This article has been accepted for publication in IEEE Transactions on Dependable and Secure Computing. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TDSC.2024.3521451

Feature Reconstruction Attacks and Countermeasures of DNN Training in Vertical Federated Learning

Peng Ye, Zhifeng Jiang, Wei Wang, Bo Li, and Baochun Li

Abstract—Federated learning (FL) has increasingly been deployed, in its vertical form, among organizations to facilitate secure collaborative training. In vertical FL (VFL), participants hold disjoint features of the same set of sample instances. The one with *labels* - the *active party*, initiates training and interacts with other participants - the *passive parties*. It remains largely unknown *whether* and *how* an active party can extract private feature data owned by passive parties, especially when training deep neural network (DNN) models.

This work examines the feature security problem of DNN training in VFL. We consider a DNN model partitioned between active and passive parties, where the passive party holds a subset of the input layer with some features of binary values. Though proved to be NP-hard. we demonstrate that, unless the feature dimension is exceedingly large, it remains feasible, both theoretically and practically, to launch a reconstruction attack with an efficient search-based algorithm that prevails over current feature protection. We propose a novel feature protection scheme by perturbing intermediate results and fabricated input features, which effectively misleads reconstruction attacks towards pre-specified random values. The evaluation shows it sustains feature reconstruction attack in various VFL applications with negligible impact on model performance.

Index Terms—DNN, Vertical Federated Learning, Feature Recovery Attack, Feature Protection Scheme.

1 Introduction

The sustained technological advances in machine learning (ML) have transformed many industries in a profound way. Companies in the internet, finance, retail, and healthcare industries are now building advanced ML models to enable new AI-driven applications, service models, and intelligent decision making. They require collecting a large volume of training data from diverse sources, often practically infeasible. In reality, data are usually dispersed in siloed organizations and data sharing is strictly forbidden – it raises serious privacy and security concerns, as well as potentially violates government regulations, such as CCPA [1] in America, GDPR [2] in Europe, and PIPEDA [3] in Canada. Thus, ensuring data privacy is of paramount importance.

Federated learning (FL) has emerged as a new private-preserving learning paradigm to break data silos [4]–[6]. It enables multiple parties to collaboratively train a global ML model over siloed data while preserving data privacy. FL has been increasingly deployed among companies to form a data federation. In this paper, we consider a typical application scenario called *vertical federated learning* (VFL) [4], [5], [7]–[9], in which participants own disjoint features (i.e., *attributes*) of the same set of sample instances, as illustrated in Figure 1. Only one participant has *labels*, known as the *active party*, and utilizes the joint feature data of its own and from the others, known as the *passive parties*, to train an ML model. For example, an online retailer and a social network company can have an overlapping user base. The former has user browsing history (feature A) and item ordering records (labels), while the latter has accumulated a rich set of user profiles (feature B) through its social network app. Together, they form a joint dataset with user features vertically partitioned between the two participants. The online retailer, being the active party, can partner with the social network company to train a better recommendation model over the joint dataset.

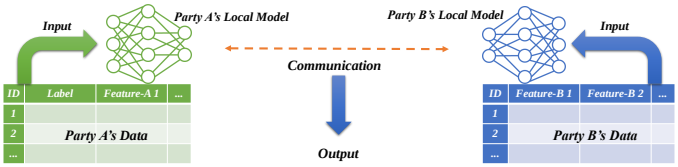


Fig. 1. An illustration of VFL. The two parties hold disjoint features of the same set of sample instances. The active party (Party A) has labels and interacts with the passive party (Party B) to train an ML model which itself is partitioned between the two parties.

With its recent successes, deep neural networks (DNN) become particularly appealing in VFL [10]–[12]. Depending on how features are partitioned, a DNN model is split between different participants, where the passive party holds a subset of a few bottom layers, and the active party holds the rest of the neural network. Each model partition is maintained as a private local model. The active party initiates the training and iteratively interacts with the passive party.

Clearly, raw data is not exposed in the training process; yet the intermediate results exchanged between the two par-

- *Peng Ye, Zhifeng Jiang, Wei Wang, and Bo Li are with the Department of Computer Science and Engineering, Hong Kong University of Science and Technology, Hong Kong. E-mail: {pyeac, zjiangaj, weiwa, bli}@cse.ust.hk.*
- *Baochun Li is with the Department of Electrical and Computer Engineering, University of Toronto, Canada. Email: bli@ece.toronto.edu.*

ties contain a rich set of information that may reveal private information. Prior works show that private labels owned by an active party can be possibly inferred by a passive party from the received gradient updates, resulting in *label leakage* [8], [13], [14]. In this paper, we study a dual security problem concerning the *feature reconstruction attack*, where an active party attempts to uncover the private features owned by a passive party. Compared with label leakage, feature reconstruction seems to be easier, as the active party has a richer set of information to exploit. This turns out to be not the case. As a matter of fact, in this paper, we will show that in general it is impossible to launch feature attack without extra knowledge on the feature characteristics. There have been a few recent works addressing feature attack in VFL [7], [9], [15], which either focus on non-DNN models or require some strong assumptions. For instance, it is assumed that the adversary knows the weights of the entire model [15] or has some auxiliary feature data [9], which are usually inaccessible in practice.

To the best of our knowledge, this paper makes the first attempt to study the feature reconstruction attack of DNN training in VFL. We consider a DNN model jointly trained by two participants, where the passive party holds a subset of the input layer, and the active party holds the remainder of the model. This design has a number of benefits: (1) for any DNN model, it achieves the same accuracy as centralized training without structural adaptation or hyper-parameter tuning; (2) it exposes minimum attack surface for label inference [8]; (3) it provides native support of the recently proposed privacy-preserving framework [16]. We assume that the active party is an *honest-but-curious adversary* with no additional knowledge beyond its own data (e.g., features and labels), local models, and the intermediate results received from the passive party. We first show that it is impossible for an active party to reconstruct *general features* that can take arbitrary values.

As reconstructing general features is infeasible, in this paper, we consider the attack on categorical features of *binary values*[1], which are commonly observed in training data containing sensitive information (e.g., gender, marital and employment status). We show that the problem of binary feature reconstruction can be reduced from the Exact Cover problem, which is NP-hard [17]. We propose a robust search-based attack algorithm that can successfully reconstruct the binary features. We further demonstrate that such an attack cannot be effectively defended by conventional random masking approaches, nor can it be guarded by the recently proposed privacy-preserving framework for VFL training [16]. To defend this attack, we propose an efficient feature protection scheme, in which we first perturb the intermediate output using a *rank-reduction* technique with negligible impact on model performance. Then we insert some fabricated (randomly generated) binary features to masquerade as the input features. We show that this can effectively lead the attacker to recover the fabricated features instead of the original input features.

We have evaluated the effectiveness of our feature reconstruction attack and defense scheme in training DNN models over six VFL datasets. Experimental results show that our attack can completely recover all input binary features, and our defense scheme effectively misleads the reconstruction attack to the fabricated binary features, with negligible model performance loss.

1. Our attack is not limited to the binary features but also effective to general categorical features that take known values (Section 7).

2 Related Work

Data reconstruction attacks. Our proposed feature attack is one type of general *data reconstruction attacks*, which seek to recover the private input data. In VFL, there are two categories of data reconstruction attacks: (1) feature inference attacks, where an active party attempts to recover a passive party's input features; and (2) label inference attacks, where a passive party tries to discover the active party's labels.

Feature attacks. This implies an attack on a passive party's input features. Since an active party possesses far more information, thus, it is in a more advantageous position for such attacks. In [7], Weng *et al.* devised a reverse multiplication attack method against logistic regression with the assistance of a corrupted third-party coordinator and a reverse sum attack method against XGBoost by encoding magic numbers in the gradients. In [15], Luo *et al.* designed an equality solving attack for linear regression models, a path restriction attack for decision tree models, and a generative regression network for attacking more complex models. This work adopts a white-box setting, which requires an active party to know the entire model weights including the passive party's local model. In [9], Jiang *et al.* proposed a gradient-based inversion attack, which can recover a passive party's input under both white-box and black-box settings with the assistance of a set of auxiliary data used in training.

Label attacks. In [8], Fu *et al.* proposed a label inference attack based on the semi-supervised learning technique, which can recover an active party's labels using its local bottom model and a small set of auxiliary data. In [14], Liu *et al.* presented a gradient inversion attack, which can infer the labels from batch-averaged gradients when the top model is a softmax function on the sum of intermediate results and the loss function is cross entropy. In [13], Li *et al.* considered a two-party split learning scenario and designed two attack mechanisms to extract labels from the norm and direction of intermediate gradients.

3 Threat Model and Formulation

In this section, we formally describe the DNN model training in VFL and present the threat model. We show that it is impossible for an active party to reconstruct general features owned by the passive party that can take arbitrary values.

