Harmful Bias: A General Label-Leakage Attack on Federated Learning from Bias Gradients

Nadav Gat
Tel Aviv University
Tel Aviv, Israel
nadavgat@mail.tau.ac.il

Mahmood Sharif
Tel Aviv University
Tel Aviv, Israel
mahmoods@tauex.tau.ac.il

Abstract

Federated learning (FL) enables several users to train machine-learning models jointly without explicitly sharing data with one another. This regime is particularly helpful in cases where keeping the data private and secure is essential (e.g., medical records). However, recent work has shown that FL does not guarantee privacy—in classification tasks, the training-data labels, and even the inputs, may be reconstructed from information users share during training.

Using an analytic derivation, our work offers a new label-extraction attack called Label Leakage from Bias Gradients (LLBG). Compared to prior work, ours makes fewer assumptions and applies to a broader range of classical and modern deep learning models, regardless of their non-linear activation functions. Crucially, through experiments with two datasets, nine model architectures, and a wide variety of attack scenarios (e.g., with and without defenses), we found that LLBG outperformed prior attacks in almost all settings explored, pushing the boundaries of label-extraction attacks.

CCS Concepts

• **Security and privacy** → **Privacy-preserving protocols**; • **Computer systems organization** → **Neural networks**.

Keywords

Federated Learning, Privacy Attacks, Gradient Leakage

ACM Reference Format:
Nadav Gat and Mahmood Sharif. 2024. Harmful Bias: A General Label-Leakage Attack on Federated Learning from Bias Gradients. In *Proceedings of the 2024 Workshop on Artificial Intelligence and Security (AISec '24), October 14–18, 2024, Salt Lake City, UT, USA.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3689932.3694768

1 Introduction

Federated learning (FL) [26] has become a popular regime for collaboratively training machine-learning models. In FL, participants can train models on private data without explicitly sharing raw samples. For instance, in the common horizontal FL setting [42], participants compute model updates locally, and only share these updates with a server coordinating the training process, in lieu of sharing actual training records.

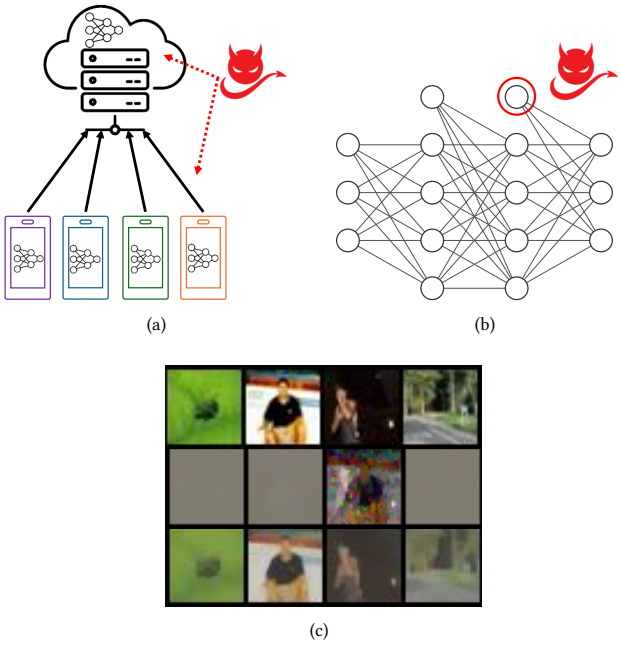

Figure 1: (a) We consider a horizontal FL setup where a passive adversary may observe user updates. (b) Specifically, the adversary only requires the gradients w.r.t. the last layer's bias to launch attacks. (c) Reconstruction attacks leveraging our LLBG attack (bottom) better resemble original images (top) than prior attacks' reconstructions (middle).

As the data itself remains on participants' devices, FL is often employed for training models on sensitive data. For instance, Google makes famous use of FL by collaboratively training models for its smart text-selection feature on user data [15]. Naturally, as users may type sensitive information they may want to avoid sharing with unexpected parties, enhancing the privacy of FL protocols is of utter importance. Consequently, parties interested in deploying FL are increasingly interested in techniques for assessing and improving the extent to which FL preserves privacy (e.g., [16]).

Indeed, although sensitive data is not explicitly shared in FL, several attacks have demonstrated means to reconstruct inputs and extract labels from user updates [12, 37, 43, 46, 51]. For instance, some attacks pose input reconstruction as an optimization problem to estimate users' local private data based on the updates they send and the model weights [12, 43, 51]. These attacks are particularly

This work is licensed under a Creative Commons Attribution-NonCommercial-NoDerivs International 4.0 License.

AISec '24, October 14–18, 2024, Salt Lake City, UT, USA
© 2024 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-1228-9/24/10
https://doi.org/10.1145/3689932.3694768

pernicious as they violate users' implicit assumptions that their data is protected because they only share updates. Other attacks enable adversaries to extract labels [37, 46, 51]. Besides violating users' expectations about what data is being shared, extracted labels may supply attackers with sensitive information about the user. For example, if the training data consists of medical records, label extraction may reveal how many patients included in the dataset have a certain illness. Moreover, label extraction is also crucial for input-reconstruction attacks (e.g., [12, 43]), as such attacks rely on correct label extraction to operate successfully.

In this work, we study label-extraction attacks against FL when used to train deep-learning-based classifiers. The general training setup and threat model can be seen in Fig. 1(a). Based on a mathematical analysis of model gradients, we characterize the connection between user updates and the training-data labels. Equipped with this understanding, we develop the Label Leakage from Bias Gradients (LLBG) attack. Compared to previous attacks derived via related techniques [37], LLBG makes fewer assumptions about the models. Accordingly, LLBG applies in a wider range of settings, against a wider range of classical and modern model architectures.

Moreover, the analysis enabling the attack is less tightly coupled to the model's internal activations during training, leading to more precise estimates of unknown variables and higher attack success rates (ASRs)—i.e., higher fractions of correctly extracted labels. We extensively evaluate LLBG, using two datasets (`CIFAR100` [21] and `ImageNet` [31]) and nine model architectures, and compare it with previous attacks. When tested in variety of settings—with trained and untrained models, against different FL meta-algorithms (i.e., FedAVG and FedSGD), with different activations, among others—LLBG consistently attained the highest ASRs, with a few exceptions.

In a nutshell, our main contributions are:

(1) We analyze models' gradients and derive a simple relation between them and training samples' labels (§4).
(2) Leveraging the analysis, we propose LLBG—a general label-extraction attack, which relies on less assumptions compared to previous work, and therefore is applicable to more settings (§5). Notably, LLBG requires only access to the gradient of the last layer's bias (Fig. 1(b)).
(3) We evaluated LLBG extensively, contrasting it with prior attacks, and found it attains state-of-the-art results in almost all cases explored, and succeeds in new settings where previous attacks fail (§6). We also find correct label extraction crucial for high-quality data reconstruction (§6.7, Fig. 1(c)).
(4) We experimented with different defenses, testing their efficacy against prior attacks and ours, finding LLBG more robust against state-of-the-art defenses (§6.8). However, a simple, new defense we propose succeeds to deter LLBG.

Before delving into the contributions, we next present related work and background (§2) and specify the threat model (§3).

2 Background and Related Work

2.1 Federated Learning (FL)

FL enables various clients to jointly train machine-learning models in a distributed manner, while avoiding the need to share their raw data [26]. In its most common regime, called horizontal FL [26, 40], FL starts with a model of a particular architecture with randomly initialized weights, and proceeds iteratively, performing the following actions in each round:

(1) The server selects a set of clients to participate in the round, and sends them an updated copy of the model;
(2) Selected clients run an optimization algorithm, each using their private data, yielding updates (i.e., gradients) of the model parameters;
(3) The selected clients send the updates to the server; and
(4) The server aggregates the updates received and updates its copy of model.

On top of different choices of optimization algorithm for step 2, two different meta-algorithms are commonly used for the user optimization—FedSGD and FedAVG. In the FedSGD algorithm, the user only performs a single iteration of optimization on the data, while in the FedAVG algorithm several iterations are performed consecutively (possibly on different data in each iteration) before updates are aggregated and sent to the server. In this work, we focus our experiments (§6) on FedAVG, as it is more general (FedSGD is a specific case where the number of local steps is one), communication efficient, and results in more accurate models [26].

Other regimes than horizontal FL exist [40], including, but not limited to, vertical FL, where clients have different feature sets pertaining to the same samples, and federated transfer learning where clients have different features belonging to different samples. Our work targets horizontal FL (referred to as FL from here onwards, for simplicity), as it is more commonly encountered in practice [40] and the literature (e.g., [12, 37, 43, 46, 51]).

