Label Inference Attacks Against Vertical Federated Learning

Chong Fu[1], Xuhong Zhang[1,2], Shouling Ji[1,2], Jinyin Chen[3], Jingzheng Wu[4], Shanqing Guo[5], Jun Zhou[6], Alex X. Liu[6], and Ting Wang[7]

[1]Zhejiang University, [2]Binjiang Institute of Zhejiang University, [3]Zhejiang University of Technology, [4]Institute of Software, Chinese Academy of Sciences, [5]Shandong University, [6]Ant Group, [7]Pennsylvania State University

E-mails: {fuchong, zhangxuhong, sji}@zju.edu.cn, chenjinyin@zjut.edu.cn, jingzheng08@iscas.ac.cn, guoshanqing@sdu.edu.cn, {jun.zhoujun, alexliu}@antfin.com, inbox.ting@gmail.com

Abstract

As the initial variant of federated learning (FL), horizontal federated learning (HFL) applies to the situations where datasets share the same feature space but differ in the sample space, e.g., the collaboration between two regional banks, while trending vertical federated learning (VFL) deals with the cases where datasets share the same sample space but differ in the feature space, e.g., the collaboration between a bank and an e-commerce platform.

Although various attacks have been proposed to evaluate the privacy risks of HFL, yet, few studies, if not none, have explored that for VFL. Considering that the typical application scenario of VFL is that a few participants (usually two) collaboratively train a machine learning (ML) model with features distributed among them but labels owned by only one of them, protecting the privacy of the labels owned by one participant should be a fundamental guarantee provided by VFL, as the labels might be highly sensitive, e.g., whether a person has a certain kind of disease. However, we discover that the bottom model structure and the gradient update mechanism of VFL can be exploited by a malicious participant to gain the power to infer the privately owned labels. Worse still, by abusing the bottom model, he/she can even infer labels beyond the training dataset. Based on our findings, we propose a set of novel label inference attacks against VFL. Our experiments show that the proposed attacks achieve an outstanding performance. We further share our insights and discuss possible defenses. Our research can shed light on the hidden privacy risks of VFL and pave the way for new research directions towards more secure VFL.

1 Introduction

With more and more countries realizing the importance of security and privacy of user data, regulations like General Data Protection Regulation (GDPR) [48] are issued to put forward strict requirements for the use of user data. Particularly, any company cannot directly disclose its user data to any other company, which makes it hard, if not impossible, for companies to train a more accurate ML model with a joint data pool. To address this issue, the emergence of FL provides an alternative. As a privacy-preserving distributed ML technique, it allows multiple participants to collaboratively train a ML model by periodically exchanging intermediate computation results without revealing their raw data. Worldwide IT companies put much effort into developing FL systems [24]. To date, several open source FL systems are developed, such as TensorFlow Federated (TFF) from Google [15], Federated AI Technology Enabler (FATE) from Tencent Webank [53], PySyft from OpenMined [34], and PaddleFL from Baidu [3].

According to the sample and feature space of data parities, FL can be categorized into HFL and VFL. HFL is designed for the situations where datasets share the same feature space but differ in the sample space, e.g., the collaboration between two regional banks, while VFL deals with the cases where datasets share the same sample space but differ in the feature space, e.g., the collaboration between a bank and an e-commerce platform. FL is claimed to be privacy-preserving as local data never leaves participants' local machines, but actually FL *does* leak private information indirectly. Recent studies have thoroughly analyzed the privacy and security risks of HFL, such as data leakage from gradients [62], membership inference attacks [33], property inference attacks [30], and backdoor attacks [2].

However, the privacy risks of VFL remain underexplored. Moreover, VFL is being used more and more widely in industry [16, 50, 52, 54]. One of its typical usage scenarios, which is the focus of this paper, is that a few participants (usually two) collaboratively train a model with features distributed among them but labels owned by only one of them [50, 52, 54]. For example, a car rental company with plenty of labels but limited user features for risk evaluation might want to improve its model performance by incorporating more user features from other businesses, e.g., a bank [52]. The role of other partici-

Chong Fu and Xuhong Zhang are the co-first authors. Shouling Ji is the corresponding author.

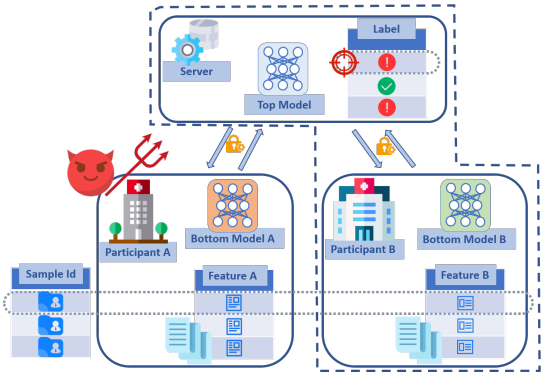


Figure 1: Illustration of the label inference attacks against VFL. In practice, the server is usually managed by the participant who holds the labels, as outlined by the dotted line. The participant without the labels is the adversary, whose goal is to infer the labels.

pants is simply providing more features and they usually ask for money as return, since the final trained model is private to the participant with the labels.

As VFL is different from HFL in many aspects including usage scenarios, the architecture of the federated model and the training algorithm, VFL has unique privacy issues. In this paper, we focus on a new label leakage privacy risk of VFL. In the VFL architecture as shown in Figure 1, there is only one participant owning the labels, which is different from HFL where every participant has its own labeled samples. Ensuring the privacy of the private labels should be a fundamental guarantee provided by VFL, as the labels might be the key asset of the participant or highly sensitive. Moreover, an adversarial participant may try to build a similar business with the stolen labels or sell the sensitive labels to underground industries.

Compared with HFL, it is more challenging for the adversarial participant in VFL to infer private information. In general, an adversarial participant in HFL controls a complete local model and has access to gradients of all the parameters of this model, which can be abused to infer private information. However, in VFL, the adversarial participant only controls part of the federated model, which cannot run independently, and only has access to the gradients of this incomplete model.

Despite these challenges, we discover that the current design of VFL has an inherent vulnerability that can be leveraged by an adversarial participant to conduct label inference attacks. A thorough study of this vulnerability is imperative for the wide applications of VFL. Specifically, we find that the bottom model controlled by a malicious participant might naturally have the ability to infer the privately owned labels of other participant(s). The reason is that during training, the gradients propagated from the server help the bottom model learn a good feature representation with respect to the labels and thus can serve as a pre-trained model for label inference attacks. The inference accuracy is determined by the expressiveness of this trained malicious bottom model. It is worth noting that the bottom model empowers the malicious participant to infer the label of any sample, not just the ones in the training dataset, as long as the input features to the bottom model are available. In some special cases, the gradients from the server can also directly leak the label information.

Based on the above findings, we propose a set of label inference attacks. To take advantage of the trained bottom model, we first introduce a passive label inference attack with "model completion". We show that with the help of a small amount of auxiliary labeled data, the malicious participant can fine-tune his/her trained bottom model into a complete label inference model in a semi-supervised manner. Experiments show that with only 70 auxiliary labeled samples, the passive label inference attack achieves an F1 score of 0.7614 on a real-world medical image dataset with tens of thousands of training samples. To increase the expressiveness of the malicious bottom model so as to further improve the attack performance, we further introduce an active label inference attack to trick the federated model to rely more on the malicious bottom model. Our experiment shows that the active attack successfully boosts the attack performance. Moreover, we use visualization techniques such as GradCAM [42] and t-SNE [46] to explain why the active label inference attack works. Then, we introduce a special direct label inference attack which infers labels by analyzing the signs of gradients from the server. It is only applicable when the VFL architecture is not designed with split learning [47]. Our experiments show that the direct label inference attack can achieve a label inference accuracy of 1.00 on every evaluated dataset.

Finally, we evaluate four possible defenses: noisy gradients, gradient compression, privacy-preserving deep learning [43] and discreteSGD. We find that though some of these defenses can mitigate the direct label inference attack, they are not effective for our passive and active label inference attacks. This motivates better defenses to enhance the privacy of VFL.

The main contributions of this paper are summarized as follows:

- To the best of our knowledge, we are the first to evaluate the privacy risks of VFL. We reveal and shed lights on the new label leakage issue of VFL.
- We present three types of label inference attacks against VFL, including the direct label inference attack, the passive label inference attack with model completion, and the active label inference attack with the malicious local optimizer. These attacks cover multiple practical VFL settings.
- We evaluate our attacks on various tasks with real-world datasets under both two-participant and multi-participant settings and achieve outstanding attack performance. Further, we share insights about the underlying working mechanism of the active label inference attack, and present readily comprehensible proofs. We also evaluate four possible defenses against our attacks and find that they are not effective, which motivates future work on better defenses.

2 Background

In this section, we briefly introduce HFL and VFL. They both have essential industrial applications. For example, HFL is used to improve the quality of virtual keyboard search suggestion without directly gathering user data [57]; VFL is used to train a joint model to predict traffic violations by a car rental service provider and a bank [52].

2.1 Horizontal Federated Learning

HFL is suitable for the situations where all participants' local datasets share the same feature space but differ in the sample space. For example, hospital A has the medical records of a group of citizens and labels of whether they have a kind of disease. Hospital B, located in another city, has the same kind of medical records and labels of another group of citizens. If the two hospitals aggregate all their samples, they can build a larger dataset and then train a more accurate model. However, directly sharing the private data of users is forbidden by law. This case reflects the dilemma of "horizontally isolated data". HFL can be applied to solve this dilemma.

In HFL, the server runs a global model, and every participant runs a local model. All local models share the same model architecture with the global model. One classic training algorithm of HFL is the model averaging [29]. In model averaging, for one training iteration, every participant trains his/her local model for several epochs on the local dataset, then submits the updated local model to the server; the server then averages every submitted local model to get the updated global model. In this way, participants can jointly train a global model without sharing their raw data [59].

2.2 Vertical Federated Learning

VFL [18, 28, 56] is designed for the situation where participants' datasets share the same sample space but differ in the feature space. For example, hospital A has CT scans of a group of patients and labels of whether they have lung cancer. Hospital B has MRI scans of the same group of patients. If these two hospitals put their data together, they can train a better ML model to predict lung cancer according to both CT and MRI scans. Unfortunately, they cannot directly share data or labels to each other – disclosing patients' private data to other organizations is strictly forbidden by law.