Throughout this paper, we use boldface upper case letters (e.g., $\boldsymbol{A}$) to denote matrices and boldface lower case letters (e.g., $\boldsymbol{x}$) to denote vectors. We use $\mathbf{0}$ to denote the zero vector. Vectors are by default column vectors while row vectors are denoted by the transpose of column vectors (e.g., $\boldsymbol{x}^\top$). The i-th coordinate of vector $\boldsymbol{x}$ is denoted by $\boldsymbol{x}_i$. We use $[n]$ to represent the set $\{1, 2, \ldots, n\}$ for positive integer n. The notation $\{0,1\}^n$ denotes the set of all n-dimensional binary vectors (i.e., $\{\boldsymbol{x} \in \mathbb{R}^n : \boldsymbol{x}_i \in \{0,1\} \text{ for all } i \in [n]\}$).

3.1 Formalizing DNN Training Workflow

We now describe the overall workflow of DNN training in VFL. In each iteration, VFL runs a forward pass to make predictions and a backward pass to update parameters (only a forward pass is needed in the inference phase). In the forward pass, a passive party B computes the output of its local model using its own data and sends the intermediate results to an active party A. Then the active party A aggregates the first layer output and runs the top model to obtain the final output. In the backward pass, the active party A computes the gradients using the labels and updates all the local parameters. The passive party B receives intermediate gradients from the active party A and computes the local model gradients.

Formally, let d_A and d_B denote the number of input features of party A and B, respectively. Consider a neural network with a weight matrix $\boldsymbol{W} \in \mathbb{R}^{k\times(d_A+d_B)}$ in the input layer, where $d_A + d_B$ is the total input dimension and k is the number of neurons in the second layer. In each iteration, party A's input is a vector $\boldsymbol{x}_A \in \mathbb{R}^{d_A}$ and party B's input is a vector $\boldsymbol{x}_B \in \mathbb{R}^{d_B}$. The weight $\boldsymbol{W}$ is vertically partitioned into two matrices $\boldsymbol{W}_A \in \mathbb{R}^{k\times d_A}$ and $\boldsymbol{W}_B \in \mathbb{R}^{k\times d_B}$, such that party A owns $\boldsymbol{W}_A$ and party B owns $\boldsymbol{W}_B$. The remaining parameters, denoted by $\boldsymbol{\theta}$, are owned by the active party A.

In each iteration, party B sends an intermediate result $\boldsymbol{z}_B = \boldsymbol{W}_B\boldsymbol{x}_B$ to party A. Then party A computes $\boldsymbol{z}_A = \boldsymbol{W}_A\boldsymbol{x}_A$ and $\boldsymbol{z} = \boldsymbol{z}_A + \boldsymbol{z}_B$. After obtaining $\boldsymbol{z}$, the aggregated output of the first layer, party A completes the forward pass by computing $f_{\boldsymbol{\theta}}(\boldsymbol{z})$, where $f_{\boldsymbol{\theta}}$ denotes the remaining forward computation, which is done only by party A.

In the backward pass, party A uses the label y to compute the gradients of loss L w.r.t. $\boldsymbol{\theta}$ and $\boldsymbol{z}$. The gradient $\frac{\partial L}{\partial \boldsymbol{W}_A}$ is obtained by $\frac{\partial L}{\partial \boldsymbol{z}}\boldsymbol{x}_A^\top$. Thus all parameters maintained by party A can be updated by the gradient descent method. To update $\boldsymbol{W}_B$, party A passes $\frac{\partial L}{\partial \boldsymbol{z}}$ to party B. Party B can then calculate $\frac{\partial L}{\partial \boldsymbol{W}_B} = \frac{\partial L}{\partial \boldsymbol{z}}\boldsymbol{x}_B^\top$.

3.2 Threat Model

In our threat model, we assume that the active party acts as a semi-honest adversary. That is, the adversary strictly adheres to the training protocol but attempts to extract private information from its own perspective. The adversary's view (i.e., the active party) encompasses its own input data, local model parameters, and the intermediate results received in each iteration. The active party possesses no knowledge beyond this information, including details about the passive party such as its model weights. The goal of the adversary is to perform a data reconstruction attack, whereby the active party tries to reconstruct the passive party's input data.

Formally, suppose the training process runs for T rounds. In the t-th iteration, an active party receives an intermediate result $\boldsymbol{z}_B^t$ from a passive party. It knows its own private data $\boldsymbol{x}_A^t$, the label y^t, and its local model weights $\boldsymbol{W}_A^t$ and $\boldsymbol{\theta}^t$. With these information (i.e., $\{\boldsymbol{x}_A^t, y^t, \boldsymbol{W}_A^t, \boldsymbol{\theta}^t, \boldsymbol{z}_B^t\}_{t=1,\dots,T}$), the goal of a data reconstruction attack is to recover the passive party's private data $\{\boldsymbol{x}_B^t\}_{t=1,\dots,T}$.

Extracting input features turns out to be particularly challenging. In practice, even reconstructing a single feature can raise substantial security concerns. Thus, it is also critical to consider a data reconstruction attack, where the goal is to reconstruct one feature, i.e., $\{(\boldsymbol{x}_B^t)_i\}_{t=1,\dots,T}$ for some $i \in [d_B]$.

3.3 Privacy Leakage and Binary Assumption

During training (or inferencing), what an active party A receives is a matrix product $\boldsymbol{z}_B = \boldsymbol{W}_B\boldsymbol{x}_B$ sent from a passive party B. Since both $\boldsymbol{W}_B$ and $\boldsymbol{x}_B$ are unknown to party A. Party A cannot recover the data by solving linear equations. Indeed, one can show that even for a malicious active party A (i.e., it can send some specially crafted values instead of the true gradients to the passive party), it is impossible to infer the passive party's input data $\boldsymbol{x}_B$. Noticing what party A receives is a matrix product $\boldsymbol{z}_B = \boldsymbol{W}_B\boldsymbol{x}_B$, there are an infinite number of possible pairs $(\boldsymbol{W}_B, \boldsymbol{x}_B)$ that generate the same $\boldsymbol{z}_B$. Since the active party only sees the intermediate results $\boldsymbol{z}_B$, it is impossible for the active party to distinguish them, let alone recover them. We now state this impossibility result formally in the following theorem:

Theorem 1. *Suppose $\{\boldsymbol{z}_B^t\}_{t=1,\dots,T}$ is the set of intermediate results received by party A during training, which is generated by some initial weight $\boldsymbol{W}_B^0$ and input data $\{\boldsymbol{x}_B^t\}_{t=1,\dots,T}$. Suppose $\boldsymbol{x}_B^t$'s are not all zero. Then there are infinite possible input data that can generate this set. Thus, party A cannot reconstruct party B's input.*

Proof. In the t-th iteration ($1 \le t \le T$), party B sends $\boldsymbol{z}_B^t = \boldsymbol{W}_B^{t-1}\boldsymbol{x}_B^t$ to party A. Then it receives gradient $\boldsymbol{g}^t$ (w.r.t. $\boldsymbol{z}_B^t$) from party A and update the weight by $\boldsymbol{W}_B^t = \boldsymbol{W}_B^{t-1} - \eta_t\boldsymbol{g}^t(\boldsymbol{x}_B^t)^\top$, where η_t is the learning rate in t-th iteration (for the inference phase, just set the learning rate to be 0).

Suppose $\boldsymbol{W}_B^0$ and $\{\boldsymbol{x}_B^t\}_{t=1,\dots,T}$ is a pair of initial weight and input data that generates set $\{\boldsymbol{z}_B^t\}_{t=1,\dots,T}$. Let $\boldsymbol{U} \in \mathbb{R}^{d_B\times d_B}$ be an arbitrary unitary matrix. We prove that adopting $\boldsymbol{W}_B^0\boldsymbol{U}^\top$ and $\{\boldsymbol{U}\boldsymbol{x}_B^t\}_{t=1,\dots,T}$ as initial weight and input data leads to the same set of intermediate output $\{\boldsymbol{z}_B^t\}_{t=1,\dots,T}$.

Consider the first iteration, the party B first computes $\boldsymbol{W}_B^0\boldsymbol{U}^\top\boldsymbol{U}\boldsymbol{x}_B^1$, which is exactly the same as $\boldsymbol{z}_B^0 = \boldsymbol{W}_B^0\boldsymbol{x}_B^1$. Since it sends the same intermediate output to party B. It receives the same gradient $\boldsymbol{g}^1$. It then computes $\boldsymbol{W}_B^0\boldsymbol{U}^\top - \eta_t\boldsymbol{g}^t(\boldsymbol{U}\boldsymbol{x}_B^t)^\top = (\boldsymbol{W}_B^0 - \eta_t\boldsymbol{g}^1(\boldsymbol{x}_B^1)^\top)\boldsymbol{U}^\top = \boldsymbol{W}_B^1\boldsymbol{U}^\top$ to update its local weight.

We can then prove by induction that in the t-th iteration, the intermediate output sent to party A is exactly $\boldsymbol{z}_B^t$ and the local weight held by party B is $\boldsymbol{W}_B^t\boldsymbol{U}^\top$. We have shown that this holds for $t = 1$.