2.2 Attacking FL

Attacks with varied goals have been proposed against FL. Some privacy attacks seek to guess properties of client data (e.g., whether certain records are used for training) [27], while others attempt to reconstruct client inputs [12, 43, 51]. In contrast, attacks on FL's integrity aim to prevent or delay the convergence of training (e.g., [3, 4, 13]), or even to instill backdoors in the model that have little-to-no effect on its performance on benign inputs, but lead to predictable behavior (e.g., misclassification) upon the introduction of minimal triggers to inputs during inference (e.g., [2, 3]). Naturally, test-time attacks against machine learning (e.g., [6, 32, 34]) not specifically designed for FL may also be applied against models trained via FL.

Differently from these lines of work, we primarily study label extraction from updates shared by clients [37, 46, 51]. The most related work to ours is that of Wainakh et al. [37]. They introduced the Label Leakage from Gradients (LLG) attack for label reconstruction, surpassing previous attack's performance by a wide margin. LLG is based on a theoretical analysis of model gradients extending that of Zhao et al. [46], and is guaranteed to accurately extract labels when the training batch-size equals one.

In this setting, assuming non-negative activation functions (e.g., ReLU), LLG works by returning the index of the column of model's last linear layer's weight gradients (∇W_L; sent by clients) with negative values. The assumption of non-negative activation is crucial for LLG, as it guarantees that only classes included in the batch have negative values in their respective columns of ∇W_L. However,

this assumption prevents LLG from applying to popular architectures with possibly negative activations (e.g., Vision Transformers [9] with GELUs [18] or Multi-Layer Perceptrons with Tanh or LeakyReLU activations).

LLG was also applied to batch sizes >1. For batches where each class appears at most once, LLG is guaranteed to attain an ASR of 100%. Several input-reconstruction attacks rely on this fact to justify the assumption that attackers have the true labels before initiating input reconstruction [12, 41, 43]. However, in realistic settings, client batches often have several samples of the same label [19, 26]. In such settings, LLG does not attain perfect ASR, leaving significant room for improvement. In this work, we simplify the theoretical analysis employed by LLG and relax its assumptions (§4), leading to a more general attack with higher ASRs (§5–6).

2.3 Defending FL

Various privacy-enhancing methods for FL have been proposed in recent years. Offering an elegant, attack-agnostic privacy definition seeking to limit records' impact on computation, differential privacy (DP) has become a gold-standard definition of privacy [10]. Accordingly, mechanisms have been developed for attaining DP in deep learning and FL [1, 28]. To achieve DP in FL each client clips and introduces noise to updates, before sharing them with servers. Here, clipping consists of normalizing vectors to an L_2-norm of ρ if their norm is higher, while noise is sampled i.i.d. for each coordinate from a Gaussian distribution with mean μ=0 and a standard deviation of σ. As this defense is often found effective (e.g., [8, 50]), we evaluate our attacks against it (§6.8).

Another heuristic-based defense we experiment with is (lossy) gradient compression [30, 49]. Here, coordinates of the updates with absolute values below a certain threshold are replaced with zero, making the updates sparse. This method has also been suggested to make communication in FL more efficient, and has been demonstrated to have little-to-no harm on the trained model's accuracy as long as the threshold is sufficiently large [24].

Previous work [37, 51] has shown that both label- and data-reconstruction attacks in FL are less effective when the batch size of the training data is increased. Therefore, increasing the training batch size (when possible, depending on the amount of data users have) may also be considered as a defense. We also experiment with this defense (§6.3).

Adopting on secure multiparty computation methods, secure aggregation enables the clients to aggregate their updates in a privacy-preserving manner [5]. In return, secure aggregation was deployed to enable computing updates for larger batch sizes (across clients), thus hindering attack success, and to prevent adversaries from tying updates and extracted labels (or reconstructed inputs) to specific clients. Unfortunately, however, secure aggregation may render it more challenging to defend against other categories of attacks, such as ones backdooring models [45].

3 Threat Model

This work studies label-extraction attacks against horizontal FL, where clients jointly train a classifier, aiming to minimize the cross-entropy loss on their data. We assume that the model is a neural network with a linear layer as its last layer. This is a general setting which applies to different domains and data formats (images, text etc.). We assume a realistic, constrained adversary that can only obtain user updates, and not the data or other statistics about it.

Unlike active, malicious adversaries [47, 48], ours cannot manipulate the model weights or deviate from the FL protocol to learn more information about the user data. If the attacker controls a client or the server, they must update models according to user updates, and they cannot independently influence the model weights, deviating from the FL protocol. Said differently, we study an honest-but-curious adversary, which could be an eavesdropper on the communication between clients and the server, or even a legitimate participant in the FL protocol (e.g., the server itself) seeking to learn as much information as possible about other participants [29].

For instance, the adversary could be a company running a standard FL protocol with its users and attempting to learn sensitive information about the data (Fig. 1(a)). The adversary may have access to auxiliary data [32] they could run through the model to assess how updates behave for certain samples and conduct more powerful attacks. We consider this threat in one version of our attack (LLBG_{aux}), but primarily focus on adversaries without auxiliary data.

Our attack is general and does not make assumptions about the data distribution. However, because class imbalance is rather common in FL—in fact, FL is primarily useful because individual clients may not have sufficiently representative data for training well-performing models [26]—we mainly evaluate attacks considering an uneven distribution of classes between clients. Specifically, similar to prior work [26, 37], we assume that the labels of each client data are unbalanced such that half of the client's samples are from one class, a quarter are from another class, and the last quarter are chosen randomly i.i.d. [37]. However, we also demonstrate our attack's effectiveness when samples are chosen uniformly at random across classes (§6.4).

4 Gradient Analysis

We now present the analysis informing LLBG by tying the gradients computed by clients to the samples' labels. We assume a general setting where FL is used to train a neural network for classification, with a linear layer as its last layer, while employing the cross-entropy loss. As opposed to Wainakh et al. [37], we do *not* assume the non-linear activations are non-negative. This assumption is rather restricting when using classical architectures (e.g. Multi-Layered Perceptron with Tanh or LeakyReLU activation), and does not hold for certain modern architectures (e.g. Vision Transformer [9] with GELU [18] activations).

As the loss we analyze is the most commonly used, the following analysis applies to most classification models published and researched. For a model M, a multidimensional input x, and a label index $c \in \{1, ..., n\}$, the cross-entropy loss is defined as:

$$L_M(x, c) = -\ln \frac{\exp(\hat{y}_c)}{\sum_j \exp(\hat{y}_j)}$$

where $\hat{y} = (\hat{y}_1, \hat{y}_2, ..., \hat{y}_n)$ are the model's logits.

Next, we will analyze the model gradients w.r.t. the last layer's bias ∇b_L, denoted as β, demonstrating their connection with the labels. To this end, we first analyze the gradient of the loss w.r.t. the model's output, $\nabla \hat{y}$. Given a batch $X = (x_1, ..., x_B)$ with B samples

and labels $C = (c_1, ..., c_B)$, training typically aims to decrease the average cross-entropy loss:

$$L_M(X, C) = -\frac{1}{B}\sum_{k=1}^{B} \ln \frac{\exp\left(\hat{y}_{c_k}\right)}{\sum_j \exp\left(\hat{y}_{j_k}\right)}$$

where $\hat{y}_{j_k}$ is the j'th logit coordinate when x_k is given as input.

