HashVFL: Defending Against Data Reconstruction Attacks in Vertical Federated Learning

Pengyu Qiu, Xuhong Zhang, Shouling Ji, Chong Fu, Xing Yang✉, Ting Wang

***Abstract*—Vertical Federated Learning (VFL) is a trending collaborative machine learning model training solution. Existing industrial frameworks employ secure multi-party computation techniques such as homomorphic encryption to ensure data security and privacy. Despite these efforts, studies have revealed that data leakage remains a risk in VFL due to the correlations between intermediate representations and raw data. Neural networks can accurately capture these correlations, allowing an adversary to reconstruct the data. This emphasizes the need for continued research into securing VFL systems.**

Our work shows that hashing is a promising solution to counter data reconstruction attacks. The one-way nature of hashing makes it difficult for an adversary to recover data from hash codes. However, implementing hashing in VFL presents new challenges, including vanishing gradients and information loss. To address these issues, we propose HashVFL, which integrates hashing and simultaneously achieves learnability, bit balance, and consistency.

Experimental results indicate that HashVFL effectively maintains task performance while defending against data reconstruction attacks. It also brings additional benefits in reducing the degree of label leakage, mitigating adversarial attacks, and detecting abnormal inputs. We hope our work will inspire further research into the potential applications of HashVFL.

***Index Terms*—Vertical Federated Learning, Deep Hashing.**

I. Introduction

Machine learning algorithms, particularly Deep Neural Networks (DNNs), have seen significant growth in recent decades [1]–[3]. DNNs have been applied in finance [4], [5], biomedicine [6], [7], and even military operations [8], [9]. However, data security and privacy are of the utmost importance in these sensitive fields, and strict laws and regulations like GDPR [10] and CCPA [11] limit the flow of data. This creates a dilemma between the need for large amounts of data in machine learning models and the restrictions on data flow.

Vertical federated learning (VFL) [12]–[15] is a trending paradigm that addresses a common dilemma faced by companies that share the same user group but differ in the features.

P. Qiu, S. Ji, C. Fu are with the College of Computer Science and Technology at Zhejiang University, Hangzhou, Zhejiang, 310027, China. E-mail:{qiupys,sji,fuchong}@zju.edu.cn

X. Zhang is with the School of Software Technology at Zhejiang University, Ningbo, Zhejiang, 315048, China. E-mail: zhangxuhong@zju.edu.cn

X. Yang is with the Hefei Interdisciplinary Center, National University of Defense Technology, Hefei, Anhui, 230037, China. E-mail: yangxing.1983@163.com. He is also the corresponding author of this paper.

T. Wang are with the College of Information Science and Technology at Pennsylvania State University, State College, PA, 16801, United States. E-mail: inbox.ting@gmail.com

Fig. 1. A typical scenario for using VFL involves Party A, an e-commerce company with features 1 and 2, and Party B, a bank with features 3, 4, 5 and the label. Together, they train a model that predicts loan approval decisions.

The concept is illustrated in Fig. 1. Suppose a bank, Party B, needs to improve its loan approval prediction and requires additional information from an e-commerce company, Party A. In VFL, instead of directly exchanging user data, each party uploads the intermediate results of their user features calculated by a bottom model to a neutral party for further processing. This way, the raw data remains confidential.

The main challenge in VFL is ensuring these intermediate results' privacy and security. Current frameworks adopt Secure Multi-Party Computation (SMC) methods, such as Homomorphic Encryption (HE) [16], [17], to provide these guarantees. HE allows computations to be performed in an encrypted environment, ensuring no one can access the intermediate results in plain text.

However, recent research has shown that such methods are insufficient. In particular, the studies in [18]–[21] have demonstrated that an adversary can reconstruct the intermediate results, and even the raw data of the target party by using the sample's posteriors and the parameters of the VFL model. The reason behind these successful data reconstruction attacks is the ability of deep neural networks (DNNs) to model the correlation between the intermediate calculations and the raw inputs. For instance, generative adversarial networks (GANs) [22], [23] have achieved remarkable success in image reconstruction. Researchers have investigated several ways [24], [25] to defend against data reconstruction attacks in VFL, such as increasing the correlation distance between the input and the corresponding activations. These methods do reduce information leakage, but not completely. In the worst case where the adversary knows the model and the output for a target sample [26], there remains a chance of the reconstruction attack.