Suppose this holds for iteration 1 to $t-1$. In t-th iteration party B sends $\boldsymbol{W}_B^{t-1}\boldsymbol{U}^\top\boldsymbol{U}\boldsymbol{x}_B^t = \boldsymbol{W}_B^{t-1}\boldsymbol{x}_B^t = \boldsymbol{z}_B^t$ to party A. Note that party A receives $\{\boldsymbol{z}_B^1, \dots, \boldsymbol{z}_B^t\}$ until iteration t. The gradient it sends back to party B must be $\boldsymbol{g}^t$. Therefore the weight held by party B will be updated to $\boldsymbol{W}_B^{t-1}\boldsymbol{U}^\top - \eta_t\boldsymbol{g}^t(\boldsymbol{U}\boldsymbol{x}_B^t)^\top = (\boldsymbol{W}_B^{t-1} - \eta_t\boldsymbol{g}^t(\boldsymbol{x}_B^t)^\top)\boldsymbol{U}^\top = \boldsymbol{W}_B^t\boldsymbol{U}^\top$.

Thus $\boldsymbol{W}_B^0\boldsymbol{U}^\top$ and $\{\boldsymbol{U}\boldsymbol{x}_B^t\}_{t=1,\dots,T}$ generate the same set $\{\boldsymbol{z}_B^t\}_{t=1,\dots,T}$. As there are infinite unitary matrices of size $d_B \times d_B$, also infinite pairs of initial weight and input data.

Since $\{\boldsymbol{z}_B^t\}_{t=1,\dots,T}$ is the only information that party A receives, an attack algorithm will always output the same

for these pairs. However, $\{\boldsymbol{U}\boldsymbol{x}_B^t\}_{t=1,\ldots,T}$ varies for different $\boldsymbol{U}$. Thus, such an attack algorithm doesn't exist. □

Remark. *In the above proof, we allow the intermediate gradient $\boldsymbol{g}^t$ to be generated arbitrarily. That is, party A doesn't have to follow protocol. This is called the malicious adversary setting. A malicious adversary is more powerful than a semi-honest one at attacking. Therefore, we actually prove a stronger result - even a malicious adversary cannot reconstruct the passive party's input.*

Remark. *We use vanilla stochastic gradient descent (SGD) in the proof of Theorem 1. It can be directly extended to other popular variants such as SGD with momentum [18], RMSprop [19], and Adam [20] because the update only depends on historical gradients. This theorem indicates that one cannot distinguish between infinite possible inputs. Thus there is no way to recover the data.*

This illustrates that an attack is not possible when the attacker has zero knowledge about the data. In practice, however, the active party may know certain properties of the passive party's input. Noticing that the impossibility result relies on the fact that performing a unitary transform on the input doesn't change the active party's view. Thus, an intermediate result corresponds to an infinite number of possible inputs, which are indistinguishable to an attacker. However, when an attacker knows certain properties of the input features, the number of possible inputs could be drastically reduced (even to only one), making it possible for the attacker to perform attacks.

Taking this into account, in this work, we consider the situation that some of the passive party's input features are discrete. Such discrete features are common in real-world scenarios such as marital status, exam results, medical test outcome, economic status, ethnicity, and citizenship, which are usually sensitive.

Specifically for simplicity, in this work, we only consider binary features, i.e., discrete features that take values of 0 or 1. This is partly because binary features are commonly observed in real-world datasets. For example, in healthcare, binary features are usually employed to represent whether a person has some symptoms or diseases (e.g., the COVID-19 and monkey-pox dataset we used in Section 6).

The binary features can also come from feature engineering. In practice, it is common to convert a categorical feature to a one-hot representation, which introduces many binary features. One-hot encoding is frequently used when the raw feature contains many categories but is nominal, i.e., there is no quantitative relationship between different values. For instance, blood types have four categories. Simply assigning them with different numbers implicitly introduces an order between them, which may hinder the model from learning the true relationship. We will illustrate that our attack and defense methods can be easily extended to handle discrete features that take more than two values.

4 Binary Feature Inference Attacks

In this section, we present our approach that leverages the existence of binary features to launch effective attacks. Then we propose an algorithm that can effectively recover the binary feature.

The intermediate results sent from the passive party B can be represented as a matrix product $\boldsymbol{Z}_B = \boldsymbol{X}_B\boldsymbol{W}_B^\top \in \mathbb{R}^{n\times k}$, where $\boldsymbol{X}_B$ is an $n \times d_B$ matrix with each row containing one input data record, and $\boldsymbol{W}_B \in \mathbb{R}^{k\times d_B}$ is the first layer weights owned by party B. Here we consider that the intermediate results are generated in one batch (during training) or during the inference phase, thus the weights won't change.

As discussed in Section 3.3, it is infeasible for the active party A to recover the passive party's input data $\boldsymbol{X}_B$ since the weight matrix $\boldsymbol{W}_B$ is unknown to party A. So we assume that some of the passive party's input features are binary. Our goal is to recover those binary features.

Our approach is based on the observation that every binary feature, which corresponds to a column of $\boldsymbol{X}_B$, also lies in the column space of $\boldsymbol{Z}_B$. Thus, we aim to identify a binary vector in the column span of $\boldsymbol{Z}_B$. This can be achieved by enumerating all possible binary vectors (2^n vectors in total) and then verifying each vector by solving linear equations. We will show how to reduce the number of enumerations to 2^d through linear algebraic manipulations.

4.1 Attack by Solving Linear Equations

Consider the problem of finding binary vectors in the column space of the intermediate results $\boldsymbol{Z}_B$. We can first construct a matrix $\boldsymbol{Z} \in \mathbb{R}^{n\times d_B}$ such that $\boldsymbol{Z}_B$ and $\boldsymbol{Z}$ shares the same column space by picking linearly independent columns. Without loss of generality, we assume that $\boldsymbol{Z}$ has full rank, because otherwise we can further remove some columns while keeping the column space unchanged and replace d_B by the actual rank, which is always at most d_B.

We now focus on matrix $\boldsymbol{Z}$. We can find a $d_B \times d_B$ submatrix $\boldsymbol{Z}'$ of $\boldsymbol{Z}$ with full rank. Now suppose $\boldsymbol{x}$ is a binary vector that lies in the column space of $\boldsymbol{Z}$. This means there exists a vector $\boldsymbol{w}$ such that $\boldsymbol{Z}\boldsymbol{w} = \boldsymbol{x}$. Since $\boldsymbol{Z}'$ is a submatrix of $\boldsymbol{Z}$, we have that $\boldsymbol{Z}'\boldsymbol{w}$ is also a binary vector. Conversely, given a binary vector $\boldsymbol{x}' \in \mathbb{R}^{d_B}$, there is a unique $\boldsymbol{w}$ such that $\boldsymbol{Z}'\boldsymbol{w} = \boldsymbol{x}'$ since $\boldsymbol{Z}'$ is invertible. Therefore, to find all binary vectors in the column span of $\boldsymbol{Z}$, we only have to find all binary vectors in the column span of $\boldsymbol{Z}'$. We describe the details of such an algorithm in Algorithm 1.

Algorithm 1 Attack by Solving Linear Equations

Input: Matrix $\boldsymbol{Z} \in \mathbb{R}^{n\times d_B}$ with $rank(\boldsymbol{Z}) = d_B$

```
Find a d_B × d_B submatrix Z' from Z, where rank(Z') = d_B
T ← ∅.
for x' in {0,1}^{d_B} \ {0} do
    w ← Z'^{-1} x'
    x ← Z w
    if x is binary then
        T ← T ∪ {x}
    end if
end for
return T
```

Output: A set T of binary vectors

4.2 NP-hardness

The above attack algorithm is equivalent to finding binary vectors in the column span of a given matrix. We now prove

Theorem 2, which shows this task is indeed NP-hard. The basic idea is to transform an Exact Cover instance to some matrix $\boldsymbol{Z}$ so that one can solve the Exact Cover instance by finding a binary solution to $\boldsymbol{Zw} = \boldsymbol{x}$. Thus this above problem is at least hard as the Exact Cover problem, which is NP-hard.

Theorem 2. *Given a matrix $\boldsymbol{Z}$, deciding whether there is a nonzero binary vector in the column span of $\boldsymbol{Z}$ is NP-hard.*

Proof. Our proof relies on the fact that the Exact Cover problem is NP-hard. Given a set of n elements $U = \{u_1, \ldots, u_n\}$ and a collection $C = \{S_1, \ldots, S_m\}$ of subsets of U. The Exact Cover problem is to decide whether there is a sub-collection $C' \subseteq C$ that covers every element exactly once, i.e. $|\{j | u_i \in S_j \text{ and } S_j \in C'\}| = 1$ for all $i \in [n]$.