We denote the gradient of the loss w.r.t. a single output coordinate $\hat{y}_i$ by d_i:

$$d_i := \frac{\partial L_M(X, C)}{\partial \hat{y}_i} =$$

$$-\frac{1}{B}\sum_{k=1}^{B}\left(\frac{\partial \ln \exp\left(\hat{y}_{c_k}\right)}{\partial \hat{y}_i} - \frac{\partial \ln \sum_j \exp\left(\hat{y}_{j_k}\right)}{\partial \hat{y}_i}\right)$$

$$= -\frac{1}{B}\sum_{k=1}^{B}\left(\mathbb{I}\{i = c_k\} - \frac{\exp\left(\hat{y}_{i_k}\right)}{\sum_j \exp\left(\hat{y}_{j_k}\right)}\right)$$

where $\mathbb{I}\{\alpha = \beta\} = 1$ if $\alpha = \beta$, and $= 0$, otherwise. Opening parentheses, we get:

$$d_i = -\frac{1}{B}\sum_{k=1}^{B}\mathbb{I}\{i = c_k\} + \frac{1}{B}\sum_{k=1}^{B}\frac{\exp\left(\hat{y}_{i_k}\right)}{\sum_j \exp\left(\hat{y}_{j_k}\right)}$$

$$= -\frac{\lambda_i}{B} + \frac{1}{B}\sum_{k=1}^{B}\frac{\exp\left(\hat{y}_{i_k}\right)}{\sum_j \exp\left(\hat{y}_{j_k}\right)} \tag{1}$$

where λ_i is the number of samples in the training batch with label i. Notice that the average probability assigned to class i (the second term, on the right hand side of (1)) lays $\in [0, 1]$. When $i \notin C$, meaning no sample in the training batch has label i, we have $\lambda_i = 0$ and thus $d_i \in [0, 1]$. On the other hand, if $i \in C$, we have:

$$-\frac{\lambda_i}{B} \leq d_i \leq 1 - \frac{\lambda_i}{B}$$

Hence, if the value of d_i is negative, we can conclude with certainty that $i \in C$. Notice, however, that the opposite is not true: if $i \in C$ it is not guaranteed that $d_i < 0$, since the second term (on the right hand side) of (1) may be larger in absolute value than the first.

Although the gradient d w.r.t. the outputs $\hat{y}$ is not usually shared explicitly in FL, the gradients of the weights and biases of the linear layers of the model are. We now analyze the gradient of the bias of the last linear layer, denoted by β. From the chain rule we have:

$$\beta_i := \nabla b_L^i = \frac{\partial L\left(X, C\right)}{\partial b_L^i} = \frac{\partial L\left(X, C\right)}{\partial \hat{y}_i} \cdot \frac{\partial \hat{y}_i}{\partial b_L^i}$$

$$= d_i \cdot \frac{\partial\left(\left(W_L^i\right)^T a_{L-1} + b_L^i\right)}{\partial b_L^i} = d_i \cdot 1 = d_i \tag{2}$$

where W_L^i, b_L^i are the weights connected to the i^{th} output (which represents the i^{th} class) and the corresponding bias term, a_{L-1} is the output of the second-to-last layer and $\hat{y}_i = {W_L^i}^T a_{L-1} + b_L^i$ from the definition of a linear layer. Combining (1) with (2) we get:

$$\beta_i = d_i = -\frac{\lambda_i}{B} + \frac{1}{B}\sum_{k=1}^{B}\frac{\exp\left(\hat{y}_{i_k}\right)}{\sum_j \exp\left(\hat{y}_{j_k}\right)} \tag{3}$$

Untrained Models As mentioned by Wainakh et al. [37], the magnitude of the second term in Eqn. 3—the average probability assigned to class i by the model over the batch—depends on the model's confidence in the prediction, which is usually dependent on the training stage. For randomly initialized and untrained models, the output probability distribution should be close to uniform, and in that case we have:

$$\frac{1}{B}\sum_{k=1}^{B}\frac{\exp\left(\hat{y}_{i_k}\right)}{\sum_j \exp\left(\hat{y}_{j_k}\right)} \approx \frac{1}{B}\sum_{k=1}^{B}\frac{1}{n} = \frac{1}{n}$$

For large values of n (e.g., 100 in `CIFAR100` [21] or 1,000 in `ImageNet` [31], respectively), this term is close to 0 and we have:

$$\beta_i \approx -\frac{\lambda_i}{B}$$

Trained Models For a trained model, we can usually assume that its accuracy is rather high, meaning for most samples (x, c) in the dataset:

$$\forall w \neq c. \frac{\exp \hat{y}_c}{\sum_j \exp \hat{y}_j} > \frac{\exp \hat{y}_w}{\sum_j \exp \hat{y}_j}$$

Applying this to the second term of (3), we assume that the expressions within the sum that contribute most are those corresponding to batch indices where the label of the sample is i (the coordinate of the gradient analyzed in that equation). Therefore:

$$\beta_i \approx -\frac{\lambda_i}{B} + \frac{1}{B}\sum_{k;c_k=i}\frac{\exp\left(\hat{y}_{i_k}\right)}{\sum_j \exp\left(\hat{y}_{j_k}\right)}$$

Examining the sum on the right we get:

$$\sum_{k;c_k=i}\frac{\exp\left(\hat{y}_{i_k}\right)}{\sum_j \exp\left(\hat{y}_{j_k}\right)} = \sum_{k;c_k=i} p_i^{(k)}$$

where $p_i^{(k)}$ is the estimated probability of class i given sample x_k as input. Adversaries do not know $p_i^{(k)}$, as it is not shared by clients during training. However, they may approximate it as the average confidence level of the model when correctly predicting class i. Denoting this average as v_i, and using the fact that there are exactly λ_i samples in the batch with label i, we can approximate:

$$\beta_i \approx -\frac{\lambda_i}{B} + \frac{1}{B}\sum_{k;c_k=i} v_i = -\frac{\lambda_i}{B} + \frac{1}{B}\left(\lambda_i \cdot v_i\right)$$

$$= -\frac{\lambda_i}{B} \cdot (1 - v_i) \tag{4}$$

We offer means to estimate the vector v in §5.2 to enable attacks against trained models. This approximation is also valid for untrained models, however in this case $v_i \approx \frac{1}{n}$ and so its contribution to this equation is negligible when n is large.

Comparison With Prior Work In past work, Wainakh et al. [37] proposed an analysis similar to ours, on the gradient of the weights of the last linear layer (instead of the gradient of the bias). They derived an equation resembling Eqn. 2 that relates weight gradients to the gradient of the output:

$$\nabla W_L^i = d_i \cdot a_{L-1} \tag{5}$$

where ∇W_L^i is the i'th column of the gradient of the last layer's weights, a_{L-1} is the output of the previous layer, and d is the gradient of the output (as before). They suggested that, assuming non-negative activations (i.e., if a_{L-1} only has non-negative values), negative weight gradients correspond to labels used at least once in updates. Accordingly, in Wainakh et al.'s analysis, the relation

between the weight gradients and the training labels has another multiplicative factor, which may be distributed differently across different models, potentially obfuscating it compared to our analysis. Furthermore, if the activation function is not non-negative, the sign of the weights gradient ∇W_L may be different than d_i, thus hindering label extraction. In contrast, our analysis enables our attack to be deployed successfully against models with different activation functions at the last layer, including ones which could be negative (e.g., LeakyReLU, Tanh, GELU).

FedSGD vs. FedAVG It is important to note that our analysis applies for cases where clients only take a single local step and send the gradients to the server (i.e. FedSGD). In cases where clients take several local steps, updating their local model after each (i.e. FedAVG), the updates shared are the difference between weights before and after running all local steps. In such cases, our analysis does not strictly hold. Still, even in these settings we find that the analysis provides a good approximation for gradient behavior when the learning rate is sufficiently small, as the model does not drastically change between steps. This is also evident from the high ASRs attained against FedAVG (§6).

5 Technical Approach

Using the observations from §4, we now define the LLBG attack. The attack has two variants, applicable for models at different stages of training. We first describe our attack against a randomly initialized, untrained model (§5.1). Subsequently, we present an attack well-suited for trained models that achieve high accuracy on the classification task (§5.2).

5.1 Untrained Models

In this scenario we assume the only information the attacker receives is the update of the model parameters θ w.r.t. a training batch, $\nabla\theta$. In fact, the only part of the update needed for our attack is the gradient of the last linear layer's bias (see Fig. 1(b)), ∇b_L, which we denote as β. From the analysis in §4 we know that every negative coordinate in the vector β indicates the matching label exists at least once in the batch. We also know that in this case the contribution of sample x with label c in a batch to β is $\approx -\frac{1}{B} \cdot e_c$, where e_i is the one-hot vector with 1 in its i'th coordinate and 0 in the rest. We refer to this quantity as impact, and denote it by m. From these observations we derive a two-stage attack (see Alg. 1).

In the first stage of the attack, labels guaranteed to be in the training batch (labels corresponding to negative indices in the gradient) are added to the reconstruction. After label i is added to the reconstruction, as preparation for the second stage, its contribution to the gradient is corrected, by reducing the i^{th} coordinate of the bias gradient by the impact m.

In the second stage, we rely on the corrections done in the first stage being close to accurate, and add the label corresponding to the negative index of the bias gradient with the highest absolute value (or the positive with smallest value if one does not exist). In this stage we correct for each reconstructed label by reducing m from the corresponding coordinate of the bias gradient as well. The second stage is run repeatedly until B samples are reconstructed.