To eliminate the reversibility, we propose a new VFL framework called HashVFL that uses hashing. The one-way nature of hashing allows our framework to block all attempts to recover intermediate calculations or raw inputs from the hash codes. However, the integration of hashing makes it difficult for models to learn during training because the gradients disappear. Moreover, hashing discards information, and to preserve

privacy, the length of the hash code must be minimized. These two factors will inevitably affect the model's performance. Given the above considerations, the design of our HashVFL framework addresses the following three challenges:

- **Learnability.** The challenge lies in balancing the trade-off between preserving privacy through hashing and ensuring the learnability of the model. The solution is to identify hash functions with easily estimable gradients so that the model can continue to train and maintain its performance.
  Our solution: To achieve the desired trade-off between privacy preservation and learnability, we use the following approach: 1) We add a $Sign$ function to binarize the intermediate calculations of each party. This is a common technique used in hashing [27] and binary neural networks [28]. 2) We employ the *Straight-Through Estimator* [29], [30] in back-propagation. This allows the gradient to pass through the $Sign$ function exactly as it is, avoiding gradient vanishing.
- **Bit balance.** Hashing leads to the loss of information in each bit. Furthermore, to minimize the risk of information leakage, it is desired to limit the length of the hash codes. To address this challenge, we introduce the concept of *bit balance*. This refers to maximizing the information carried by each bit, given a limited hash code length. Ideally, we aim for half of the samples to take a value of 1/-1 on each bit. This maximizes the information that the whole hash code can carry.
  Our solution: To address the above requirement, we propose the use of *Batch Normalization* (BN) [31]. BN normalizes the intermediate calculations of each dimension in a batch of samples to have a standard normal distribution. It means that roughly half of the samples in a batch will have positive values, and half will have negative values at each dimension. Hence, incorporating BN helps address the issue of the effectiveness of the bit.
- **Consistency.** Since the binarization maps intermediate results into the same latent space, intuitively, a sample's hash codes from different parties should be consistent. One way to do this is to add the constraint in training by comparing the difference of hash codes between parties. However, this approach may result in a high computational overhead if there are many parties, which will definitely limit the application of VFL.
  Our solution: To address the high computational overhead when comparing the hash codes of samples across many parties, a solution is to pre-define a set of binary codes for each class [32]. This way, each party only needs to compare their hash codes with these binary codes, reducing the complexity from $O(N^2)$ to $O(N)$, where N is the number of parties, making it suitable for scenarios with many parties. Additionally, the calculated differences between the sample's and target binary codes can also guide the optimization, as shown in [33].

Experimental results demonstrate that our proposed HashVFL maintains the performance of the main task across various data types. Furthermore, HashVFL provides additional advantages by reducing label leakage, mitigating adversarial attacks, and detecting abnormal inputs. Additionally, we assess HashVFL's performance in different conditions, such as varying numbers of parties, and find that it effectively handles various scenarios.

Our contributions are as follows:

- We propose a novel approach for enhancing data security and privacy in the VFL by integrating hashing techniques.
- We address three key challenges in the design of the hashing-based VFL framework, i.e., learnability, bit balance, and consistency, and present a practical solution.
- We conduct a comprehensive empirical evaluation of our framework, HashVFL, demonstrating its ease of use, versatility, and effectiveness in defending against data reconstruction attacks.

TABLE I
SYMBOLS AND NOTATIONS.

Notations	Definition
P_i, D_i	the i-th party in VFL, and P_i's dataset
$\mathcal{U}_i$, $\mathcal{F}_i$	D_i's sample/user space and feature space
f_i, f_{top}	P_i's bottom model and the top model
θ_i, θ_{top}	f_i's parameters, and f_{top}'s parameters
$\mathbf{x}_i^{(u)}$, $\mathbf{v}_i^{(u)}$, $\mathbf{h}_i^{(u)}$	a sample u's feature vector of D_i, $\mathbf{x}_i^{(u)}$'s corresponding output from f_i, and the hash code of $\mathbf{v}_i^{(u)}$

II. BACKGROUND

This section serves as an introduction to the background information relevant to VFL, data reconstruction attacks, hashing, and our threat model. TABLE I summarizes the necessary notations in this paper.

