adversarial initializations.

(B, N)	MNIST A	MNIST P	MNIST R	CIFAR10 A	CIFAR10 P	CIFAR10 R	ImageNet A	ImageNet P	ImageNet R
(200, 20)	.522	.436	**.720**	.454	.670	**.695**	.090	.948	**.355**
(200,50)	.690	.302	.428	.662	.494	.452	.381	.763	.304
(200,100)	.782	.196	.218	.846	.280	.269	.653	.500	.240
(200,200)	.859	.121	.086	.954	.124	.096	.886	.233	.113
(500,20)	.535	.451	**.915**	.452	.689	**.870**	.096	.939	**.490**
(500,50)	.697	.301	.624	.653	.505	.614	.387	.767	.426
(500,100)	.792	.205	.397	.845	.290	.422	.646	.508	.358
(500,200)	.871	.129	.185	.950	.119	.177	.892	.240	.199
(1000,20)	.539	.444	**.950**	.441	.703	**.915**	.102	.942	**.595**
(1000,50)	.705	.300	.760	.648	.504	.724	.388	.770	.516
(1000,100)	.796	.203	.540	.844	.297	.556	.655	.514	.457
(1000,200)	.871	.124	.293	.951	.120	.256	.892	.238	.288
(3000,20)	.541	.442	**1.**	.441	.696	**.945**	.101	.934	**.640**
(3000,50)	.704	.299	.888	.646	.503	.812	.386	.764	.586
(3000,100)	.797	.203	.746	.840	.286	.711	.649	.518	.579
(3000,200)	.873	.129	.504	.951	.122	.414	.889	.243	.404

TABLE 5: **Effect of Mini-Batch Size and Number of Neurons on Data Extraction.** Success of our adversarial weight initialization is dependent on the mini-batch size **B** and the number of neurons **N** that corresponds to the number of weights rows. The results depict the percentage of *active neurons* (A), *extraction-precision* (P), and *extraction-recall* (R). All numbers are averaged over 10 runs with different adversarial initializations.

of individual training data points over a mini-batch of 100 data points and 1000 neurons.

Table 4 depicts the results, averaged over ten different random adversarial initializations. We can see that the best scaling factor for MNIST, when it comes to the *extraction-recall*, is $s = 0.7$. With this scaling factor, we are able to extract on average 54.0% of the individual training data points which were involved in the users' gradient computations. This is an improvement by around factor nine to the passive attack. For CIFAR10 and Imagenet, the best scaling factors concerning *extraction-recall* are $s = 0.95$, and $s = 0.99$, which allow for a perfect reconstruction of 54.0%, and 45.7% of the individual training data points, respectively, for 1000 neurons and a mini-batch size of 100 data points. Thereby, the active attack is more than twice as successful as the passive attack for extracting individual training data points in these datasets. Figure 11, Figure 12, and Figure 13 in the Appendix D show the visual reconstruction results of the best run for the MNIST, CIFAR10, and ImageNet dataset, respectively. For CIFAR10, we additionally present extraction success when all local data stems from the same single class.



(a) $s = 1$, Baseline (b) $s = 0.99$ (ImageNet)

(c) $s = 0.95$ (CIFAR10) (d) $s = 0.7$ (MNIST)

Figure 4: **Influence of s on Trap Weight Distribution.** When $s = 1$, the distribution of weights follows the standard Gaussian normal distribution (here $\sigma = 0.5$). This corresponds to the baseline of random initialization. For ImageNet and IMDB (b) and CIFAR10 (c), the difference in distribution to the random Baseline (a) is negligible.

Similar improvements of performance could be achieved for the IMDB dataset. The best extraction was achieved also with $s = 0.99$, for which, with 1000 neurons and mini-batches of 100 data points, we obtained an *extraction-recall* of 65.4%, which is around 2.5 time as high as the passive attack, see Table 2.

In Figure 4, we show the influence of the scaling factor s, our method's hyperparameter, on the distribution of our trap weights. The case $s = 1$ corresponds to the baseline where positive components in the trap weights are not scaled down. The figure shows that the more s deviates from 1, the larger the difference between a random distribution and our trap weights. For CIFAR10 and ImageNet ($s = 0.95$, and $s = 0.99$), our trap weights's distribution is very close to the the random distribution, making our trap weights more stealthy. The best scaling factor for MNIST, $s = 0.7$, is significant smaller than for ImageNet and CIFAR10 due to the sparsity in the data (the background in MNIST images consists of zero pixels). Our experiments indicate that with decreasing sparsity and increasing data dimensionality, s approaches 1. Especially the last observation makes sense since scaling more positive components with a factor closer to 1 is in effect of the weighted sum equivalent to scaling fewer positive components with a factor much smaller than 1. Thereby, our trap weights increase in stealthiness with increasing complexity of the data to be extracted.

As hypothesized above, from Table 4, we furthermore confirm that the *extraction-recall* of our attack is related to the percentage of *active neurons*: When very few neurons are activated, it is not possible to extract large numbers of individual data points due to the lack of gradients to extract them from. However, when the percentage of *active neurons* is high, the *extraction-recall* also becomes very small, which is due to the fact that each neuron gets activated by several input data points, and thereby, individual extraction is impossible.

Attacker without Auxiliary Data. We experiment with an attacker who does not have access to a small mini-

batch of data from the users' distribution to tune the scaling factor s of our trap weights. In this setup, the only knowledge an attacker holds is about the dimensionality of the users' data which it needs to instantiate an adequate model architecture. We evaluate three attacks in this setup. *1) Exploiting passive data leakage and composing a tuning dataset:* the attacker randomly initializes the model in a first round of the protocol. Our results in Table 1 show that also randomly initialized models' gradients leak significant fractions of the users' data (MNIST 6.1%, CIFAR10 25.9%, and ImageNet 21.7%). By plotting the user's gradients and eyeballing which data points resemble natural images, the attacker can build a tuning set for s. Since we only require a maximum of 100 data points to find the optimal values for s per dataset in Table 4, the attacker only has to inspect the gradients of 17, 4, and 5 users for MNIST, CIFAR10, and ImageNet, respectively in the first round of the protocol. On the selected data, they can tune s and use it in every subsequent iteration. We performed tuning on 100 data points obtained through passive extraction and obtained the same s as through tuning on a random mini-batch of data (0.7, 0.95, and 0.99 for MNIST, CIFAR10, and ImageNet, respectively). *2) Exploiting raw passive data leakage:* Since manually, selecting suitable data points is time-consuming, we propose an alternative approach where the attacker uses all extracted data points with are in a valid range for input pixels ([0,1]) from the passive extraction on non-adversarially initialized model weights in the first round of the protocol. These data points are not necessarily individually extracted user data points as we show in Figure 9 in Appendix D.2. But the attacker can still consider them as a tuning dataset for s and evaluate the *extraction-recall* on this dataset when initializing the shared model with different trap weights to tune s. Our results in Table 13 show that for CIFAR10 and ImageNet, the best s found on these passively reconstructed data points are equal to the best s obtained directly by tuning on one mini-batch of the original data. For MNIST, the s on the extracted gradients differs slightly from the original best s (0.75 vs. 0.7). We suspect these changes to result from MNIST data being much sparser (many more zero features) than the extracted gradients in Figure 9a. *3) Using a surrogate dataset of same data dimensions:* Lastly, the attacker can tune s on a surrogate dataset of the same dimension (but potentially different distribution) than the users' data. We compare *extraction-recall* of an adversarial weight initialization with s found on a surrogate dataset and the optimal s^* found on the actual dataset for Fashion MNIST, SVHN, CIFAR100, and Open Images [30] in Table 6. Our results highlight that extraction with the surrogate s obtained through tuning on MNIST, CIFAR10, and ImageNet, already yields a significantly higher success than passive extraction on non-manipulated weights. Furthermore, the closer the surrogate dataset's distribution is to the users' dataset, the closer s and s^*. Especially for CIFAR10 and CIFAR100, and ImageNet and Open Images, we find that $s = s^*$ which leads to highest extraction success.

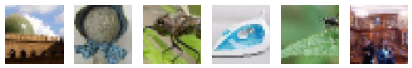
Impact of Data Labels. Additionally, we investigated whether this high reconstruction success could also be achieved =in a non-IID setting when users hold local mini-batches of data that belongs to one single class, different from the other users. This is a particularly challenging setting for prior work on optimization-based attacks that end up reconstructing average points rather than individual points exactly. Instead, Figure 12b and Table 12 in Appendix D.2 show on CIFAR10, how our method remains able to perfectly extract individual data points from the gradients even when all points stem from the same class.

Dataset	Passive R	s	R with s	s^*	R with s^*
Fashion MNIST	0.09	0.7	0.22	0.77	**0.31**
SVHN	0.22	0.95	0.26	0.97	**0.40**
CIFAR100	0.25	0.95	**0.42**	0.95	**0.42**
Open Images	0.21	0.99	**0.44**	0.99	**0.44**

TABLE 6: **Surrogate Data for Tuning s.** We report *extraction-recall* for passive extraction and extraction under adversarial weight initializations. For the latter, we compare the extraction under an s found on a surrogate dataset, and the optimal s^* found through tuning on 100 data points from the given datasets. As surrogate datasets, we use MNIST for Fashion MNIST, CIFAR10 for SVHN and CIFAR100, and ImageNet for Open Images. Results are averaged over 5 runs.

(a) Original Data. Dimension=(224,224,3).

(b) VGG7 Extracted. Dimension=(56,56,3).



(c) ResNet Extracted. Dimension=(28,28,3).

Figure 5: **Extraction from Standard Architectures.** We extract individual user data points from the first fully-connected layer after the convolutional layers. The compression of extracted data in comparison to the original data results from the pooling layers in the architectures.