VFL is designed to solve this dilemma. There are two kinds of popular VFL architectures: VFL without model splitting and VFL with model splitting [47]. In both architectures, there is a trusted third-party server which preserves labels, and participants that have vertically partitioned data. Every training iteration of VFL can be divided into two steps. The first step is the federated forward propagation. All participants do local forward propagation using their local data and bottom models, then submit local outputs to the server. The server uses the aggregated outputs from all participants to compute the final prediction and then computes the corresponding loss value. The second step is the federated backward propagation. The server does backward propagation and computes the gradients of the loss w.r.t. the outputs from every participant. The gradients are sent back to every participant. Then each participant continues the federated backward propagation and updates its bottom model.

For VFL without model splitting, every participant runs a bottom model, and the server does not run any model. Each participant's bottom model gives an output; then, the server simply sums up all outputs to get the final output. For more detailed description of the training algorithm of VFL without model splitting, please refer to Appendix A. Actually, for VFL without model splitting, each participant *has access to the output layer*, which might bring a significant label leakage risk. We will introduce this risk in detail and show how to exploit it in Section 3.5.

VFL with model splitting is designed with the idea of split learning [47]. The whole ML model is split into a top model and some bottom models at a specific middle layer, referred to as the *cut layer*. As shown in Figure 1, every participant runs a bottom model that learns hidden representations of his/her local data. The server runs a top model to aggregate hidden representations from every participant, and then computes the final output. Backward propagation is completed by sharing gradients of the cut layer. For more detailed description of the training algorithm of VFL with model splitting, please refer to Appendix A. In VFL with model splitting, participants have no access to the last layer of the DNN. Therefore, the labels on the server are more secure.

After the training process is finished, input features and trained bottom models are still preserved by every participant. Hence, at the inference time, VFL requires all participants to get involved. This is different from the situations of HFL, where the trained global model is shared to every participant, and participants can do inference individually.

In real world applications, a fully trusted third-party server is hard to implement, thus the server in VFL is usually managed by one of the participants, as shown in Figure 1. In this architecture, there is one participant holding both labels and part of the vertically partitioned data. In addition to a bottom model, it also runs the top model. However, this architecture has no fundamental difference from the VFL with model splitting described above. Actually, this architecture suits best for the scenarios that one participant, who exclusively owns the labels but possesses few features, wants to improve its model performance by incorporating more features from other participants that might come from different business domains. Although the private labels never leave the hand of the participant, in the following, we show that they might still get leaked via label inference attacks.

3 Label Inference Attacks

In this section, we first share the insights on why VFL is vulnerable to label inference attacks, and then we design three kinds of label inference attacks based on the insights.

3.1 Possible Privacy Leakage in VFL

In the training process of VFL, participants and server do not exchange data or labels directly; instead, they exchange intermediate computation results. However, there are still risks of privacy leakage. We discover that the current design of VFL has inherent vulnerabilities to label inference attacks. In the following, we share our findings on two components of VFL that may cause label leakage: leakage from the trained bottom model and leakage from the gradients.

Leakage from the Trained Bottom Model. VFL with/without model splitting requires a bottom model trained locally by each participant to embed the input features to a latent space, which avoids the raw features being directly sent to the server. Additionally, this bottom model is within the full control of a potential adversarial participant during both the training and inference stages. Though this design protects the privacy of features from participants, it also empowers an adversary to infer the privately owned labels. The fundamental reason is that the bottom model is optimized by VFL to provide a more indicative feature representation to predict the labels. The expressiveness of a malicious bottom model depends on how well it got trained and how close it is to the final prediction layer. We will introduce two kinds of label inference attacks that exploit the expressiveness of the trained bottom model.

Leakage from the Gradients. In VFL, every participant can receive the gradients of the loss w.r.t. the outputs sent to the server. As the loss is computed by measuring the error between the predicted label and the ground-truth label, its gradients contain hidden information about labels. It can be further inferred that the closer a given layer is to the final prediction layer, its gradients contain more information about the labels. Therefore, instead of relying on the expressiveness of the trained bottom model, an adversary might be able to perform label inference attacks simply based on the received gradients. In this paper, we introduce a label inference attack by analyzing the sign of the received gradients under the setting of VFL without model splitting.

3.2 Threat Model

We assume that K participants (where $K \geq 2$) jointly train a ML model using the VFL framework described in Section 2.2. Each participant holds partial features for the ML model. We assume that the features held by each participant are useful for the prediction task, which is reasonable because only if a participant has valuable features, will it be allowed to join the VFL collaboration. The labels of the training dataset are privately owned by one participant. This participant also controls the server running the top model. One of the rest of the $K-1$ participants is the *adversary* with the goal to infer the privately owned labels. Note that the adversary's goal is to infer the labels of any interested samples, not just the ones in the training dataset.

An illustration of our label inference attacks is shown in Figure 1. At each training round, each participant receives the gradients of the loss w.r.t. his/her bottom model's outputs from the server, and then uses them to update his/her bottom model. The adversary can optionally exploit the received gradients to conduct attack in the training stage. Additionally, once the training process is over, each participant gets a trained bottom model. With the help of a small amount of auxiliary labeled samples, the adversary can further train a model for label inference based on the trained bottom model. Finally, to infer the label of an interested sample, the adversary also needs to have the sample features required by the trained bottom model.

3.3 Passive Label Inference Attack through Model Completion

Labels preserved by the server are considered to be well protected because any participant cannot access the top model. However, we demonstrate that an adversary can infer the labels based on his/her locally owned bottom model. As discussed in Section 3.1, the trained bottom model of an adversary can transform the features he/she owns to a very indicative representation to predict the labels. Therefore, the adversary can fine-tune the bottom model with an additional classification layer for label inference using a small amount of auxiliary labeled data. We name this attack as *the passive label inference attack through model completion*.

Model Completion. After the training process, the adversary gets his/her trained bottom model. Then, the adversary adds additionally randomly initialized layers on top of the trained bottom model to make a "complete model" for label inference. We refer to these additional layers as *the inference head*. We assume that the adversary has a small amount of auxiliary labeled data to fine-tune the complete model. Specifically, in our design, a few dozens of auxiliary labeled samples are usually sufficient, e.g., 40 auxiliary labeled samples versus 50,000 labeled training samples in our experiments. In practice, it is not hard for an adversary to obtain this small amount of labels. For example, if the adversary is a company, it can simply buy the labels from its employees. To mitigate the issue of limited auxiliary labeled data, the adversary can fine-tune the complete model in a semi-supervised manner. We choose the state-of-the-art semi-supervised learning algorithm MixMatch [5] for computer vision (CV) datasets and datasets with numerical or categorical features. Vanilla MixMatch uses various strategies, including training on pseudo labels, regularization, and data augmentation. However, in

Algorithm 1 Local malicious optimization of the adversary's bottom model

Require: Momentum parameter β, the gradient scaling factor's resetting parameter γ, maximum gradient scaling factor r_{max}, minimum gradient scaling factor r_{min}, learning rate η, initial bottom model parameters Θ, initial gradient velocity v.

1: **while** stopping criterion not met **do**
2: Receive G_{output} from the server
3: $G \leftarrow Backward(G_{output})$
4: **for** each parameter θ in Θ and its gradient g_θ in G **do**
5: $v_\theta \leftarrow \beta \cdot v_\theta + (1-\beta) \cdot g_\theta$
6: **if** is not the first criterion **then**
7: $r_\theta \leftarrow 1.0 + \gamma \cdot (v_\theta \div v_{last})$
8: $r_\theta \leftarrow Max(r_\theta, r_{min})$
9: $r_\theta \leftarrow Min(r_\theta, r_{max})$
10: $v_\theta \leftarrow r_\theta \cdot v_{last}$
11: **end if**
12: $v_{last} \leftarrow v_\theta$
13: $\theta \leftarrow \theta - \eta \cdot v_\theta$
14: **end for**
15: **end while**

implementation, we omit the part of data augmentation (such as image flipping, image random cropping) in MixMatch to make it suitable for tasks of other domains rather than just CV. Specially, for text datasets, we choose the state-of-the-art semi-supervised learning algorithm for natural language processing, MixText [8]. For more detailed description of the customized MixMatch algorithm, please refer to Appendix B.

Once the semi-supervised training is finished, the adversary gets a fully-trained complete model. This model can predict a label for every datum of the adversary. In this way, the adversary successfully infers the label of any interested sample with his/her available features, no matter whether the sample is within the training dataset or not. In the whole attack pipeline, the adversary keeps being "honest but curious", i.e., he/she strictly follows the rule of VFL both in the training and inference stages. We name this attack as *the passive label inference attack* as the adversary does not attack in the training or inference stage.

3.4 Active Label Inference Attack with the Malicious Local Optimizer

In this attack, we show that the adversary can actively trick the federated model to rely more on his/her bottom model so as to increase its expressiveness. With better expressiveness of the bottom model, the adversary can train a more accurate complete model for label inference. Here the adversary is assumed to actively do some malicious actions in the training stage.

Specifically, instead of using normal optimizers such as Adam or SGD with momentum, the adversary uses a specially designed malicious local optimizer. The key intuition is that the adversary can accelerate the gradient descent on his/her bottom model, and thus submits better features to the server in each iteration. Finally, it makes the top model rely more on the bottom model of the adversary rather than other participants. In other words, the adversary can maliciously scale up the learning rate when he/she trains his/her bottom model. However, there is a challenge: a larger learning rate does not always lead to more efficient gradient descent. An overly large learning rate deviates network parameters from the shortest path to a local optimum. For example, imagine that we are training a neural network by gradient descent, and the network is near a local minimum point in the parameter space; if we use an overly large learning rate for gradient descent, the network parameters will tend to oscillate around the local minimum point [45].

To address this challenge, we design an adaptive malicious local optimizer, which adaptively scales up the gradient of each parameter in the adversary's bottom model. The detailed algorithm of the malicious local optimizer is shown in Algorithm 1. G_{output} is the gradient of the loss w.r.t. the output of the adversary's bottom model. G is the gradients of the loss w.r.t. the parameters of the adversary's bottom model. For each parameter θ, r_θ is its adaptive gradient scaling factor, and v_θ is the exponential moving average of each scaled gradient in previous iterations (also known as the velocity). First, the adversary receives G_{output} from the server, then computes G by backward propagating G_{output} (line 2-3). Afterwards, v_θ is computed according to the definition of exponential moving average (line 5). Next, if it is not the first iteration, for each parameter θ and its gradient g_θ of the bottom model, r_θ is computed as $r_\theta \leftarrow 1.0 + \gamma \cdot (v_\theta \div v_{last})$, where γ is a hyperparameter set to 1.0 in our experiments and v_{last} is the value of v_θ in the last iteration (line 6-7). Further, r_θ is bounded by the maximum/minimum value to improve the stability of the algorithm (line 8-9), and then v_θ is scaled with r_θ (line 10). Finally, v_{last} is updated with v_θ, and then θ is updated with v_θ (line 12-13).