Algorithm 1: Label Leakage from Bias Gradient (LLBG) attack against untrained models

Input: $\beta = \nabla b_L$, batch size B

1 $m \leftarrow -\frac{1}{B}$;
2 $C' \leftarrow [\,]$; // Initialize labels list
3 **for** $i \leftarrow 1$ **to** n **do** // Guaranteed labels
4 **if** $\beta_i < 0$ **then**
5 $C' \leftarrow C' + [i]$;
6 $\beta_i \leftarrow \beta_i - m$; // Sample impact
7 **while** $|C'| < B$ **do** // Heuristic for rest
8 $l \leftarrow \arg\min\{\beta\}$;
9 $C' \leftarrow C' + [l]$;
10 $\beta_l \leftarrow \beta_l - m$;
11 **return** C'

5.2 Trained Models

The analysis in §4 showed that the contribution of each sample (x, c) to β_c is $\approx -\frac{1}{B} \cdot (1 - v_c)$ (Eqn. 4), where v_c is the average probability given by the model when correctly predicting a sample is of class c. Accordingly, while Alg. 1 can be applied against trained models, the approximation of $v_c \approx \frac{1}{n} \approx 0$ it relies on would be invalid. Thus, to improve ASRs against trained models, it would be helpful to accurately approximate v_c.

In some cases, v may be publicly available, alongside models' architectures and weights. For instance, if confidence calibration is applied to the model [14]—i.e., confidence is adjusted such that estimated probability also corresponds to accuracy—v could be derived from any public information about model accuracy.

If v is not publicly available, adversaries would need to find alternative ways to approximate it. We offer two approaches differing in their assumptions concerning the attacker's knowledge for doing so. The first, used in an attack variant we denote by LLBG_{aux}, assumes the availability of auxiliary data. The adversary passes the data through the model, averaging the output probabilities across samples, to estimate the average confidence v_i for each class.

In the second approach, used in attack variant denoted by LLBG_γ, v is set to a constant γ for all classes. While we use this approach as part of an ablation to measure the utility of approximating v, it may be used by an attacker as it only requires an estimate of the overall model confidence—a single statistic instead of n statistics (one per each class in the dataset).

Given the estimate of v for different classes, we derive the attack against trained models (see Alg. 2). The difference from the attack outlined in Alg. 1 is in Lines 6 and 10, where we take the model confidence into account when we calculate the contribution of each sample to the gradient.

6 Evaluation

We extensively evaluated the effectiveness of LLBG and compared it to prior attacks in different settings, including ones previously explored (e.g., [37]), and new ones made possible by our analysis. We started by considering a basic setting, with several untrained models of varied complexities (§6.2); we then explored the effects of

Algorithm 2: LLBG attack against Trained Models

Input: $\beta = \nabla b_L$, batch size B, average confidence of model per class $v = (v_1, \ldots, v_n)$

$m \leftarrow -\frac{1}{B}$;
$C' \leftarrow [\,]$; // Initialize labels list
for $i \leftarrow 1$ **to** n **do** // Guaranteed labels
  if $\beta_i < 0$ **then**
    $C' \leftarrow C' + [i]$;
    $\beta_i \leftarrow \beta_i - m \cdot (1 - v_i)$; // Sample impact
while $|C'| < B$ **do** // Heuristic for rest
  $l \leftarrow \arg\min\{\beta\}$;
  $C' \leftarrow C' + [l]$;
  $\beta_l \leftarrow \beta_l - m \cdot (1 - v_l)$
return C'

increasing batch size (§6.3); we also considered different label distributions (§6.4); we then tested attacks with trained models (§6.5); we further assessed attacks with different non-linear activation functions, including ones with negative activations (§6.6); we also measured the extent to which more accurate label reconstruction improves existing input-reconstruction attacks (§6.7); lastly, we tested different defenses to mitigate the attacks (§6.8). We provide the code for our attack and all experiments reported in a GitHub repository [11].

6.1 Experimental Setup

Data and Models We evaluated attacks with the `CIFAR100` [21] and `ImageNet` [31] datasets, two standard image datasets commonly used in the field [12, 37, 46, 51], containing 100 and 1,000 classes, respectively. We also attacked models of varying complexities: For `CIFAR100`, we used a small `MLP` with three hidden layers and ReLU activation, a small `CNN` with four convolutional layers and ReLU activation, `VGG19` [33] and `ResNet32` [20], and, for `ImageNet`, we used `VGG19`, `ResNet50` [17], $\texttt{EfficientNet}_{\texttt{B0}}$ [36], `MNasNet-A1` [35], `ShuffleNetV2` [25], and `ViT` [9].

Meta-Algorithms and Data Distribution In all experiments, we report results with FedAVG as the FL meta-optimization method, as it is more commonly used due to its performance and communication efficiency. We exclude FedSGD results for brevity, as they were consistent with FedAVG's. Unless otherwise mentioned, we considered unbalanced client batches, as defined by Wainakh et el. [37] and explained in §3. Namely, $\frac{1}{2}$ of the labels in each batch wre of a random class a, another $\frac{1}{4}$ were of a random class b, and the rest were each chosen at random, i.i.d. from all classes. The batch size, unless otherwise reported, was B=128.

Metrics We assessed attacks via attack success rates (ASRs)—i.e., the percentage of labels correctly extracted from a single batch by attacks. To reliably measure ASRs, we repeated each experiment 100 times and reported the average ± the standard deviation of the ASR across runs. For each attack we considered, we ran experiments with the exact same 100 batches. When presenting results, we mark the best result for each experiment in **bold**.

Baseline Attack The baseline for our attack is LLG, the state-of-the-art attack we aim to improve upon [37]. We tested the LLG variant which assumes no auxiliary dataset, similarly to our LLBG attack. As aforementioned, LLG bears several similarities to LLBG, differing in two primary ways. First, instead of approximating β to extract labels, LLG leverages

$$G = \left(\vec{1}\right)^T \cdot \nabla W_L$$

Where $\vec{1}$ is a vector of length n with 1 in every coordinate, and ∇W_L are the gradients of the last linear layer's weights. The summation of each column of the weights gradient results in a single value for each label index. Assuming non-negative activations, if a label is present in the batch, the entire corresponding column would be negative, resulting in a negative sum, and enabling label extraction.

Second, instead of setting $m = -\frac{1}{B}$ (as determined by our closed-form analysis), LLG's impact parameter is estimated empirically:

$$m = \frac{1}{B} \sum_{i;G_i<0} G_i \left(1 + \frac{1}{n}\right)$$

Namely, m is approximated by summing all negative indices and dividing by the batch size. The multiplicative factor is meant to correct for positive terms added by other labels, and was empirically found to boost LLG's success.

Ablation Attack To evaluate the contribution of each of the differences between out attack and the baseline, we define an attack called Empirical Bias Impact (EBI) that, similarly to LLBG, exploits bias gradients to extract labels, but more closely resembles LLG by estimating the impact empirically. Particularly, EBI runs according to Alg. 1, but uses the following term to estimate the impact:

$$m = \frac{1}{B} \sum_{i;\beta_i<0} \beta_i \tag{6}$$

6.2 Attacking Untrained Models

In the most basic setting, we tested label reconstruction on untrained `CIFAR100` models; Tab. 1 reports the results. In almost all cases, LLBG achieved near-perfect ASR, bypassing the baselines' ASRs by ∼20%. An exception was `ResNet32`, for which all attacks attained lower ASRs than against other models, and LLBG's ASR fell >15% behind the baselines.

We discovered the reason for LLBG's failure against `ResNet32` can be explained by the high probability assigned to (incorrect) classes, even when untrained (∼0.89 entropy for `ResNet32` compared to ∼6.64 for other architectures), violating LLBG's assumption that untrained models assign roughly uniform probability to all labels (§4). Similar findings held when attacking an untrained `ResNet50` model on `ImageNet` (Tab. 2). In this case, however, the difference in ASR between attacks was smaller for most models. In contrast, for $\texttt{EfficientNet}_{\texttt{B0}}$, LLG achieved an extremely low ASR, compared to the near-perfect success of LLBG and EBI.

6.3 Varying Batch Size (B)

We also examined how varying the batch size (B) affected ASRs. Here, we experimented on `CIFAR100` with the `VGG19` architecture, considering $B \in \{256, 512, 1024\}$, in addition to B=128 previously reported in Tab. 1. The results are reported in Tab. 3. LLBG's ASRs

Table 1: ASRs vs. untrained models on `CIFAR100`.