Impact of Mini-Batch Sizes. We also set out to investigate the impact of the mini-batch size B and the number of weight rows that we can use for extraction. Table 5 depicts the resulting metrics. The metrics show that the smaller the mini-batch sizes are, and the more weight rows there are for extraction, the more individual training data points can be individually reconstructed. For 3000 weight rows, even up to 50% of the individual training data points for mini-batch sizes as large as 200 in the MNIST dataset can be perfectly extracted. Small mini-batches of 20 training data points are entirely extractable without any loss in this setting. Also for the IMDB dataset, smaller batch-sizes for the same number of neurons yield much higher *extraction-recall*, and embeddings of data from small mini-batches of 20 training data points are perfectly extractable, see Table 2. This suggests that in practice, the success of the extraction attack can be significantly increased by the central party demanding smaller mini-batch sizes from the users or initializing larger models.

Impact of Lossy Layers. For perfect extraction of data points in CNN architectures, our attack requires the input to the first fully-connected layer to have at least as many parameters as the original input data point. CNN architec-

local epochs	local B	MNIST ↑Δ_{acc}	MNIST P	MNIST R	CIFAR10 ↑Δ_{acc}	CIFAR10 P	CIFAR10 R
1	10	0.296	0.238	0.704	0.264	0.279	0.584
	20	0.220	0.175	0.496	0.238	0.277	0.466
	40	0.289	0.109	0.280	0.188	0.127	0.213
2	10	0.384	0.252	0.704	0.316	0.311	0.560
	20	0.772	0.184	0.478	0.296	0.295	0.476
	40	0.671	0.111	0.264	0.282	0.126	0.247
3	10	0.604	0.241	0.712	0.372	0.329	0.596
	20	0.790	0.189	0.544	0.420	0.277	0.496
	40	0.823	0.111	0.283	0.324	0.138	0.257
4	10	0.644	0.251	0.692	0.396	0.332	0.632
	20	0.848	0.178	0.494	0.440	0.288	0.478
	40	0.825	0.113	0.273	0.415	0.138	0.256
5	10	0.604	0.270	0.732	0.412	0.354	0.620
	20	0.870	0.178	0.494	0.492	0.289	0.502
	40	0.873	0.119	0.283	0.461	0.139	0.277

TABLE 7: **Local Accuracy Improvement and Extraction Success with FedAvg.** We present results for FedAvg where each user holds five mini-batches of size B and computes 1,2,3,4, or 5 local epochs of training with the FC-NN from Table 9. The Δ_{acc} indicates the accuracy improvement on the user's local data w.r.t. the received shared model (initially around 10% accuracy). The *extraction-precision* (P) and *extraction-recall* (R) for every B stay at the same high level even after multiple local epochs of training.

tures can contain pooling layers to reduce input size. In Appendix D.3, we evaluate the impact of pooling on the fidelity of the extracted data. Our evaluation shows that pooling results in some form of compression of the user's input data, see for example Figure 16a. In Appendix B.2, we show how the central party can implement an alternative to pooling for size-reduction in CNNs based on convolutional layers which still allows for prefect extracability, as long as there are enough model parameters. We also evaluate the effect of dropout on the fidelity of the extracted data. Figure 15 and 17 visualize the effect of pure dropout, while Figure 16 and 18 visualize the joint effect of dropout and pooling. To increase fidelity of extraction under lossy layers, an attacker can apply post-processing, such as de-compression.

VGG and ResNet. In addition to our custom FC-NN and CNN architecture from Table 9, we experimented with a VGG7 and a ResNet20 [44] architecture. For VGG7, we initialize all convolutional layers as illustrated in Figure 6 in Appendix B.1, and the fully-connected layer directly after the convolutional layers with our trap weights. For the ResNet20, we only initialize the first convolutional layer according to Figure 6. Thanks to the skip connections, the remaining convolutional layers can remain unchanged, apart from the convolutional filters whose output is added to the output of the skip connections. These need to be set to zero, such that the input data can be propagated unaltered over the skip-connections to the fully-connected layer that we initialize with our trap weights. We set the last pooling layer before this fully-connected layer in ResNet20 to implement average pooling. Our extraction results for ImageNet are depicted in Figure 5. The compression of extracted data in comparison to the original data results from the pooling layers in both architectures that reduce input dimensions.

Impact of Local Mini-Batch-Averaging. Additionally, we looked into the effect of averaging over the gradients of multiple mini-batches, *e.g.* the average of gradients received from multiple users. The results in Table 11 in Appendix D.2 show that through averaging, the attack success is significantly reduced. Already when averaging over 20 mini-batches of size $B = 100$ in the MNIST dataset, the average *extraction-recall* drops from 54.0% to 2% because multiple data points overlay in the gradients. This highlights that the central party needs to perform the extraction before the averaging operation. The following section shows that this simple change to the protocol is easily implemented by an actively dishonest central party, even for standard FL libraries.

FedAvg. To validate our theoretical insights from Section 5.3 which highlights that even under FedAvg, perfect extraction of individual data points is possible, we run FedAvg experiments in which users hold five mini-batches of $\{10,$ 20, or $40\}$ different data points (yielding a total of 50, 100, and 200 local data points per user), and perform $\{1,$ 2, 3, 4, or $5\}$ local training epochs. In Table 7, we depict results from the FC-NN architecture from Table 9 on MNIST and CIFAR10. Our results highlight that the while accuracy on the users' data significantly increases through the local training, the extraction success stays constant over multiple local epochs. We even observe a slight increase in *extraction-recall*. For example, we can extract 58.4% of data points from users who hold five mini-batches with ten data points each after one epoch, while this number increases to 62% after five local epochs of training. This finding is congruent with our finding in Table 3, where we show that extraction success increases with convergence. For CNNs, the extraction success degrades over multiple local epochs of training. This is due to the convolutional filters that, after a local update, do not have the zero-elements anymore which prevent features in the forward-pass from overlapping. For our CNN architecture from Table 9, we report ℓ_2-distances between original and extracted data of $[3.81\text{e}{-5}, 5.06\text{e}{-5}, 0.07, 0.27, 0.92]$ and $[1.4\text{e}{-3}, 138.94, 199.69, 264.45, 269.32]$ for CIFAR10 and ImageNet after 1,2,3,4, and 5 epochs, respectively.

TensorFlow Federated. We experimented with TensorFlow Federated [2]—a standard open source library for FL deployments. In Appendix D.4, we show that a dishonest central party only requires minimal code changes to implement our trap weights.

7.4. Comparison To Previous Work

To compare the success of our attack, we compare to the three approaches conceptually closest to ours. These are [15] which is the first to describe individual extractability of single-data point gradients, [12] which relies on direct extraction from a fully-connected layer in the model architecture, and [37] which exploits manipulations model parameters to extract data from user-gradients.

Comparison to [15]. For the sake of correctness, we build on their code base and adopt it to also run with neural architectures we used in our other experiments. We use the parameters that [15] found to perform best. Here, we are mainly interested in the quality of the other attack's reconstruction in comparison to our method, and in the number of passes over the model, *i.e.* the computing time required to obtain the reconstructions. We perform evaluation on the MNIST and CIFAR10 datasets.

Figure 19 in Appendix D shows an example of the gradient-inversion fidelity obtained with [15] for MNIST with $B = 1$. Within 10^4 iterations, the pixel-wise ℓ_2 errors observed go as low as 10^{-4} for FC-NNs with architecture as depicted in Table 9 in the Appendix, and 10^{-3} for LeNet-5. Similar results for CIFAR10 are available in Appendix 20 in Section D. In contrast to these results, our attack allows us to extract data points perfectly, *i.e.* with ℓ_2 error of zero, and without *any* back-propagation iterations. These results make clear that gradient inversion in practice suffers from local minima and requires a very large number of iterations to converge to a comparable reconstruction to our method. On a practical note, reconstruction of a single CIFAR10 image with 32 restarts from a seven-layer FC-NN takes on average 1 hour and 3 minutes on a high-end GPU, in comparison to milliseconds needed for extraction with the help of our trap weights. Most importantly, it is clear that reconstruction is not a lossless process and full data recovery is almost never possible, even in the simple cases where gradients of only a single data point are considered. To better understand limitations of prior literature we refer the reader to [48].

Comparison to [12]. The success of [12]' data extraction depends on the size of their imprinting module. Using their code-base and extending it with our success metrics, we evaluated what size of imprinting module they require to obtain the same extraction recall as we do, *i.e.*, to extract the same number of data points from the model gradients perfectly. We compared our methods for all three vision dataset, using a batch-size of $B = 100$. For ImageNet and CIFAR10, following their baseline, we instantiated their model with a ResNet18, for MNIST, we used LeNet5. We always inserted their imprinting module before the first layer to allow for perfect extractability with their method. Our results show that to obtain the same extraction recall as we do (46%, 54%, and 54% for ImageNet, CIFAR10, and MNIST, respectively), their imprinting module needs to be of size roughly 150, 200, and 400, respectively. The fact that they require the largest imprint module for MNIST is due to the similarity in the data points (sparsity in the background with all zero pixels) which makes their binning less effective.

Comparison to [37]. Note that [37]'s main goal is not to extract large amounts individual user *data points* but user *updates*, by circumventing the secure aggregation [6] used to protect the FL protocol. The updates (gradients) that [37] recover do usually not correspond to full and perfectly individual data points. Instead, for FC-NNs, their extracted gradients will resemble our passive extraction results from Figure 8 where most of the gradients are a blurry overlay of all underlying data points. For CNNs, their results will not be able to extract any individual data point since non-maliciously initialized convolution filters overlay input features. Thereby, their attack mainly violates confidentiality of the users' model updates in a setup where users believe to obtain protection though an aggregate with other users. Still, the resulting gradients can then be used as a departure point for additional privacy-attacks, such as reconstruction. In contrast, our work directly violates the users' privacy by manipulating the shared model weights to make individual data points directly extractable from the model updates sent from users to the central party. To assess individual extractability in their setup, we use their gradient suppression and model inconsistency attack to make all but one user in a round of the FL protocol return zero gradients. The one target-user receives a randomly initialized FC-NN (Gaussian with $\sigma = 0.5$) with architecture from Table 9. Note that the data extraction from gradients in this setup corresponds to our passive extraction. For MNIST, with $B = 100$, extraction in their setup yields 5% of perfectly extractable data points, while our trap weights yield 54%. In the same setup for CIFAR10, their method yields 26%, in contrast to our trap weights which yield again 54%.