By using this mechanism, the malicious local optimizer scales up gradients with adaptive scaling factors. To illustrate how the adaptive scaling factor works, we take one parameter as a simplified case, which only has two optimization directions: increase or decrease. If the velocity (of this parameter) has an opposite sign with the velocity of the last iteration, it is considered an oscillation signal. Correspondingly, the malicious local optimizer will decrease the scaling factor. This intuition is similar to the classic optimization algorithm Rprop [40]. On the contrary, if the signs of the velocities remain the same in a series of iterations, it can be inferred that the parameter is steadily optimized in one direction. Then the malicious local optimizer will increase the scaling factor for faster optimization. Therefore, the adaptive malicious local optimizer is able to accelerate the gradient descent of the adversary's bottom model while avoiding the

negative influence of overly large local gradients.

By using the malicious local optimizer in the training stage, the adversary can obtain a trained bottom model with more hidden information about labels. We will visually prove this point in Section 5.3. Then the adversary can follow the model completion step in the passive inference attack to get the final label inference model. The active label inference attack is expected to boost the label inference accuracy and will be evaluated in Section 5.1.

3.5 Direct Label Inference Attack

Instead of relying on the trained bottom model, as discussed in Section 3.1, an adversary can also directly utilize the received gradients to infer the labels of the training examples. As a preliminary study, we only consider the VFL without model splitting. Under this setting, the adversary is able to receive the gradients of the final prediction layer. Taking such an advantage, he/she can directly infer labels by analyzing the signs of the gradients received from the server. We name this attack as *the direct label inference attack*. The idea is inspired by the work of Zhao et al. [60]. Their work focuses on the data reconstruction attack in HFL, but one step of their method is relevant to inferring labels. They demonstrated that the signs of the gradients in HFL can leak the labels of the training samples. We extend this idea to the field of VFL, and present mathematical analysis and experiments to demonstrate how the customized method works for label inference in VFL without model splitting. The direct label inference attack works for the mainstream loss functions used for the classification tasks, including the cross-entropy loss, the weighted cross-entropy loss, and the negative log likelihood loss.

Here we take the cross-entropy loss as an example. Generally, for the vertical federated model (without model splitting) trained with the cross-entropy loss, we have:

$$loss(x,c) = -log\frac{e^{\sum_k y_c^k}}{\sum_j e^{\sum_k y_j^k}} \tag{1}$$

where x is the features of one sample, and c is the ground-truth label. $y^k = [y_1^k, y_2^k, ...]$ is the activations of the output layer (logits) of the k^{th} participant's bottom model, and y_i^k is logits for the i^{th} class. $y = y^1 + y^2 + ... + y^K$ is the aggregated logits from K participants. Assuming that the adv^{th} participant is the adversary, then the gradient of the loss w.r.t. the i^{th} logit from the adversary is

$$\begin{aligned} g_i^{adv} = \frac{\partial loss(x,c)}{\partial y_i^{adv}} &= -\frac{\partial log e^{y_c^{adv}} - \partial log \sum_j e^{y_j^{adv}}}{\partial y_i^{adv}} \\ &= \begin{cases} -1 + \frac{e^{y_i^{adv}}}{\sum_j e^{y_j^{adv}}} & if\ i = c \\ \frac{e^{y_i^{adv}}}{\sum_j e^{y_j^{adv}}} & if\ i \neq c \end{cases} \end{aligned} \tag{2}$$

Table 1: Model architectures. "FCNN-3" refers to the 3-layer fully connected neural network.

Dataset	Bottom Model Architecture	Top Model Architecture
CIFAR-10	ResNet-18	FCNN-4
CIFAR-100	ResNet-18	FCNN-4
CINIC-10	ResNet-18	FCNN-4
Yahoo Answers	BERT	FCNN-4
Criteo	FCNN-3	FCNN-3
BHI	ResNet-18	FCNN-4

We have $g_i^{adv} < 0$ if $i = c$ and $g_i^{adv} > 0$ if $i \neq c$ because $\sum_j e^{y_j^{adv}} \in (0,1)$. Therefore, the sign of g_i^{adv} indicates whether the i^{th} label is the ground-truth label. As the server sends back g_i^{adv} to the adversary in every iteration, the adversary can infer the label of the current training sample by the sign of g_i^{adv}.

For the weighted cross-entropy loss, we have:

$$loss(x,c) = w_c \times (-log\frac{e^{\sum_k y_c^k}}{\sum_j e^{\sum_k y_j^k}}) \tag{3}$$

where w_c is the weight of the ground-truth label, and the definitions of other symbols are the same as above. Apparently, the new introduced weight $w_c (w_c > 0)$ has no impact on the fact that the sign of g_i^{adv} indicates whether the i^{th} label is the ground-truth label. As for the negative log likelihood loss, it is the same as the cross-entropy loss in practice.

To draw a conclusion, for VFL without model splitting, the adversary can directly infer labels from the signs of the gradients sent back by the server. The downside of this attack is that it is only able to infer the labels of training examples, as no gradients are available at the inference time. However, in order to infer the label of an arbitrary sample, the adversary can use the obtained labels of the training examples as auxiliary labeled data to further conduct the designed passive label inference attack.

4 Experimental Setup

4.1 Datasets and Model Architectures

We choose six datasets to evaluate our label inference attacks: CIFAR-10, CIFAR-100 [22], CINIC-10 [11], Yahoo Answers [39], Criteo [23] and Breast Histopathology Images Dataset (BHI) [32]. CIFAR-10 is selected for the convenience of later visualized analysis. CIFAR-100 is selected to prove the feasibility of our attacks on datasets with many class labels, e.g., 100 classes. CINIC-10, which is 4.5 times the size of CIFAR-10, is selected to prove that our attack is also effective on large datasets. Yahoo Answers is selected to prove that our method is applicable for various data types and model architectures, e.g., text data and BERT [12]. Criteo is a real-world dataset for predicting ad click-through rate, which is selected to show that our attack is applicable in commercial

Table 2: Attack performance of our passive and active label inference attacks. Since BHI is an unbalanced dataset, we use F1 score as the metric for evaluation. "Passive" or "Active" means that the VFL model is under the passive or active label inference attack.

Dataset	Train Set Size	Test Set Size	Number of Classes	Known Label Quantity Per Class	Metric	Attack Performance			
						Train Set		Test Set	
						Passive	Active	Passive	Active
CIFAR-10	50,000	10,000	10	4	Top-1 Acc	0.8024	**0.8484**	0.6299	**0.6342**
CIFAR-100	50,000	10,000	100	4	Top-5 Acc	0.6267	**0.6732**	0.4319	**0.4700**
CINIC-10	180,000	90,000	10	4	Top-1 Acc	0.7206	**0.7818**	0.5440	**0.5995**
Yahoo Answers	50,000	20,000	10	10	Top-1 Acc	0.6335	**0.6424**	0.6370	**0.6419**
Criteo	80,000	20,000	2	50	Top-1 Acc	0.6828	**0.6879**	0.6785	**0.6830**
BHI	69,181	17,296	2	35	F1 Score	0.7614	**0.7824**	0.7519	**0.7673**

scenarios where categorical features and continuous features are commonly used. BHI is a real-world medical dataset to demonstrate the threats of label inference attacks in medical scenarios. Table 1 shows an overview of the datasets and their corresponding neural network architectures. The choices of hyperparameters are based on the standard models from ML literature, and all models use initialization methods in [19].

To fit these datasets for the scenario of VFL, we need to preprocess them. For CV datasets with image classification tasks including CIFAR-10, CIFAR-100 and CINIC-10, each sample (an image) is cut in half from the middle line with each participant holding a half, as the common setting for experiments of VFL [26, 28]. For Criteo, the dataset with categorical and numerical features, we first project all features into a hash space of 2^{13} dimensions, then cut the feature dimensions in half, and then the benign participant and the adversary both holds 2^{12} dimensions. While splitting raw features before projecting them into the hash space is a more realistic approach on this dataset, we choose the approach above to guarantee that the features assigned to each participant are valuable for the original task to some extent, which is corresponding to the assumption described in Section 3.2. For Yahoo Answers, the dataset for text classification, each sample (one paragraph of text) is split into two paragraphs with each participant holding one of them. As for BHI, each patient in the dataset has several medical image patches. Thus, we use it for both *two-participant and multi-participant* experiments. In both settings, we assign the available image patches of a patient with the same label to each participant in a round-robin manner.

For more details of these datasets, please refer to Appendix C.

4.2 Experiment Environment and Parameter Settings

All experiments are performed on a workstation equipped with Intel(R) Xeon(R) CPU E5-2650 v4 @ 2.20GHz, 32GB RAM, and four NVIDIA GTX 1080Ti GPU cards. We use PyTorch to implement DNNs. We assume the adversary has 40, 400, 40, 100, 100 and 70 auxiliary labeled samples for CIFAR-10, CIFAR-100, CINIC-10, Yahoo Answers, Criteo and BHI, respectively. As for other parameter settings of the passive and active label inference attacks, please refer to Appendix D.

5 Attack Evaluation

5.1 Attack Performance

With the settings in Section 4, on all the six evaluated datasets, the federated models get good performance with respect to the original tasks (top-1 accuracy of 0.8280, 0.7369, 0.7167, 0.7132 on CIFAR-10, CINIC-10, Yahoo Answers and Criteo, respectively; 0.7511 top-5 accuracy on CIFAR-100; 0.8340 F1 score on BHI). Then, in this section, we evaluate the performance of our label inference attacks.

Performance of the Passive Label Inference Attack. The experimental results of the passive label inference attack on the six datasets are reported in Table 2. Our attack achieves good performance with the top-1 accuracy/F1 score of around 0.75 on CIFAR-10, CINIC-10 and BHI. On the datasets with more difficult tasks such as CIFAR-100 and Criteo, our attack still achieves a good performance with a top-5/top-1 accuracy of around 0.70. On Yahoo Answers, our attack achieves a top-1 accuracy of around 0.65. Considering that the state-of-the-art performance on Yahoo Answers is a top-1 accuracy of 0.7762 [44], the performance of our attack is as expected on this dataset. Further, we find that by exploiting the labels inferred by the passive attack, the adversary can even gain additional power to infer other private information related to the labels. For details, please refer to Appendix E.