Model	LLG	EBI	LLBG
MLP	81.93 ± 1.94	79.11 ± 1.38	**99.56 ± 0.39**
CNN	81.24 ± 1.91	78.93 ± 1.40	**99.58 ± 0.39**
VGG19	81.57 ± 1.78	78.96 ± 1.38	**99.62 ± 0.39**
ResNet32	**56.32 ± 8.24**	50.40 ± 9.90	33.01 ± 3.79

Table 2: ASRs vs. untrained models on `ImageNet`.

Model	LLG	EBI	LLBG
VGG19	97.92 ± 1.22	**99.16 ± 0.59**	**99.86 ± 0.30**
ResNet50	**62.90 ± 5.86**	55.88 ± 7.45	44.91 ± 4.08
$\texttt{EfficientNet}_{\texttt{B0}}$	16.45 ± 3.26	**99.17 ± 0.59**	**99.86 ± 0.30**
MNasNet-A1	91.11 ± 4.07	**99.17 ± 0.59**	**99.86 ± 0.30**
ShuffleNetV2	91.02 ± 5.11	**99.16 ± 0.61**	**99.86 ± 0.30**

Table 3: ASRs vs. untrained `VGG19` on `CIFAR100`, with varied batch sizes.

Batch Size	LLG	EBI	LLBG
128	81.57 ± 1.78	78.96 ± 1.38	**99.62 ± 0.39**
256	69.03 ± 1.61	68.00 ± 1.11	**99.36 ± 0.24**
512	49.07 ± 2.25	47.15 ± 1.65	**99.55 ± 0.13**
1024	39.25 ± 2.79	37.35 ± 2.27	**99.97 ± 0.04**

Table 4: ASRs vs. untrained models on `CIFAR100` with uniform labels.

Model	LLG	EBI	LLBG
MLP	74.75 ± 3.25	80.54 ± 3.00	**100.00 ± 0.00**
CNN	76.56 ± 3.02	80.72 ± 3.43	**100.00 ± 0.00**
VGG19	75.37 ± 3.12	79.94 ± 3.74	**100.00 ± 0.00**
ResNet32	**71.33 ± 3.29**	67.39 ± 3.81	66.77 ± 3.53

remained almost constant (even rising slightly) when increasing B, while other attacks' ASRs dropped significantly. Said differently, the results suggest LLBG were less effected by the batch size increase compared to the other attacks.

6.4 Attacks With Random Label Distribution

We also tested attacks with randomly (i.i.d.) distributed labels, instead of unbalanced distributions, simulating settings where (some) clients may have large amounts of representative private data. Here too, we considered untrained models on the `CIFAR100` dataset. This time, however, we set B to 100 to minimize multiple occurrences of the same label. Results are reported in Tab. 4. In this setting, LLBG achieved perfect ASRs, surpassing other attacks by 20%-25%, except for the untrained `ResNet32`, which violates LLBG's assumptions.

Table 5: ASRs vs. trained `ImageNet` models.

Model	LLG	EBI	LLBG_{aux}
VGG19	80.90 ± 7.64	80.31 ± 7.97	**85.66 ± 4.33**
ResNet50	79.02 ± 9.36	78.57 ± 9.33	**82.74 ± 4.16**
$\texttt{EfficientNet}_{\texttt{B0}}$	74.85 ± 10.04	87.14 ± 6.69	**93.52 ± 2.15**
ShuffleNetV2	**79.10 ± 8.28**	76.89 ± 7.92	**78.21 ± 7.29**
MNasNet-A1	80.37 ± 8.73	78.95 ± 9.06	**82.13 ± 7.86**

Table 6: LLBG_γ ASRs vs. trained `ImageNet` models.

Model	$\gamma = 0.5$	$\gamma = 0.7$	$\gamma = 0.9$
VGG19	71.34 ± 10.88	**78.87 ± 8.73**	70.74 ± 11.15
ResNet50	64.05 ± 12.65	**75.11 ± 12.51**	73.90 ± 10.94
$\texttt{EfficientNet}_{\texttt{B0}}$	78.15 ± 9.91	**85.51 ± 8.40**	77.00 ± 10.42
ShuffleNetV2	65.74 ± 11.89	**75.29 ± 9.94**	71.02 ± 12.11
MNasNet-A1	68.64 ± 11.57	**77.77 ± 10.47**	72.09 ± 10.57

6.5 Attacking Trained Models

As discussed by Wainakh et al. [37] and in §4, trained models' gradients are distributed differently compared to untrained models, rendering data and label reconstruction more challenging. We did not apply LLBG in this setting, since one of its assumptions (namely, the model's output distribution being near uniform) does not hold for trained models. Instead, we experimented with LLBG_{aux} and LLBG_γ, comparing them to LLG and EBI.

To approximate v for the LLBG_{aux} attack, we ran, for each class, a batch of 10 samples pertaining to the class through the model, applied the softmax function to the output, and averaged the probability estimates assigned to said class. We ran the experiments on `ImageNet`, with pre-trained weights from the official PyTorch implementation [7]. Tab. 5 reports the results. The LLBG_{aux} attack had the highest ASRs in almost all cases (with the exception of `ShuffleNetV2`, where its ASR was comparable to LLG's), attaining the most significant advantage against LLG when attacking the $\texttt{EfficientNet}_{\texttt{B0}}$ model (~19% higher ASR). We also note that LLBG_{aux} performed better against the trained `ResNet50` model than LLBG's performance against the untrained counterpart. We believe this can be explained by the irregular behavior of the untrained model (§6.2), which LLBG_{aux} accounts for.

Tab. 6 shows LLBG_γ's ASRs against trained `ImageNet` models with different values of γ. Interestingly, LLBG_γ achieved high ASRs, scoring >10% higher than LLG for $\texttt{EfficientNet}_{\texttt{B0}}$, and lagging 1–3% ASRs behind LLG and EBI for the other models. The best results, across models, were achieved with γ=0.7, suggesting that the average confidence level of all models were similar. Indeed, we were able to verify this empirically by calculating the average $\bar{v}^{(M)}$ of the empirical model confidence vector v (used for LLBG_{aux}) for each model M:

$$\bar{v}^{(M)} = \frac{1}{n}\sum_{i=1}^{n} v_i$$

These values are reported in Tab. 7.

Table 7: Average model confidence $\bar{v}$ calculated for trained `ImageNet` models.

Model	$\bar{v}$
VGG19	0.6269
ResNet50	0.6940
$\text{EfficientNet}_{\text{B0}}$	0.6324
ShuffleNetV2	0.6363
MNasNet-A1	0.6735

Table 8: ASRs vs. untrained `MLP`s on `CIFAR100`, with varied activations.

Activation	LLG	EBI	LLBG
ReLU	81.93 ± 1.94	79.11 ± 1.38	**99.56 ± 0.39**
LeakyReLU	81.95 ± 1.95	79.11 ± 1.38	**99.56 ± 0.39**
Sigmoid	82.88 ± 1.61	82.80 ± 1.62	**97.62 ± 0.94**
Tanh	36.72 ± 21.10	79.16 ± 1.34	**99.48 ± 0.40**

6.6 Different Activation Functions

As discussed in §4, unlike LLG's, our analysis does not assume non-negative activation functions, rendering LLBG more general. To empirically asses the effect of this fact, we ran attacks against a small `MLP` with several different activation functions: ReLU, Sigmoid, LeakyReLU with α=0.01, and Tanh (the latter two emitting possibly negative activations). Results are reported in Tab. 8. Sigmoid and LeakyReLU had little impact on ASRs compared to ReLU activations. However, the Tanh activation, emitting outputs $\in [-1, 1]$, led to a major drop (>50%) in LLG's ASR. In comparison, LLBG's ASR remained high and roughly unchanged across all activations.

We also tested attacks against the popular Vision Transformer (`ViT`) architecture [9], as it contains non-linear Gaussian Error Linear Unit (GELU) [18] activations, emitting both negative and positive activations. We ran the experiment with `ImageNet` dataset and B=64 (due to memory constraints), considering both trained and untrained models. For trained models, we approximated v for LLBG_{aux} in the same manner as in §6.5. We also ran LLBG_{γ} with $\gamma \in \{0.5, 0.7, 0.9\}$. Tab. 9 reports the results.

Note that since the activation function is not non-negative and there may be more negative columns in the weights gradient than samples in the batch, the LLG attack was slightly modified, adding the negative columns with highest sum (in absolute value) first, in order to ensure that at most B labels are reconstructed.

