User Label Leakage from Gradients in Federated Learning

Aidmar Wainakh, Fabrizio Ventola, Till Müßig, Jens Keim,
Carlos Garcia Cordero, Ephraim Zimmer, Tim Grube, Kristian Kersting, and
Max Mühlhäuser

Technical University of Darmstadt, Germany
firstname.lastname@tu-darmstadt.de

Abstract. Federated learning enables multiple users to build a joint model by sharing their model updates (gradients), while their raw data remains local on their devices. In contrast to the common belief that this provides privacy benefits, we here add to the very recent results on privacy risks when sharing gradients. Specifically, we propose Label Leakage from Gradients (LLG), a novel attack to extract the labels of the users' training data from their shared gradients. The attack exploits the direction and magnitude of gradients to determine the presence or absence of any label. LLG is simple yet effective, capable of leaking potential sensitive information represented by labels, and scales well to arbitrary batch sizes and multiple classes. We empirically and mathematically demonstrate the validity of our attack under different settings. Moreover, empirical results show that LLG successfully extracts labels with high accuracy at the early stages of model training. We also discuss different defense mechanisms against such leakage. Our findings suggest that gradient compression is a practical technique to prevent our attack.

Keywords: Label leakage · Federated learning · Gradient attack · Privacy attack.

1 Introduction

In an increasingly interconnected world, the abundance of data and user information has brought machine learning (ML) techniques into daily life and many services. Arguably, the most common ML approaches work in a centralized fashion, typically requiring large amounts of user data to be collected and processed by central service providers. This data can be of sensitive nature, raising concerns about the handling of data in accordance with user expectations and privacy regulations (e.g., GDPR).

Federated Learning (FL) is an emerging ML setting that allegedly enables service providers *and* users to utilize the power of ML without exposing the user's personal information. The general principle of FL consists of cooperating to train an ML model in a distributed way. Users are given a model, which they can locally train with their sensitive data. Afterward, users only share the model gradients of their training endeavors with a central server. The users' gradients are aggregated

to establish the joint model [24]. This general principle is currently believed to reduce the impact on users' privacy compared to the classical centralized ML setting, since personal information does not leave the user, and sharing learning gradients does not supposedly reveal information about the user [37].

However, a considerable number of recent works have shown that gradients can be exploited to reconstruct the users' training data in FL [3,31,9,36]; while protecting the users' ground-truth labels from possible leakage has received only limited attention [37,34,17], mainly focusing on gradients generated from a small number of data samples (small batches) or binary classification tasks. Label leakage, however, is an important setting for FL. Both, FL and distributed ML are used in many applications where labels can contain highly sensitive information. For example, in the medical sector, hospitals employ distributed learning to collaboratively build ML models for disease diagnosis and prediction [13]. Building models in this and many other settings, while maintaining the users' privacy, is crucial. Leaking the labels of the user's data in such scenarios might disclose the user's diseases, which is a severe violation of privacy. It is essential to highlight this issue and investigate to what extent gradients can leak information about labels. For this purpose, developing privacy attacks that exploit gradients is of high importance.

Triggered by this, we propose Label Leakage from Gradients (LLG), a novel attack to extract ground-truth labels from shared gradients trained with mini-batch stochastic gradient descent (SGD) for multi-class classification. LLG is based on a combination of mathematical proofs and heuristics derived empirically. Our attack exploits two properties that the gradients of the last layer of a neural network have: (**P1**) Inspired by [34], we observe that the direction of these gradients indicates whether a label is part of the training batch. (**P2**) The gradient magnitude can hint towards the number of occurrences of a label in the batch. In our Work-in-Progress paper [30], we have introduced the fundamentals of these properties and the initial idea of LLG. Here, we formalize the properties, provide their mathematical proofs, extend our threat model, and conduct an extensive evaluation, as follows.

- We consider four benchmark datasets, namely MNIST, SVHN, CIFAR-100, and CelebA. Results show that LLG achieves high success rate despite the datasets having different classification targets and complexity levels.
- We study LLG considering different capabilities of the adversary. Experiments demonstrate that an adversary with an auxiliary dataset, similar to the training dataset, can adequately extract labels with accuracy $> 98\%$ at the early stage of the model training.
- We show that our simple attack can outperform one of the state-of-the-art optimization-based attacks, Deep Leakage from Gradients (DLG) [37], under several settings. Our approach is orders of magnitude faster than DLG.
- We also investigate the effectiveness of the attack on various model architectures including simple Convolutional Neural Network (CNN), LeNet [16], and ResNet20 [12]. Results suggest that LLG is not sensitive to the complexity of the model architecture.

- We illustrate the influence of the model convergence status on LLG. Findings reveal that LLG can perform best at the early stages of training and still poses a serious threat for well-trained models.
- Finally, we test LLG against two defense mechanisms: noisy gradients and gradient compression (pruning). Results show that gradient compression with $\geq 80\%$ compression ratio can render the attack ineffective.

We proceed as follows. We start off by reviewing the background and our problem setting in Section 2. Next, in Section 3, we present related work on information leakage from gradients. We elaborate on our findings regarding gradients properties in Section 4. Our attack is then explained in Section 5. Before concluding, we present the results of our evaluation in Section 6.

2   Background

In this section, we present the fundamentals of FL and our problem setting including the threat model.

2.1   Federated Learning

Federated Learning (FL) is a machine learning setting that enables a set of N users to collaboratively train a joint model under the coordination of a central server [14]. For each round t of the global training process, a subset of users $K_t \ll N$ is selected to train the model locally on their data. In particular, they optimize the model weights $\boldsymbol{W}$ based on the gradients $\nabla \boldsymbol{W}$, which are the derivative of the loss function l w.r.t. the weights $\boldsymbol{W}$. Users can take one step of gradient descent (FedSGD [24]) or multiple steps (FedAvg [24]) before sharing the gradients $\nabla \boldsymbol{W}$ (or model updates) with the server. The server calculates a weighted average to aggregate the gradients from the K_t users, and updates the global model $\boldsymbol{W}(t+1) = \boldsymbol{W}(t) - \eta \sum_{k=1}^{K_t} \frac{n_k}{n} \nabla \boldsymbol{W}^k$ where η is the learning rate, n_k number of data samples of user k, and n is total number of data samples. This process is repeated until the model potentially converges [24].

2.2   Problem setting

We consider an FL scenario where a user trains a neural network (NN) model for one iteration on a batch of their data samples, following the FedSGD algorithm [24]. Then, the user shares the gradients resulted from training with the server. We assume that the model consists of L layers and is trained with cross-entropy loss [10] over one-hot labels for a multi-class classification task. For our attack, we focus on the gradients $\nabla \boldsymbol{W}_L$ w.r.t. the last-layer weights $\boldsymbol{W}_L$ (between the output layer and the layer before). The gradient vector $\nabla \boldsymbol{W}_L^i$ represents gradients connected to label i on the output layer. We note g_i to refer to the sum of the elements of $\nabla \boldsymbol{W}_L^i$: $g_i = \mathbb{1}^T \cdot \nabla \boldsymbol{W}_L^i$. Next, we define our threat model in FL w.r.t. three aspects: adversary access point, mode, and observation.