8. Defending Privacy in Federated Learning

This section discusses potential mitigations against our trap weights attack. We start with explaining DP which provides formal privacy guarantees and then move on to defenses specifically tailored to our trap weights attack.

8.1. Formal Guarantees: Differential Privacy

To bound the leakage of private information from model gradients, a gold standard for reasoning about privacy guarantees is the framework of differential privacy (DP) [11].

There exist three main ways of integrating DP in the FL protocol, namely Centralized Differential Privacy (CDP), Local Differential Privacy (LDP), and Distributed Differential Privacy (DDP).

In CDP, users clip their gradients locally according to a clip norm c and the central party performs the addition of noise with a scale dependent on the noise multiplier σ [40]. CDP cannot provide DP guarantees with a malicious central party because this central party can simply extract user data before adding noise or not add noise at all.[2]

LDP reduces the trust required in the central party since every user locally adds noise to their gradients according to their privacy requirements [47]. Independent of other users, the noise is drawn from $\mathcal{N}(0, \sigma^2 c^2)$. However, previous work has shown that this setup leads to poor privacy-utility trade-offs, such that LDP is not popular in practical applications [51].

DDP is supposed to combine the advantages of CDP and LDP. In DDP, before aggregation, each user locally adds some (small) amount of noise to their gradients [46]. The noise distribution depends on the number M of other selected users. It is specified by $\mathcal{N}\left(0, \frac{\sigma^2}{\mathsf{M}-1} c^2\right)$ [46]. While the individual noise levels do not offer sufficient protection, the aggregates provide rigorous privacy guarantees. There exist different forms of performing the aggregation. One popular approach is to use secure aggregation (SA) [6], which adds significant computational overhead and requires tailored DP mechanisms that operate on integer values, *e.g.* [4], [26]. Finally, prior work has shown that in FL, SA can be eluded [37]. This motivates defenses dedicated to protecting specifically against our trap weights attack.

2. For a private aggregation of sensitive statistics (instead of high-dimensional ML model gradients), there exist solutions of CDP without a trusted aggregator [41], [42].

8.2. Specific Defenses against Trap Weights

The following defenses can be applied to mitigate the success of our trap weights attack. Since these defenses do not provide rigorous theoretical privacy guarantees but rather empirical protection, we recommend combining them with DP.

Hardware-Based Protection. Using protocols that rely on Trusted Execution Environments (TEE), *e.g.* [35] prevents the central party from performing active manipulations. However, TEEs are prone to side-channel attacks [9], [24]. Hence, there is a remaining risk for the privacy of users' data.

Local Averaging and Large Mini-Batches. Our results in Table 5 and Table 11 highlight that calculating gradients over large mini-batches and local averaging reduces the fraction of data points that can be perfectly reconstructed. We, therefore, argue that users should perform gradient calculation on large mini-batches of local data points and average gradients over multiple mini-batches before sending them to the central party. This is, however, only possible if users have actual control on the local execution of the FL protocol and the execution is not inaccessibly encapsulated inside an application.

Choice of the Activation Function. Our trap weights are designed to exploit properties of the ReLU activation function, namely the fact that it yields zero-gradients for inputs at some neurons. Yielding zero-gradients is not unique to the ReLU activation. Other popular activation functions such as sigmoid and tanh have flat areas that also yield zero-gradients. Hence, by adapting the initialization of our trap weights to these functions' properties, we could also achieve perfect extractability of individual data points with these functions. However, using activation functions, such as leaky ReLU, which propagate information on every input through each neuron, can prevent individual extractability of training data points.

Lossy Layers. Our evaluation in Section 7 highlights that the application of layers that compress the input data or cause information loss, such as pooling or dropout reduce fidelity of the extracted data. Therefore, relying on architectures that have aggressive compression and/or dropout reduces the leakage of individual user data to the central party.

Given that in FL, the central party is in charge of instantiating the shared model (with its hyperparameters, such as the activation function), users can *only* rely on additional protection through the model itself *if this central party is trusted*. An untrusted central party, in contrast, has incentives to choose model architectures and hyperparameters that facilitate data extraction.

9. Discussion and Future Directions

In this section, we first discuss the detectability of our adversarial initialization and integration of our attack in the training process of the shared model. We then analyze the potential and capabilities of adversarial weight initialization for future privacy attacks. Finally, we argue that dedicated privacy-protection should be implemented as a default option into FL protocols to prevent accidental or malicious privacy leakage.

9.1. Detectability of Trap Weights

To detect the presence of our trap weights, the users can apply one of the following two strategies: (1) analyzing the weights of the shared model in *one* or *multiple* iterations over the FL protocol, or (2) analyzing the behavior of the shared model on their data.

Analyzing Model Weights. Assuming the user has access to the model only in **one iteration** of the protocol[3], they can run a detection method that aims at deliberately looking for characteristic elements of our trap weights, such as a normal distribution with high standard deviation, or the presence of higher absolute values for negative components than positive components in the first fully-connected layer's weight matrix. Figure 3 shows that even when initialized with a uniform distribution and relatively low deviation, model weights after several epochs of training resemble more a normal distribution and exhibit a larger standard deviation, making the former characteristic of our trap weights an unreliable attack detector. When it comes to the magnitude of positive and negative components, Figure 4 shows that for s close to one, the distribution of the model weights still resembles a standard normal. However, the more s deviates from one, the more the distribution of weights deviates from a standard normal distribution. Yet, without knowledge of the prior training procedure and the other users' data, we argue that a target user can still not determine with certainty whether the received model weights are the result of the prior training or of a manipulation [43].

Having access to the shared model over **multiple iterations** of the FL protocol *additionally* enables users to compare the received shared model's parameters to the parameters from previous FL iterations. Therefore, the success of detection boils down to the following question: *Can a local user tell that a given set of model parameters came from legitimate updates of other local users?* We argue that this is not possible, even for non-FL setups where the entire training procedure is transparent. The stochastic nature of training algorithms, combined with the non-determinism of modern hardware, makes it difficult to reproduce training runs [25]. Because of this reproducibility error, an attacker can assemble a mini-batch of natural data points that produce *any* desired gradient update [43]. In other words, given two different sets of model weights, the user cannot tell if the gradient descent step between these weights was a result of a legitimate optimization step. This is exacerbated in FL because the data of any given user is invisible to other users, further complicating the verification of gradient descent integrity.

Analyzing Model Performance. In addition to analyzing the received model weights for detection of our attack, a user can also evaluate the *functionality* of the shared model. We observe that, for the vast majority of classification tasks, the model's loss across training data points significantly reduces after just a few iterations. Thus, after the initial training iterations, FL users would expect to encounter low loss values for their own examples. However, research has shown that, in particular for users whose data

3. Given that in practical deployments of FL, $\mathsf{N} >> \mathsf{M}$, an individual user will be sampled for participation very rarely.

stems from the tails of the data distribution, FL does not necessarily lead to an improvement of model accuracy on their data [54]. Therefore, detection mechanisms that rely on analyzing the convergence of the model's accuracy over multiple FL iterations are also no reliable detectors of our manipulation.

9.2. Training Success and Model Performance

Increasing the utility of the shared model over the course of training in the FL protocol is important because the central party is expected to provide a well-performing model after several training iterations. Therefore, the central party in our attack leverages two main points over the course of the protocol. (1) Instead of aiming at reconstructing the user data over all communications, it sends the adversarially (re-)initialized model out only at a few communication rounds. In all other rounds, it sends out the actual shared model for training without adversarially re-initializing it. (2) Instead of targeting all users, the central party only targets a subset of users, and send out an adversarially initialized model to them, and the continuously trained shared model to all other users. The central party can even combine both strategies by sending out an adversarially initialized model only to a subset of users in a few iterations.

9.3. The Power of Weight Initialization

In general, even outside of the FL context, our attack shows that controlling and manipulating the weights of neural networks opens a new attack surface against ML. We argue that weight manipulation could be used to design further privacy attacks outside of the FL context. Our adversarial initialization of convolutional and fully-connected model layers is able to transmit input data points to any subsequent layer in the model, practically modulating data perfectly over them. Additionally by setting our trap weights, we can increase the leakage of individual training data points from model gradients. Therefore, the trap weights basically create a simple if-else logic based on $>$ and $<$ relations between weighted inputs to model neurons. Future work could investigate whether the weights could also be set in order to implement more complex logical structures and if-else cases depending on the input. Based on these, it might be possible to craft hybrid attacks that first initialize the model weights and then use that to later extract information, for example, on membership of individual data points, or these data points' sensitive attributes. Note that adversarially setting weights also does not need to be limited to initializing the model weights. Instead, given an already initialized (and trained model), it might be possible to craft additional training data that leads to the weights taking the adversarial values that an attacker wants.

9.4. Using Dedicated Privacy Protection in FL

FL was originally designed as an alternative to centralized ML in which no large datasets would have to be moved from users to a central party in order to train an ML model on the joint data. The approach does not only reduce communication costs but also spares the central party from having to build up the infrastructure by outsourcing training and data storage costs to the users. Indeed, FL is more communication cost effective since the data itself is not shared directly.

Attacks like our trap weights highlight, however, that the protocol does not guarantee protection for the individual users' private training data. This is not surprising since nothing in the design the FL protocol protects against leakage of private information. Without dedicated privacy-protection, the central party even has an upper hand over how much data a local model will leak, as demonstrated in our work. However, FL is still often marketed as a data-minimizing technology. Our work highlights that such marketing is misleading since in order to deploy FL as a privacy-technology, it is necessary to implement dedicated additional protection methods, such as the ones discussed in Section 8.We argue that, to prevent malicious or accidental leakage in FL, these protection methods should be implemented in FL as a default when deploying the protocol to actual users. That is, vanilla federated learning does not provide privacy advantages for users—unless it is combined with additional defense methods, such as DP learning.