In both the untrained and trained cases, LLG's ASRs dropped to ~0%. In the untrained case, both EBI and LLBG achieved high ASRs with <1% difference. In the trained case, LLBG_{aux} had the highest ASRs with a margin of >5% , with EBI still having a high ASR.

For the LLBG_{γ} the attack again has the most success with $\gamma = 0.7$, although in this case the gap between this attack and the best (LLBG_{aux}) is more significant. We calculate the average confidence level of the model $\bar{v}$, similarly to §6.5, and find that for `ViT`, $\bar{v} =$ 0.6892, explaining why setting γ=0.7 leads to successful attacks.

Overall, these results demonstrate LLBG's chief advantage: by exploiting bias gradients (∇b_L) instead of weight gradients (∇W_L) to extract labels, LLBG makes fewer assumptions about activations than LLG, leading to a more general attack with higher ASRs against a wide range of models with varied activation functions.

Table 9: ASRs against untrained and trained `ViT` models on `ImageNet`, with a batch size of 128.

Attack	γ	Untrained	Trained
LLG	N/A	2.19 ± 1.89	0.34 ± 0.72
EBI	N/A	**95.36 ± 1.88**	84.41 ± 8.77
LLBG	N/A	**94.90 ± 0.69**	N/A
LLBG_{aux}	N/A	N/A	**90.20 ± 6.00**
LLBG_{γ}	0.5	N/A	70.25 ± 12.87
LLBG_{γ}	0.7	N/A	81.22 ± 11.32
LLBG_{γ}	0.9	N/A	76.58 ± 10.54

6.7 Impact of Label Extraction on Input Reconstruction

Although our main focus in this work is exploring and enhancing label extraction, we also wanted to test whether increasing ASRs in this task impact the performance of input-reconstruction attacks. Input-reconstruction attacks often assume accurate knowledge of labels. However, in reality, adversaries need to infer labels (potentially, with some errors) to reconstruct inputs. Next, we show that more accurate label extraction, as attained by LLBG, results in markedly better input-reconstruction success. In our experiments, we ran input reconstruction with a small batch size, substantially smaller than the ones considered in §6 because (1) input-reconstruction quality deteriorates for large batch sizes; and (2) running input reconstruction on large batches is computationally expensive.

In particular, we experimented with a `MLP` with Tanh activations, training on `CIFAR100` with batch size B=4 and an unbalanced partition of the data, meaning in this case each batch has two samples with the same label, and the other two with random labels. We used Geiping et al.'s Inverting Gradients attack [12] to reconstruct input, with the default hyperparameters.[1]

At the end of input reconstruction, we measured several common image-similarity metrics to assess how similar are reconstructed inputs to the original data. Specifically, we measured Mean Squared Error (MSE); Peak Signal to Noise Ratio (PSNR), which measures the ratio between the maximum value of the signal (the image) and the noise; Learned Perceptual Image Patch Similarity (LPIPS, [44]), which compares the embeddings of a pre-trained model (namely, `AlexNet` [22]) corresponding to the original and reconstructed samples; and Structural Similarity Index (SSIM, [38]) which estimates the perceptual differences between two images via the mean and variance of different patches in the image.

Tab. 10 reports the results, averaged over 100 repetitions of each experiment condition. Even for this relatively small batch size, LLBG had a significant advantage over LLG when attacking a model for which the latter's assumptions do not hold, i.e. the last activation is not non-negative. While LLBG has a perfect ASR for every experiment, LLG reconstructed $< \frac{1}{2}$ of the labels correctly, on average,

[1]Using the Adam optimizer with a learning rate of 0.1, β_1=0.9, and β_2=0.99 for 24,000 steps.

Table 10: Comparing the impact of label extraction using LLBG vs. LLG on input reconstruction. Experiments were on the CIFAR100 dataset, using an untrained MLP with Tanh activations, with a batch size 4. Arrows pointing to the top (resp. bottom) indicate that higher (resp. lower) metric values are better. The ASR column reports label-extraction success, while the columns to the right report input-reconstruction metrics.

Attack	ASR ↑	MSE ↓	PSNR ↑	LPIPS ↓	SSIM ↑
LLG	44.89 ±28.29	0.042 ±0.024	16.14 ±3.32	0.302 ±0.132	0.389 ±0.181
LLBG	**100.00 ±0.00**	**0.014 ±0.006**	**19.317 ±1.881**	**0.060 ±0.016**	**0.711 ±0.036**

with a large standard deviation. In a few cases the LLG attack succeeded in extracting all labels correctly, but there were many cases where the reconstruction was completely wrong and cases where only 1 or 2 labels (out of 4) were reconstructed correctly.

Quantitatively, the data reconstruction using correct batch labels (reconstructed using the LLBG attack) had much higher quality (i.e. higher fidelity to the original images) compared to to using only some correct labels (reconstructed using the LLG attack). This is evident from the significant difference between the two rows of Tab. 10 in each of the data-reconstruction metrics, with the attack relying on LLBG achieving better values across the board. Qualitatively, the images reconstructed while relying on the LLBG attack appeared similar to the original batch for most experiments (a randomly chosen example is shown in Fig. 1(c)). In contrast, the attack relying on LLG usually produced a medium fidelity reconstruction of the samples for which the correct label was reconstructed (although noisier compared to the other attack), while generating complete noise or repeating the medium fidelity reconstructions to some degree. All in all, these results emphasize the importance of correct label extraction for data reconstruction.

6.8 Defenses

We now experiment with different defenses presented in §2.3.

Local DP We started with local DP, implemented by applying clipping and introducing noise to the user's gradients. When applying DP in a training process, a privacy budget ε needs to be set in advance, and the noise added in each step of the training process is calculated accordingly. Since in this case we do not run a full training process and only compute gradients for a single batch, we simply run with different noise and clipping levels, covering a wide range of options. We first experimented with different clipping values ρ, setting the Gaussian noise's standard deviation to σ=0.1. We ran the attack against an untrained VGG19 model with the CIFAR100 dataset and a batch size of 128. Tab. 11 reports the results, including the trivial case where $\rho = \infty$ (meaning no clipping is done). To assess the effect of DP on the model's utility after training, we trained an instance of the VGG19 model for each setting reported while applying the noise and clipping of the setting during training, and report the test accuracy of the final model in Tab. 11 as well. These results are reported in Tab. 11 as well.

Table 11: ASRs vs. untrained VGG19 on CIFAR100, with different ρ values for gradient clipping and $\sigma = 0.01$ for sampling Gaussian noise, and relative accuracy (rightmost column) of model after training with DP, compared to the baseline of 70.60%.

ρ	LLG	EBI	LLBG	Rel. Acc.
0.1	41.82 ± 3.55	**62.93 ± 3.74**	33.17 ± 1.56	−66.80%
0.5	**84.77 ± 2.56**	**85.02 ± 1.85**	70.85 ± 2.44	−54.47%
1	84.83 ± 1.95	83.13 ± 1.79	**95.46 ± 1.21**	−44.22%
1.5	84.77 ± 2.21	83.05 ± 1.77	**95.46 ± 1.21**	−38.80%
∞	84.77 ± 2.21	82.84 ± 1.75	**95.40 ± 1.21**	−21.14%

Table 12: ASRs vs. untrained VGG19 on CIFAR100, with batch size of 128, different σ values for sampling Gaussian noise, and ρ=1 for gradient clipping, and Relative Accuracy (rightmost column) of model after training with DP, compared to baseline of 70.60%.

σ	LLG	EBI	LLBG	Rel. Acc.
0	81.63 ± 1.78	78.96 ± 1.38	**99.63 ± 0.47**	+0.65%
0.01	84.97 ± 2.01	83.42 ± 1.73	**95.62 ± 1.14**	−38.05%
0.1	37.01 ± 4.26	55.73 ± 4.20	**75.19 ± 3.43**	−69.60%
0.3	21.17 ± 3.65	29.14 ± 3.70	**56.43 ± 8.88**	−69.60%
0.5	17.38 ± 3.90	22.34 ± 3.70	**41.82 ± 12.96**	−69.60%

Setting $\rho \geq 1$ had little effect on all attacks in this setting, which means that most gradients had lower norm and were not clipped. For LLG and EBI significant privacy gains were only achieved for ρ=0.1, while the ASR of LLBG dropped for the case where ρ=0.5. Of all three attacks, the EBI was least affected, having only ∼ 20% decrease in ASR, while LLG had the largest decrease, of ∼82%. Still, the accuracy loss of the trained model was substantial for all the settings, and especially for the case where ρ=0.1. This puts into question the usefulness of this defense.