Now consider an instance of the Exact Cover problem: $U = \{u_1, \ldots, u_n\}$ and $C = \{S_1, \ldots, S_m\}$. We construct following three matrices. Let $\boldsymbol{Z}_1 = \boldsymbol{I}_{m+1}$ be an $(m+1)$-dimensional identity matrix, $\boldsymbol{Z}_2 = (z_{ij}) \in \mathbb{R}^{n \times (m+1)}$ where

$$z_{ij} = \begin{cases} 1 & \text{if } j \in [m] \text{ and } u_i \in S_j \\ 0 & \text{if } j \in [m] \text{ and } u_i \notin S_j \\ -1 & \text{if } j = m+1 \end{cases},$$

and $\boldsymbol{Z}_3 = [2|S_1|, \ldots, 2|S_m|, -2n] \in \mathbb{R}^{1 \times (m+1)}$. Stacking the three matrices we can obtain

$$\boldsymbol{Z} = \begin{bmatrix} \boldsymbol{Z}_1 \\ \boldsymbol{Z}_2 \\ \boldsymbol{Z}_3 \end{bmatrix} \in \mathbb{R}^{(n+m+2) \times (m+1)}.$$

We then show that if we could decide whether there exists $\boldsymbol{w} \in \mathbb{R}^{m+1}$ such that $\boldsymbol{x} = \boldsymbol{Zw}$ is nonzero and binary, then we can decide whether there exists an exact cover.

Suppose there is a sub-collection C' that covers every element exactly once. Let $\boldsymbol{w}_{m+1} = 1$ and for $j \in [m]$ let

$$\boldsymbol{w}_j = \begin{cases} 1 & \text{if } S_j \in C' \\ 0 & \text{if } S_j \notin C' \end{cases}.$$

Then $\boldsymbol{Z}_1\boldsymbol{w} = \boldsymbol{w}$ is nonzero and binary. The i-th element of $\boldsymbol{Z}_2\boldsymbol{w}$ is $\sum_{j=1}^{m} z_{ij}\boldsymbol{w}_j - 1 = |\{j | u_i \in S_j \text{ and } S_j \in C'\}| - 1 = 0$. And $\boldsymbol{Z}_3\boldsymbol{w} = 2(\sum_{S_j \in C'} |S_j| - n) = 0$. Therefore $\boldsymbol{Zw}$ is nonzero and binary.

Now, suppose there exists $\boldsymbol{w} \in \mathbb{R}^{m+1}$ that $\boldsymbol{x} = \boldsymbol{Zw}$ is nonzero and binary. From $\boldsymbol{Z}_1\boldsymbol{w} = \boldsymbol{w}$, we know that $\boldsymbol{w}$ is also binary. Since $\boldsymbol{Z}_3\boldsymbol{w} = 2\sum_{j=1}^{m} |S_j|\boldsymbol{w}_j - 2n\boldsymbol{w}_{m+1}$ is a multiple of 2, it must be 0. Thus $\boldsymbol{w}_{m+1}$ must be 1 otherwise we will have $\boldsymbol{w} = \boldsymbol{0}$, contradicting that $\boldsymbol{Zw}$ is nonzero. So if we let $C' = \{S_j | \boldsymbol{w}_j = 1, j \in [m]\}$, we have $\sum_{S_j \in C'} |S_j| = \sum_{j=1}^{m} |S_j|\boldsymbol{w}_j = n\boldsymbol{w}_{m+1} = n$. The i-th element of $\boldsymbol{Z}_2\boldsymbol{w}$ is $\sum_{j=1}^{m} z_{ij}\boldsymbol{w}_j - 1 = |\{j | u_i \in S_j \text{ and } S_j \in C'\}| - 1$, which is either 0 or 1, indicating that $|\{j | u_i \in S_j \text{ and } S_j \in C'\}|$ should be 1 or 2 for each $i \in [n]$. But we have $\sum_{i=1}^{n} |\{j | u_i \in S_j \text{ and } S_j \in C'\}| = \sum_{S_j \in C'} |S_j| = n$. Thus $|\{j | u_i \in S_j \text{ and } S_j \in C'\}|$ must be 1, namely, each element is covered exactly once. Therefore C' is an exact cover.

We have shown that there is a nonzero binary vector in the column span of $\boldsymbol{Z}$ if and only if there is an exact cover. Thus if we could decide the existence of a nonzero binary vector in the column span, then we can solve the Exact Cover problem. Applying the NP-hardness of the Exact Cover problem completes the proof. □

Remark. *In Theorem 2 we add one restriction that $\boldsymbol{x}$ should be a nonzero vector, i.e. $\boldsymbol{x} \neq \boldsymbol{0}$. Since $\boldsymbol{A0} = \boldsymbol{0}$, the zero vector $\boldsymbol{0}$ is always a trivial solution. Thus, it reveals no information about the actual input. Also, an input feature that contains only 0 contributes nothing to the training process and is impossible to be detected by the attacker.*

4.3 Attack by Solving Linear Regression

While we have an algorithm that effectively launches attacks, its success relies on solving linear equations during the verification step. However, this approach may be susceptible to small perturbations, such as numerical errors or noise introduced by the passive party. To enhance the robustness of the attack algorithm, we replace the linear equation solving process with linear regression. To avoid enumerating all 2^n binary vectors, we adopt a Leverage Score Sampling technique [21]. Algorithm 2 gives the details.

Algorithm 2 Attack by solving Linear Regression

Input: Matrix $\boldsymbol{Z} \in \mathbb{R}^{n \times d_B}$ with $rank(\boldsymbol{Z}) = d_B$

1: Use leverage score sampling to randomly sample and rescale r rows and obtain $\boldsymbol{Z}' = \boldsymbol{DSZ}$, where $\boldsymbol{S} \in \mathbb{R}^{r \times n}$ is a sampling matrix that samples r rows $R = \{i_1, \ldots, i_r\}$ and $\boldsymbol{D} \in \mathbb{R}^{r \times r}$ is a diagonal matrix that rescales the values in each row
2: $T \leftarrow \{\boldsymbol{e}\}$ where $\boldsymbol{e} = [1, 0, \ldots, 0]^\top \in \mathbb{R}^n$
3: **for** $\boldsymbol{x}'$ in $\{0,1\}^r \setminus \{\boldsymbol{0}\}$ **do**
4: $\quad \boldsymbol{w}' \leftarrow \operatorname{argmin}_{\boldsymbol{w}} \|\boldsymbol{Z}'\boldsymbol{w} - \boldsymbol{D}\boldsymbol{x}'\|_2^2$
5: $\quad$ Create an n-dimensional vector $\boldsymbol{x}$, set

$$\boldsymbol{x}_i \leftarrow \begin{cases} \boldsymbol{x}'_j & \text{if } i = i_j \text{for some } j \\ 0 & \text{if } i \notin R \text{ and } (\boldsymbol{Zw}')_i < 0.5 \\ 1 & \text{otherwise} \end{cases}$$

6: $\quad T \leftarrow T \cup \{\boldsymbol{x}\}$
7: **end for**
8: $\boldsymbol{x}^* \leftarrow \operatorname{argmin}_{\boldsymbol{x} \in T} \min_{\boldsymbol{w}} \|\boldsymbol{Zw} - \boldsymbol{x}\|_2^2$

Output: Vector $\boldsymbol{x}^* \in \mathbb{R}^n$

The correctness of the algorithm is established through the following theorem, showing that it finds a good approximate solution to the linear regression problem.

Theorem 3. *Given an $n \times d_B$ matrix $\boldsymbol{Z}$ and $\epsilon > 0$, let $\boldsymbol{x}^{opt} = \operatorname{argmin}_{\boldsymbol{x} \in \{0,1\}^n \setminus \{\boldsymbol{0}\}} \min_{\boldsymbol{w}} \|\boldsymbol{Zw} - \boldsymbol{x}\|_2^2$. By choosing $r = O(d_B \log d_B / \epsilon^2)$, Algorithm 2 outputs a vector $\boldsymbol{x}^*$ such that*

$$\min_{\boldsymbol{w}} \|\boldsymbol{Zw} - \boldsymbol{x}^*\|_2^2 \leq (1+\epsilon) \min_{\boldsymbol{w}} \|\boldsymbol{Zw} - \boldsymbol{x}^{opt}\|_2^2$$

with constant probability. Moreover, Algorithm 2 runs in time $O(nr \cdot 2^r)$.

Our proof relies on the following lemma, which states the error bound provided by Leverage Score Sampling.

Lemma 4 (Leverage Score Sampling [21])**.** *Given an $n \times d_B$ matrix $\boldsymbol{A}$, an n-dimensional vector $\boldsymbol{b}$, and $\epsilon > 0$. Let $\boldsymbol{p}_i = \|\boldsymbol{U}_{(i)}\|_2^2 / d$ be normalized leverage scores, where $\boldsymbol{U}$ is the matrix containing left singular vectors of $\boldsymbol{A}$ and $\boldsymbol{U}_{(i)}$ is the i-th row of $\boldsymbol{U}$.*

Let $\boldsymbol{S} \in \mathbb{R}^{r\times n}$ and $\boldsymbol{D} \in \mathbb{R}^{r\times r}$ be the sampling and rescaling matrix generated from distribution $\boldsymbol{p}$, where $r = O(d_B \log d_B/\epsilon^2)$. With constant probability we have

$$\|\boldsymbol{A}\tilde{\boldsymbol{x}} - \boldsymbol{b}\|_2^2 \le (1+\epsilon)\min_{\boldsymbol{x}} \|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}\|_2^2,$$

where $\tilde{\boldsymbol{x}} = \operatorname{argmin}_{\boldsymbol{x}} \|\boldsymbol{DSA}\boldsymbol{x} - \boldsymbol{DSb}\|_2^2$.