Access point. An adversary might be able to access the gradients by compromising the user's device as the gradients are calculated on the user side before

sharing with the server. We assume that the user's device can be compromised partially, such that the adversary has no access to the training data or labels [32]. Such scenario can apply, for example, to several online ML applications, where the training data is not stored but used for training on-the-fly. The server also can access the gradients of an individual user, in case no secure aggregation [6] is used. In addition, if the connection between the server and the users is not secure, the gradients might be intercepted by an external eavesdropper.

Mode. We assume the adversary to act in a passive mode. The adversary may analyze the gradients to infer information about the users, but without hindering or deviating from the regular training protocol. This adversary mode is widely common in privacy attacks [27,33,23,37], where the focus is on disclosing information rather than disturbing the system.

Observation. The adversary might be capable to observe different amounts of information to launch their attack. We consider three possibilities.

1. Shared gradients: the adversary has access only to the shared gradients. This can apply for an external eavesdropper or an adversary with limited access to the user's device.
2. White-box model: in addition to the gradients, the adversary is aware of the model architecture and parameters. In the case of a curious server or compromised user, the adversary might have this kind of information.
3. Auxiliary knowledge: the adversary has access to an auxiliary dataset, which contains the same classes as the original training dataset. This is a common scenario in real-world cases; given that NNs need a considerable amount of data for training, a typical adopted strategy is to train the model on publicly available datasets, eventually, fine-tuning the model on ad-hoc labeled data.

3   Related Work

Several attacks in the literature exploit the shared gradients to (1) reconstruct user training data or (2) extract the ground-truth labels.

Data reconstruction. Aono et al. [2,3] are the first to discuss reconstructing data from gradients on simple NNs with a training batch of one sample. Wang et al. [31] moved on to generative attacks, leveraging a Generative Adversarial Network (GAN) to reconstruct the input data in a CNN. In contrast, Zhu et al. [37] introduce an optimization-based attack; the attacker generates dummy input data and output labels, then optimizes them using L-BFGS [20] to generate dummy gradients that match the shared ones. As an improvement, Geiping et al. [9] propose using cosine similarity and the Adam optimization algorithm. Wei et al. [32] provide a framework for evaluating the optimization-based attacks considering multiple factors, e.g., optimizer, activation, and loss functions. Qian et al. [28] theoretically analyze the limits of [37] considering fully-connected NNs and vanilla CNNs. They also propose a new initialization mechanism to speed up the attack convergence. Unlike previous approaches, Enthoven et al. [8] introduce an analytical attack that exploits fully-connected layers to reconstruct the input data on the server side, and they extend this exploitation to CNNs. Recently,

Zhu et al. [36] propose a recursive closed-form attack. They demonstrate that one can reconstruct data from gradients by recursively solving a sequence of systems of linear equations. Overall, all the aforementioned attacks, except for [37], only focus on reconstructing the input training data while overlooking the leakage of data labels, which can be of high sensitivity.

Label extraction. While the data reconstruction attacks attracted considerable attention in the research community, a very limited number of approaches were proposed on the label leakage. As part of the optimization approach of Zhu et al. [37], the ground-truth labels are extracted. However, the approach requires a learning phase where the model is sensitive to the weight initialization and can be hard to converge. Moreover, it was found to extract wrong labels frequently [34] and it is effective only for gradients aggregated from a batch size < 8 [25]. Zhao et al. [34] propose a more reliable analytical approach, which exploits the observation that gradients of classification (cross-entropy loss) w.r.t. the last layer weights have negative values for the correct labels. However, their approach is limited to one-sample batch, which is uncommon case in real-world applications of FL. Li et al. [17] propose also an analytical approach based on the observation that the gradient norms of a particular class are generally larger than the others. However, their approach is tailored only for a binary classification task in vertical split learning. Overall, the existing approaches are not well generalized to arbitrary batch sizes nor number of classes. Also, the influence of different model architectures on these approaches is yet to be investigated.

4   Gradient Analysis

In gradient descent optimization, the values of gradient determine how the parameters of a model need to be adjusted to minimize the loss function. Through an empirical analysis, we carefully derive two properties for the sign and magnitude of the gradients that indicate the ground-truth labels. In this section, we formalize these properties, next, in Section 5, we use them as a base for our attack.

Property 1. *For label i and last layer L in an NN model, when $\nabla \boldsymbol{W}_L^i < 0$, label i is present in the training batch on which gradient descent was applied*[1].

Proof. We consider an NN model for a classification task. The model is trained using the cross-entropy loss over labels encoded with a one-hot encoding. This loss function l is defined as $l(\boldsymbol{x}, c) = -\ln \frac{e^{y_c}}{\sum_j e^{y_j}}$, where $\boldsymbol{x}$ is a multidimensional input instance and c represents the ground-truth label of $\boldsymbol{x}$. While, $\boldsymbol{y} = [y_1, y_2, ..., y_n]$ is the output vector of the model where each $y_i \in \boldsymbol{y}$ is the score predicted for the ith class, y_c is the score assigned to the ground-truth label, and n is the total number of classes.

Given a batch size B, we have a set $\boldsymbol{X}$ of B samples and the set of their labels C. Thus, we can define a training batch as a set composed of the pairs

---

[1] This property is a generalization of the main observation in [34].

$\{(\boldsymbol{x}_1, c_1), \ldots, (\boldsymbol{x}_B, c_B)\}$. Therefore, we can redefine the loss function as the loss $l(\boldsymbol{x}, c)$ averaged over a batch of B labeled samples

$$l(\boldsymbol{X}, C) = -\frac{1}{B} \sum_{k=1}^{B} \ln \frac{e^{y_{c(k)}}}{\sum_j e^{y_{j(k)}}}, \tag{1}$$

where $c(k)$ is the ground-truth label for the kth sample in the batch, and $y_{c(k)}$ is the corresponding output score when $\boldsymbol{x}_k$ is given as input to the model. We note that the gradient d_i of the loss w.r.t. an output y_i is

$$d_i = \frac{\partial l(\boldsymbol{X}, C)}{\partial y_i} = -\frac{1}{B} \sum_{k=1}^{B} \left(\frac{\partial \ln e^{y_{c(k)}}}{\partial y_i} - \frac{\partial \ln \sum_j e^{y_{j(k)}}}{\partial y_i} \right) \tag{2}$$

$$= -\frac{1}{B} \sum_{k=1}^{B} \left(\mathbb{1}(i = c(k)) - \frac{e^{y_{i(k)}}}{\sum_j e^{y_{j(k)}}} \right) \tag{3}$$