10. Conclusion

In this work, we presented a new privacy attack against FL that is based on an active attacker who holds the ability to maliciously manipulate the shared model and its weights. Our attack allows for perfect reconstruction of a significant portion of the users' private training data. Even for very high-dimensional complex datasets, such as ImageNet, we are able to perfectly extract roughly 50% of the individual data points from mini-batches of sizes as large as 100. The extraction is computationally highly efficient and even allows to perfectly extract individual training data points from data mini-batches containing all data points from one single class.

Our attack underscores the deficiency of the "data never leaves the device" approach to preserving privacy. For FL to have a chance of truly preserving privacy, it must incorporate appropriate mitigations against our attack. Those either have expensive overheads or are tailored for this specific attack (see Section 8).

Acknowledgments

We would like to acknowledge our sponsors, who support our research with financial and in-kind contributions: Amazon, Apple, CIFAR through the Canada CIFAR AI Chair, DARPA through the GARD project, Intel, Meta, NFRF through an Exploration grant, NSERC through the COHESA Strategic Alliance, the Ontario Early Researcher Award, and the Sloan Foundation. Resources used in preparing this research were provided, in part, by the Province of Ontario, the Government of Canada through CIFAR, and companies sponsoring the Vector Institute.

References

[1] Implementation of FedAvg in TensorFlow Federated. https://github.com/tensorflow/federated/tree/v0.19.0/tensorflow_federated/python/examples/simple_fedavg. Accessed: 10/2021.

[2] TensorFlow Federated. https://www.tensorflow.org/federated. Accessed: 10/2021.

[3] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, and *et al*. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems*. 2016.

[4] Naman Agarwal, Peter Kairouz, and Ziyu Liu. The skellam mechanism for differentially private federated learning. *Advances in Neural Information Processing Systems*, 34, 2021.

[5] Keith Bonawitz, Hubert Eichner, Wolfgang Grieskamp, Dzmitry Huba, Alex Ingerman, Vladimir Ivanov, Chloe Kiddon, Jakub Konečnỳ, Stefano Mazzocchi, Brendan McMahan, et al. Towards federated learning at scale: System design. *Proceedings of Machine Learning and Systems*, 1:374–388, 2019.

[6] Keith Bonawitz, Vladimir Ivanov, Ben Kreuter, Antonio Marcedone, H Brendan McMahan, Sarvar Patel, Daniel Ramage, Aaron Segal, and Karn Seth. Practical secure aggregation for privacy-preserving machine learning. In *proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 1175–1191, 2017.

[7] Gregory Cohen, Saeed Afshar, Jonathan Tapson, and Andre Van Schaik. Emnist: Extending mnist to handwritten letters. *2017 International Joint Conference on Neural Networks (IJCNN)*, 2017.

[8] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009.

[9] Ghada Dessouky, Tommaso Frassetto, and Ahmad-Reza Sadeghi. {HybCache}: Hybrid {Side-Channel-Resilient} caches for trusted execution environments. In *29th USENIX Security Symposium (USENIX Security 20)*, pages 451–468, 2020.

[10] Di Xie, Jiang Xiong, and Shiliang Pu. All you need is beyond a good init: Exploring better solution for training extremely deep convolutional neural networks with orthonormality and modulation. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2017.

[11] Cynthia Dwork. Differential privacy. 2006.

[12] Liam H Fowl, Jonas Geiping, Wojciech Czaja, Micah Goldblum, and Tom Goldstein. Robbing the fed: Directly obtaining private data in federated learning with modified models. In *International Conference on Learning Representations*, 2021.

[13] Stephen Gallant. Perceptron-based learning algorithms. *IEEE Transactions on neural networks*, 1(2):179–191, 1990.

[14] Karan Ganju, Qi Wang, Wei Yang, Carl A. Gunter, and Nikita Borisov. Property inference attacks on fully connected neural networks using permutation invariant representations. In David Lie, Mohammad Mannan, Michael Backes, and XiaoFeng Wang, editors, *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, pages 619–633. ACM, 2018.

[15] Jonas Geiping, Hartmut Bauermeister, Hannah Dröge, and Michael Moeller. *Inverting Gradients – How easy is it to break privacy in federated learning?* 2020. 23 pages, 20 figures. The first three authors contributed equally.

[16] Nastaran Gholizadeh and Petr Musilek. Distributed learning applications in power systems: A review of methods, gaps, and challenges. 14(12):3654, 2021. PII: en14123654.

[17] Raja Giryes, Guillermo Sapiro, and Alex M. Bronstein. Deep neural networks with random gaussian weights: A universal classification strategy? 64(13):3444–3457, 2016.

[18] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. Deep sparse rectifier neural networks. pages 315–323, 2011.

[19] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Proceedings of the Advances in Neural Information Processing Systems (NeurIPS)*, Montreal, Canada, December 2014.

[20] Kathrin Grosse, Thomas A. Trost, Marius Mosbach, and Michael Backes. Adversarial initialization - when your network performs the way i want -. 2019.

[21] Saqib Hakak, Suprio Ray, Wazir Zada Khan, and Erik Scheme. A framework for edge-assisted healthcare data analytics using federated learning. In *2020 IEEE International Conference on Big Data (Big Data)*. IEEE, 2020.

[22] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *2015 IEEE International Conference on Computer Vision (ICCV)*. IEEE, 2015.

[23] Briland Hitaj, Giuseppe Ateniese, and Fernando Perez-Cruz. Deep models under the gan: information leakage from collaborative deep learning. 2017.

[24] Patrick Jauernig, Ahmad-Reza Sadeghi, and Emmanuel Stapf. Trusted execution environments: properties, applications, and challenges. *IEEE Security & Privacy*, 18(2):56–60, 2020.

[25] Hengrui Jia, Mohammad Yaghini, Christopher A. Choquette-Choo, Natalie Dullerud, Anvith Thudi, Varun Chandrasekaran, and Nicolas Papernot. Proof-of-learning: Definitions and practice, 2021.

[26] Peter Kairouz, Ziyu Liu, and Thomas Steinke. The distributed discrete gaussian mechanism for federated learning with secure aggregation. In *International Conference on Machine Learning*, pages 5201–5212. PMLR, 2021.

[27] Latif U. Khan, Walid Saad, Zhu Han, Ekram Hossain, and Choong Seon Hong. Federated learning for internet of things: Recent advances, taxonomy, and open challenges. 23(3):1759–1799, 2021.

[28] Alex Krizhevsky. Learning multiple layers of features from tiny images. Technical report, 2009.

[29] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009.

[30] Alina Kuznetsova, Hassan Rom, Neil Alldrin, Jasper Uijlings, Ivan Krasin, Jordi Pont-Tuset, Shahab Kamali, Stefan Popov, Matteo Malloci, Tom Duerig, and Vittorio Ferrari. The open images dataset v4: Unified image classification, object detection, and visual relationship detection at scale. *arXiv:1811.00982*, 2018.

[31] Yann LeCun, Corinna Cortes, and C. J. Burges. *MNIST handwritten digit database*. 2010.

[32] Andrew L. Maas, Raymond E. Daly, Peter T. Pham, Dan Huang, Andrew Y. Ng, and Christopher Potts. Learning word vectors for sentiment analysis. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pages 142–150, Portland, Oregon, USA, June 2011. Association for Computational Linguistics.

[33] Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Aguera y Arcas. Communication-efficient learning of deep networks from decentralized data. pages 1273–1282, 2017.

[34] Luca Melis, Congzheng Song, Emiliano de Cristofaro, and Vitaly Shmatikov. Exploiting unintended feature leakage in collaborative learning. In *2019 IEEE Symposium on Security and Privacy (SP)*, pages 691–706. IEEE, 19/05/2019 - 23/05/2019.

[35] Fan Mo, Hamed Haddadi, Kleomenis Katevas, Eduard Marin, Diego Perino, and Nicolas Kourtellis. Ppfl: privacy-preserving federated learning with trusted execution environments. In *Proceedings of the 19th Annual International Conference on Mobile Systems, Applications, and Services*, pages 94–108, 2021.

[36] Dinh C. Nguyen, Ming Ding, Pubudu N. Pathirana, Aruna Seneviratne, Jun Li, and H. Vincent Poor. *Federated Learning for Internet of Things: A Comprehensive Survey*, volume 23. 2021.

[37] Dario Pasquini, Danilo Francati, and Giuseppe Ateniese. Eluding secure aggregation in federated learning via model inconsistency. *ArXiv*, abs/2111.07380, 2021.

[38] Le Trieu Phong, Yoshinori Aono, Takuya Hayashi, Lihua Wang, and Shiho Moriai. Privacy-preserving deep learning: Revisited and enhanced. pages 100–110. Springer, Singapore, 2017.

[39] Rudiger Pryss, Manfred Reichert, Jochen Herrmann, Berthold Langguth, and Winfried Schlee. Mobile crowd sensing in clinical and psychological trials – a case study. In *2015 IEEE 28th International Symposium on Computer-Based Medical Systems*. IEEE, 2015.

[40] Swaroop Ramaswamy, Om Thakkar, Rajiv Mathews, Galen Andrew, H Brendan McMahan, and Françoise Beaufays. Training production language models without memorizing user data. *arXiv preprint arXiv:2009.10031*, 2020.

[41] Edo Roth, Daniel Noble, Brett Hemenway Falk, and Andreas Haeberlen. Honeycrisp: large-scale differentially private aggregation without a trusted core. In *Proceedings of the 27th ACM Symposium on Operating Systems Principles*, pages 196–210, 2019.

[42] Edo Roth, Hengchu Zhang, Andreas Haeberlen, and Benjamin C Pierce. Orchard: Differentially private analytics at scale. In *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*, pages 1065–1081, 2020.