With this lemma in hand, we can then prove Theorem 3 as follows.

Proof. By Lemma 4, we have

$$\|\boldsymbol{Z}\boldsymbol{w}' - \boldsymbol{x}^{opt}\|_2^2 \le (1+\epsilon)\min_{\boldsymbol{w}} \|\boldsymbol{Z}\boldsymbol{w} - \boldsymbol{x}^{opt}\|_2^2,$$

where $\boldsymbol{w}' = \operatorname{argmin}_{\boldsymbol{w}} \|\boldsymbol{DSZ}\boldsymbol{w} - \boldsymbol{DS}\boldsymbol{x}^{opt}\|_2^2$, with constant probability.

If $\boldsymbol{S}\boldsymbol{x}^{opt}$ is nonzero, consider the vector $\boldsymbol{x}$ generated in the algorithm with $\boldsymbol{x}' = \boldsymbol{S}\boldsymbol{x}^{opt}$, it is easy to see that $\boldsymbol{S}\boldsymbol{x} = \boldsymbol{x}' = \boldsymbol{S}\boldsymbol{x}^{opt}$. Thus $\boldsymbol{w}' = \operatorname{argmin}_{\boldsymbol{w}} \|\boldsymbol{DSZ}\boldsymbol{w} - \boldsymbol{DS}\boldsymbol{x}\|_2^2$. Recall that R is the set of rows sampled in the algorithm, we have

$$\begin{aligned}
\|\boldsymbol{Z}\boldsymbol{w}' - \boldsymbol{x}\|_2^2 &= \sum_{i\in R}[(\boldsymbol{Z}\boldsymbol{w}')_i - \boldsymbol{x}_i]^2 + \sum_{i\notin R}[(\boldsymbol{Z}\boldsymbol{w}')_i - \boldsymbol{x}_i]^2 \\
&\le \sum_{i\in R}[(\boldsymbol{Z}\boldsymbol{w}')_i - \boldsymbol{x}_i^{opt}]^2 + \sum_{i\notin R}[(\boldsymbol{Z}\boldsymbol{w}')_i - \boldsymbol{x}_i^{opt}]^2 \\
&= \|\boldsymbol{Z}\boldsymbol{w}' - \boldsymbol{x}^{opt}\|_2^2
\end{aligned}$$

because $[(\boldsymbol{Z}\boldsymbol{w}')_i - \boldsymbol{x}_i]^2 = \min([(\boldsymbol{Z}\boldsymbol{w}')_i - 0]^2, [(\boldsymbol{Z}\boldsymbol{w}')_i - 1]^2) \le [(\boldsymbol{Z}\boldsymbol{w}')_i - \boldsymbol{x}_i^{opt}]^2$ for $i \notin R$.

Since $\boldsymbol{x} \in T$, by the definition of $\boldsymbol{x}^*$, we have

$$\begin{aligned}
\min_{\boldsymbol{w}} \|\boldsymbol{A}\boldsymbol{w} - \boldsymbol{x}^*\|_2^2 &\le \min_{\boldsymbol{w}} \|\boldsymbol{A}\boldsymbol{w} - \boldsymbol{x}\|_2^2 \\
&\le \|\boldsymbol{A}\boldsymbol{w}' - \boldsymbol{x}\|_2^2 \\
&\le \|\boldsymbol{A}\boldsymbol{w}' - \boldsymbol{x}^{opt}\|_2^2 \\
&\le (1+\epsilon)\min_{\boldsymbol{w}} \|\boldsymbol{A}\boldsymbol{w} - \boldsymbol{x}^{opt}\|_2^2.
\end{aligned}$$

If $\boldsymbol{S}\boldsymbol{x}^{opt}$ is a zero vector, $\operatorname{argmin}_{\boldsymbol{w}} \|\boldsymbol{DSA}\boldsymbol{w} - \boldsymbol{DS}\boldsymbol{x}^{opt}\|_2^2$ is also a zero vector. Thus we have

$$\|\boldsymbol{x}^{opt}\|_2^2 \le (1+\epsilon)\min_{\boldsymbol{w}} \|\boldsymbol{A}\boldsymbol{w} - \boldsymbol{x}^{opt}\|_2^2.$$

Since $\boldsymbol{e} \in T$ and $\boldsymbol{x}^{opt}$ is nonzero, we have

$$\begin{aligned}
\min_{\boldsymbol{w}} \|\boldsymbol{A}\boldsymbol{w} - \boldsymbol{x}^*\|_2^2 &\le \min_{\boldsymbol{w}} \|\boldsymbol{A}\boldsymbol{w} - \boldsymbol{e}\|_2^2 \\
&\le \|\boldsymbol{e}\|_2^2 \\
&\le \|\boldsymbol{x}^{opt}\|_2^2 \\
&\le (1+\epsilon)\min_{\boldsymbol{w}} \|\boldsymbol{A}\boldsymbol{w} - \boldsymbol{x}^{opt}\|_2^2.
\end{aligned}$$

The algorithm enumerates all binary vectors in $\mathbb{R}^r$ and for each vector it solves a least square problem, which can be done in $O(nr)$ time because the pseudo inverse of $\boldsymbol{A}'$ can be precomputed. Thus the total time complexity is $O(nr \cdot 2^r)$. □

Note that Algorithm 2 is presented in a way that it outputs a unique binary vector. In scenarios where multiple binary features exist, a small threshold can be set, enabling the algorithm to output all binary vectors with errors below the threshold. Also note that the value of r critically determines the complexity of Algorithm 2. However, our evaluation in Section 6 shows that, for the purpose of our attack, choosing $r = d_B + 1$ is sufficient, without the need of matching the theoretical bound. This allows Algorithm 2 to recover the input binary features efficiently.

5 Defense

In this section, we describe a novel masquerade mechanism designed to safeguard the binary features of the passive party from reconstruction attempts by the active party. Our key idea is to inject random binary vectors into the column space of $\boldsymbol{Z}_B$. By incorporating these fabricated random binary vectors, the attacking party will encounter difficulty in distinguishing between the true binary features and the decoy vectors, thus ensuring the effective protection of the authentic binary feature data.

We now describe the details. We introduce two matrices $\boldsymbol{P} \in \mathbb{R}^{k\times(d_B-1)}$ and $\boldsymbol{Q} \in \mathbb{R}^{(d_B-1)\times d_B}$. In the forward pass, the passive party generates a random bit a for each input data point $\boldsymbol{x}_B$, which is set to either 0 or 1 with equal probability. The passive party then sends

$$\boldsymbol{z}_B = \begin{bmatrix}\boldsymbol{P}|\boldsymbol{u}\end{bmatrix}\begin{bmatrix}\boldsymbol{Q}\boldsymbol{x}_B \\ a\end{bmatrix}$$

to the active party. In the backward pass, the passive party receives $\frac{\partial L}{\partial \boldsymbol{z}}$ and updates $\boldsymbol{P}$, $\boldsymbol{Q}$, and $\boldsymbol{u}$ through gradient descent.

However, adding only one fabricated feature can be still vulnerable. The attacker can adaptively adjust the strategy by first recovering the fabricated feature and then picking out those data points with values 0 on this feature (data points with values 1 can also be attacked in a similar way). We demonstrate this through experiments in Section 6.

To address this issue, we introduce additional fabricated features. Since the adaptive attack relies on picking data points that have the same fabricated features, by including m fabricated binary features, the number of data points sharing the same values on the fabricated features will be less than $n/2^m$, where n is the total number of data points. Therefore, we can effectively invalidate the attack as long as $2^m \ge n$. In Section 6, we show that as the number of fabricated features increases, the model accuracy degrades. We thus set $m = \lceil \log_2 n \rceil$, the minimum value that ensures data security.

Note that implementing the masquerade method does not affect the communication cost, as the sizes of the tensors transmitted remain the same. However, it does introduce additional computation in the first layer. Nevertheless, our experiments in Section 6 demonstrate that the extra training cost is minimal since other layers remain unchanged.