$$= -\frac{1}{B} \sum_{k=1}^{B} \mathbb{1}(i = c(k)) + \frac{1}{B} \sum_{k=1}^{B} \frac{e^{y_{i(k)}}}{\sum_j e^{y_{j(k)}}} \tag{4}$$

$$= -\frac{\lambda_i}{B} + \frac{1}{B} \sum_{k=1}^{B} \frac{e^{y_{i(k)}}}{\sum_j e^{y_{j(k)}}}, \tag{5}$$

where λ_i is the number of occurrences (frequency) of samples with label i in the training batch. When $i \notin C$, $\lambda_i = 0$, and $\frac{e^{y_i}}{\sum_j e^{y_j}} \in (0, 1)$, thus, $d_i \in (0, 1)$. Instead, when $i \in C$, we have $-\frac{\lambda_i}{B} \leq d_i \leq 1 - \frac{\lambda_i}{B}$. Hence, if the gradient d_i is negative, we can conclude that label $i \in C$. Of course, the d_i value moves in this range accordingly to the status of the network weights optimization, e.g. if $i \in C$ and the network performs poorly, then, d_i will be closer to $-\frac{\lambda_i}{B}$. However, the gradients $\boldsymbol{d}$ w.r.t. the outputs $\boldsymbol{y}$ are usually not calculated or shared in FL, but only $\nabla \boldsymbol{W}$, the gradients w.r.t. the model weights $\boldsymbol{W}$. We write the gradient vector $\nabla \boldsymbol{W}_L^i$ w.r.t. the weights $\boldsymbol{W}_L^i$ connected to the ith output representing the ith class confidence in the output layer as follows

$$\nabla \boldsymbol{W}_L^i = \frac{\partial l(\boldsymbol{X}, C)}{\partial \boldsymbol{W}_L^i} = \frac{\partial l(\boldsymbol{X}, C)}{\partial y_i} \cdot \frac{\partial y_i}{\partial \boldsymbol{W}_L^i} \tag{6}$$

$$= d_i \cdot \frac{\partial (\boldsymbol{W}_L^{i^T} \boldsymbol{a}_{L-1} + b_L^i)}{\partial \boldsymbol{W}_L^i} \tag{7}$$

$$= d_i \cdot \boldsymbol{a}_{L-1}, \tag{8}$$

where $\boldsymbol{y} = \boldsymbol{a_L}$ is the activation function of the output layer, b_L^i is the bias, and $y_i = \boldsymbol{W}_L^{i^T} \boldsymbol{a}_{L-1} + b_L^i$. When non-negative activation functions (e.g. Sigmoid or ReLU) are used, $\boldsymbol{a}_{L-1}$ is non-negative. Consequently, $\nabla \boldsymbol{W}_L^i$ and d_i have the same sign. Considering Eq. (5), we conclude that negative $\nabla \boldsymbol{W}_L^i$ indicates that the label i is present in the ground-truth labels set C of the training batch.

Property 2. *In untrained models, the magnitude of the gradient $g_i = \mathbb{1}^T \cdot \nabla \boldsymbol{W}_L^i$ is proportional to the number of occurrences λ_i of label i in the training batch.*

Proof. Based on Eq. (8), we have $g_i = \mathbb{1}^T \cdot \nabla \boldsymbol{W}_L^i = d_i \left(\mathbb{1}^T \cdot \boldsymbol{a}_{L-1} \right)$. We substitute d_i with its expression from Eq. (5) as follows

$$g_i = \left(-\frac{\lambda_i}{B} + \frac{1}{B} \sum_{k=1}^{B} \frac{e^{y_{i(k)}}}{\sum_j e^{y_{j(k)}}} \right) \left(\mathbb{1}^T \cdot \boldsymbol{a}_{L-1} \right) . \tag{9}$$

An untrained model typically performs poorly predicting the ground-truth labels. Meaning, when the label i is present in batch $\lambda_i > 0$, then $y_i \approx 0$ (the predicted score is close to zero). Therefore, $\frac{e^{y_{i(k)}}}{\sum_j e^{y_{j(k)}}}$ is also close to zero. Consequently, we can write

$$g_i \approx -\frac{\lambda_i}{B} \left(\mathbb{1}^T \cdot \boldsymbol{a}_{L-1} \right) , \tag{10}$$

thus, g_i is proportional to λ_i. We refer to $-\frac{\mathbb{1}^T \cdot \boldsymbol{a}_{L-1}}{B}$ as *impact m*, therefore, $g_i \approx \lambda_i m$. We define the impact m as *the change of the gradient value caused by a single occurrence of a label in the training batch*. This value is negative and constant across labels, thus, label-agnostic.

However, in our empirical analysis we notice that in some cases the score y_i might be high while the label i is not present in the training batch. This comes as a result of misclassification and leads to a positive shift in the gradient values. We call this shift *offset s*, and based on Eq. (9), we can write

$$s_i = \left(\frac{1}{B} \sum_{k=1}^{B} \frac{e^{y_{i(k)}}}{\sum_j e^{y_{j(k)}}} \right) \left(\mathbb{1}^T \cdot \boldsymbol{a}_{L-1} \right) . \tag{11}$$

This offset value varies from a label to another, so it is a label-specific value. Using our defined parameters impact m and offset s_i, we can reformulate Eq. (9) as follows $g_i = \lambda_i m + s_i$. From this equation, it follows easily that the number of occurrences λ_i of label i can be derived from the parameters m, s_i, and g_i.

To demonstrate the two gradient properties, we randomly initialize the weights of a CNN composed by three convolutional layers. Then, we check the gradients g_i by evaluating the network on a batch of samples taken from the MNIST dataset [16]. We repeat the experiment 1,000 times with different batch sizes $B \in \{2, 8, 32, 128\}$. Fig. 1(a) depicts the distribution of the resulting gradients, where the y-axis shows the gradient values and the x-axis represents the number of occurrences for a label $i : \forall i \in [1, n]$.