Performance of the Active Label Inference Attack. To show that our active label inference attack can further boost the attack performance, we use the malicious local optimizer in Section 3.4 to update the adversary's bottom model in the training stage. The attack performance is shown in Table 2. On all the six datasets, the active label inference attack outperforms the passive label inference attack. For example, on CINIC-10, the active attack boosts the top-1 inference accuracy from 0.7206 to 0.7818 on the training dataset and from 0.5440 to 0.5995 on the testing dataset. The main rea-

son is that the malicious local optimizer can make the top model rely more on the adversary's bottom model, and thus the adversary's bottom model learns a more indicative feature representation to predict the labels. More analysis and proofs are presented in Section 5.3.

Comparison to the Direct Semi-supervised Learning. The key insight for our passive and active label inference attacks is that the bottom model of the adversary is trained to transform the input features to an indicative representation to predict the labels. To show the capability of the trained bottom model, we compare the label inference accuracy when the adversary uses the direct semi-supervised learning, i.e., the adversary uses the same model architecture but with a randomly initialized bottom model instead. The same semi-supervised training strategy is employed with the same amount of auxiliary labeled samples. For simplicity, we here use CIFAR-10 as an example dataset to conduct the evaluation. The results are reported in Table 3. It can be seen that our passive label inference attack achieves much better label inference accuracy than the direct semi-supervised learning. As a particular case in Table 3, when the adversary has only ten labeled samples, i.e., one labeled sample for each class, the top-1 inference accuracy of the direct semi-supervised learning is only 0.1157 on the training dataset. This is as expected because it is almost impossible for the direct semi-supervised learning to fit well with so few labeled samples. However, with the same ten labeled samples, our passive label inference attack achieves a good top-1 inference accuracy of 0.6554. This proves the inference capability of the trained bottom model, as the only difference between our passive label inference attack and the direct semi-supervised learning is whether the bottom model has been pre-trained. This experiment also explains why our passive attack based on model completion is effective even with few auxiliary labels. Firstly, the inference capability of the trained bottom model is strong. Secondly, the state-of-the-art semi-supervised learning algorithms further empower the inference ability of the attacker's complete model.

Performance of the Direct Label Inference Attack. As stated in Section 3.5, the direct label inference attack is applicable for VFL frameworks without model splitting and can only infer the labels of samples in the training dataset. We evaluate its performance on all the six datasets and our experiment shows that the direct label inference attack achieves a top-1 accuracy of 1.0000 on every dataset, which is as expected because the direct label inference attack is based on deterministic mathematical derivation.

Impact on the Performance of the Original Task of a VFL Model. To be stealthy, attacks against VFL should not degrade the performance of its original task evidently. The direct/passive label inference attack does not attack in the training stage, so there is no degradation on the federated model's performance on the original task. However, the active label inference attack may influence the performance of the original task, since it manipulates the training process. To evaluate the overhead of the active label inference attack, we conduct experiments to compare the original task's performance on the test datasets with/without the active label inference attack. The results are shown in Table 4. It can be seen that the active label inference attack has a very small impact on the performance of the original task. For example, on CIFAR-10, with the active attack, the federated model's top-1 accuracy on the original task decreases by 0.0141, while on Criteo, the decrease is only 0.0004. On BHI, the active attack even slightly boosts the federated model's F1 score on the original task, from 0.8340 to 0.8504. The underlying reason is that though the active label inference attack tricks the federated model to rely more on the adversary's bottom model, it does not "force" the federated model to do so. If the adversary's bottom model cannot be optimized to improve the performance of the federated model, as discussed in Section 3.4, the malicious local optimizer will catch the signal of oscillation and then gradually stop the active attack by automatically decreasing the gradient scaling factor.

Table 3: The impact of the amount of auxiliary labeled data on CIFAR-10, and the attack performance comparison between the passive label inference attack and the direct semi-supervised learning. Attack performance is measured by top-1 accuracy.

Known Label Quantity	**Passive Label Inference**		**Direct Semi**	
	Training Dataset	Test Dataset	Training Dataset	Test Dataset
10	**0.6554**	**0.5235**	0.1157	0.1138
20	**0.7080**	**0.5542**	0.1187	0.1166
40	**0.8024**	**0.6299**	0.1698	0.1683
120	**0.8406**	**0.6305**	0.1866	0.1846
320	**0.8544**	**0.6392**	0.3286	0.3218

Table 4: The impact of the active label inference attack on the performance of the original task.

Dataset	**Metric**	**Model Performance under:**	
		No Attack	Active Attack
CIFAR-10	Top-1 Acc	**0.8280**	0.8139
CIFAR-100	Top-5 Acc	**0.7511**	0.7500
CINIC-10	Top-1 Acc	0.7369	**0.7400**
Yahoo Answers	Top-1 Acc	**0.7167**	0.7120
Criteo	Top-1 Acc	**0.7132**	0.7128
BHI	F1 Score	0.8340	**0.8504**

5.2 Sensitivity Evaluation

In this subsection, we study various of factors that may affect the performance of our passive and active attacks. Experimental results of more sensitivity evaluations can be found in the supplementary material of this paper [14].

Impact of the Quantity of the Adversary's Features. Our passive or active label inference attack trains a complete model based on the adversary's bottom model. Then the complete model is used to infer labels, taking the adversary's

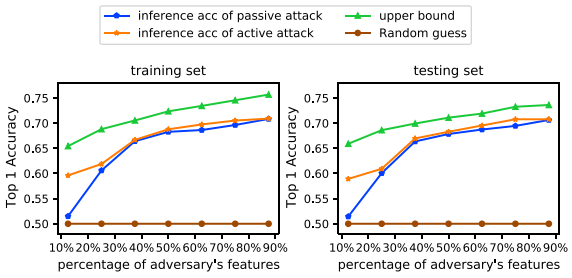


Figure 2: The impact of the quantity of the adversary's features.

available local data as input. Therefore, the quantity of the adversary's local features could determine the upper bound of the attack performance. We conduct experiments on Criteo to investigate the impact of the adversary's feature quantity. All the 2^{13} feature dimensions are firstly randomly sorted. Then, different fractions of the feature dimensions are assigned to the adversary. The upper bound is obtained using all the labels to directly train an inference model with the adversary's features. As shown in Figure 2, the adversary can hardly infer labels when he/she has only 12.5% of the features with the passive attack. However, with the active attack, the top-1 inference accuracy is boosted to around 0.60. When the adversary has over 35% of the features, the top-1 inference accuracy of both attacks is above 0.65. Further, as expected, the inference performance of both attacks is within the upper bound. We can draw three conclusions from this evaluation. First, generally, the more features the adversary has, the better the attack performance is. Second, our active label inference attack steadily outperforms the passive label inference attack, even when the adversary has a small quantity of features. Third, the attack performance of the passive or active attack is limited within the upper bound determined by the quantity of the adversary's features.

Amount of Auxiliary Labeled Data. Our passive and active label inference attacks need additional labeled samples to train the complete model via semi-supervised learning. The amount of available auxiliary labeled data may impact the attack performance. Therefore, we evaluate the accuracy of our passive label inference attack on CIFAR-10 with different amounts of auxiliary labeled samples. As shown in Table 3, more auxiliary labeled samples indeed increase the attack accuracy. However, as the number of auxiliary labeled samples grows, the attack accuracy increases more and more slowly. In practice, an attacker does not need a large amount of auxiliary labeled samples to get a reasonable attack accuracy. For example, 40 auxiliary labeled samples, which is much smaller comparing with the 50,000 training samples, help the passive attack reach a top-1 accuracy of 0.8024.

Multi-participant Experiment. As the number of participants increases in VFL, the contribution from each participant might be weakened. This indicates that the latent feature representation from an adversary might also be less indicative

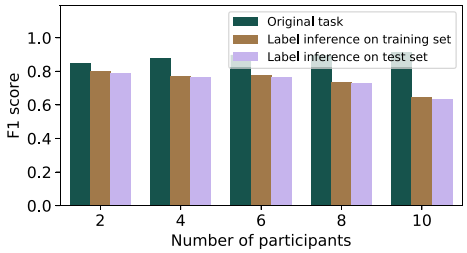
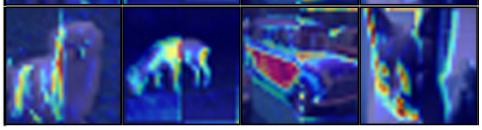


Figure 3: Performance of the active attack in multi-party setting on BHI.

Table 5: GradCAM visualization of the passive and active label inference attacks on CIFAR-10. The left half of image is the datum of the adversary.

Original Image in CIFAR-10	
GradCAM under Passive Label Inference Attack	
GradCAM under Active Label Inference Attack	

for label inference. To study this factor, we evaluate the performance of our active label inference attack with different number of participants on the BHI dataset. The input features are evenly divided among the participants.

As shown in Figure 3, the performance of the active label inference attack, measured by the F1 score, degrades when the number of participants increases. This is consistent with our expectation, as the top model's reliance on the features provided by the adversary's bottom model degrades. However, the F1 score of the active label inference attack is still over 0.70 when there are up to eight participants. When there are ten participants, the F1 score drops to around 0.60, while it is worth to mention that different from HFL, which may involve hundreds of participants in practice, VFL in practice usually has only several participants with two-participant VFL as the most common case [7]. The situations where a large number of participants join in VFL is rare.

Even under the multi-participant scenario, we find that the adversary can add overlapping features to his/her bottom model to boost the attack performance. VFL assumes that the features from the participants are disjoint and indeed participants usually agree on the feature set that each one should provide. However, in practice, it is very common for two participants to have overlapping features, e.g., a bank and a financial company are very likely to both have user income features. Therefore, the adversary can maliciously add the overlapping features to its bottom model to have better capability of inferring the labels. In our experiments with ten

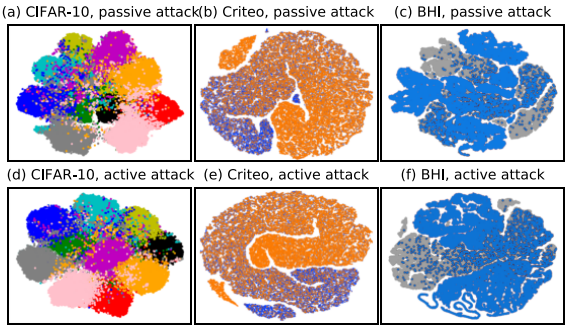


Figure 4: t-SNE projection of the outputs of the adversary's bottom model. Different color represents different labels.

participants, the adversary can only achieve a F1 score of 0.6446, if the features are evenly assigned to the participants. However, if the adversary intentionally adds 2 overlapping features, the F1 score is boosted to 0.7638.