Algorithm 3 Masquerade Defense with Multiple Fabricated Features

Input: Input data $\boldsymbol{x}_B \in \mathbb{R}^{d_B}$, random binary values $a_i \in \{0,1\}$ for $i \in [m]$

1: $\boldsymbol{z}_B \leftarrow \begin{bmatrix}\boldsymbol{P}|\boldsymbol{u}_1|\cdots|\boldsymbol{u}_m\end{bmatrix}\begin{bmatrix}\boldsymbol{Q}\boldsymbol{x}_B \\ a_1 \\ \vdots \\ a_m\end{bmatrix}$
2: Send $\boldsymbol{z}_B$ to the active party
3: Receive $\frac{\partial L}{\partial \boldsymbol{z}}$ from the active party
4: Update $\boldsymbol{P}$, $\boldsymbol{u}$, and $\boldsymbol{Q}$ by gradient descent

This article has been accepted for publication in IEEE Transactions on Dependable and Secure Computing. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TDSC.2024.3521451

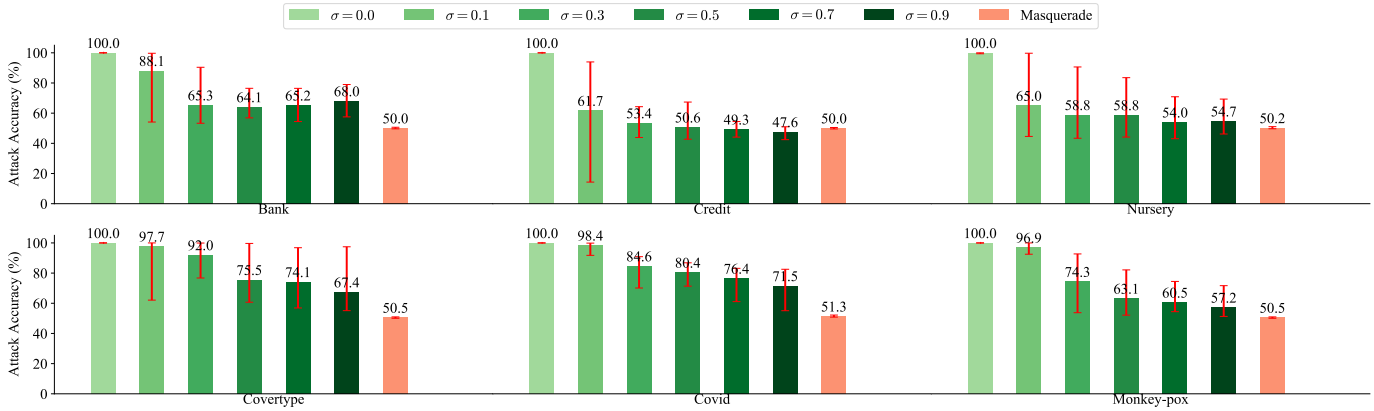


Fig. 2. Attack accuracy: varying σ and Masquerade

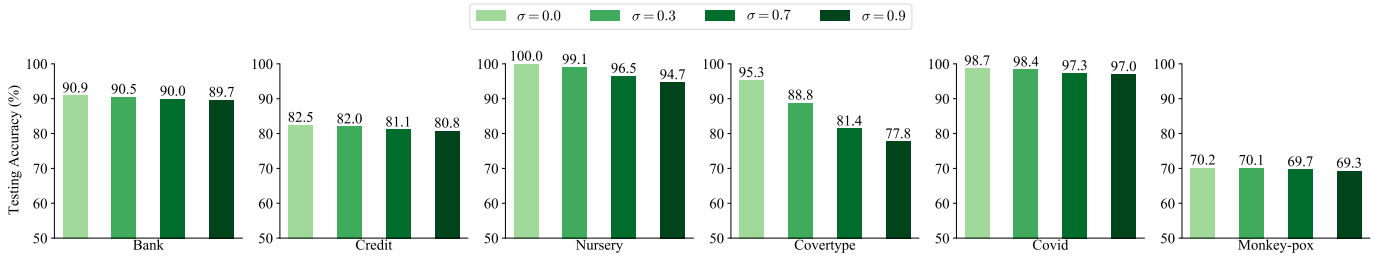


Fig. 3. Model accuracy: varying σ

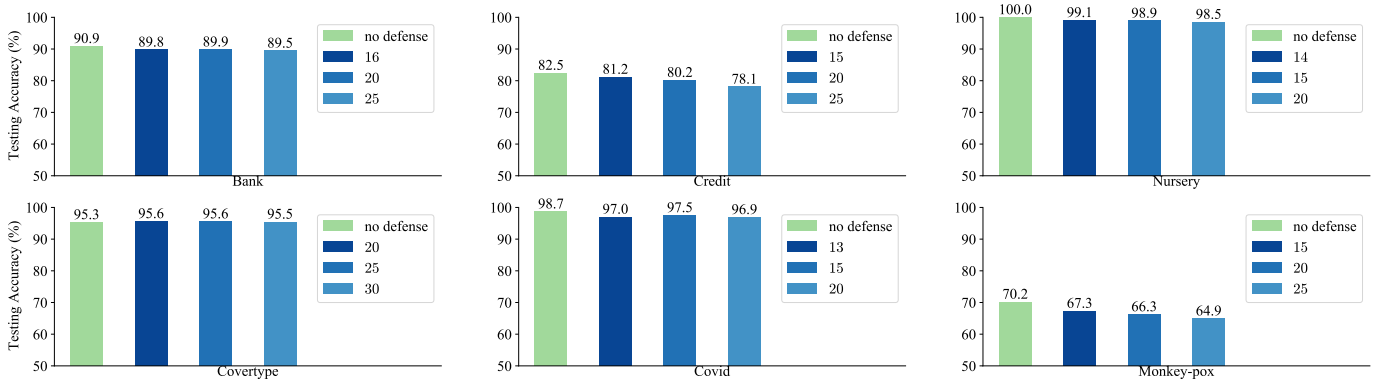


Fig. 4. Model accuracy: varying number of fabricated features

6 Evaluation

In this section, we evaluate the effectiveness of the proposed attacks and defense approaches through experiments.

6.1 Experimental Setup

We implement the FL algorithm with our attack and defense methods in PyTorch [22]. The experiments were conducted on a computer equipped with Intel(R) Xeon(R) CPU @ 2.20GHz and 16GB RAM, running Ubuntu 20.04.4 LTS.

For each dataset, we randomly split it so that 90% of the data records are used for training and the remaining 10% are for testing. We optimize the neural network for 100 epochs using SGD with momentum, where the momentum is set as 0.9. The base learning rate is 0.1 and we reduce the learning rate by a factor of 10 after 30, 60, and 90 epochs, respectively. We adopt cross-entropy loss with a 10^{-4} weight decay as the objective function. Such a hyper-parameter setting is commonly used in training neural networks (e.g., [23], [24]). Each experiment is repeated 20 times.

6.2 Datasets and Models

In our experiments, we use the following six public datasets from the UCI machine learning repository [25] and Kaggle[2].

- Bank [26] is a dataset that contains information about a bank's 41188 clients with 20 attributes. The goal is to predict whether the client will subscribe to a term deposit. We split the input features so that the passive party owns 8 features including a binary feature "contact".

2. https://www.kaggle.com

- Credit [27] is a dataset that consists of 30000 consumers' credit information where each consumer has 23 attributes. The task is to predict whether a cardholder will have a default payment. We split the features so that the passive party has 10 input features, among which "gender" is a binary attribute.
- Nursery [28] dataset consists of 12960 records of nursery-school applications, where each record contains 8 features. The target is the final evaluation of every application, which is divided into 5 levels. We split the features so that the passive party has 6 features including one binary feature about the financial standing of the families.
- Covertype [29] is a dataset of 581012 data records with each record contains 54 attributes extracted from the observation of a certain area. The objective is to determine the forest cover type out of 7 types. We split the features so that the passive party owns 10 features, including 4 binary features that are converted from a categorical feature by one-hot encoding.
- COVID [30] is a dataset that contains 5434 records about whether a person has COVID or not along with various symptoms presence. The target is to predict whether a person has got COVID or not based on the symptoms. We split the features so that the passive party owns 12 features, all of which are boolean features that represent the presence of some symptoms.
- Monkey-pox [31] is a synthetic dataset generated based on an article published by the BMJ [32]. It contains features about the symptoms and infection of other diseases of 25000 patients and the target is to predict whether they have monkey-pox. We split the dataset so that the passive party owns 8 features, all of which are binary features.

Table 1 summarizes the information about the datasets. We use d and d_B to represent the total number of features and the number of features owned by the passive party. The number of classes is denoted by C. The column "NN Structure" depicts the number of neurons in each hidden layer of the neural network. Different model structures (i.e., number of neurons in hidden layers) are employed for different datasets.

TABLE 1
Description of datasets and models used in our experiments.

Dataset	#Instances	d	C	d_B	NN Structure
Bank	41188	20	2	8	$\{60, 30, 10\}$
Credit	30000	23	2	10	$\{100, 50, 20\}$
Nursery	12960	8	5	6	$\{200, 100\}$
Covertype	581012	54	7	10	$\{200, 200, 200\}$
COVID	5434	20	2	12	$\{200, 100\}$
Monkey-pox	25000	9	2	8	$\{100, 50, 20\}$

6.3 Attacks

Effectiveness. We first conduct experiments on training neural networks without adding noise. After training, we first extract intermediate results by feeding the entire dataset into the model. Then we perform our attack algorithm to recover the input binary features. Our experimental results show that we can recover the input binary features with 100% accuracy.