We can see that there are no negative gradients at $\lambda_i = 0$ (framed in red), i.e., the negative gradients always correspond to an existing label in the batch $\lambda_i > 0$, which confirms Property 1. For all the batch sizes, we notice that the values of the gradients decrease consistently with the increase of the occurrences. This, in turn, confirms Property 2 and our definition of the impact parameter. We

(a) Gradients distribution   (b) Gradients heatmap   (c) After calibration
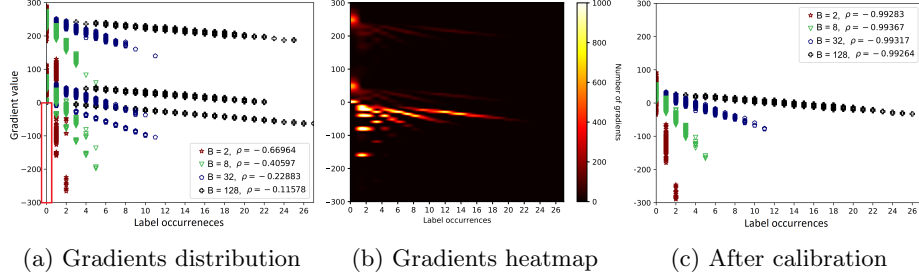
Fig. 1: Distribution of gradients obtained from a randomly initialized CNN on a batch of samples of MNIST varying the batch size in $\{2, 8, 32, 128\}$: (a) the distribution shows the correlation between the gradients and the label occurrences, (b) heatmap shows that the majority of the gradients have negative values when the corresponding label is present in the batch, (c) gradients after calibration exhibit a more prominent correlation with label occurrences. Given this strong correlation, it is possible to accurately estimate the label occurrences in the training batch basing on the gradient values.

also observe that the decrease of gradient values is roughly constant regardless of the label. This confirms the impact being label-agnostic, as we described earlier. Furthermore, we notice that the magnitude of the impact is negatively correlated with the batch size. Meaning, the more samples are present in a batch, the smaller are the changes of the gradients for a different number of occurrences. This is also clear from the definition of impact in Eq. (10). In Fig. 1(b), we can see the heatmap of (a), where it is possible to observe that the majority of the gradients have negative values, while only part of the gradients (18%) are positive, shifted by an offset s_i defined in Eq. (11).

5   Label Extraction

In this section, we present our approach, Label Leakage from Gradients (LLG), to extract the ground-truth labels from shared gradients. We first introduce different methods to estimate our attack parameters, impact and offset. Then, we explain the attack.

5.1   Attack parameters estimation

In the light of different threat models, we developed several methods to estimate the impact and offset.

Shared gradients. In this scenario, the adversary has access only to the shared gradients. As mentioned earlier, the impact refers to the change in the value of the gradients corresponding to one occurrence of a label. Our intuition is that a good estimation for the impact is obtained by averaging the gradients of the labels in the batch over the batch size B. Based on Property 1, we know that all

negative gradients are indicating existing labels in the training batch. Therefore, we average only the gradients with negative values. Consequently, this average is an underestimation since some gradients may be positive because they are shifted with an offset. We empirically observed that multiplying by a factor that depends on the total number of classes n is a good additive correction, precisely, we multiply by $(1 + 1/n)$. Thus, we estimate the impact m as follows

$$m = \frac{1}{B} \sum_{i:g_i<0}^{n} (g_i) \left(1 + \frac{1}{n}\right) . \tag{12}$$

For this threat model, the offset s_i cannot be estimated, thus, considered to be zero in the attack.

White-box model. When the adversary additionally has access to the model architecture, they can use it to generate more gradients and gain more insights about the behavior of the gradients in this model. Thus, better estimations for the impact and offset can be achieved. For that, they can use batches of dummy data samples, e.g., dummy images of zeros (black), ones (white), or random pixels. More precisely, we form a collection of dummy batches, each batch contains data samples assigned to one label i. For impact estimation, we pass these batches to the model, one at a time, and calculate the average $\bar{g}_i$ for all the batches corresponding to each label $i \in [1, n]$. Then, we average over all classes n and the batch size B as follows

$$m = \frac{1}{nB} \sum_{i=1}^{n} (\bar{g}_i) \left(1 + \frac{1}{n}\right) . \tag{13}$$

As mentioned earlier, we assume the offset s_i to be an approximation of misclassification penalties, when the model mistakenly predicts i to be ground-truth. Based on this intuition, we estimate the offset s_i by passing batches full of other labels $\forall j \in [1, n] \colon j \neq i$, each batch full of one label, one batch per run. We repeat this for various batch sizes, in total for t runs. In these runs, the gradients of label i reflect to some extent the misclassification penalties. Therefore, we calculate the mean of these gradients to be our estimated offset, thus, we have

$$s_i = \frac{1}{t} \sum_{k=1}^{t} (g_{i_k}) . \tag{14}$$

Auxiliary knowledge. In this scenario, the adversary has access to the shared gradients, model, and auxiliary data that contains the same classes as the training dataset. Here, the adversary can follow the same methods of the white-box scenario, however, using real input data instead of dummy data. This in turn is expected to yield better estimations for the impact and offset

To demonstrate the quality of our offset estimation, we calibrate the gradients of Fig. 1(a) by subtracting the estimated offset and plot the results in Fig. 1(c). We can see how the gradients become mainly negative values and strongly correlated with the label occurrences (Pearson correlation coefficient $|\rho| > 0.99$) [4]. The calibration process eliminates the effect of the offset and makes the gradient values more consistent, thus, easier to be used for extracting the labels.

Algorithm 1: Label Leakage from Gradients (LLG)

Data: $G = \{g_1, .., g_n\}$: set of gradients, m: impact, $S = \{s_1, .., s_n\}$: set of offsets, B: batch size.

Result: E: list for extracted labels.

```
1  for g_i ∈ G do                    7  G ← G − S;
2      if g_i < 0 then               8  while |E| < B do
3          append i to E;            9      Select g_i : g_i = min(G);
4          g_i ← g_i − m;           10      append i to E;
5      end                          11      g_i ← g_i − m;
6  end                             12  end
```

5.2   Label Leakage from Gradients Attack

LLG extracts the ground-truth labels from gradients by exploiting Property 1 and 2. The attack consists of three main steps summarized in Algorithm 1.

1. We start with extracting the labels based on the negative values of the gradients (Property 1). Thus, the corresponding label of each negative gradient is added to the list of the extracted labels E. Every time we add a label to E, we subtract the impact from the corresponding gradient (Property 2). As Property 1 holds firm in our problem setting, we can guarantee 100% correctness of the extracted labels in this step (Lines 1-5).
2. We calibrate the gradients by subtracting the offset. In case the offset is not estimated, it is considered to be zero. This step increases the correlation between the gradient values and label occurrences, which facilitates better label extraction based on these values (Line 7).
3. After calibration, the minimum gradient value (negative with maximum magnitude) is more likely corresponding to a label occurred in the batch (see Fig. 1(c)). Therefore, we select the minimum and add the corresponding label to the extracted labels. We repeat Step (3) until the size of the extracted labels list matches the batch size. Assuming that the batch size is known or can be guessed by the adversary (Lines 8-11).

Finally, the output of the LLG attack is the list of extracted labels E, precisely, the labels existing in the batch and how many times they occur.

6   Empirical Evaluation

We evaluate our attack effectiveness with varying settings including: different threat models, model architectures, and model convergence status. We also test the robustness of LLG against two defense mechanisms, namely noisy and compressed gradients[2]. For the sake of simplicity, we refer to $g_i = \mathbb{1}^T \cdot \nabla \boldsymbol{W}_L^i$ as the gradient of label i in the rest of this section.