[43] Ilia Shumailov, Zakhar Shumaylov, Dmitry Kazhdan, Yiren Zhao, Nicolas Papernot, Murat A. Erdogdu, and Ross Anderson. *Manipulating SGD with Data Ordering Attacks*. 2021.

[44] sjmikler. Github: Resnets in tensorflow 2.0 on cifar-10, 2019. https://github.com/sjmikler/resnets-in-tensorflow2, on commit "547d131382438ef76e315dde06a6870737f1fbad".
[45] Jingwei Sun, Ang Li, Binghui Wang, Huanrui Yang, Hai Li, and Yiran Chen. Provable defense against privacy leakage in federated learning from representation perspective. *arXiv preprint arXiv:2012.06043*, 2020.
[46] Stacey Truex, Nathalie Baracaldo, Ali Anwar, Thomas Steinke, Heiko Ludwig, Rui Zhang, and Yi Zhou. A hybrid approach to privacy-preserving federated learning. In *Proceedings of the 12th ACM workshop on artificial intelligence and security*, pages 1–11, 2019.
[47] Stacey Truex, Ling Liu, Ka-Ho Chow, Mehmet Emre Gursoy, and Wenqi Wei. Ldp-fed: Federated learning with local differential privacy. In *Proceedings of the Third ACM International Workshop on Edge Systems, Analytics and Networking*, pages 61–66, 2020.
[48] Aidmar Wainakh, Ephraim Zimmer, Sandeep Subedi, Jens Keim, Tim Grube, Shankar Karuppayah, Alejandro Sanchez Guinea, and Max Mühlhäuser. Federated learning attacks revisited: A critical discussion of gaps, assumptions, and evaluation setups, 2021.
[49] Yi Wang, Imane Lahmam Bennani, Xiufeng Liu, Mingyang Sun, and Yao Zhou. Electricity consumer characteristics identification: A federated learning approach. 12(4):3637–3647, 2021.
[50] Zhibo Wang, Mengkai Song, Zhifei Zhang, Yang Song, Qian Wang, and Hairong Qi. Beyond inferring class representatives: User-level privacy leakage from federated learning. In *IEEE INFOCOM 2019 - IEEE Conference on Computer Communications*. IEEE, 2019.
[51] Kang Wei, Jun Li, Ming Ding, Chuan Ma, Howard H Yang, Farhad Farokhi, Shi Jin, Tony QS Quek, and H Vincent Poor. Federated learning with differential privacy: Algorithms and performance analysis. *IEEE Transactions on Information Forensics and Security*, 15:3454–3469, 2020.
[52] Yuxin Wen, Jonas A. Geiping, Liam Fowl, Micah Goldblum, and Tom Goldstein. Fishing for user data in large-batch federated learning via gradient magnification. In Kamalika Chaudhuri, Stefanie Jegelka, Le Song, Csaba Szepesvari, Gang Niu, and Sivan Sabato, editors, *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pages 23668–23684. PMLR, 17–23 Jul 2022.
[53] Hongxu Yin, Arun Mallya, Arash Vahdat, Jose M. Alvarez, Jan Kautz, and and Pavlo Molchanov. *See through Gradients: Image Batch Recovery via GradInversion*. 2021.
[54] Tao Yu, Eugene Bagdasaryan, and Vitaly Shmatikov. Salvaging federated learning by local adaptation, 2021.
[55] Bo Zhao, Konda Reddy Mopuri, and Hakan Bilen. *iDLG: Improved Deep Leakage from Gradients*. 2020.
[56] Ligeng Zhu and Song Han. Deep leakage from gradients. In *Federated Learning*, pages 17–31. Springer, Cham, 2020.

A. Extended Related Work on Passive Data Reconstruction Attacks

Table 8 summarizes these data reconstruction attacks (described below) and compares them with our attack.

Method	Attacker U	Attacker S	Rep. C	Rep. U	Rep. ID (stronger →)	Label-Free	B 1	B ≥ 1 (stronger →)	Opt.-/ Train.-Free
DMU-GAN [23]	✓		✓					✓	
mGAN-AI [50]		✓		✓				✓	
DLG [56]		✓			✓	✓	✓		
iDLG [55]		✓			✓	✓	✓		
GradInv [53]		✓			✓	✓		✓	
trap weights [Ours]		✓			✓	✓		✓	✓

TABLE 8: **Comparison of data reconstruction attacks.** KEY– U: User, S: Server, C: Class, ID: Individual Data Points, B: Mini-batch size, Opt.: Optimization, Train.: Training, Rep.: Representative.

Class-wise Representation Reconstruction Attacks. Hitaj *et al*. [23] were the first to propose a GAN-based data reconstruction attack, called DMU-GAN. The attacker must know the dataset's classes, and the reconstructed data points are generic representations of class-wise properties rather than individual user data points or classes. Wang *et al*. [50] suggested mGAN-AI, which extends DMU-GAN's reconstruction attack to per-user class-wise representations, but still does not extract individual data points. Additionally, both methods require access to data from the same distribution as the users' data. [45] observes that class-wise representations are embedded in model updates even without the need to reconstruct them using a GAN, and suggest defenses.

Algorithm 2: Optimization-Based Data Reconstruction [56].

Input: Gradients, $G_i^{[t]}$, received from victim user u_i at iteration t, Shared model $f_{\mathcal{W}}^{[t]}(\cdot)$ at iteration t.
Output: Reconstructed training data, $(\mathbf{x}_i^*,\ y_i^*)$

$(\hat{\mathbf{x}}^{[1]}, \hat{y}^{[1]}) \leftarrow (\mathcal{N}(0,1), \mathcal{N}(0,1))$ ▷ Initialize
for $\hat{t} \in [1, \hat{T}]$ **do**
  $\hat{G}^{[\hat{t}]} = \nabla_{\mathcal{W}} \mathcal{L}(f_{\mathcal{W}}^{[t]}(\hat{\mathbf{x}}^{\hat{t}}), \hat{y}^{\hat{t}})$ ▷ Dummy gradients
  $D^{[\hat{t}]} = \|G_i^{[t]} - \hat{G}^{[\hat{t}]}\|^2$ ▷ Dummy vs user
  $\hat{\mathbf{x}}^{[\hat{t}+1]} \leftarrow \hat{\mathbf{x}}^{[\hat{t}]} - \alpha \nabla_{\hat{\mathbf{x}}^{[\hat{t}]}} D^{[\hat{t}]}$,
  $\hat{y}^{[\hat{t}+1]} \leftarrow \hat{y}^{[\hat{t}]} - \alpha \nabla_{\hat{y}^{[\hat{t}]}} D^{[\hat{t}]}$
end for
$(\mathbf{x}_i^*,\ y_i^*) \leftarrow (\hat{\mathbf{x}}^{[\hat{T}+1]},\ \hat{y}^{[\hat{T}+1]})$

B. Generalization of Data Extraction Attack to Convolutional Neural Networks

So far, both the passive and active attacks we described are tailored to extracting data from the gradients computed to update a fully-connected layer. However, modern neural network architectures often rely on convolutional layers to model image and text data alike. It is difficult to directly apply our attack strategy to these convolutional layers because they rely on the weight sharing principle: to decrease the effective number of parameters that need to be trained, the same weight values are applied to multiple locations of the image to extract patterns regardless of their location in the image. In this section, we thus propose a second instantiation of our adversarial weight initialization strategy that generalizes extraction attacks to convolutional neural networks (CNNs).

Our solution reduces networks with convolutional layers to the setting we previously considered with fully-connected neural networks. To do so, we observe that a CNN typically composes a few convolutional layers with fully-connected layers. We thus initialize the weights of the convolutional layers such that they transmit the model input *unaltered* up to the fully-connected layers of the model architecture.

There are two important requirements for our approach to transmitting, or forwarding, model inputs through convolutional layers. The first is to make sure that no feature of the input data is lost. This requires having at least as many parameters at every convolutional layer as the number of input features. The second is to make sure that

(a) 2D Example. (b) 3D Example.

Figure 6: **Size-Preserving Adversarially Initialized Convolutional Filters.** Adversarially initialized convolutional filters that transmit their input to the next layer. The numbers indicated in the input and feature map represent the features, the numbers in the filter represent the weight initialization. Grey layers indicate random weight values while white layers indicate zero weights.

different features do not get overlaid. We explain how to ensure this next.

B.1. Preserving Input-Size

Two Dimensional Input. In general, preserving input size over a convolutional layer can be achieved through an adequate combination of padding, stride, and filter sizes. Specifically, we use stride *one* and an adequate zero-padding to preserve the size of the layer input. In order to transmit the input features, we create a filter with uneven dimensions (w, h), where $w = h$, and we initialize it with *zero* everywhere apart from the element in the middle which we set to *one*. For a two dimensional input (*e.g.* a grey-scale image), the described filter perfectly transmits the information to the next layer and creates a feature map that exactly replicates the input. See Figure 6a for this adversarially initialized filter.

Three Dimensional Input. Some input data to CNNs is distributed over several input channels, such as color images, that consist of three channels. At every layer, we, therefore need three adversarially initialized convolutional filters to "transmit a copy" of the input channels. A standard architecture can have many more filters per layer, which can, in the case of our attack be randomly initialized since they will be ignored by the attacker. Assume now that the original input features at the current layer l_i are distributed over a_{l_i} of the total $b_{l_{i-1}}$many feature maps. For example, in the first model layer, $a_{l_i} = b_{l_{i-1}}$ corresponds to the number of color channels required to encode the image. In subsequent layers, the remaining $b_{l_{i-1}} - a_{l_i}$ many feature maps contain random noise, introduced by random filters that do not transmit the input features (*e.g.* Filter 1 in Figure 6b). We denote the indices of the feature maps where the input features are located by $\vec{\alpha_{l_i}}$. We then need a_{l_i} many filters, initialized as described above to transmit the information to the next layer. The filters differ from each other only by the placement of the matrix that contains the *one* element. This placement must correspond to different indices in $\vec{\alpha_{l_i}}$. See Figure 6b for a visualization of this setting. Note that the placement of the feature-transmitting filters at layer l_i will determine the indices $\vec{\alpha_{l_{i+1}}}$ of the feature maps that are input to the next layer.