We also experimented with adding Gaussian noise of different standard deviations σ, while setting the gradient clipping value ρ=1. For this experiment we again attacked the untrained VGG19 model with CIFAR100 and batch size of 128. We experimented with $\sigma \in \{0.01, 0.1, 0.3, 0.5\}$ and also trivial case where $\sigma = 0$ (i.e., no noise added). Tab. 12 reports the results.

Except for two data points, the results show that ASRs of all attacks were indeed inversely correlated to σ. The exceptions were the ASRs of LLG and EBI for σ=0.01, which were higher compared to results for σ=0. This is due to the fact that indices correlating to labels that were in the training batch but had a non-negative gradient were changed to a negative value by the noise addition, leading these to be added to the reconstruction and its accuracy rising. For higher σ values this effect was outweighed by the information obfuscation achieved by adding noise. The LLG ASRs came close to the rate of the random attack at $\sigma \geq 0.3$, for EBI this is true only for σ=0.5, and LLBG remained well above the random attack for all values of σ tested. Although it was also affected in a

Table 13: ASRs vs. untrained VGG19 on CIFAR100, with different values of p for gradient compression, and Relative Accuracy (rightmost column) of model after training with compression, compared to baseline of 70.60%.

p	LLG	EBI	LLBG	Rel. Acc.
0	81.57 ± 1.78	78.96 ± 1.38	**99.62 ± 0.39**	+0.00%
0.1	82.13 ± 7.86	78.96 ± 1.38	**99.62 ± 0.39**	+0.24%
0.4	81.53 ± 1.78	76.02 ± 1.23	**90.89 ± 1.41**	+0.12%
0.8	78.32 ± 1.45	76.02 ± 1.22	**83.65 ± 1.87**	−2.78%
0.9	77.52 ± 1.38	76.02 ± 1.23	**82.62 ± 2.11**	−5.91%
0.99	**73.88 ± 2.37**	50.70 ± 3.49	64.20 ± 3.08	−38.00%

significant manner (dropping from ∼99% to ∼41%) by this defense, it was more robust against it, compared to the other two attacks.

Clipping alone had no noticeable effect on accuracy, compared to the case where no clipping is applied. However, any addition of noise had a major effect on the accuracy of the trained model, and for $\sigma > 0.01$ reduced the accuracy to 1%, equivalent to random guessing for CIFAR100. These results also suggest that DP may have too high a cost to be relevant defense against label leakage.

Gradient Compression We also experimented with gradient compression, described in §2.3. We implemented compression as described by Lin et al. [24]—i.e. compressing the aggregated gradient of each weight and bias vector separately, with the threshold for each gradient set according to its p^{th} percentile value. We experiment with an untrained VGG19 model, on the CIFAR100 dataset with batch size of 128, and values $p \in \{0, 0.1, 0.4, 0.8, 0.9, 0.99\}$ (for p=0 no compression is performed). Results are reported in Tab. 13. To assess the effect of compression on the model's utility after training, we also trained an instance of the VGG19 model for each setting (including $p = 0$ where no compression is applied, which is the baseline) and reported its final test accuracy (also in Tab. 13).

Unlike the results for compression reported by Wainakh et al. [37], which showed it to be a highly effective defense, lowering the ASR of LLG to below random guess when p=0.8, we found its effect against LLG to be minor, while having a large effect against EBI only when $p = 0.99$. While there is a non-negligible effect against LLBG when $p \geq 0.4$, the ASR remained relatively high for p<0.99. The main difference from the experiment reported by Wainakh et al. is the dataset used, where in that case it was the MNIST dataset [23], which only has 10 classes compared to 100 for CIFAR100. These results suggest that for unbalanced data compression is only effective as a defense in cases where $B \gg n$, since only then does compression remove sufficient information from gradients.

To achieve a significant decrease in ASR of all attacks via compression (at p=0.99) a substantial loss of utility of the final model is incurred. This may be avoided by changing the training procedure (e.g. adding more training epochs) but also indicates that this defense has an unfavorable utility-privacy trade-off.

Removing the Bias Another defense we considered is one tailored specifically against our LLBG attack: it removes the bias parameter from the last linear layer of the model, resulting in a linear layer consisting of only a weights matrix. With this defense the LLBG and EBI attacks are not applicable, as there is no bias gradient (in the last layer) to exploit for label extraction, while the LLG attack remains applicable, since neither its analysis nor its implementation require the bias gradient. Note that this defense is only applicable when training a model from scratch. If a partially trained (or pre-trained) model is used, removing some of its parameters will harm its performance and will defeat the purpose of using such a model.

Table 14: ASRs vs. VGG11 on CIFAR100, with and without bias at the last layer, considering different training stages.

	Untrained, w/	Trained, w/	Untrained, w/o	Trained, w/o
LLG	79.34 ± 1.48	**79.20 ± 7.14**	79.19 ± 1.54	78.06 ± 7.40
EBI	79.05 ± 1.37	**79.36 ± 7.12**	N/A	N/A
LLBG	**99.63 ± 0.39**	N/A	N/A	N/A
LLBG_{aux}	N/A	75.04 ± 7.48	N/A	N/A

We experimented with the VGG11 model [33] on CIFAR100, with four different settings: randomly initialized or trained [39] and with the original architecture or without the last bias. The accuracy of the trained models was roughly the same (∼68%) with and without the bias parameters in the last layer. In other words, removing the bias from the last layer did not impact the model accuracy.

We report attacks' ASRs in Tab. 14. When including weight biases, LLBG was the strongest attack against untrained models, while the remaining attacks were comparable for both untrained and trained models. When removing bias, EBI, LLBG, and LLBG_{aux} all became non-applicable as they require the bias gradients (i.e., the defense managed to hinder them without harming model accuracy). As for LLG, however, it remained mostly unaffected by the training status and the removal of the bias parameter, meaning other defenses remain necessary for mitigating it.

7 Conclusion

We propose LLBG, a novel label-reconstruction attack against FL derived from deep learning models' last layer bias gradients. Specifically, we offer two LLBG variants tailored for FL at different stages: one for randomly initialized, untrained models, and another for trained models. We tested LLBG under a wide variety of settings, including with two popular datasets, a collection of models with varying complexities and sizes, different training stages, batch sizes, and defenses. We found that LLBG achieved the highest success rates compared to prior leading attacks in almost all settings, and enabled label reconstruction with remarkable success in settings where past attacks may not apply. Our work also offers simple, yet effective means to mitigate the LLBG variants.

Acknowledgments

This work has been supported in part by grant No. 2023641 from the United States-Israel Binational Science Foundation (BSF); by a grant from the Blavatnik Interdisciplinary Cyber Research Center (ICRC); by Intel® via a Rising Star Faculty Award; by a gift from KDDI Research; by Len Blavatnik and the Blavatnik Family foundation; by a Maof prize for outstanding young scientists; by the Ministry of Innovation, Science & Technology, Israel (grant number 0603870071); by NVIDIA via a hardware grant; and by a grant from the Tel Aviv University Center for AI and Data Science (TAD).

References

[1] Martin Abadi, Andy Chu, Ian Goodfellow, H Brendan McMahan, Ilya Mironov, Kunal Talwar, and Li Zhang. 2016. Deep learning with differential privacy. In *CCS*.

[2] Eugene Bagdasaryan, Andreas Veit, Yiqing Hua, Deborah Estrin, and Vitaly Shmatikov. 2020. How to backdoor federated learning. In *AISTATS*.

[3] Gilad Baruch, Moran Baruch, and Yoav Goldberg. 2019. A little is enough: Circumventing defenses for distributed learning. In *NeurIPS*.

[4] Peva Blanchard, El Mahdi El Mhamdi, Rachid Guerraoui, and Julien Stainer. 2017. Machine learning with adversaries: Byzantine tolerant gradient descent. In *NeurIPS*.

[5] Keith Bonawitz, Vladimir Ivanov, Ben Kreuter, Antonio Marcedone, H Brendan McMahan, Sarvar Patel, Daniel Ramage, Aaron Segal, and Karn Seth. 2017. Practical secure aggregation for privacy-preserving machine learning. In *CCS*.

[6] Nicholas Carlini, Chang Liu, Úlfar Erlingsson, Jernej Kos, and Dawn Song. 2019. The Secret Sharer: Evaluating and Testing Unintended Memorization in Neural Networks. In *USENIX Security*.

[7] Torch Contributors. [n. d.]. MODELS AND PRE-TRAINED WEIGHTS. https://pytorch.org/vision/stable/models.html. Last accessed on 09-01-2024.