A. Vertical Federated Learning

Consider a set of N parties $\{P_1, P_2, \cdots, P_N\}$ working on a classification task, each having its own dataset $\{D_1, D_2, \cdots, D_N\}$. Each dataset D_i can be described as $(\mathcal{U}_i, \mathcal{F}_i)$, where $\mathcal{U}_i$ is the sample/user space and $\mathcal{F}_i$ is the feature space.

Before training, the parties must establish an overlapped sample space $\mathcal{U}$ as the intersection of all the sample spaces, i.e., $\mathcal{U} = \bigcap_{i=1}^{N} \mathcal{U}_i$. Then, the features of the samples in $\mathcal{U}$ from different $\mathcal{F}_i$ will be aligned based on their new indices in $\mathcal{U}$.

After the preparation of training set, P_i trains its bottom model, f_i. Let $\mathbf{x}_i^{(u)}$ denotes the feature vector (raw input) of the sample u from $\mathcal{F}_i$. The function of f_i is to map it into a $\tilde{d}$-dimensional latent space, i.e., $f_i(\mathbf{x}_i; \theta_i) : \mathbb{R}^{d_i} \rightarrow \mathbb{R}^{\tilde{d}}$, where θ denotes the model's parameters, and d_i refers to the size of $\mathbf{x}_i$'s dimension. We use $\mathbf{v}_i^{(u)}$ to represent u's output of f_i.

Then, $\mathbf{v}_i^{(u)}$ will be sent to a neutral third party's server for aggregation and further calculation. Specifically, let $\mathbf{v}_{cat}^{(u)} = [\mathbf{v}_1^{(u)}, \mathbf{v}_2^{(u)}, \cdots, \mathbf{v}_N^{(u)}]$ denote the concatenated vector of the sample u and f_{top} denote the top model deployed at the server. f_{top} is supposed to learn a mapping from $\mathbf{v}_{cat}^{(u)}$ to $\mathbf{v}_{top}^{(u)}$, where $\mathbf{v}_{top}^{(u)}$ is also the posterior for classification. Formally, f_{top} can be presented as $f_{top}(\mathbf{v}_{cat}; \theta_{top}) : \mathbb{R}^{N \times \tilde{d}} \rightarrow \mathbb{R}^{C}$, where C denote the number of classes.

Finally, $\mathbf{v}_{top}^{(u)}$ will be sent to the party who owns the label. Then, the party calculates the loss, e.g., cross-entropy loss, and the corresponding gradients. Using the chain rule, each model's parameters can be updated by passing the gradients. The above process can be formulated as:

$$\min_{\{\theta_i\}_{i=1}^{N},\theta_{top}} \mathbb{E}_{u\in\mathcal{U}}[\ell(\mathbf{x}_1^{(u)},\mathbf{x}_2^{(u)},\cdots,\mathbf{x}_N^{(u)},y;\{\theta_i\}_{i=1}^{N},\theta_{top})],$$

where ℓ denotes the loss function, and y is the label of u.

During the training process, intermediate calculations and gradients are transmitted, which are confidential information. To ensure privacy protection, existing frameworks, such as FATE [34], PySyft [35], TF Encrypted [36], and CrypTen [37], employ homomorphic encryption (HE) [38]. HE allows for vector operations, such as addition and multiplication, to be performed on encrypted data. In a nutshell, HE provides a secure environment for mathematical operations on sensitive information.

B. Data Reconstruction Attack

One persistent criticism of deep neural networks (DNNs) is their potential to violate user privacy by leaking information through the collected data. This issue is particularly prevalent in computer vision tasks that involve images.

In [39], Zhu et al. demonstrated that sensitive information can be reconstructed from leaked gradients. They achieved this by allowing the gradients of generated samples to closely approximate the gradients of target samples, resulting in high performance. The attack can be formulated as follows:

$$\tilde{\mathbf{x}}^*,\tilde{y}^* = \arg\min_{\tilde{\mathbf{x}},\tilde{y}} = \left\| \frac{\partial \ell(f(\tilde{\mathbf{x}},\theta),\tilde{y})}{\partial\theta} - \nabla\theta \right\|^2,$$

where $\tilde{\mathbf{x}}^*$, $\tilde{y}^*$ are the reconstructed sample and its inferred label; $\ell(\cdot)$ denotes the loss function; f, θ denote the model and its parameters; and $\nabla\theta$ is the target sample's gradients.