---

[2] We are working on releasing our source code to the public.

6.1   Experimental setup

Default model. We use a CNN model with three convolutional layers (see Appendix, Table 1) as our default model for a classification task. The activation function is Sigmoid, and we use SGD as an optimizer with learning rate 0.1 and cross-entropy as loss function. When applying the attack, we feed the model with a batch of size 2^k with $k \in [0, 7]$. The label distribution in the batch can be *balanced* or *unbalanced*. For balanced data, the samples of the batch are selected randomly from the dataset. For unbalanced data, we select 50% of the batch samples from one random label i and 25% from another label j. The remaining 25% of the batch is chosen randomly. We initialize the model with random weights and repeat each experiment 100 times, then report the mean values for analysis and discussion.

Datasets. We conduct our experiments on four widely used benchmark datasets: MNIST [16] consists of 70,000 grey-scale images for handwritten digits, with totally 10 classes. SVHN [26] has 99,289 color images of house numbers with 10 classes. CIFAR-100 [15] contains 60,000 color images with 100 classes. CelebA [21] is a facial attributes dataset with 202,599 images. In our experiments, we consider only the hair color attribute with 5 classes.

Metrics. To measure the attack effectiveness, we use the attack success rate (ASR) metric, which is expressed as the ratio of the correctly extracted labels over the total number of the extracted labels, i.e., batch size. We also employed the Hellinger distance [7] to measure the distance between the distribution of the extracted labels and the ground-truth. However, during our experiments, we observed that both aforementioned metrics yielded very similar measurements, therefore, we present only the ASR metric.

Baselines. We compare our attack with two baselines. First, the DLG attack [37], which aims to reconstruct the training data and labels using an optimization approach. For our experiments, we run DLG for 100 iterations and focus only on the label reconstruction results. Second, we consider a uniform distribution-based random guess as a baseline. An adversary without any shared gradients might partially succeed in guessing the existing labels frequency by assuming that the labels distribute uniformly, especially in the case of large balanced batches. The random guess serves as a risk assessment curve. Having any attack performing better than the random guess means that there is information leakage.

6.2   Attack success rate

We consider three different scenarios for the observation capabilities of the adversary (see Section 2.2). Based on these scenarios, the estimation of the impact and offset parameters differs (see Section 5.1), while the same attack applies for all. We refer to the application of our attack under these different scenarios as follows: (1) LLG for the shared gradients scenario, (2) LLG* for the white-box model, and (3) LLG+ for auxiliary knowledge. During our experiments, we observed a very limited difference in the ASR of the LLG attacks for balanced and unbalanced batches, therefore, and because the unbalanced data is closer

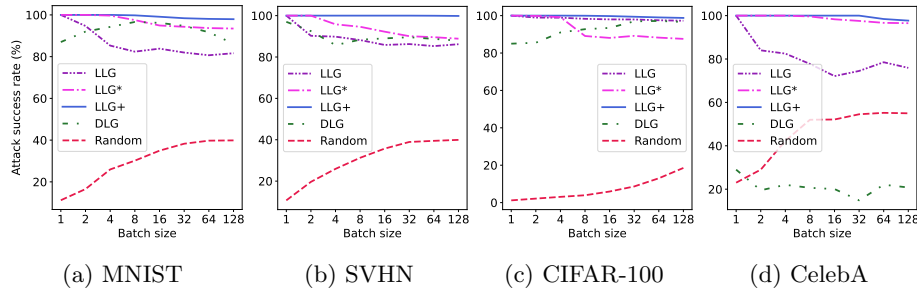(a) MNIST   (b) SVHN   (c) CIFAR-100   (d) CelebA

Fig. 2: Attack success rate of (1) LLG with shared gradients, (2) LLG* with white-box model, (3) LLG+ with auxiliary knowledge, (4) DLG [37], and (5) random guess on MNIST, SVHN, CIFAR-100, and CelebA. Label extraction is based on gradients generated from passing one unbalanced batch to a randomly initialized CNN. DLG runs for 100 iterations. LLG methods outperform the baselines in most of the cases.

to real-world scenarios, we focus on presenting the unbalanced data case, while providing balanced data results in the Appendix, Fig. 6.

Fig. 2 illustrates the ASR scores (y-axis) with batches of different sizes (x-axis). We can see that all LLG variants show some level of ASR degradation with bigger batches. This is expected as the gradients' variance becomes smaller with bigger batches as we can see also in Eq. (10). Thus, small estimation errors in our attack parameters (impact and offset) will notably decrease the success rate.

Overall, LLG+ outperforms all the other LLG variants and DLG. The LLG and LLG* scores range from 100% to a minimum of 77% across the different datasets. Whereas LLG+ remarkably exhibits a high level of stability for various batch sizes and number of classes (in datasets) with $ASR > 98\%$. This mainly reflects the quality of our estimation methods for impact and offset. In contrast, DLG achieves varying accuracy scores, however, no clear behavior can be concluded w.r.t. the changes in the batch sizes. This might be due to the fact that DLG requires a training phase, which is highly sensitive to model initialization, i.e., it might fail to converge for some randomly initialized models or might take different amounts of time for reaching a specific accuracy, unlike LLG, which yields more deterministic results, orders of magnitude faster. For example, the execution time of the experiments illustrated in Fig. 2(a) is as follows: LLG 54s, LLG* 32.2m, LLG+ 14.6m, DLG 17.4h, and Random 50s, using Tesla GPU V100-SXM3-32GB.

The ASR of each LLG attack is similar to some extent on MNIST and SVHN respectively. This can be explained by the fact that both datasets have the same number of classes, i.e. 10. On CIFAR-100 (100 classes), interestingly, we notice that LLG performs quite closely to LLG+ (both have $ASR > 96\%$), as shown in Fig. 2(c), while it drops to around 75% on CelebA (5 classes). We conclude that LLG performs better for datasets with bigger number of classes. LLG*, with its dummy data for the parameter estimation, shows a notable drop on CIFAR-100. It is known that the complexity of CIFAR-100 images is higher than the one

Fig. 3: Attack success rate of LLG+ and DLG on un-balanced batches of different sizes from MNIST with different model architectures: CNN, LeNet, ResNet20. LLG+ achieves around 100% accuracy on CNN and LeNet while its accuracy slightly decreases on ResNet20. DLG achieves > 80% for most batch sizes on CNN, while drops remarkably on LeNet and ResNet20.
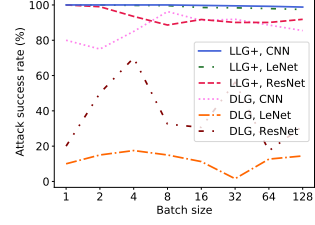
of MNIST and SVHN. Therefore, we can conclude that the complexity of the dataset might influence LLG* in a negative way, while it has no observable effect on LLG and LLG+. For DLG, we notice a remarkable decrease in accuracy on CelebA. This can be due to that fact that the images are of higher dimensions (178x218), unlike the other datasets. Thus, the convergence of the attack is much more difficult.