5.3 Why the Active Label Inference Attack Works

To better understand how the active label inference attack boosts attack performance compared to the passive label inference attack, we use a visualization tool, GradCAM [42], on CIFAR-10. GradCAM can highlight the important regions in the image for predicting the class. We find that if the adversary uses the malicious local optimizer to update his/her bottom model in the training stage, the top model will pay more attention to the adversary's local data in the end. Table 5 shows some image samples in CIFAR-10 and their GradCAM results under the passive and active label inference attacks. As described in Section 4.1, the left half of the image is the datum of the adversary, while the right half of the image is the datum of the benign participant. It can be seen that after training with the malicious local optimizer, the whole federated model relies more on the datum of the adversary.

From above analysis, it can be further inferred that with the malicious local optimizer's help, the adversary's bottom model learns more indicative representations of his/her local features. To prove this, we use t-SNE [46] to visualize the distribution of the adversary's bottom model's outputs. As shown in Figure 4, the adversary's bottom model learns better in separating samples from each class with the active attack. As expected, the experimental results here is correlated to the active attack performance in Table 2. For example, from Figure 4 (a) and (d), it is evident that the bottom model learns much better representations with the active attack, which is corresponding to the fact shown in Table 2 that the active attack effectively boosts the top-1 inference accuracy from 0.8024 to 0.8484. However, on Criteo, the bottom model seems to learn slightly better representations, which is shown in Figure 4 (b) and (e). Correspondingly, Table 2 shows that although the active attack successfully boosts the attack performance (from 0.6828 to 0.6879, top-1 accuracy), the boost is not as effective as that on CIFAR-10.

To sum up, with the active attack, the adversary gets a trained bottom model containing more information about labels. Thus, after the model completion, the complete model performs better in predicting the labels, bringing better label inference performance.

6 Defenses

6.1 Possible Defenses

Some defense approaches may mitigate our label inference attacks. In the training process of VFL, the only information sent to the adversary is the gradient from the server. Thus, defensive strategies can be applied to the gradients to prevent information leakage from the server to the adversary. We consider four possible defense approaches against label inference attacks: noisy gradients, gradient compression, privacy-preserving deep learning and discreteSGD (a customized version of signSGD [4]). These defense approaches are commonly used by prior works to mitigate possible information leakage in FL [20, 30, 43, 62].

Noisy Gradients. As discussed in [62], adding noise to gradients is a common defense strategy in FL. In VFL, the server can add laplacian noise to gradients before sending them to participants.

Gradient Compression. Another defense strategy is sharing fewer gradients, which is also known as gradient compression. Gradient compression is a strategy designed both for communication efficiency and privacy protection [21]. Its key idea is only sharing a proportion of gradients with the largest absolute values. HFL can still produce highly-accurate global model even when only 10% of the gradient values are shared [43].

Privacy-preserving Deep Learning. Privacy-preserving deep learning is a comprehensive privacy-enhancing method introduced in [43]. It includes three defense strategies: differential privacy, gradient compression, and random selection. In each iteration, the server does the following steps to protect the gradients: (1) randomly selects one gradient value, generates noise, and adds the noise to the gradient value; (2) if the gradient value after adding noise is larger than a threshold value τ, keeps it, otherwise sets it to zero; (3) loops the first two steps until θ_u fraction of gradient values are gathered. Both θ_u and τ are hyperparameters to balance the trade-off between model performance and defense performance.

DiscreteSGD. SignSGD [4] is proposed to reduce the communication cost among workers in HFL and prevent privacy leakage caused by gradients. Since it only preserves the signs of gradients, it further boosts communication efficacy and enhances privacy protection. However, it is not proper to naively apply signSGD to VFL. The reason is that in HFL, the shared gradients from the sever to the participant include the updates of all the model parameters, while the shared gradients in VFL only include the gradients of the loss w.r.t. the outputs of

a participant's bottom model. A participant needs to do local backward propagation based on the shared gradients to get updates of all parameters of his/her bottom model. Therefore, VFL is more sensitive to modifications on the shared gradients. Our preliminary experiment shows that directly applying signSGD to VFL is very likely to make a VFL model fail to converge on the original task.

With the above consideration, we evaluate a customized version of signSGD by keeping partial magnitude information of the shared gradients, which we name as *discreteSGD*, as a possible defense. Specifically, the server first observes the distribution of the shared gradients in the first epoch. The mean and the standard deviation of the distribution are denoted as μ and σ, respectively. According to the three-sigma rule [37], the server sets an interval as $[\mu - 2\sigma, \mu + 2\sigma]$. The gradients outside of the interval are regarded as outliers and discarded. Then, the server slices the interval into N sub intervals. In the following training process, before sending gradients to a participant, the server first rounds each gradient value to the nearest endpoint of the sub intervals, e.g., a gradient value of 1.6 with sub intervals [0, 1] and [1, 2] will be rounded to the endpoint 2. The hyperparameter N controls how much magnitude information of the shared gradients is preserved. While VFL fails to converge with signSGD, discreteSGD is a more suitable defense solution. As shown in Figure 5 (j)-(l), our experiment proves that with N set to a reasonable value like 24, discreteSGD brings little degradation to a VFL model's performance.

6.2 Defense Evaluation

Defense Against the Active Label Inference Attack. We evaluate the four defense approaches introduced in Section 6.1 against label inference attacks on three datasets: CIFAR-10, Criteo and BHI. Similar experiments can be extended to other datasets. The results are reported in Figure 5.

From Figure 5 (a)-(c), for noisy gradients, we do experiments on several scales of laplacian noise to evaluate its defense performance against the active label inference attack. It can be seen that noise at a small scale cannot mitigate the risk of label leakage. Large-scale noise can successfully mitigate label inference attacks, but with the cost of significantly degrading the federated model's performance on the original task. A funny thing is that on CIFAR-10, the noise with a scale of $1e-3$ even makes our attack stronger, possibly because noise at this scale has a similar effect with adversarial training [31] that makes the federated model learn a more robust representation of the adversary's data.

From Figure 5 (d)-(f), for gradient compression, we evaluate its defense performance with different compression rates. On CV datasets such as CIFAR-10 and BHI, gradient compression can successfully mitigate label inference attacks, but with the cost of significantly degrading the federated model's performance on the original task. For example, on BHI, when

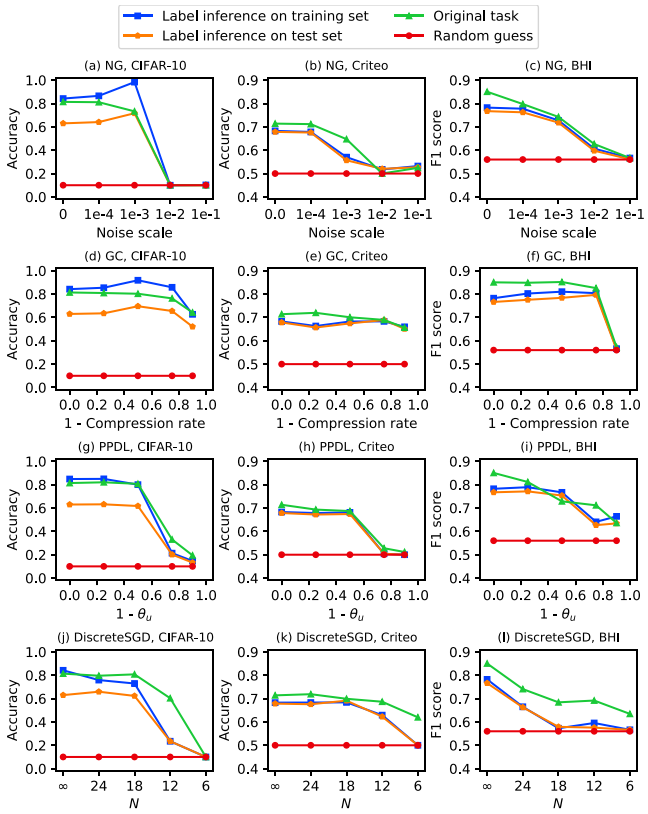


Figure 5: Defense performance of four defense approaches with different parameter settings against the active label inference attack. (a)-(c): noisy gradients (NG), (d)-(f): gradient compression (GC), (g)-(i): privacy-preserving deep learning (PPDL) and (j)-(l): discreteSGD. The symbol ∞ in (j)-(l) represents keeping gradients continuous, i.e., not applying discreteSGD.

the compression rate is 0.9, both the F1 score of the adversary's inference performance and the federated model's F1 score on the original task decrease by around 0.30. On dataset with numerical and categorical features like Criteo, gradient compression seems to have no apparent effect. As shown in Figure 5 (e), both the federated model's performance on the original task and the attack performance have no significant change as the compression rate varies from 0.1 to 0.9.

From Figure 5 (g)-(i), for privacy-preserving deep learning, we evaluate its defense performance with different settings of the hyperparameter θ_u. On all three datasets, the defense of privacy-preserving deep learning can mitigate label inference attacks with the hyperparameter θ_u set to 0.25 or lower. However, the degradation of the federated model's performance on the original task is also significant, which is different from the results reported on HFL [43]. We believe that this is related to the different nature of HFL and VFL. As mentioned in 6.1, VFL is more sensitive to modifications on the shared gradients from the server to participants. This explains why privacy-preserving deep learning might be a good defense for HFL but not suitable for VFL.

Table 6: Defenses against the direct label inference attack on CIFAR-10.

Defense Approach	Parameter	Parameter Set Value	Model Accuracy	Attack Accuracy
Noisy Gradients	Noise Scale	1e-4	0.8347	0.8063
		1e-3	0.8318	0.4906
		1e-2	0.7191	0.2452
		1e-1	0.1000	0.1265
Gradient Compression	Compression Rate	75%	0.8248	0.9997
		50%	0.8259	0.9931
		25%	0.8049	0.9245
		10%	0.1000	0.0058
Privacy-preserving Deep Learning	θ_u	0.75	0.8189	0.3904
		0.50	0.8216	0.3891
		0.25	0.1993	0.0972
		0.10	0.1000	0.0430
Discrete SGD	N	24	0.8145	0.9763
		18	0.7962	0.9330
		12	0.7471	0.9399
		6	0.6575	0.9087

We evaluate the defense performance of discreteSGD with different settings of the hyperparameter N in Figure 5 (j)-(l). The results are similar to the above three possible defenses. On all the three datasets, as the hyperparameter N decreases (the gradients become more discretized), both the attack performance and the federated model's performance on the original task degrade significantly, which shows that discreteSGD cannot effectively defend against the active label inference attack. We believe that this is due to the inner nature of our label inference attacks. As our attacks extract information about labels from a part of the federated model, i.e., one of the bottom models, the attack performance is highly correlated to how well the adversary's bottom model is trained. However, in spite of this inherent trade-off, discreteSGD seems to be able to reduce the cost, i.e., the degradation of the federated model's performance on the original task. For example, on CIFAR-10, assuming that the defense goal is to decrease the adversary's top-1 inference accuracy on the training/testing dataset to around 0.2500; privacy-preserving deep learning can achieve this goal with the cost that the federated model's top-1 accuracy on the original task degrades from 0.8139 to around 0.4000. While for discreteSGD, the degradation is from 0.8139 to around 0.6000. Although the cost is still high, discreteSGD outperforms the above three defenses in this aspect, showing that gradient discretization is possibly a more promising defense against the label inference attacks.