For the bank, credit, and nursery datasets, there is only one input binary feature on the passive side. In our experiments, our attack algorithm outputs the input binary feature as expected. For the Covid and monkey-pox datasets, there are more than one binary feature. Our attack can recover all the input binary features.

For the covertype dataset, the passive party has a categorical feature, which is converted to 4 binary features by one-hot encoding. The 4 binary vectors don't overlap at any coordinate. Thus, in this case, the element-wise sum of any subset of the 4 binary vectors is also a binary vector in the column space of the intermediate output. As a result, our attack will be able to identify it as well. In our experiment, the attack algorithm successfully finds the binary features along with their combinations (totally $2^4 - 1 = 15$ binary vectors, corresponding to all non-empty subsets of the 4 binary features), from which we can reconstruct the input categorical feature.

Effectiveness in the Presence of Gaussian Noise. If the passive party adds some noise before transmitting the intermediate results, an attack by solving linear equations becomes ineffective since the linear equations no longer hold. To handle this, we propose Algorithm 2, which relies on solving linear regression.

We test this attack over six datasets with Gaussian noise added to the intermediate results. To measure the effectiveness of our attack, we compare the output of our attack algorithm with the true input features. We introduce the concept of attack accuracy defined as,

$$\max_{\boldsymbol{x} \in \text{ input features}} \frac{1}{n} \sum_{i=1}^{n} \mathbb{I}(\boldsymbol{x}_i^* = \boldsymbol{x}_i),$$

where $\boldsymbol{x}^*$ is the output of Algorithm 2 and $\mathbb{I}$ is the indicator function. That is, we consider all input features and find the one that has the most coordinates that match the algorithm's output. For one-hot features, we also take their combinations into account. Clearly, a high attack accuracy indicates the effectiveness of the attack while a low attack accuracy means the protection is successful.

In the experiments we set $r = d_B + 1$ in Algorithm 2. Since it is a randomized algorithm, we run it for 20 times and choose the solution that has the minimum error. We present the range of attack accuracy under different σ in Figure 3. The error bars represent the maximum and minimum attack accuracy in our experiments.

From the results, we can observe that the Gaussian noise defense method cannot provide satisfactory protection. For example, on the bank dataset, the lowest average attack accuracy we can achieve is still over 60%. If we look at the maximum attack accuracy, the security issue becomes more serious. For the covertype dataset, even if the average attack accuracy is lower than 70%, the maximum attack accuracy can be more than 95%, which means most of the input samples are leaked. Thus, it cannot guarantee that our data is secure. Also, we can see that different datasets require different σ to reach the lowest attack accuracy. It would be hard to determine an appropriate value of σ.

Performance. When Gaussian noise is introduced, the performance of the neural network may be affected. In addition to attack accuracy, we also evaluate the effect on model accuracy when we add noise to the intermediate output. We measure the model accuracy under different levels of noise (i.e. different σ) on every dataset. The results are depicted in Figure 2, which shows that the model accuracy drops as σ increases. Therefore, although masking the intermediate output with Gaussian noise can provide a certain level of protection, it also sacrifices model quality.

6.4 Countermeasure

The masquerade defense method aims at misleading the attacker to randomly generated binary features. Thus, the attacker will only find the fabricated binary features after performing the attack so that the true input features will be properly protected.

We evaluate our masquerade defense over the six datasets. In our experiments, we set the number of fabricated features m to be 1 and compare the fabricated feature and the solution produced by the attack algorithm. We find that the solution that the attack algorithm outputs matches one of the fabricated features as expected. This indicates that our defense effectively misguides the attacker to the randomly generated binary features, and therefore protects true input features. We also plot the attack accuracy in Figure 3. The values are all around 50% because the fabricated features are randomly set to be 0 or 1 with equal probability.

TABLE 2
Attack accuracy under adaptive attack

Dataset	m	Accuracy
Bank	1	93.8%
Bank	2	90.2%
Credit	1	97.7%
Credit	2	97.7%

As we analyzed in Section 5, the attacker can adaptively change the way of attack to recover the true input under masquerade defense. We train models on the bank and credit datasets using masquerade with $m = 1$ and 2. Then we perform adaptive attack on the intermediate results. Table 2 shows it is still possible to attack with high accuracy. Thus we have to set m large enough to prevent such a kind of attack.

TABLE 3
Training time with and with out defense

Dataset	Time w/o defense	Time with defense
Bank	41.8	46.7
Credit	34.5	38.2
Nursery	18.1	19.9
Covertype	1560	1712.5
Covid	7.7	8.3
Monkey-pox	27.6	30.3

We also wonder how this masquerade method affects model performance because the fabricated features may introduce some noise in the training process. For each dataset, we set m (the number of fabricated features) to be large enough so that 2^m is greater than the total number of samples. As we showed in Section 5, such an m can protect the input data against adaptive attack. We adopt various m and measure the model performance. The results are shown in Figure 4. We can see the accuracy degrades as the number of fabricated features increases. Moreover, we list the time usage of training a model with and without defense in Table 3. The results show that the masquerade method won't increase the training time too much since it only modifies the first layer.

7 Discussion

The attack mechanisms proposed are applicable under a variety of scenarios. For instance, one might think of simply adding a bias term or representing a binary feature by other values instead of 0 and 1 to invalidate the attack. However, we can pick one row of $\boldsymbol{Z}_B$ and subtract this row from other rows to eliminate the bias term. Then we can apply our attack to the resulting matrix.

Noticeably, in this work, we focus on the attack and countermeasure only for two-party VFL. However, our attack methods can be easily extended to the multi-party scenario. More specifically, if the passive parties send their intermediate results to the active party directly, we can perform the attack on the output sent from each passive party exactly the same way as in the two-party scenario. When the passive parties use secure aggregation to sum their intermediate results, our attack algorithm is still applicable, with the dimension d_B replaced by the total dimension of input features owned by all passive parties. Even if they adopt the protection proposed in [16], the sum of intermediate results of all parties (including the active party) is still exposed to the active party. Our proposed attack algorithm can be applied to the sum to extract binary features as long as the total dimension is still within reach.

Moving beyond binary features, for instance, categorical features, a common way to do feature engineering is one-hot encoding. In this case, our attack can find the converted binary features (and their sums), from which we can recover the categorical features. In the case that a categorical feature is transformed into a single multi-valued feature, our attack still works if the attacker knows what the values are.

A limitation is that our attack methods are based on the fact that the cut layer is the input layer. If we cut at the other layers (e.g., the second layer), our attack algorithms cannot work because the linearity the algorithms rely on no longer holds after the nonlinear activation functions are introduced.

When the cut layer is not the input layer, our impossibility and hardness results also hold, which means it is at least hard as in the case that the cut layer is the input layer. Also, the well-known universal approximation theorem [33] suggests that the output can be arbitrary when the local model contains two or more layers. Thus it will be much more difficult for the attacker to recover the input - for any input, there exist some model weights that generate a specific output.

8 Conclusion

In this paper, we take the initiative to study the feature security problem of DNN training in VFL. We first prove

that feature attacks are not possible when the attacker has zero knowledge of the dataset. We then focus on client data with binary features, and show that unless the feature space is exceedingly large, we can precisely reconstruct the binary features in practice with a robust search-based attack algorithm. We proceed to present a defense mechanism that overcomes such binary feature vulnerabilities by misleading the adversary to search for fabricated features. Our experiments show that our feature reconstruction attack is extremely effective against VFL on realistic DNN training tasks. Yet, the defense method proposed can effectively thwart the attack with a negligible loss in model accuracy.

REFERENCES

[1] "California Consumer Privacy Act (CCPA)," https://oag.ca.gov/privacy/ccpa, 2018.

[2] "Regulation (EU) 2016/679 of the European Parliament and of the Council of 27 April 2016 on the protection of natural persons with regard to the processing of personal data and on the free movement of such data, and repealing Directive 95/46/EC (General Data Protection Regulation)," https://eur-lex.europa.eu/eli/reg/2016/679/oj, 2016.

[3] "California Consumer Privacy Act (CCPA)," https://www.priv.gc.ca/en/opc-news/news-and-announcements/2018/an_181010/, 2018.

[4] Q. Yang, Y. Liu, T. Chen, and Y. Tong, "Federated machine learning: Concept and applications," *ACM Transactions on Intelligent Systems and Technology (TIST)*, vol. 10, no. 2, pp. 1–19, Feb. 2019.

[5] 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.

[6] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas, "Communication-efficient learning of deep networks from decentralized data," in *Proc. AISTATS*, 2017.

[7] H. Weng, J. Zhang, F. Xue, T. Wei, S. Ji, and Z. Zong, "Privacy leakage of real-world vertical federated learning," Apr. 2021.