6.3   Model architecture

Here, we study the influence of the model architecture on our attack and DLG; for that, we consider two models besides our default CNN: (1) LeNet [16], a basic CNN that contains 3 convolutional layers with 2 maximum pooling layers as shown in the Appendix, Table 2. (2) ResNet20 [12], a modern CNN architecture, which introduces the concept of "identity shortcut connection" that skips one or more layers to avoid the problem of vanishing gradients in deep neural architectures. ResNet20 contains 20 layers in total: 9 convolutional layers, 9 batch normalization, and 2 linear layers.

The two main conditions for Property 1 to hold are: (1) using the cross-entropy loss and (2) having a non-negative activation function in the last layer before the output. Thus, we assume the labels extracted in the first step of our attack (see Algorithm 1, Line 1-5) based on this property to be correct regardless of the rest of the model architecture. While the next steps of LLG are based on our impact definition (in Eq. (10)), where the assumption is that a randomly initialized model performs poorly, and this might vary from one model architecture to another.

To run our analysis, we pass one batch from MNIST of varying size and measure the ASR of LLG+ and DLG. As we can see in Fig. 3, our attack performs best on CNN and LeNet, achieving approximately 100% of success rate, while a degradation starts from batches with size > 2 for ResNet20. This is mainly due to the ResNet20 architecture that prevents the vanishing gradients problem in deep neural networks. In other words, it implicitly alters and controls the gradient values in order to not let them vanish (gradients close to zero) or to explode (gradients close to $+\infty$) during training. On the other hand, DLG shows much higher sensitivity towards the model architecture. As we can see it achieves > 80% for most batch sizes on CNN, while drops remarkably on more complex models, LeNet and ResNet20. Such a strong influence of the model architecture

Fig. 4: Influence of model convergence status on ASR of LLG, LLG*, LLG+, and random guess for CNN with unbalanced batches from MNIST dataset. On the left y-axis, the label extraction accuracy is plotted, while on the right y-axis, the model test accuracy can be seen. The number of training iterations ($\times 10^3$) is on the x-axis. All different LLG methods achieve remarkable accuracy even if the models are well-trained and gradients become less informative.
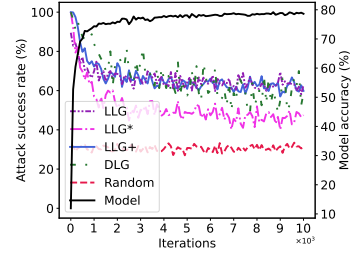
on DLG is expected, as DLG is includes an optimization phase, where optimizing complex models typically requires much more iterations.

6.4   Model convergence status

The gradients guide the model towards a local minimum of the loss function. As the model converges to this minimum, the information included in the gradients becomes less prominent. Therefore, we expect the convergence status of the model to have a strong influence on our attack effectiveness. All the previous experiments are conducted for one inference iteration, i.e. passing one batch to the model and calculate the corresponding gradients. In this section, we go further with training the model and observe the implications on the attack. As the focus in this experiment is on the effect of a specific convergence status on the attack, regardless how this status is achieved, we train the model locally. The CNN model is trained with batches of size 8 for 10^4 iterations. For every iteration, a new set of gradients are generated and attacked by DLG and LLG variants, where the impact and offset are dynamically estimated. Fig. 4 depicts the attack ASR (on the left y-axis) versus the model accuracy at testing time (on the right y-axis), while the x-axis represents the number of training iterations. We can see that the growth of the model accuracy incurs a notable decreasing of the ASR for all LLGs. However, although the model converges close to 80% accuracy, LLG and LLG+ keep achieving ASR > 60%, considerably higher than the random guess which is around 32%. Meaning, our attacks are still able to take advantage of the reduced information in gradients on the course of the whole training process. Similarly, DLG shows degradation in accuracy, yet it remains effective for well-trained models.

6.5   Defense mechanisms

As our attack is mainly based on the gradients, thus, sensitive to changes in their values, obfuscating them can be a direct mitigation mechanism. In this experiment, we apply two obfuscation techniques: noisy gradients and gradients compression, on a randomly initialized CNN model.

Noisy gradients. Many researchers consider adding noise to gradients is the de facto standard for privacy-preserving ML [18]. In the FL scenario, the user can add

(a) Pure noise   (b) Differential privacy   (c) Gradient compression
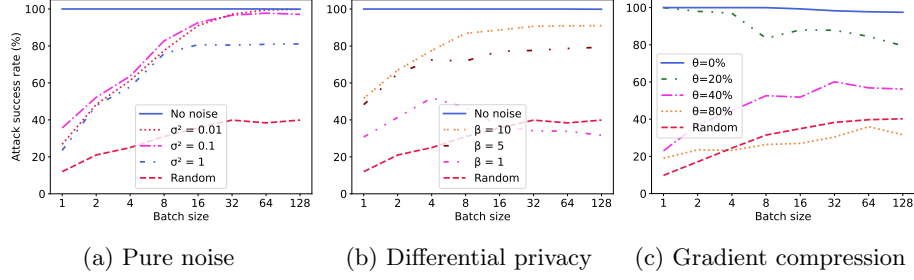
Fig. 5: Effectiveness of different defenses against LLG+ on an unbalanced batch from MNIST with a randomly initialized CNN: (a) defense by adding Gaussian distributed noise to gradients with variances $\sigma^2 \in \{0.01, 0.1, 1\}$, (b) defense by differential privacy with clipping bound $\beta \in \{1, 5, 10\}$, (c) defense by pruning gradients with varying compression ratios $\theta \in \{20\%, 40\%, 80\%\}$. Pure noise is not successful in eliminating the risk completely, since LLG+ maintains higher ASR than the random guess. While, differential privacy mitigates the attack with $\beta = 1$ and batch size $B > 16$. Gradient compression is effective when a high compression ratio ($\geq 80\%$) is used with $B \geq 4$.

the noise before sharing the gradients with the server [22], thus, protecting against external eavesdroppers and curious servers. In this experiment, we evaluate our attack against two techniques of noise addition. (1) Pure noise: we add noise directly to the gradients, similar to [32,37], where no formal privacy properties are discussed. (2) Differential privacy: following differentially private SGD [1], we clip the gradients to bound the global sensitivity, then add noise. The clipping is defined as $\nabla \boldsymbol{W} \leftarrow \nabla \boldsymbol{W} / max\left(1, \frac{\|\nabla \boldsymbol{W}\|_2}{\beta}\right)$, where β is the gradient norm bound.