In the last convolutional layer before the fully-connected layer that we want to transmit the input to, the filters containing noise should be initialized such that they yield negative input to the ReLU function. Thereby, the output of the last convolutional layer becomes zero everywhere apart from the feature maps produced by the filters transmitting input data features. The flattened output then serves as input to the fully-connected layer, and reconstruction can be conducted as described in Section 6.

B.2. Reducing Input-Size

Two Dimensional Input. The reduction of size in convolutional layers can be achieved by increasing the stride. However, thereby, the number of features in the next feature map is reduced such that this feature map cannot accommodate all features from the previous layer. To overcome this, we propose distributing the features of one input feature map over several feature maps in the following layer. Figure 7a depicts this approach for two-dimensional inputs. Note that the stride is set to the dimensions of the convolutional filters (w, h) to prevent features from overlapping in the following layer. Additionally, to transmit all the features, the dimensions of the filters w and h $(w = h)$ need to be integer dividers of the previous feature map's dimensions. Finally, in total, for each layer that reduces the size of the input by a factor $\frac{1}{w}$, we require w^2 many filters to transmit every feature from one input feature map. Hence, assuming that at layer l_i the original features are distributed over a_{l_i} many input feature maps, we require $a_{l_i} \cdot w^2$ many filters to transmit all original input features.

Three Dimensional Input. The same approach as for the two dimensional input can be extended to the case with three input dimensions. The approach is visualized in Figure 7b. For improved visualization, we do not present the feature maps in the layer's output which contain only noise. Again, in both the two and three dimensional case, in the last convolutional layer before flattening, the noise filters should produce negative input to the ReLU function. This enables only extracting the original input features and no noise from the following fully-connected layer.

B.3. Reducing Detectability

In principle, our adversarial weight initialization for CNNs only requires the number of filters per layer that actually transmit the features. However, using only a small number of filters, *e.g.* *one* as in the case of the

(a) 2D Example. (b) 3D Example.

Figure 7: **Size-Reducing Adversarially Initialized Convolutional Filters.** Adversarially initialized convolutional filters that transmit their input to the next layer. The numbers indicated in the input and feature map represent the features, the numbers in the filter represent the weight initialization. Grey layers indicate random weight values while white fields indicate zero weights. Feature maps in the 3D example that only contain noise are suppressed for improved visualization.

size-preserving adversarial convolutional filters, leads to models architectures that deviate strongly from standard architectures. Therefore, we propose using a standard number of convolutional filters in every layer and initializing the filters that are not used to transmit features at random. Additionally, to prevent the simple detection strategy which relies on probing after every convolutional layer whether its input is equal to its output, one can replace the ones in the adversarially initialized convolutional filters by other positive constants. Data extraction at the fully-connected layer then yields data points where features of the original input data are scaled by (multiple different) factors. By applying the inverse of the factors encoded in the model weights this scaling can then be reverted. As a consequence, the rescaled extracted data points still perfectly correspond to the input data.

C. Additional Material

The following table describes the model architectures both for the FC-NNs and CNNs use throughout the paper. Note that the our method could also be applied to much larger CNNs with more layers: in fact, as long as each layer contains as many parameters as the data holds input features, our approach is applicable.

FC-NN Architecture	VGG-inspired CNN Architecture
Dense(n=1000, act=relu)	Conv(f=128, k=(3,3), s=1, p=same, act=relu)
Dense(n=3000, act=relu)	Conv(f=256, k=(3,3), s=1, p=same, act=relu)
Dense(n=3000, act=relu)	Conv(f=512, k=(3,3), s=1, p=same, act=relu)
Dense(n=2000, act=relu)	Flatten
Dense(n=1000, act=relu)	Dense(n=1000, act=relu)
Dense(n=#classes, act=None)	Dense(n=#classes, act=None)

TABLE 9: **Architectures of models used in the experiments on image data.** f: number of filters, k: kernel size, s: stride, p: padding act: activation function, n: number of neurons.

D. Additional Experimental Results

This section presents additional experimental results.

D.1. Passive Extraction

Figure 8 shows extraction from a randomly initialized FC-NN with architecture presented in Table 9.

IMDB-Model Architecture
Embedding(feat=10000, dim=250)
Dense(n=1000, act=relu))
Dense(n=1, act=None)

TABLE 10: **Architecture of models used in the experiments on the IMDB dataset.** feat: vocabulary size, dim: embedding size, act: activation function, n: number of neurons.

Figure 8: **Baseline: Passive Attack.** Data from the CIFAR10 dataset, extracted from the gradients of the first 30 weight rows at the first fully-connected layer of a randomly initialized FC-NN with architecture from Table 9.

D.2. Trap Weights and Active Extraction

Figures 11 and 13 depict the extracted data points for MNIST and ImageNet, respectively.

We, furthermore, study partial extractablity, *i.e.*, the case when a data point is not individually extractable, but still leaks meaningful private information about a training data point. Partial leakage occurs when an extracted gradient represents the overlay of only a few data points. In this case, the individual signal of each data point is still distinguishable, see for example the first data point in the third row of Figure 8. We plot in Figure 10 by how many data point each of the neurons with our trap weight initialization gets activated. This corresponds to the number of data points that will be present in the overlay of the respective gradients. We can see that nearly as many neurons get activated by two data points as by one data point (*i.e.* perfect extractability). In general, with our trap weights, neurons get activated by small numbers of data points. This indicates that the central party can still extract meaningful partial information on many data points, also if these are not perfectly extractable. Results an average over five runs with different trap weight initializations for a mini-batch of 100 data points from the ImageNet dataset.

To provide additional insights on data points that can and cannot be individually extracted, in Figure 14 which

B, Num	A	P	R
(20,1)	.496	.486	.950
(20,5)	.787	.213	.572
(20,10)	.851	.157	.412
(20,20)	.898	.116	.251
(50,1)	.687	.307	.790
(50,5)	.901	.107	.230
(50,10)	.928	.080	.138
(50,20)	.953	.053	.067
(100,1)	.800	.200	.562
(100,5)	.936	.066	.116
(100,10)	.966	.046	.054
(100,20)	.982	.028	.020

TABLE 11: **Effect of Mini-Batch Averaging.** Success of our adversarial weight initialization on MNIST under averaging over multiple mini-batches on the same model parameters. The number of mini-batches is denoted by **Num** and their respective size by **B**. The results depict the percentage of *active neurons* (A), *extraction-precision* (P), and *extraction-recall* (R) for extracting from 1000 neurons at the first layer of the FC-NN depicted in Table 9. All numbers are averaged over 10 runs with different adversarial initializations.

data points can and cannot be extracted. For the individually extractable data points, we, furthermore, depict how often each of them is individually extractable, *i.e.* for how many neurons this data point is the only one activating it. We see that our trap weights first amplify natural leakage *i.e.*, data points that are extractable from random weights are usually also extractable with our trap weight and our trap weights make other data points extractable. Second, our trap weights yield redundancy, *i.e.* data points are extractable multiple times from different weight rows' gradients.

We depict extraction success for local averaging over multiple mini-batches in Table 11.

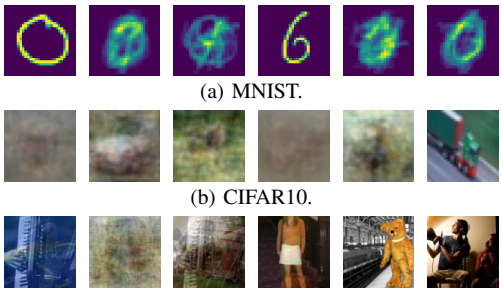
We also study the non-IID setup where users hold data from a single class, different from other users in the protocol. We present the *extraction-recall* and *extraction-precision* per class on the CIFAR10 dataset in Table 12.

Class	P (Passive)	P (Active)	R (Passive)	R (Active)
0	.064	**.570**	.185	**.352**
1	.041	**.276**	.208	**.560**
2	.056	**.480**	.195	**.384**
3	.044	**.318**	.208	**.489**
4	.056	**.516**	.225	**.426**
5	.045	**.356**	.238	**.534**
6	.049	**.358**	.209	**.442**
7	.051	**.367**	.205	**.515**
8	.055	**.536**	.209	**.386**
9	.043	**.395**	.240	**.559**

TABLE 12: **Non-IID Extraction on CIFAR10.** Success of our adversarial weight initialization (active) versus non-manipulated model weights (passive) on CIFAR10 in a non-IID setup where each user only holds data from a single class, different from all other users. The results depict the *extraction-precision* (P) and *extraction-recall* (R) for extracting from 1000 neurons at the first layer of the FC-NN depicted in Table 9. All numbers are averaged over 10 runs with different (adversarial) initializations. While in both passive and active extraction, extraction success between the classes differs, our adversarial weight initialization significantly increases leakage over all classes. Results are averaged over 5 runs.

s	MNIST R	CIFAR10 R	ImageNet R
.650	0.468	0.0	0.0
.700	0.477	0.0	0.0
.750	**0.603**	0.0	0.0
.800	**0.603**	0.085	0.0
.900	0.531	0.121	0.0
.910	0.504	0.147	0.0
.920	0.513	0.178	0.0
.930	0.459	0.210	0.0
.940	0.450	0.222	0.0
.950	0.513	**0.238**	0.0
.960	0.378	0.229	0.0
.970	0.450	0.191	0.073
.980	0.315	0.012	0.232
.990	0.360	0.0	**0.422**
.995	0.342	0.0	0.330
.999	0.270	0.0	0.0

TABLE 13: **Tuning Factor s on Data Points from Passive Extraction.** We model an attacker who does not hold auxiliary data to tune the scaling factor s. Such an attacker can, during the firs round, of the protocol extract the data points from the non-manipulated model weights' gradients. The points (we select those with features in range [0,1], see Figure 9), can be used to tune s. The identified optimal s (bold) w.r.t. the *extraction-recall* are close (0.75 or 0.80, MNIST) or identical (0.95, CIFAR10 and 0.99, ImageNet) to the original datasets' optimal s, 0.7, 0.95, and 0.99 for MNIST, CIFAR10, ImageNet.