Defense Against the Direct Label Inference Attack. We evaluate the four defense approaches against the direct label inference attack on CIFAR-10 (similar evaluation can be extended to other datasets). The results are reported in Table 6. All the evaluated defense approaches except discreteSGD can mitigate the direct label inference attack. This is as expected because the direct label inference attack relies on the signs of gradients, so any modification on the gradients' signs, e.g., adding noise to flip the gradients' signs and pruning some gradients to zero, can effectively defend against this attack. In our experiments, privacy-preserving deep learning achieves the best defense performance since it degrades the top-1 accuracy of the direct label inference attack from 1.0000 to 0.3891 without decreasing the federated model's accuracy on the original task. Gradient compression seems to have the advantage of maintaining the performance of the original task. However, gradient compression cannot effectively degrade the attack performance, which is due to that it prefers to prune gradients with small absolute values to zero, but the gradient related to the ground truth label may not be included in these small gradients. Noisy gradients can significantly mitigate the label inference attack, since adding sufficient scale of noise is likely to flip the signs of gradients. Its side effect is that noise of an over large scale harms the performance of the original task. Privacy-preserving deep learning combines gradient compression and noisy gradients. As a result, with a suitable scale of noise and compression rate, it can effectively mitigate the direct label inference attack while maintaining the performance of the original task. As for discreteSGD, it is not effective against the direct label inference attack because it hardly changes the signs of gradients.

Summary. We evaluate four mainstream defense approaches including a customized version of signSGD that is popular in the recent two years. The experiment results show that though some of these defense approaches can effectively mitigate the direct label inference attack, they are not effective against the passive and active label inference attacks. However, actually, even if the direct label inference attack is mitigated, it can still infer the labels for a decent number of samples. With the help of these inferred labels, the adversary can simultaneously use the passive label inference attack to get a better attack performance. Therefore, our label inference attacks reveal the urgency of mitigating the label privacy risks in real-world VFL applications.

7 Future Work

Evaluation on Other ML Models. This paper focuses on VFL with DNNs as the ML model, while we have not evaluated VFL built with other types of ML models, such as VFL with logistic regression [7, 18, 28, 56], and VFL with gradient boosting tree models [9]. In addition, our experiments are conducted on convolutional neural networks, fully connected neural networks and transformers, but other types of neural networks like graph neural networks are also used for VFL [61]. VFL is a fast-developing research area both in the academy and industry. More ML models and training algorithms will be modified to serve VFL. Therefore, it is interesting to evaluate the privacy and security risks of VFL built with other ML models.

Towards Better VFL. As shown in Section 6, mainstream defense strategies are not effective when faced with our label

inference attacks against VFL. This calls for new defense strategies designed for VFL. In addition, is there any other privacy or security risk of the current VFL framework? How to defend VFL against the possible information leakage and security risks? These are open questions worth exploring.

8 Related Work

8.1 Information Leakage in FL

Prior works have proposed various attacks to exploit possible data leakage in HFL, including membership inference attacks, property inference attacks, class representative reconstruction attacks, and dataset reconstruction attacks. Our label inference attacks are different from these attacks. First, our work focuses on the data leakage in VFL, while to the best of our knowledge, prior works focus on the data leakage in HFL. Second, our work shows a new type of data leakage – label leakage, while prior works focus on inferring membership, sample property, and sample features.

Inferring Membership. Membership inference attacks aim to infer whether a particular sample appears in the training dataset [1, 38, 41]. Melis et al. [30] showed that in HFL, if the federated model uses embedding layers for text data, the curious server can infer whether a certain word appears in a participant's data by observing the non-zero gradients of the embedding layer. Nasr et al. [33] demonstrated that an adversarial participant in HFL can also infer the membership of other participants by exploiting the privacy vulnerabilities of the gradient descent algorithm, and the adversarial participant can craft adversarial model updates to gain more information about the membership of other participants' datasets.

Inferring Class Representatives. Hitaj et al. [20] demonstrated that an adversarial participant in HFL can use a generative adversarial network (GAN) to reconstruct the class representatives of other participants, and by submitting malicious model updates, an adversary can deceive other participants into leaking more information about their data.

Inferring Properties. Melis et al. [30] demonstrated that an adversarial participant in HFL can infer the properties of other participants' input samples by leveraging the exchanging gradients, and the properties are not limited to be related to the original task. For example, the original task is gender classification, but the adversary can infer races of training samples.

Reconstructing Training Samples. A more challenging attack is to completely reconstruct training samples of participants in FL. Zhu et al. [62] showed that exchanging gradients in collaborative learning can be exploited to do such an attack. They assumed the server is curious about the data of participants, and showed that by optimizing dummy data to match the gradients submitted by participants, the server can reconstruct the *exact* data of every participant. Wang et al. [51] demonstrated that an adversarial participant can use a GAN with a multi-task discriminator to simultaneously discriminate the participant identity of input samples and reconstruct the samples of other participants.

8.2 Other Attacks Against FL

Backdoor Attacks. Backdoor attack, also known as trojaning attack against ML [6, 17, 27, 36], is a hot research topic in recent years. A trojaned ML model behaves normally with normal input samples, but if the input sample has a certain *trojan trigger*, the trojaned model will have abnormal behaviors. Bagdasaryan et al. [2] first showed that HFL faces the danger of backdoor attacks: a malicious participant can trojan the global model by scaling up model updates. Xie et al. [55] considered the scenario where there are multiple compromised adversarial participants, and proposed the distributed backdoor attack (DBA) against HFL.

Byzantine Attacks. HFL also faces possible threats of byzantine attacks. In HFL, one or more participants may submit abnormal intermediate results to the server so that the server cannot get a converged model [13].

8.3 Defense in FL

Mainstream defense approaches aiming at mitigating information leakage from gradients includes adding noise to gradients [62], gradient compression [25], and randomly pruning part of the gradients to zero [43]. Privacy-preserving deep learning [43] is a defense approach combining all the three mechanisms. However, our experiments in Section 6 show that these defense approaches are not effective against our passive and active label inference attacks. SignSGD [4] is a recent defense for privacy enhancement in HFL. However, our preliminary experiment shows that directly applying signSGD to VFL is very likely to make a VFL model fail to converge on the original task. Therefore, we design a customized version of signSGD, named discreteSGD for VFL. Although the cost (the degradation of the federated model's performance) is still high, discreteSGD does better than the above three defenses, showing that gradient discretization is possibly a more promising defense against our passive and active label inference attacks. Another line of works studied how to securely select and aggregate the submits from the participants [35, 58]. However, these works focus on defending against byzantine and backdoor attacks in HFL, and are not suitable to defend against our label inference attacks against VFL.

9 Conclusion

Most prior works on the privacy and security of FL focus on HFL. However, as VFL is applied more and more widely in industry, it is also urgent to evaluate the privacy and security risks of VFL. To bridge this gap, we conduct the first investigation on the privacy risks in VFL. We reveal and shed lights on the new and unique label leakage issue of VFL. As labels are

likely to be highly sensitive information, e.g., whether someone has a disease, inferring labels can be a severe privacy violation.

We design three kinds of label inference attacks covering several practical settings of VFL. We also provide insights into how the active attack affects the training of the vertical federated model with visualization techniques.

Our experiments show that the proposed label inference attacks are powerful against VFL on real-world large-scale datasets. We also evaluate possible defenses including gradient compression, noisy gradients, privacy-preserving deep learning and discreteSGD. Though some of these defenses can effectively mitigate the threat of the direct label inference attack, they are not effective against our passive and active label inference attacks. This should motivate future works on better defenses.

10 Acknowledgements

This work was partly supported by the National Key Research and Development Program of China under No. 2020YFB2103802, NSFC under No. 62102360, 61772466, 62072406, U1836202, and U1936215, and the Zhejiang Provincial Natural Science Foundation for Distinguished Young Scholars under No. LR19F020003. Ting Wang is partially supported by the National Science Foundation under Grant No. 1953893, 1953813, and 1951729.

References

[1] M. Backes, P. Berrang, M. Humbert, and P. Manoharan. Membership privacy in MicroRNA-based studies. In *CCS*, 2016.

[2] E. Bagdasaryan, A. Veit, Y. Hua, D. Estrin, and V. Shmatikov. How to backdoor federated learning. In *AISTATS*, 2020.

[3] Baidu. PaddleFL. https://github.com/PaddlePaddle/PaddleFL.

[4] J. Bernstein, Y. Wang, K. Azizzadenesheli, and A. Anandkumar. signSGD: Compressed optimisation for non-convex problems. In *ICML*, 2018.

[5] D. Berthelot, N. Carlini, I. Goodfellow, N. Papernot, A. Oliver, and C. Raffel. MixMatch: A holistic approach to semi-supervised learning. In *NeurIPS*, 2019.

[6] A. N. Bhagoji, S. Chakraborty, P. Mittal, and S. Calo. Analyzing federated learning through an adversarial lens. In *ICML*, 2019.

[7] C. Chen, J. Zhou, L. Wang, X. Wu, W. Fang, J. Tan, L. Wang, X. Ji, A. Liu, H. Wang, et al. When homomorphic encryption marries secret sharing: Secure large-scale sparse logistic regression and applications in risk control. *arXiv preprint arXiv:2008.08753*, 2020.

[8] J. Chen, Z. Yang, and D. Yang. MixText: Linguistically-informed interpolation of hidden space for semi-supervised text classification. In *ACL*, 2020.

[9] K. Cheng, T. Fan, Y. Jin, Y. Liu, T. Chen, and Q. Yang. SecureBoost: A lossless federated learning framework. *arXiv preprint arXiv:1901.08755*, 2019.

[10] A. Cruz-Roa, A. Basavanhally, F. González, H. Gilmore, M. Feldman, S. Ganesan, N. Shih, J. Tomaszewski, and A. Madabhushi. Automatic detection of invasive ductal carcinoma in whole slide images with convolutional neural networks. In *Proceedings of the Medical Imaging 2014: Digital Pathology*, volume 9041, 2014.

[11] L. N. Darlow, E. J. Crowley, A. Antoniou, and A. J. Storkey. CINIC-10 is not ImageNet or CIFAR-10. *arXiv preprint arXiv:1810.03505*, 2018.