In [40], Pasquini et al. showed that an adversary could reconstruct the target image with knowledge of the model and the target image's posteriors under split learning scenarios. Ergodan et al. [41] showed that the knowledge can further be relaxed to the model structure's copy and the target sample's intermediate calculations. In [26], He et al. unified model stealing and data reconstruction attack. The attack can be formulated as follows:

$$\begin{cases} \tilde{\mathbf{x}}^* = \arg\min_{\tilde{\mathbf{x}}} \ell(f_{\tilde{\theta}}(\tilde{\mathbf{x}}), f_\theta(\mathbf{x})) + L(\tilde{\mathbf{x}}) \\ \tilde{\theta}^* = \arg\min_{\tilde{\theta}} \ell(f_{\tilde{\theta}}(\tilde{\mathbf{x}}), f_\theta(\mathbf{x})), \end{cases}$$

where $\tilde{\mathbf{x}}^*$ is the reconstructed sample; f, θ denote the model and its parameters; $\tilde{\theta}^*$ is the approximated parameters; $\ell(\cdot,\cdot)$ measures the distance between two terms; and $L(\tilde{\mathbf{x}})$ denotes the penalty function of $\tilde{\mathbf{x}}$ to guide the generation. For example, in [26], they used the Total Variation term [42] to smooth the noise. The optimization was alternated between samples and parameters to achieve the best results.

In [18], Luo et al. revealed that the encryption mechanism in VFL cannot prevent the adversary from reconstructing the data by a generative adversarial network. Weng et al. [19] also verified the conclusion across more machine learning algorithms. Moreover, in [20], Qiu et al. also showed that the reconstructed intermediate calculations could reflect the topology information used in graph neural networks.

In summary, side-channel information, such as gradients and intermediate calculations, can potentially reveal sensitive information due to the approximation capabilities of DNNs.

C. Hashing

Conventional hash functions, such as MD5 [43], are data-independent, meaning they do not retain information about the input data. They take an input of arbitrary length and produce a fixed-length output, commonly referred to as a 'fingerprint' or 'message digest', through various mathematical operations.

In contrast to conventional hash functions, data-dependent hash methods retain information about the input data in their design. This is required for tasks such as similar image retrieval or product recommendations. An example of a data-dependent hash method is Locality-Sensitive Hashing (LSH) which is widely used for Approximate Nearest Neighbor (ANN) search.

Specifically, for two samples u and v, LSH requires that their hash codes should have the property:

$$Pr[h(\mathbf{x}^{(u)}) = h(\mathbf{x}^{(v)})] : \begin{cases} \geq p_1 & if \quad d(\mathbf{x}^{(u)},\mathbf{x}^{(v)}) \leq d_1 \\ \leq p_2 & else \quad d(\mathbf{x}^{(u)},\mathbf{x}^{(v)}) \geq d_2, \end{cases}$$

where $\mathbf{x}^{(u)}$ denotes the sample u's feature vector, $h(\cdot)$ is the hash function, $d(\cdot)$ is the distance calculation function, p_1, p_2, and d_1, d_2 are the specific values of probability and distance. The property means that for two samples' feature vectors, if their distance is less than d_1, their hash code should at least have the probability p_1 to have the same value; on the contrary, if it is less than d_2, the probability of their hash codes are same should not beyond p_2.

Recently, there has been a growing body of work [27], [44]–[46] that has demonstrated the ability of DNNs to maintain the data-dependent property of hashing for approximate nearest neighbor search. These methods extract abstract representations of the data using DNNs and then binarize the representations in order to retain the correlation between the samples and maintain the effectiveness of retrieval. Our HashVFL also leverages these works to address learnability.

D. Threat Model

The threat model is based on the assumption of honest-but-curious adversaries [47], meaning that all parties and the server will follow the requirements specified in VFL but may try to learn information about the intermediate calculations and raw data located on the target party.

Additionally, it is assumed that the adversary knows each other's bottom model and samples' hash codes, but the local data of each party is strictly confidential. This is the strongest assumption for data reconstruction attacks and if HashVFL can defend against attacks under these conditions, it is likely to be effective with weaker assumptions where the adversary has less knowledge.