[8] Fida Kamal Dankar and Khaled El Emam. 2013. Practicing differential privacy in health care: A review. *Trans. Data Priv.* 6, 1 (2013), 35–67.

[9] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. 2021. An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale. In *ICLR*.

[10] Cynthia Dwork. 2008. Differential privacy: A survey of results. In *TAMC*. 1–19.

[11] Nadav Gat and Mahmood Sharif. 2024. Implementation of LLBG. https://github.com/nadbag98/LLBG.

[12] Jonas Geiping, Hartmut Bauermeister, Hannah Dröge, and Michael Moeller. 2020. Inverting Gradients - How easy is it to break privacy in federated learning?. In *NeurIPS*.

[13] Rachid Guerraoui, Sébastien Rouault, et al. 2018. The hidden vulnerability of distributed learning in byzantium. In *ICML*.

[14] Chuan Guo, Geoff Pleiss, Yu Sun, and Kilian Q. Weinberger. 2017. On Calibration of Modern Neural Networks. In *ICML*.

[15] Florian Hartmann. 2021. Predicting Text Selections with Federated Learning. https://ai.googleblog.com/2021/11/predicting-text-selections-with.html. Last accessed on 09-01-2024.

[16] Florian Hartmann and Peter Kairouz. 2023. Distributed differential privacy for federated learning. https://ai.googleblog.com/2023/03/distributed-differential-privacy-for.html. Last accessed on 09-01-2024.

[17] K. He, X. Zhang, S. Ren, and J. Sun. 2016. Deep Residual Learning for Image Recognition. In *CVPR*. 770–778.

[18] Dan Hendrycks and Kevin Gimpel. 2016. Bridging Nonlinearities and Stochastic Regularizers with Gaussian Error Linear Units. *arXiv preprint arXiv:1606.08415* (2016).

[19] Yangsibo Huang, Samyak Gupta, Zhao Song, Kai Li, and Sanjeev Arora. 2021. Evaluating Gradient Inversion Attacks and Defenses in Federated Learning. In *NeurIPS*.

[20] Vinay Joshi, Manuel Gallo, Simon Haefeli, Irem Boybat, Nandakumar S.R., Christophe Piveteau, Martino Dazzi, Bipin Rajendran, Abu Sebastian, and Evangelos Eleftheriou. 2020. Accurate deep neural network inference using computational phase-change memory. *Nature Communications* 11 (2020).

[21] Alex Krizhevsky, Geoffrey Hinton, et al. 2009. Learning multiple layers of features from tiny images. *University of Toronto* (2009).

[22] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. 2012. ImageNet Classification with Deep Convolutional Neural Networks. In *NeurIPS*.

[23] Yann Lecun, Leon Bottou, Y. Bengio, and Patrick Haffner. 1998. Gradient-Based Learning Applied to Document Recognition. *IEEE* 86 (1998).

[24] Yujun Lin, Song Han, Huizi Mao, Yu Wang, and William J Dally. 2017. Deep gradient compression: Reducing the communication bandwidth for distributed training. *arXiv preprint arXiv:1712.01887* (2017).

[25] Ningning Ma, Xiangyu Zhang, Hai-Tao Zheng, and Jian Sun. 2018. ShuffleNet V2: Practical Guidelines for Efficient CNN Architecture Design. In *ECCV*.

[26] Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Aguera y Arcas. 2017. Communication-efficient learning of deep networks from decentralized data. In *AISTATS*.

[27] Luca Melis, Congzheng Song, Emiliano De Cristofaro, and Vitaly Shmatikov. 2019. Exploiting unintended feature leakage in collaborative learning. In *IEEE S&P*.

[28] Ahmed El Ouadrhiri and Ahmed Abdelhadi. 2022. Differential Privacy for Deep and Federated Learning: A Survey. *IEEE Access* 10 (2022), 22359–22380.

[29] Andrew Paverd, Andrew Martin, and Ian Brown. 2014. Modelling and automatically analysing privacy properties for honest-but-curious adversaries. *Tech. Rep* (2014).

[30] Le Trieu Phong and Tran Thi Phuong. 2023. Differentially private stochastic gradient descent via compression and memorization. *Journal of Systems Architecture* 135 (2023), 102819.

[31] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. 2015. ImageNet Large Scale Visual Recognition Challenge. *IJCV* 115, 3 (2015), 211–252.

[32] Reza Shokri, Marco Stronati, Congzheng Song, and Vitaly Shmatikov. 2017. Membership inference attacks against machine learning models. In *IEEE S&P*.

[33] Karen Simonyan and Andrew Zisserman. 2015. Very Deep Convolutional Networks for Large-Scale Image Recognition. In *ICLR*.

[34] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. 2014. Intriguing properties of neural networks. In *ICLR*.

[35] M. Tan, B. Chen, R. Pang, V. Vasudevan, M. Sandler, A. Howard, and Q. V. Le. 2019. MnasNet: Platform-Aware Neural Architecture Search for Mobile. In *CVPR*.

[36] Mingxing Tan and Quoc Le. 2019. EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks. In *ICML*.

[37] Aidmar Wainakh, Till Müßig, Tim Grube, and Max Mühlhäuser. 2022. Label Leakage from Gradients in Distributed Machine Learning. In *PETS*.

[38] Zhou Wang, A.C. Bovik, H.R. Sheikh, and E.P. Simoncelli. 2004. Image quality assessment: from error visibility to structural similarity. *IEEE Transactions on Image Processing* 13, 4 (2004), 600–612.

[39] weiaicunzai and Yonghye Kwon. [n. d.]. PyTorch-CIFAR100: A PyTorch implementation for CIFAR-100. https://github.com/weiaicunzai/pytorch-cifar100. Last accessed on 09-01-2024.

[40] Jie Wen, Zhixia Zhang, Yang Lan, Zhihua Cui, Jianghui Cai, and Wensheng Zhang. 2023. A survey on federated learning: challenges and applications. *IJMLC* 14, 2 (2023), 513–535.

[41] Haomiao Yang, Mengyu Ge, Dongyun Xue, Kunlan Xiang, Hongwei Li, and Rongxing Lu. 2023. Gradient Leakage Attacks in Federated Learning: Research Frontiers, Taxonomy and Future Directions. *IEEE Network* (2023), 1–8.

[42] Qiang Yang, Yang Liu, Tianjian Chen, and Yongxin Tong. 2019. Federated Machine Learning: Concept and Applications. *ACM Trans. Intell. Syst. Technol.* 10, 2 (2019).

[43] H. Yin, A. Mallya, A. Vahdat, J. M. Alvarez, J. Kautz, and P. Molchanov. 2021. See through Gradients: Image Batch Recovery via GradInversion. In *CVPR*.

[44] Richard Zhang, Phillip Isola, Alexei A Efros, Eli Shechtman, and Oliver Wang. 2018. The unreasonable effectiveness of deep features as a perceptual metric. In *CVPR*.

[45] Zhuosheng Zhang, Jiarui Li, Shucheng Yu, and Christian Makaya. 2023. Safelearning: Secure aggregation in federated learning with backdoor detectability. *IEEE Transactions on Information Forensics and Security* (2023).

[46] Bo Zhao, Konda Reddy Mopuri, and Hakan Bilen. 2020. iDLG: Improved Deep Leakage from Gradients. *arXiv preprint arXiv:2001.02610* (2020).

[47] Joshua C Zhao, Ahmed Roushdy Elkordy, Atul Sharma, Yahya H Ezzeldin, Salman Avestimehr, and Saurabh Bagchi. 2023. The Resource Problem of Using Linear Layer Leakage Attack in Federated Learning. In *CVPR*.

[48] Joshua C Zhao, Atul Sharma, Ahmed Roushdy Elkordy, Yahya H Ezzeldin, Salman Avestimehr, and Saurabh Bagchi. 2023. Secure aggregation in federated learning is not private: Leaking user data at large scale through model modification. *arXiv preprint arXiv:2303.12233* (2023).

[49] Lun Zhao, Siquan Hu, and Zhiguo Shi. 2023. Federated Learning Scheme Based on Gradient Compression and Local Differential Privacy. In *ICCECT*.

[50] Ying Zhao and Jinjun Chen. 2022. A Survey on Differential Privacy for Unstructured Data Content. *ACM Comput. Surv.* 54, 10s (2022).

[51] Ligeng Zhu, Zhijian Liu, and Song Han. 2019. Deep Leakage from Gradients. In *NeurIPS*.