[12] J. Devlin, M. Chang, K. Lee, and K. Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[13] M. Fang, X. Cao, J. Jia, and N. Gong. Local model poisoning attacks to byzantine-robust federated learning. In *USENIX Security*, 2020.

[14] C. Fu, X. Zhang, S. Ji, J. Chen, J. Wu, S. Guo, J. Zhou, A. Liu, and T. Wang. Supplementary material for the paper "Label Inference Attacks Against Vertical Federated Learning". https://github.com/FuChong-cyber/label-inference-attacks/.

[15] Google. TensorFlow Federated. https://www.tensorflow.org/federated.

[16] B. Gu, Z. Dang, X. Li, and H. Huang. Federated doubly stochastic kernel learning for vertically partitioned data. In *KDD*, 2020.

[17] T. Gu, B. Dolan-Gavitt, and S. Garg. BadNets: Identifying vulnerabilities in the machine learning model supply chain. *arXiv preprint arXiv:1708.06733*, 2017.

[18] S. Hardy, W. Henecka, H. Ivey-Law, R. Nock, G. Patrini, G. Smith, and B. Thorne. Private federated learning on vertically partitioned data via entity resolution and additively homomorphic encryption. *arXiv preprint arXiv:1711.10677*, 2017.

[19] K. He, X. Zhang, S. Ren, and J. Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *ICCV*, 2015.

[20] B. Hitaj, G. Ateniese, and F. Perez-Cruz. Deep models under the GAN: information leakage from collaborative deep learning. In *CCS*, 2017.

[21] P. Kairouz, H. B. McMahan, B. Avent, A. Bellet, M. Bennis, A. N. Bhagoji, K. Bonawitz, Z. Charles, G. Cormode, R. Cummings, et al. Advances and open problems in federated learning. *arXiv preprint arXiv:1912.04977*, 2019.

[22] A. Krizhevsky, G. Hinton, et al. Learning multiple layers of features from tiny images. 2009.

[23] Criteo Labs. Criteo dataset. https://labs.criteo.com/2014/02/download-kaggle-display-advertising-challenge-dataset/.

[24] Q. Li, Z. Wen, Z. Wu, S. Hu, N. Wang, and B. He. A survey on federated learning systems: vision, hype and reality for data privacy and protection. *arXiv preprint arXiv:1907.09693*, 2019.

[25] Y. Lin, S. Han, H. Mao, Y. Wang, and W. J. Dally. Deep gradient compression: Reducing the communication bandwidth for distributed training. *arXiv preprint arXiv:1712.01887*, 2017.

[26] Y. Liu, Y. Kang, X. Zhang, L. Li, Y. Cheng, T. Chen, M. Hong, and Q. Yang. A communication efficient collaborative learning framework for distributed features. *arXiv preprint arXiv:1912.11187*, 2019.

[27] Y. Liu, S. Ma, Y. Aafer, W. Lee, J. Zhai, W. Wang, and X. Zhang. Trojaning attack on neural networks. In *NDSS*, 2018.

[28] Y. Liu, X. Zhang, and L. Wang. Asymmetrically vertical federated learning. *arXiv preprint arXiv:2004.07427*, 2020.

[29] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas. Communication-efficient learning of deep networks from decentralized data. In *Artificial Intelligence and Statistics*, 2017.

[30] L. Melis, C. Song, E. De Cristof, and V. Shmatikov. Exploiting unintended feature leakage in collaborative learning. In *IEEE S&P*, 2019.

[31] T. Miyato, S. Maeda, M. Koyama, and S. Ishii. Virtual adversarial training: A regularization method for supervised and semi-supervised learning. *TPAMI*, 2019.

[32] P. Mooney. Breast histopathology images. https://www.kaggle.com/paultimothymooney/breast-histopathology-images.

[33] M. Nasr, R. Shokri, and A. Houmansadr. Comprehensive privacy analysis of deep learning. In *IEEE S&P*, 2019.

[34] OpenMined. PySyft. https://github.com/OpenMined/PySyft.

[35] X. Pan, M. Zhang, D. Wu, Q. Xiao, S. Ji, and Z. Yang. Justinian's GAAvernor: Robust distributed learning with gradient aggregation agent. In *USENIX Security*, 2020.

[36] R. Pang, H. Shen, X. Zhang, S. Ji, Y. Vorobeychik, X. Luo, A. Liu, and T. Wang. A tale of evil twins: Adversarial inputs versus poisoned models. In *CCS*, 2020.

[37] F. Pukelsheim. The three sigma rule. *The American Statistician*, 1994.

[38] A. Pyrgelis, C. Troncoso, and E. De Cristofaro. Knock knock, who's there? Membership inference on aggregate location data. In *NDSS*, 2018.

[39] Soumik Rakshit. Yahoo answers dataset. https://www.kaggle.com/soumikrakshit/yahoo-answers-dataset.

[40] M. Riedmiller and H. Braun. A direct adaptive method for faster backpropagation learning: The rprop algorithm. In *IEEE international conference on neural networks*, 1993.

[41] A. Salem, Y. Zhang, M. Humbert, P. Berrang, M. Fritz, and M. Backes. ML-leaks: Model and data independent membership inference attacks and defenses on machine learning models. *arXiv preprint arXiv:1806.01246*, 2018.

[42] R. R. Selvaraju, M. Cogswell, A. Das, R. Vedantam, D. Parikh, and D. Batra. Grad-CAM: Visual explanations from deep networks via gradient-based localization. In *ICCV*, 2017.

[43] R. Shokri and V. Shmatikov. Privacy-preserving deep learning. In *CCS*, 2015.

[44] C. Sun, X. Qiu, Y. Xu, and X. Huang. How to fine-tune BERT for text classification? In *China National Conference on Chinese Computational Linguistics*, 2019.

[45] I. Sutskever, J. Martens, G. Dahl, and G. Hinton. On the importance of initialization and momentum in deep learning. In *ICML*, 2013.

[46] L. Van Der Maaten and G. Hinton. Visualizing data using t-SNE. *Journal of machine learning research*, 2008.

[47] P. Vepakomma, O. Gupta, T. Swedish, and R. Raskar. Split learning for health: Distributed deep learning without sharing raw patient data. *arXiv preprint arXiv:1812.00564*, 2018.

[48] P. Voigt and A. Von dem Bussche. The EU general data protection regulation (GDPR). *A Practical Guide, 1st Ed., Cham: Springer International Publishing*, 2017.

[49] W. N. Street W. H. Wolberg and O. L. Mangasarian. Breast cancer wisconsin dataset. https://archive.ics.uci.edu/ml/datasets/Breast+Cancer+Wisconsin+(Diagnostic).

[50] G. Wang. Interpret federated learning with shapley values. *arXiv preprint arXiv:1905.04519*, 2019.

[51] Z. Wang, M. Song, Z. Zhang, Y. Song, Q. Wang, and H. Qi. Beyond inferring class representatives: User-level privacy leakage from federated learning. In *IEEE INFOCOM*, 2019.

[52] Webank. A case of traffic violations insurance-using federated learning. https://www.fedai.org/cases/.

[53] Webank. Federated AI Technology Enabler (FATE). https://fate.fedai.org/.

[54] Webank. Utilization of FATE in risk management of credit in small and micro enterprises. https://www.fedai.org/cases/.

[55] C. Xie, K. Huang, P. Chen, and B. Li. DBA: Distributed backdoor attacks against federated learning. In *ICLR*, 2020.

[56] S. Yang, B. Ren, X. Zhou, and L. Liu. Parallel distributed logistic regression for vertical federated learning without third-party coordinator. *arXiv preprint arXiv:1911.09824*, 2019.

[57] T. Yang, G. Andrew, H. Eichner, H. Sun, W. Li, N. Kong, D. Ramage, and F. Beaufays. Applied federated learning: Improving google keyboard query suggestions. *CoRR*, abs/1812.02903, 2018.

[58] D. Yin, Y. Chen, R. Kannan, and P. Bartlett. Byzantine-robust distributed learning: Towards optimal statistical rates. In *Proceedings of Machine Learning Research*, 2018.

[59] W. Zhang, Z. Li, and X. Chen. Quality-aware user recruitment based on federated learning in mobile crowd sensing. *Tsinghua Science and Technology*, 2021.

[60] B. Zhao, K. R. Mopuri, and H. Bilen. iDLG: Improved deep leakage from gradients. *arXiv preprint arXiv:2001.02610*, 2020.

[61] J. Zhou, C. Chen, L. Zheng, X. Zheng, B. Wu, Z. Liu, and L. Wang. Privacy-preserving graph neural network for node classification. *arXiv preprint arXiv:2005.11903*, 2020.

Algorithm 2 Training VFL with model splitting

Require: Top model parameters θ_{top}, bottom model parameters of K participants $\theta_1, \theta_2, ..., \theta_K$, learning rate η, ground-truth label y.

Server executes:
  Initialize θ_{top} and $\theta_1, \theta_2, ..., \theta_K$
  while stopping epoch not met **do**
    for each batch b of sample Ids **do**
      for $k = 1$ to K **do**
        $o_k \leftarrow$ **ParticipantForwardProp**(θ_k, b)
      end for
      $o_{all} \leftarrow Concat(o_1, ..., o_k)$
        ▷ concatenating bottom model outputs
      $o_{final} \leftarrow \theta_{top}(o_{all})$
      $L \leftarrow LossFunc(o_{final}, y)$
      $g_{top} \leftarrow \frac{\partial L}{\partial \theta_{top}}$ ▷ updating top model
      $\theta_{top} \leftarrow \theta_{top} - \eta \cdot g_{top}$
      for $k = 1$ to K **do**
        $g_k \leftarrow \frac{\partial L}{\partial o_k}$
        ParticipantBackProp(θ_k, g_k, o_k)
      end for
    end for
  end while

ParticipantForwardProp(θ, b):
  return bottom model forward outputs $\theta(b)$

ParticipantBackProp(θ, g_o, o):
  $g \leftarrow g_o \cdot \frac{\partial o}{\partial \theta}$
  $\theta \leftarrow \theta - \eta \cdot g$ ▷ updating bottom model

[62] L. Zhu, Z. Liu, and S. Han. Deep leakage from gradients. In *NeurIPS*, 2019.

Appendix

A Training Algorithms of VFL

The training algorithms of VFL with/without model splitting are shown in Algorithm 2 and Algorithm 3, respectively.