III. Methodology

This section provides the framework of HashVFL and the implementation details of each component.

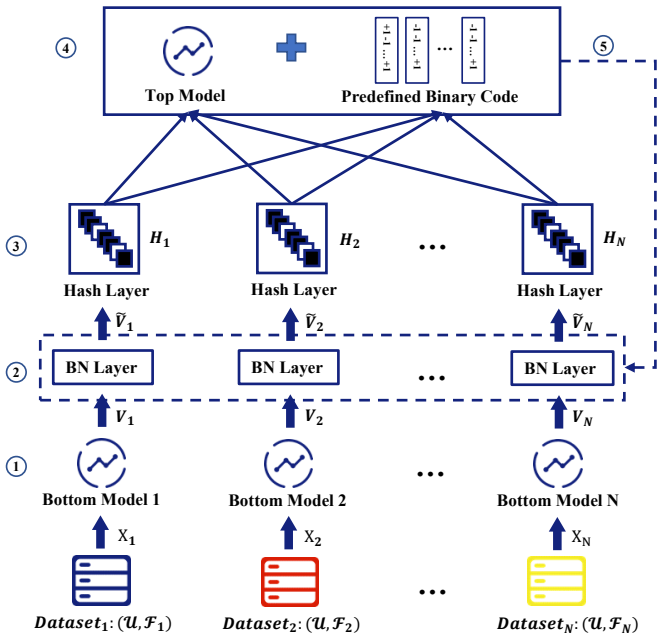

Fig. 2. Overview of HashVFL. 1) Each party uses its bottom model to extract abstractions from local data. 2) The extracted abstractions are then normalized through a BN layer. 3) Normalized abstractions obtained from 2) are binarized by the hash layer and then uploaded to the server. 4) The server uses the top model to calculate classification loss and the distance between these codes and their target pre-defined binary codes. 5) The server calculates the gradients and transmits them back to corresponding parties. Please note that the gradients pass through the hash layer as they are due to the utilization of the Straight-Through Estimator (STE), enabling updates to commence from the batch normalization layer.

A. Overview of HashVFL

Figure 2 illustrates the pipeline of HashVFL. Following is a summary of the details.

First, each party, P_i, where $i \in \{1, 2, \cdots, N\}$ and N is the number of parties, prepares the training dataset $D_i = (\mathcal{U}, \mathcal{F}_i)$. Then, P_i selects a specific model, f_i, for extracting information from the raw data. For example, if D_i is an image dataset, ResNet [48] and VGG [49] are feasible candidates, which are popular model architectures for image classification. $\mathbf{x}_i$ denotes a sample's feature vector from D_i, and $\mathbf{X}_i$ denotes a batch of samples' feature vectors.

Next, the outputs of $\mathbf{X}_i$ from f_i, i.e., $\mathbf{V}_i$, go through a Batch Normalizing transform (BN) layer, which is mandatory. The transformed outputs, $\tilde{\mathbf{V}}_i$, now achieve a balance at each dimension, thus satisfying our 'bit balance' requirement. Then, these balanced outputs have to be binarized by a hash layer. $\mathbf{H}_i$ denotes the hash codes.

Finally, $\mathbf{H}_i$ are concatenated at the server side, i.e., $\mathbf{H} = [\mathbf{H}_1, \cdots, \mathbf{H}_N]$. The top model, f_{top}, then calculates the posteriors of $\mathbf{H}$. Then, the posteriors and their ground truth $\mathbf{Y}$ will be compared to calculate classification loss, usually cross-entropy (CE) loss. Meanwhile, $\mathbf{H}_i$ is also compared to pre-defined binary codes $\mathbf{o} \in \{-1, +1\}^{C \times \tilde{d}}$, where C is the number of classes and $\tilde{d}$ is the size of the hash code, for the consistency requirement. Specifically, we calculate the distance between $\mathbf{h}_i^{(u)} \in \mathbf{H}_i$ and its target binary code $\mathbf{o}_y$, where u denotes a sample in the batch with label y, $\mathbf{h}_i^{(u)}$ refers to u's hash code, and $\mathbf{o}_y$ denotes the class y's corresponding code.