(a) MNIST.

(b) CIFAR10.

(c) ImageNet.

Figure 9: **Passive Extraction from Non-Manipulated Model Weights.** We extract from the gradients of the weight rows at the first fully-connected layer of a randomly initialized FC-NN with architecture from Table 9 and depict data points whose features are in range [0,1]. These passively extracted data points can be used by an attacker to tune the hyperparameter s for active extraction.

Finally, we study how an attacker without any prior knowledge can tune s. One way to proceed is that the attacker does not adversarially initializes the model in the first FL iteration. It then extracts data points from the gradients, which are not necessarily individually extracted data points. From these data points, the attacker keeps the one in a valid image input range with features in range [0, 1], and uses these data points for fine-tuning s. We depict the resulting data points for MNIST, CIFAR10, and ImageNet in Figure 9 and show extraction success for different s on 100 such data point in Table 13.

D.3. Extraction under Lossy Layers

We also study the effect of "lossy" layers, such as dropout and pooling on our data extraction success. Therefore, we rely on the following architecture proposed by [4] for FL, see Table 14.

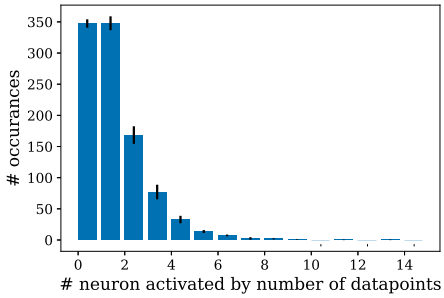
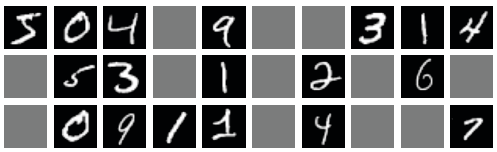
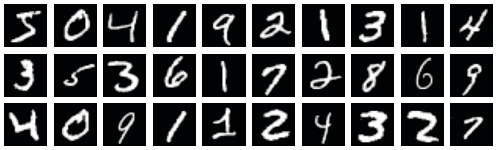


Figure 10: **ImageNet: Activation of Neurons.** To study partial extractability, we analyze by how many data points each neuron gets activated. Results are averaged over five different random and trap weight model initializations.



(a) Reconstructed data points.



(b) Original data points.

Figure 11: **MNIST. Reconstruction success of our adversarial initialization:** first 30 images from a mini-batch of 100 data points, extracted at the first fully-connected layer of the FC-NN from Table 9. Gray images indicate that the corresponding original data point could not be extracted individually from the model gradients.

Figures 15 and 16 and Figures 17 and 18 show individual effects of dropout and pooling layers on a reconstructions for mini-batches of size 1 and 20 respectively. We evaluated different dropout rates $p \in \{0.0, 0.1, 0.3, 0.5, 0.7, 0.9\}$. Note that the second dropout layer does not have a significant impact on the success of our reconstruction since we extract from the first fully-connected model layer before information can get lost due to the second dropout. To evaluate dropout without pooling, we remove the MaxPool layer, and to evaluate pooling without dropout, we set the dropout rate to $p = 0.1$. Although existence of non-invertible components compromises overall reconstruction fidelity, we observe it is often possible to still recognise individual data points.

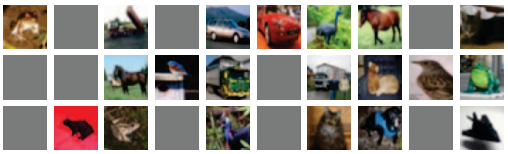
CNN Architecture by [4]
Conv(f=32, k=(3,3), s=1, p=same, act=relu)
MaxPool()
Conv(f=64, k=(3,3), s=1, p=same, act=relu)
Dropout()
Flatten
Dense(n=1000, act=relu)
Dropout()
Dense(n=#classes, act=None)

TABLE 14: **CNN Architecture by [4] used to evaluate the data extraction attack under the impact of Dropout and Pooling.** f: number of filters, k: kernel size, s: stride, p: padding act: activation function, n: number of neurons.

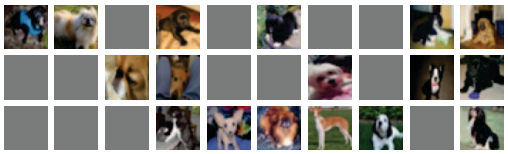
D.4. TensorFlow Federated

We adapted the implementation of FedAvg [1] provided by the developers to pass each individual gradient update through our reconstruction function. Note that the whole change took only minutes of work and required minimal code changes, such that they could easily be implemented by a dishonest central party. We pre-generated our adversarial initialized shared models with scaling factor $s = 0.7$ and $s = 0.99$ and 1000 neurons at the first fully-connected layer, and passed them to the users. The aggregator then collects the gradients and performs reconstruction.

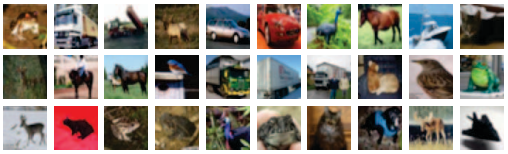
We find that our attack works consistently well against commonly used FL benchmarks integrated into the library. Over 50 users, for EMNIST [7], our trap weights yield 0.32 ± 0.07 *extraction-recall* and 0.05 ± 0.02 *extraction-precision*, versus 0.10 ± 0.05, and 0.02 ± 0.01 in the non-adversarial baseline. For CIFAR100 [28] we get 0.44 ± 0.05 *extraction-recall* and 0.22 ± 0.06 *extraction-precision*, versus 0.20 ± 0.04, and 0.04 ± 0.01 in the baseline. These results are comparable to the ones reported in the previous experiments, and confirm that our attack is practical.



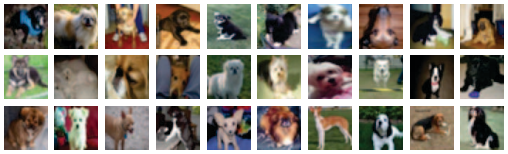
(a) Reconstructed data points from different classes.



(b) Reconstructed data points from class "dog".

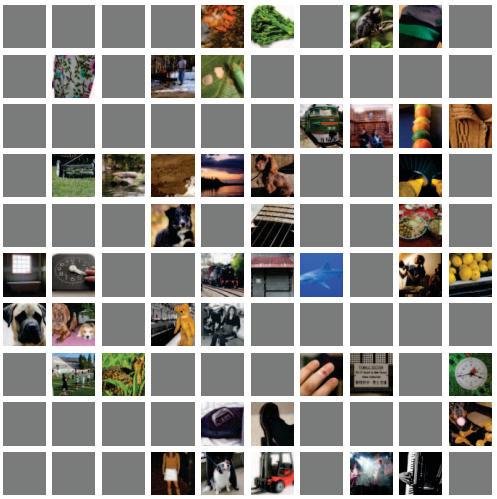


(c) Original data points from different classes.

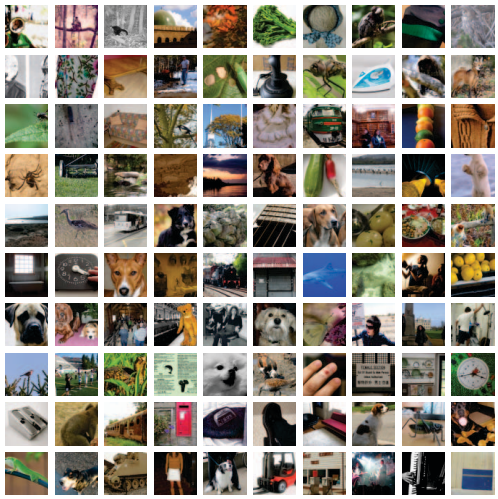


(d) Original data points from class "dog".

Figure 12: **CIFAR10 Data Extracted from Mini-Batches with Data Points from Different and the Same Class.** Reconstruction success of our adversarial initialization: first 30 images from a mini-batch of 100 data points, extracted at the first fully-connected layer of the CNN from Table 9. Gray images indicate that the corresponding original data point could not be extracted individually from the model gradients. Our extraction success for data from the same class is as high as for data from different classes.

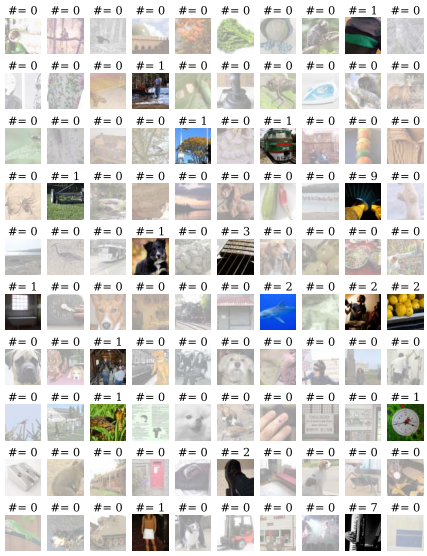


(a) Reconstructed data points.

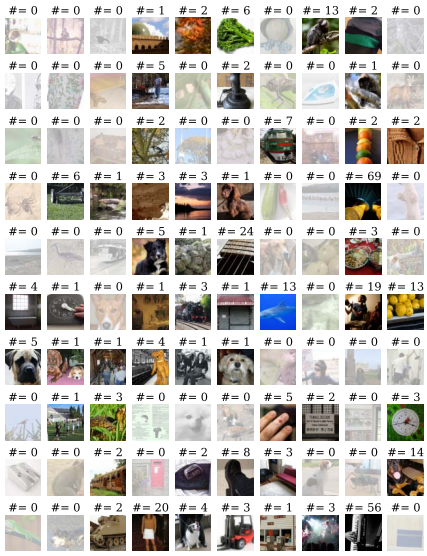


(b) Original data points.