B The Customized MixMatch

As discussed in Section 4, in order to make MixMatch applicable to tasks beyond the CV domain, we use a customized version of MixMatch without data augmentation. The algorithm of the customized MixMatch is shown in Algorithm 4, where MixUp is a data processing algorithm described in Algorithm 5, and *Sharpen* is the sharpening function used to reduce the entropy of the label distribution. The function *Sharpen* is defined as:

Algorithm 3 Training VFL without model splitting

Require: bottom model parameters of K participants $\theta_1, \theta_2, ..., \theta_K$, learning rate η, ground-truth label y.

Server executes:
  Initialize $\theta_1, \theta_2, ..., \theta_K$
  while stopping epoch not met **do**
    for each batch b of sample Ids **do**
      for $k = 1$ to K **do**
        $o_k \leftarrow$ **ParticipantForwardProp**(θ_k, b)
      end for
      $o_{sum} \leftarrow \sum_k o_k$
        ▷ aggregating bottom model outputs
      $L \leftarrow LossFunc(o_{sum}, y)$
      for $k = 1$ to K **do**
        $g_k \leftarrow \frac{\partial L}{\partial o_k}$
        ParticipantBackProp(θ_k, g_k, o_k)
      end for
    end for
  end while

ParticipantForwardProp(θ, b):
  return bottom model forward outputs $\theta(b)$

ParticipantBackProp(θ, g_o, o):
  $g \leftarrow g_o \cdot \frac{\partial o}{\partial \theta}$
  $\theta \leftarrow \theta - \eta \cdot g$ ▷ updating bottom model

$$Sharpen(p,T)_i := \frac{p_i^{\frac{1}{T}}}{\sum_{j=1}^{L} p_j^{\frac{1}{T}}} \tag{4}$$

where p is the discrete distribution, and T is a hyperparameter.

In each training iteration, the customized MixMatch generates a collection of processed labeled examples $\mathcal{X}'$ and a collection of processed unlabeled examples with guessed labels $\mathcal{U}'$. Then, it uses $\mathcal{X}'$ and $\mathcal{U}'$ to separately compute the labeled loss term $\mathcal{L}_{\mathcal{X}}$ and unlabeled loss term $\mathcal{L}_{\mathcal{U}}$:

$$\mathcal{L}_{\mathcal{X}} = \frac{1}{|\mathcal{X}'|} \sum_{x,p \in \mathcal{X}'} CE(p, p_{model}(y \mid x; \theta)) \tag{5}$$

$$\mathcal{L}_{\mathcal{U}} = \frac{1}{N|\mathcal{U}'|} \sum_{u,q \in \mathcal{U}'} \|q - p_{model}(y \mid u; \theta)\|_2^2 \tag{6}$$

where $CE(p,q)$ is the cross-entropy between distributions p and q, and N is the number of classes. Finally, the combined loss is computed as:

$$\mathcal{L} = \mathcal{L}_{\mathcal{X}} + \lambda_{\mathcal{U}} \mathcal{L}_{\mathcal{U}} \tag{7}$$

where $\lambda_{\mathcal{U}}$ is hyperparameters balancing the importance of labeled/unlabeled data. After obtaining the combined loss, the neural network model can be updated using backward propagation.

Algorithm 4 The customized MixMatch. MixMatch takes a batch of labeled data $\mathcal{X}$ and a batch of unlabeled data $\mathcal{U}$, then produces a collection of processed labeled examples $\mathcal{X}'$ and a collection of processed unlabeled examples with guessed labels $\mathcal{U}'$.

Input: Batch of labeled examples and their one-hot labels $\mathcal{X} = \big((x_b, p_b); b \in (1, ..., B)\big)$, batch of unlabeled examples $\mathcal{U} = \big(u_b; b \in (1, ..., B)\big)$, sharpening temperature T, Beta distribution parameter α for MixUp.
for $b = 1$ to B **do**
  $q_b = p_{model}(y \mid u_b; \theta)$
  $q_b = Sharpen(q_b, T)$
end for
$\hat{\mathcal{U}} = \big((u_b, q_b); b \in (1, \ldots, B)\big)$
$\mathcal{W} = Shuffle\big(Concat(\mathcal{X}, \hat{\mathcal{U}})\big)$
$\mathcal{X}' = \big(MixUp(\mathcal{X}_i, \mathcal{W}_i); i \in (1, \ldots, |\mathcal{X}|)\big)$
  ▷ Apply *MixUp* to labeled data and entries from $\mathcal{W}$
$\mathcal{U}' = \big(MixUp(\hat{\mathcal{U}}_i, \mathcal{W}_{i+|\mathcal{X}|}); i \in (1, \ldots, |\hat{\mathcal{U}}|)\big)$
  ▷ Apply *MixUp* to unlabeled data and the rest of $\mathcal{W}$
return $\mathcal{X}', \mathcal{U}'$

Algorithm 5 MixUp. MixUp takes two samples with their corresponding labels probabilities $(x_1, p_1), (x_2, p_2)$, and produces a "mixed" sample with its "mixed" labels probabilities x', p'.

Input: Two samples with their corresponding labels probabilities $(x_1, p_1), (x_2, p_2)$, Beta distribution parameter α.
$\lambda \sim Beta(\alpha, \alpha)$
$\lambda' = max(\lambda, 1 - \lambda)$
$x' = \lambda' x_1 + (1 - \lambda') x_2$
$p' = \lambda' p_1 + (1 - \lambda') p_2$
return x', p'

C Details About Datasets

Here is the detailed introduction of all the six datasets used in this paper, including Criteo, Yahoo Answers, CINIC-10, CIFAR-10, CIFAR-100 and BHI.

CIFAR-10 and CIFAR-100. CIFAR-10 [22] is a classic dataset built for image classification. It consists of 10 classes and 60,000 images in the size of 32×32 pixels. CIFAR-100 has the same size as CIFAR-10 but it has 100 classes (600 images each). Following the common practice of using CIFAR-10 or CIFAR-100, we take 50,000 samples as the training dataset and other 10,000 as the test dataset.

CINIC-10. CINIC-10 [11] is an extension of CIFAR-10 via the addition of down sampled ImageNet images, whose name means "CINIC-10 is not ImageNet nor CIFAR-10". It consists of 10 classes and 270,000 images (4.5 times of CIFAR-10) in the size of 32×32 pixels. Following the common practice of using CINIC-10, we take 180,000 samples as the training dataset and other 90,000 as the test dataset.

Criteo. Criteo [23] contains 7 days of click-through data

Table 7: Default settings for key parameters.

(a) Parameters of MixMatch.

Parameter	Description	Value
λ_u	Label guessing parameter	50
T	Sharpening parameter	0.8

(b) Parameters of the malicious local optimizer.

Parameter	Description	Value
β	Momentum parameter	0.9
γ	Resetting parameter	1.0
r_{max}	Maximum acceleration rate	5.0
r_{min}	Minimum acceleration rate	1.0

shared by Criteo Labs, which is widely used for CTR prediction benchmarking (given a user and the pages he/she visited, predict whether he/she will click on a given ad). It contains 26 anonymous categorical features and 13 continuous features. The 39 features are projected into a hash space of 2^{13} dimensions before sent to the federated model. For simplification, we randomly select 80,000 samples as the training dataset and 20,000 as the test dataset.

Yahoo Answers. Yahoo Answers [39] is a dataset for text classification tasks, which involves 10 classes (topics) such as "Society & Culture", "Science & Mathematics", "Health", "Education & Reference", etc. Each class contains 140,000 training samples and 6,000 testing samples. For simplification, we take 5,000 training samples and 2,000 testing samples for each class.

Breast Histopathology Images Dataset (BHI). Invasive Ductal Carcinoma (IDC) is the most common subtype of all breast cancers [10]. Breast Histopathology Images Dataset (BHI) [32] is used for a two-class classification task which classifies a mount slide image patch as IDC negative or IDC positive. The dataset consists of 277,524 mount slide image patches (198,738 IDC negative and 78,786 IDC positive) of breast cancer specimens. All the mount slide image patches have a size of 50×50 pixels and are marked with which patient they belong to. We use 80% of the samples as the training dataset and the rest as the test dataset.

D Parameter Settings

In experiments, the default parameters of the semi-supervised learning algorithm MixMatch are listed in Table 7(a). For MixText, we use the parameter settings in [8]. The default parameters of the malicious local optimizer are listed in Table 7(b).

E Further Exploiting the Inferred Labels

In real-world scenarios, correlated features are quite common, thus the leakage of one private feature may cause the leakage of another private feature. We conduct an experiment to show that the adversary can take advantage of the inferred labels to infer more private information on a real-world medical dataset. More details about this experiment can be found in the supplementary material [14].

Dataset Setup. The experiment is conducted on the Breast Cancer Wisconsin (BCW) dataset [49], which consists of 32 features of breast mass taken from 569 patients. We randomly select 426 samples as the training dataset and the remaining 143 samples as the testing dataset. The label is whether a patient suffers from malignant breast cancer tumor, denoted as "diagosis". The adversary is assigned with 15 features, denoted as features 1 - 15, while the benign participant is assigned with another 16 features, denoted as features 16 - 31. The remaining 32^{nd} feature is "radius mean". The adversary has auxiliary labeled data. Specifically, from the 426 samples in the training dataset, the adversary knows "diagnosis" of 40 samples, and "radius mean" of 40 samples.

Threat Model. The adversary and the benign participant jointly learn a VFL model to predict the label "diagosis". The adversary intends to infer "diagosis" by using the passive label inference attack with the auxiliary data, i.e., 40 samples labeled with "diagnosis". Then the adversary further exploits "diagosis" as an extra feature, together with his/her own features 1 - 15, to train an inference model with the purpose of inferring another feature of a patient – "radius mean". As the adversary has 40 auxiliary samples labeled with "radius mean", he/she can train the inference model in a supervised manner.

Model Architecture. We use the 3-layer fully connected neural network for both the bottom models and the top model.

Experiment Results and Analysis. The VFL model achieves a top-1 accuracy of 0.9510 on the testing dataset. With our passive label inference attack, the adversary achieves a top-1 inference accuracy of 0.8632 for the label "diagnosis". Then, the adversary trains a model using the 15 features he/she owns to infer the unknown feature "radius mean" in a supervised manner. The results show that the adversary achieves 0.6443 top-1 inference accuracy among the 426 samples in the training dataset. Further experiment shows that if the adversary uses the inferred "diagnosis" as an extra feature, he/she will achieve a top-1 inference accuracy of 0.7237. Compared with 0.6443, the boost is significant, which shows that the adversary can infer more private information by exploiting the inferred labels.

Conclusion. To sum up, our label inference attacks not only demonstrate the threat of private label leakage in real-world VFL scenarios, but also reveal that the leaked labels can cause even further privacy leakages.