We use Gaussian noise distribution, which is widely used for differential privacy. For pure noise, the variance of the noise distribution σ^2 ranges in $\{0.01, 0.1, 1\}$ with central 0. For differential privacy, we use variance $\sigma^2 = 0.1$ and varying norm bound $\beta \in \{1, 5, 10\}$, knowing that the privacy budget $\epsilon = \beta/\sigma^2$ considering one training iteration.

In Fig. 5(a), we can see that the higher the magnitude of the noise the less accurate the attack. This is expected as our attack partially uses the magnitude of the gradients to infer the labels following Property 2. Interestingly, we observe that the noise has less effect on the attack when the batch size is increasing. We illustrated earlier in Fig. 1(a, b) that the majority of gradients g_i have values close to zero when they correspond to labels not present in the batch. Adding noise to such small gradient values might lead to flipping their sign, and consequently, disrupting Property 1, which is one of the basis of our attack. For batch sizes $B < n$ number of classes, it is likely that not all labels are present in the batch, so the flipping effect is prominent on the attack success rate. Whereas, in bigger batches, it is more likely to have more various labels, thus, their gradients values are not close to zero. As a result, adding small amount of noise does not lead to sign flipping. Overall, adding noise does not eliminate the risk completely, as we

can see our attack maintains higher ASR than the random guess for all the noise variances.

Fig. 5(b) shows that adding noise of $\sigma^2 = 0.1$ with clipping bound $\beta = 1$ is effective defense against LLG+ for batch sizes > 16, where the ASR drops beyond the random guess. However, in case of $\beta = 1$, more than 45% of the gradients are clipped, which is a considerably high portion of the gradients and might lead to a significant reduction in the model accuracy.

Gradient compression. One of the main motivations for FL is reducing the communication cost by maintaining the user data local and sharing only the gradients. However, some models might contain hundreds of millions of parameters, and sending the gradients of these parameters introduces again a communication overhead. Gradient compression is one proposal to mitigate this issue [19,29], where mainly gradients with small magnitudes are pruned to zero, while further measures are taken to avoid information loss, thus, ensuring to reach the potential model accuracy.

Pruning some gradients to zero reduces the information our attack is exploiting to extract the labels. In this set of experiments, we evaluate LLG+ under various gradient compression ratios $\theta \in \{20\%, 40\%, 80\%\}$, i.e., θ percentage of the gradients to be discarded in each communication round with the server. We use the approach proposed in [19], which sends only the prominent gradients, i.e., with a magnitude larger than a specific threshold. We calculate the threshold dynamically based on the desired compression ratio for each experiment.

Fig. 5(c) illustrates that when the ratio is $\leq 20\%$, there is only a slight effect on the success rate of our attack. When the compression ratio is 80%, we notice that LLG+ becomes completely ineffective for $B \geq 4$, dropping beyond the random guess. Lin et al. [19] has shown that gradients can be compressed with ratio of 99% without losing the model accuracy by applying error compensation techniques. Thus, gradient compression with $\theta > 80\%$ can practically defend against our attack while producing accurate models.

In addition to the aforementioned defenses, cryptography-based approaches exist [5,3,11,35], which can protect gradients from external eavesdroppers and even curious servers. However, besides the computation and communication overhead introduced by these approaches, they prevent the server from evaluating the utility and benignity of users' updates.

7   Conclusion

We identified two properties of gradients of the last layer in deep neural network models trained with cross-entropy loss for a classification task. By exploiting these properties we proposed Label Leakage from Gradients (LLG), a novel attack that extracts the user ground-truth labels from shared gradients. We demonstrated the validity of our attack through mathematical proofs and empirical analysis. Results demonstrate the scalability of LLG to arbitrary batch sizes and number of classes. Moreover, we showed the success rate of our attack on various model architectures and in different stages of training. The effectiveness of noisy gradients and gradient compression as defenses was also investigated.

Acknowledgements

References

1. Abadi, M., McMahan, H.B., Chu, A., Mironov, I., Zhang, L., Goodfellow, I., Talwar, K.: Deep learning with differential privacy. In: Proceedings of the ACM Conference on Computer and Communications Security (2016). https://doi.org/10.1145/2976749.2978318
2. Aono, Y., Hayashi, T., Wang, L., Moriai, S., et al.: Privacy-preserving deep learning: Revisited and enhanced. In: International Conference on Applications and Techniques in Information Security. pp. 100–110. Springer (2017)
3. Aono, Y., Hayashi, T., Wang, L., Moriai, S., et al.: Privacy-preserving deep learning via additively homomorphic encryption. IEEE Transactions on Information Forensics and Security **13**(5), 1333–1345 (2017)
4. Benesty, J., Chen, J., Huang, Y., Cohen, I.: Pearson correlation coefficient. In: Noise reduction in speech processing, pp. 1–4. Springer (2009)
5. Bonawitz, K., Ivanov, V., Kreuter, B., Marcedone, A., McMahan, H.B., Patel, S., Ramage, D., Segal, A., Seth, K.: Practical secure aggregation for federated learning on user-held data. arXiv preprint arXiv:1611.04482 (2016)
6. Bonawitz, K., Ivanov, V., Kreuter, B., Marcedone, A., McMahan, H.B., Patel, S., Ramage, D., Segal, A., Seth, K.: Practical secure aggregation for privacy-preserving machine learning. In: Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security. pp. 1175–1191 (2017)
7. Cramér, H.: Mathematical methods of statistics, vol. 43. Princeton university press (1999)
8. Enthoven, D., Al-Ars, Z.: Fidel: Reconstructing private training samples from weight updates in federated learning. arXiv preprint arXiv:2101.00159 (2021)
9. Geiping, J., Bauermeister, H., Dröge, H., Moeller, M.: Inverting gradients - how easy is it to break privacy in federated learning? In: Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M., Lin, H. (eds.) Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual (2020), `https://proceedings.neurips.cc/paper/2020/hash/c4ede56bbd98819ae6112b20ac6bf145-Abstract.html`
10. Goodfellow, I., Bengio, Y., Courville, A.: Deep Learning. MIT Press (2016), `http://www.deeplearningbook.org`
11. Hao, M., Li, H., Luo, X., Xu, G., Yang, H., Liu, S.: Efficient and privacy-enhanced federated learning for industrial artificial intelligence. IEEE Transactions on Industrial Informatics **16**(10), 6532–6542 (2019)

12. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 770–778 (2016)
13. Jochems, A., Deist, T.M., El Naqa, I., Kessler, M., Mayo, C., Reeves, J., Jolly, S., Matuszak, M., Ten Haken, R., van Soest, J., et al.: Developing and validating a survival prediction model for nsclc patients through distributed learning across 3 countries. International Journal of Radiation Oncology* Biology* Physics **99**(2), 344–352 (2017)
14. Kairouz, P., McMahan, H.B., Avent, B., Bellet, A., Bennis, M., Bhagoji, A.N., Bonawitz, K., Charles, Z., Cormode, G., Cummings, R., et al.: Advances and open problems in federated learning. arXiv preprint arXiv:1912.04977 (2019)
15. Krizhevsky, A., Hinton, G., et al.: Learning multiple layers of features from tiny images. MIT (2009)
16. LeCun, Y., Bottou, L., Bengio, Y., Haffner, P.: Gradient-based learning applied to document recognition. Proceedings of the IEEE **86**(11), 2278–2324 (1998)
17. Li, O., Sun, J., Gao, W., Zhang, H., Yang, X., Xie, J., Wang, C.: Label leakage and protection in two-party split learning. NeurIPS 2020 Workshop on Scalability, Privacy, and Security in Federated Learning (SpicyFL) (2020)
18. Li, Q., Wen, Z., He, B.: Federated learning systems: Vision, hype and reality for data privacy and protection. arXiv preprint arXiv:1907.09693 (2019)
19. Lin, Y., Han, S., Mao, H., Wang, Y., Dally, W.J.: Deep gradient compression: Reducing the communication bandwidth for distributed training. arXiv preprint arXiv:1712.01887 (2017)
20. Liu, D.C., Nocedal, J.: On the limited memory bfgs method for large scale optimization. Mathematical programming **45**(1), 503–528 (1989)
21. Liu, Z., Luo, P., Wang, X., Tang, X.: Deep learning face attributes in the wild. In: Proceedings of the IEEE international conference on computer vision. pp. 3730–3738 (2015)
22. Lu, Y., Huang, X., Dai, Y., Maharjan, S., Zhang, Y.: Differentially private asynchronous federated learning for mobile edge computing in urban informatics. IEEE Transactions on Industrial Informatics **16**(3), 2134–2143 (2019)
23. Luo, X., Wu, Y., Xiao, X., Ooi, B.C.: Feature inference attack on model predictions in vertical federated learning. arXiv preprint arXiv:2010.10152 (2020)
24. McMahan, H.B., Moore, E., Ramage, D., Hampson, S., et al.: Communication-efficient learning of deep networks from decentralized data. arXiv preprint arXiv:1602.05629 (2016)
25. Mo, F., Borovykh, A., Malekzadeh, M., Haddadi, H., Demetriou, S.: Layer-wise characterization of latent information leakage in federated learning. arXiv preprint arXiv:2010.08762 (2020)
26. Netzer, Y., Wang, T., Coates, A., Bissacco, A., Wu, B., Ng, A.Y.: Reading digits in natural images with unsupervised feature learning (2011)
27. Pustozerova, A., Mayer, R.: Information leaks in federated learning. In: Proceedings of the Network and Distributed System Security Symposium (2020)
28. Qian, J., Hansen, L.K.: What can we learn from gradients? arXiv preprint arXiv:2010.15718 (2020)
29. Tsuzuku, Y., Imachi, H., Akiba, T.: Variance-based gradient compression for efficient distributed deep learning. arXiv preprint arXiv:1802.06058 (2018)
30. Wainakh, A., Müßig, T., Grube, T., Mühlhäuser, M.: Label leakage from gradients in distributed machine learning. In: 2021 IEEE 18th Annual Consumer Communications & Networking Conference (CCNC). pp. 1–4. IEEE (2021)

31. Wang, Z., Song, M., Zhang, Z., Song, Y., Wang, Q., Qi, H.: Beyond inferring class representatives: User-level privacy leakage from federated learning. In: IEEE INFOCOM 2019-IEEE Conference on Computer Communications. pp. 2512–2520. IEEE (2019)
32. Wei, W., Liu, L., Loper, M., Chow, K.H., Gursoy, M.E., Truex, S., Wu, Y.: A framework for evaluating client privacy leakages in federated learning. In: European Symposium on Research in Computer Security. pp. 545–566. Springer (2020)
33. Zhang, J., Zhang, J., Chen, J., Yu, S.: Gan enhanced membership inference: A passive local attack in federated learning. In: ICC 2020-2020 IEEE International Conference on Communications (ICC). pp. 1–6. IEEE (2020)
34. Zhao, B., Mopuri, K.R., Bilen, H.: idlg: Improved deep leakage from gradients. arXiv preprint arXiv:2001.02610 (2020)
35. Zhu, H., Li, Z., Cheah, M., Goh, R.S.M.: Privacy-preserving weighted federated learning within oracle-aided mpc framework. arXiv preprint arXiv:2003.07630 (2020)
36. Zhu, J., Blaschko, M.: R-gap: Recursive gradient attack on privacy. arXiv preprint arXiv:2010.07733 (2020)
37. Zhu, L., Liu, Z., Han, S.: Deep leakage from gradients. In: Advances in Neural Information Processing Systems. pp. 14747–14756 (2019)

Appendix

Layer	Size	Activation function
(input)	-	-
Conv 2D	channels x 12	Sigmoid
Conv 2D	12 x 12	Sigmoid
Conv 2D	12 x 12	Sigmoid

Table 1: Architecture of CNN, the default model in the experimental setting.

Layer	Size	Activation function
(input)	-	-
Conv 2D	1 x 6	ReLU
Maxpool	2 x 2	-
Conv 2D	6 x 16	ReLU
Maxpool	2	-
Linear	16 x 6	ReLU
Linear	120 x 84	ReLU
Linear	84 x 10	ReLU

Table 2: Architecture of LeNet network, a very common architecture adopted in computer vision.

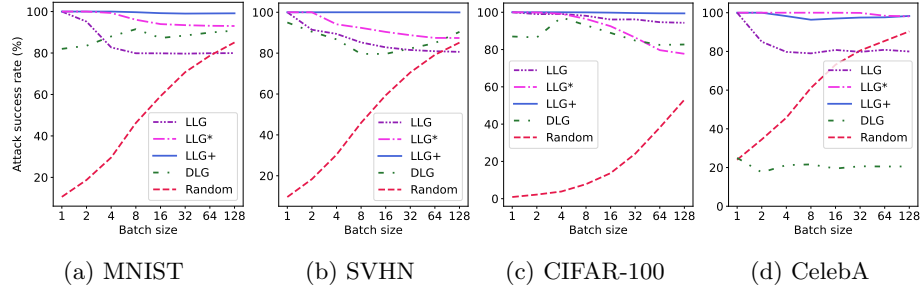(a) MNIST   (b) SVHN   (c) CIFAR-100   (d) CelebA

Fig. 6: Attack success rate of (1) LLG with shared gradients, (2) LLG* with white-box model, (3) LLG+ with auxiliary knowledge, (4) DLG [37], and (5) random guess on MNIST, SVHN, CIFAR-100, and CelebA. Label extraction is based on gradients generated from passing one balanced batch to a randomly initialized CNN. DLG runs for 100 iterations. LLG methods outperform the baselines in most of the cases.