Figure 13: **ImageNet. Reconstruction success of our adversarial initialization:** all reconstructed data points from a mini-batch of 100 data points, extracted at the first fully-connected layer of the CNN from Table 9. Gray images indicate that the corresponding original data point could not be extracted individually from the model gradients..

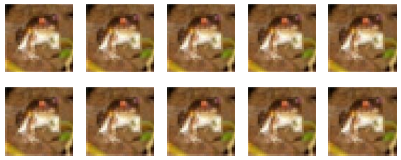


(a) Randomly initialized model weights.

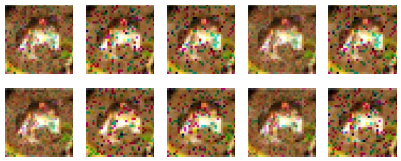


(b) Trap weights.

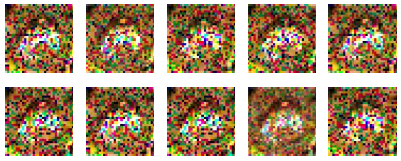
Figure 14: **ImageNet: Extractability.** Number of individual occurrences in the rescaled gradients over a mini-batch of 100 data points, extracted at the first fully-connected layer of the CNN from Table 9. To provide insights into what data points could not be individually extracted, we plot data points with zero occurrences with low saturation.



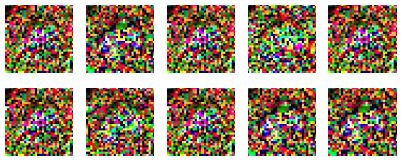
(a) Dropout with $p = 0.0$.



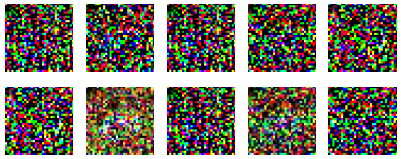
(b) Dropout with $p = 0.1$.



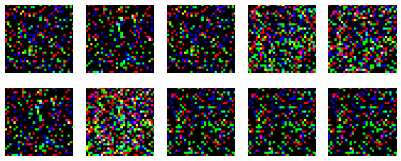
(c) Dropout with $p = 0.3$.



(d) Dropout with $p = 0.5$.



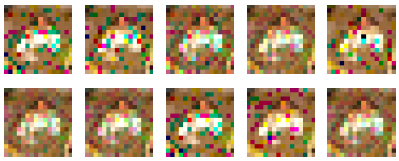
(e) Dropout with $p = 0.7$.



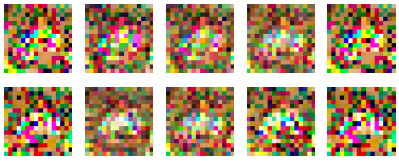
(f) Dropout with $p = 0.9$.

Figure 15: Batch size 1.

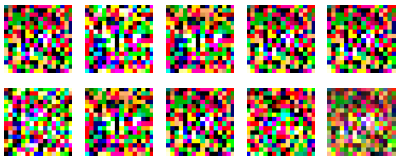
(a) Dropout with $p = 0.0$ and pooling.



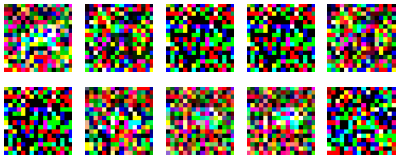
(b) Dropout with $p = 0.1$ and pooling.



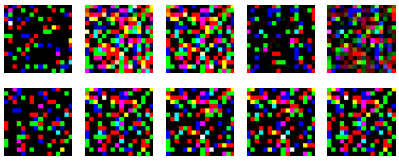
(c) Dropout with $p = 0.3$ and pooling.



(d) Dropout with $p = 0.5$ and pooling.

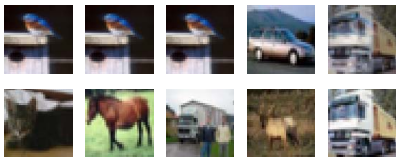


(e) Dropout with $p = 0.7$ and pooling.

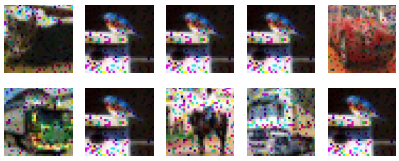


(f) Dropout with $p = 0.9$ and pooling.

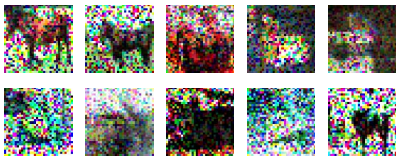
Figure 16: Batch size 1.



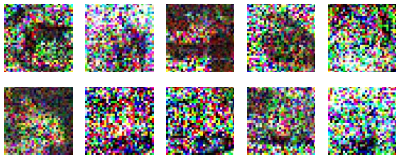
(a) Dropout with $p = 0.0$.



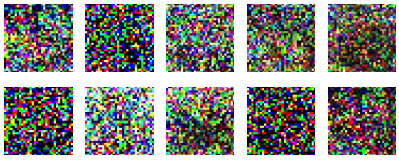
(b) Dropout with $p = 0.1$.



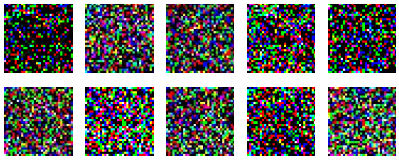
(c) Dropout with $p = 0.3$.



(d) Dropout with $p = 0.5$.



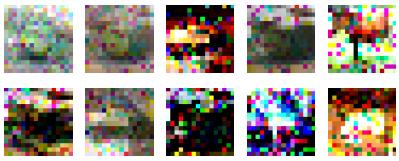
(e) Dropout with $p = 0.7$.



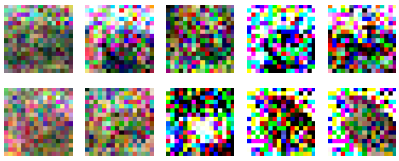
(f) Dropout with $p = 0.9$.

Figure 17: Batch size 20.

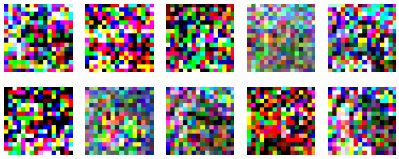
(a) Dropout with $p = 0.0$ and pooling.



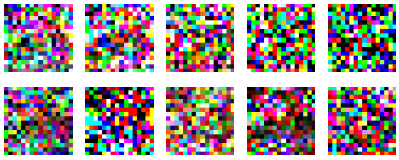
(b) Dropout with $p = 0.1$ and pooling.



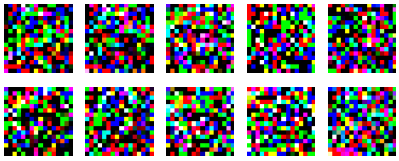
(c) Dropout with $p = 0.3$ and pooling.



(d) Dropout with $p = 0.5$ and pooling.

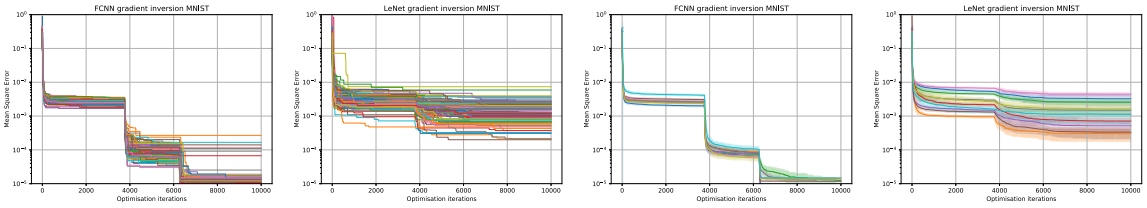


(e) Dropout with $p = 0.7$ and pooling.



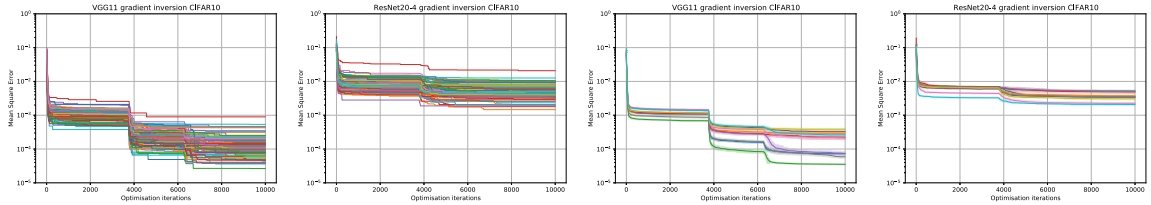
(f) Dropout with $p = 0.9$ and pooling.

Figure 18: Batch size 20.



(a) FC-NN (b) LeNet-5 (c) FC-NN (d) LeNet-5

Figure 19: **Baseline—Prior Work.** Single sample gradient inversion with untrained network using the inversion method proposed by [15] for the first 100 MNIST datapoints. (a) and (b) shows fidelity of individual datapoint reconstruction with no restarts, while (c) and (d) show 32 different optimisation start points. Error bars are a single standard deviation of individual restarts.



(a) VGG11-like (b) ResNet20-4 (c) VGG11-like (d) ResNet20-4

Figure 20: **Baseline—Prior Work.** Single sample gradient inversion with untrained network using the inversion method proposed by [15] for the first 100 CIFAR10 datapoints. (a) and (b) shows fidelity of individual datapoint reconstruction with no restarts, while (c) and (d) show 32 different optimisation start points. Error bars are a single standard deviation of individual restarts.