[8] C. Fu, X. Zhang, S. Ji, J. Chen, J. Wu, S. Guo, J. Zhou, A. X. Liu, and T. Wang, "Label inference attacks against vertical federated learning," in *31st USENIX Security Symposium (USENIX Security 22)*. Boston, MA: USENIX Association, Aug. 2022, p. 18.

[9] X. Jiang, X. Zhou, and J. Grossklags, "Comprehensive analysis of privacy leakage in vertical federated learning during prediction," *Proceedings on Privacy Enhancing Technologies*, vol. 2022, no. 2, pp. 263–281, Apr. 2022.

[10] Y. Liu, Y. Kang, C. Xing, T. Chen, and Q. Yang, "A secure federated transfer learning framework," *IEEE Intelligent Systems*, vol. 35, no. 4, pp. 70–82, 2020.

[11] Y. Hu, D. Niu, J. Yang, and S. Zhou, "Fdml: A collaborative machine learning framework for distributed features," in *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2019, pp. 2232–2240.

[12] D. Romanini, A. J. Hall, P. Papadopoulos, T. Titcombe, A. Ismail, T. Cebere, R. Sandmann, R. Roehm, and M. A. Hoeh, "Pyvertical: A vertical federated learning framework for multi-headed splitnn," *arXiv preprint arXiv:2104.00489*, 2021.

[13] O. Li, J. Sun, X. Yang, W. Gao, H. Zhang, J. Xie, V. Smith, and C. Wang, "Label leakage and protection in two-party split learning," in *International Conference on Learning Representations*, 2022.

[14] Y. Liu, T. Zou, Y. Kang, W. Liu, Y. He, Z. Yi, and Q. Yang, "Batch label inference and replacement attacks in black-boxed vertical federated learning," Feb. 2022.

[15] X. Luo, Y. Wu, X. Xiao, and B. C. Ooi, "Feature inference attack on model predictions in vertical federated learning," in *2021 IEEE 37th International Conference on Data Engineering (ICDE)*. IEEE, 2021, pp. 181–192.

[16] F. Fu, H. Xue, Y. Cheng, Y. Tao, and B. Cui, "Blindfl: Vertical federated machine learning without peeking into your data," in *Proceedings of the 2022 International Conference on Management of Data*, Jun. 2022, pp. 1316–1330.

[17] R. M. Karp, "Reducibility among combinatorial problems," *Complexity of computer computations*, pp. 85–103, 1972.

[18] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *nature*, vol. 323, no. 6088, pp. 533–536, 1986.

[19] T. Tieleman, G. Hinton *et al.*, "Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude," *COURSERA: Neural networks for machine learning*, vol. 4, no. 2, pp. 26–31, 2012.

[20] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[21] M. W. Mahoney, "Randomized algorithms for matrices and data," *Foundations and Trends® in Machine Learning*, vol. 3, no. 2, pp. 123–224, Nov. 2011.

[22] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga *et al.*, "Pytorch: an imperative style, high-performance deep learning library," in *Proceedings of the 33rd International Conference on Neural Information Processing Systems*, 2019, pp. 8026–8037.

[23] L. Yang, R.-Y. Zhang, L. Li, and X. Xie, "Simam: A simple, parameter-free attention module for convolutional neural networks," in *International conference on machine learning*. PMLR, 2021, pp. 11 863–11 874.

[24] P. Chen, S. Liu, H. Zhao, and J. Jia, "Distilling knowledge via knowledge review," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2021, pp. 5008–5017.

[25] D. Dua and C. Graff, "UCI machine learning repository," 2017. [Online]. Available: http://archive.ics.uci.edu/ml

[26] S. Moro, P. Cortez, and P. Rita, "A data-driven approach to predict the success of bank telemarketing," *Decision Support Systems*, vol. 62, pp. 22–31, 2014.

[27] I.-C. Yeh and C.-h. Lien, "The comparisons of data mining techniques for the predictive accuracy of probability of default of credit card clients," *Expert systems with applications*, vol. 36, no. 2, pp. 2473–2480, 2009.

[28] M. Olave, V. Rajkovic, and M. Bohanec, "An application for admission in public school systems," *Expert Systems in Public Administration*, vol. 1, pp. 145–160, 1989.

[29] J. A. Blackard and D. J. Dean, "Comparative accuracies of artificial neural networks and discriminant analysis in predicting forest cover types from cartographic variables," *Computers and electronics in agriculture*, vol. 24, no. 3, pp. 131–151, 1999.

[30] H. Harikrishnan, "Symptoms and covid presence," 2020. [Online]. Available: https://www.kaggle.com/datasets/hemanthhari/symptoms-and-covid-presence

[31] M. Ahmed, "Monkey-pox patients dataset." 2022. [Online]. Available: https://www.kaggle.com/dsv/4271503

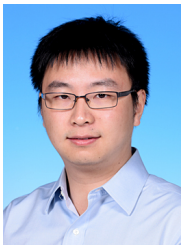
[32] A. Patel, J. Bilinska, J. C. H. Tam, D. Da Silva Fontoura, C. Y. Mason, A. Daunt, L. B. Snell, J. Murphy, J. Potter, C. Tuudah, R. Sundramoorthi, M. Abeywickrema, C. Pley, V. Naidu, G. Nebbia, E. Aarons, A. Botgros, S. T. Douthwaite, C. van Nispen tot Pannerden, H. Winslow, A. Brown, D. Chilton, and A. Nori, "Clinical features and novel presentations of human monkeypox in a central london centre during the 2022 outbreak: descriptive case series," *BMJ*, vol. 378, 2022. [Online]. Available: https://www.bmj.com/content/378/bmj-2022-072410

[33] K. Hornik, "Approximation capabilities of multilayer feedforward networks," *Neural networks*, vol. 4, no. 2, pp. 251–257, 1991.

This article has been accepted for publication in IEEE Transactions on Dependable and Secure Computing. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TDSC.2024.3521451

Peng Ye received his BEng degree from the Institute for Interdisciplinary Information Sciences at Tsinghua University in 2021. He is currently a PhD student at Hong Kong University of Science and Technology. His research focuses on federated learning and privacy-preserving learning algorithms.

Zhifeng Jiang received the BEng (Hons.) degree from the Department of Computer Science and Technology at Zhejiang University (ZJU) in 2019. Since then, he has been working towards a Ph.D. degree in the Department of Computer Science and Engineering, Hong Kong University of Science and Technology (HKUST). He is mainly interested in privacy-preserving machine learning systems, with a special focus on efficient and secure federated learning. He was a recipient of the Best Paper Runner-up Award at IEEE ICDCS 2021. He served on the artifact evaluation committees of ACM SOSP 2021, USENIX OSDI 2022, and USENIX ATC 2022.



Wang Wei received his B.Engr. and M.Engr. degrees from the Department of Electrical Engineering, Shanghai Jiao Tong University, China, in 2007 and 2010, respectively and his Ph.D. degree from the Department of Electrical and Computer Engineering, University of Toronto, Canada, in 2015. Since 2015, he has been with the Department of Computer Science and Engineering at the Hong Kong University of Science and Technology (HKUST), where he is currently an Associate Professor. He is also affiliated with the HKUST Big Data Institute. Dr. Wang's research interests cover the broad area of distributed systems, with focus on serverless computing, machine learning systems, and cloud resource management. He published extensively in the premier conferences and journals of his fields. His research has won the Best Paper Runner Up awards of IEEE ICDCS 2021 and USENIX ICAC 2013.

Bo Li is a Chair Professor in Department of Computer Science and Engineering, Hong Kong University of Science and Technology. He held the Cheung Kong Visiting Chair Professor in Shanghai Jiao Tong University (2010-2016), and was the Chief Technical Advisor for ChinaCache Corp. (NASDAQ:CCIH), one of the world leading CDN service providers. He made pioneering contributions in multimedia communications and the Internet video broadcast, in particular the Coolstreaming system, which was credited as the first large-scale Peer-to-Peer live video streaming system in the world. It attracted significant attention both from industry with substantial VC investment, and from academia in receiving the Test-of-Time Best Paper Award from IEEE INFOCOM (2015). He received 8 Best Paper Awards including IEEE INFOCOM (2021). He was the Co-TPC Chair for IEEE INFOCOM (2004).

He is a Fellow of IEEE. He received his PhD in the Electrical and Computer Engineering from University of Massachusetts at Amherst, and his B. Eng. (summa cum laude) in the Computer Science from Tsinghua University, Beijing, China.

Baochun Li received his Ph.D. degree from the Department of Computer Science, University of Illinois at Urbana-Champaign, Urbana, in 2000. Since then, he has been with the Department of Electrical and Computer Engineering at the University of Toronto, Canada, where he is currently a Professor. He holds the Bell Canada Endowed Chair in Computer Engineering since August 2005. His current research interests include cloud computing, security and privacy, distributed machine learning, federated learning, and networking. He is an IEEE Fellow and Fellow of the Canadian Academy of Engineering.