In summary, the loss function is designed as follows:

$$\mathcal{L} = CE(f_{top}(\mathbf{H}), \mathbf{Y}) + (\mathbf{1} - Cos(\mathbf{H}, \mathbf{o_Y})),$$

where $CE(\cdot, \cdot)$ denotes the CE loss term, $(\mathbf{1} - Cos(\cdot, \cdot))$ denotes the cosine distance loss term, and $\mathbf{o_Y}$ denotes $\mathbf{Y}$'s corresponding pre-defined codes.

During the back-propagation process, the gradients are passed as they are through the hash layer. Therefore, only BN layers' parameters and θ_i need to be updated. Algorithm 1 describes the mainframe of HashVFL in training.

Algorithm 1 Mainframe of HashVFL in training.

Require: $\{D_i, f_i\}_{i=1}^N$, f_{top}, pre-defined binary codes $\mathbf{o}$
Ensure: $\{f_i\}_{i=1}^N$, f_{top} for inference

for each epoch **do**
  for each batch $(\mathbf{X}, \mathbf{Y})$ **do**
    During forward process:
    for At each P_i **do**
      $\mathbf{V}_i \leftarrow f_i(\mathbf{X}_i)$
      $\tilde{\mathbf{V}}_i \leftarrow BN(\mathbf{V}_i)$
      $\mathbf{H}_i \leftarrow Sign(\tilde{\mathbf{V}}_i)$
      Send $\mathbf{H}_i$ to the server
    At the server:
    $\mathbf{H} \leftarrow concate(\{\mathbf{H}_i\}_{i=1}^N)$
    $\mathbf{o_Y} \leftarrow onehot(\mathbf{Y}) \times \mathbf{o}$
    $\mathcal{L} \leftarrow CE(f_{top}(\mathbf{H}), \mathbf{Y}) + (\mathbf{1} - Cos(\mathbf{H}, \mathbf{o_Y}))$
    During backward process:
    At the server:
    for each $\mathbf{H}_i$ **do**
      Calculate $\frac{\partial \mathcal{L}}{\mathbf{H}_i}$
      Send $\frac{\partial \mathcal{L}}{\mathbf{H}_i}$
    for At each P_i **do**
      $\frac{\partial \mathcal{L}}{\partial \tilde{\mathbf{V}}_i} \leftarrow \frac{\partial \mathcal{L}}{\mathbf{H}_i}$
      Update the following parameters

In the following, we present the details of our implementation of the BN layer, the Hash layer, and the design of pre-defined binary codes.

B. BN Layer

Batch Normalization (BN) was first proposed by Sergey et al. [31] to solve the problem of *Internal Covariate Shift*. That is, during neural networks' training, the distribution of activations will shift due to the change in networks' parameters, In this paper, however, we use the design of BN to address our proposed bit balance.

Formally, given a batch of samples $\mathcal{B} = \{\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \cdots, \mathbf{x}^{(m)}\}$, where m denotes the size of the batch, a BN layer first normalizes each $\mathbf{x}^{(i)}$ with batch mean $\mu_{\mathcal{B}} : \frac{1}{m}\Sigma_{i=1}^m \mathbf{x}^{(i)}$ and batch variance $\sigma_{\mathcal{B}}^2 : \frac{1}{m}\Sigma_{i=1}^m (\mathbf{x}^{(i)} - \mu_{\mathcal{B}})^2$, i.e., $\bar{\mathbf{x}}^{(i)} : \frac{\mathbf{x}^{(i)} - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}}$, where $\bar{\mathbf{x}}^{(i)}$ denotes normalized value, and set ϵ to prevent from division by zero. Hence, we have $\Sigma_{i=1}^m \bar{\mathbf{x}}^{(i)} = 0$ and $\frac{1}{m}\Sigma_{i=1}^m \bar{\mathbf{x}}^{2(i)} = 1$, if we neglect ϵ. In such a way, we can guarantee that in each batch, the samples are evenly assigned positive and negative values on each bit.

However, simply normalizing each layer's input may change what the layer can represent [31]. Therefore, there are two more parameters γ and β in the BN layer to scale and shift the normalized value: $\tilde{\mathbf{x}}^{(i)} = \gamma\bar{\mathbf{x}}^{(i)} + \beta$. This way, the BN layer can recover the original activations if that is optimal for training.

During inference, for a batch of samples $\mathcal{B}_{inf}$, we transform them with $\tilde{\mathbf{x}} = \frac{\gamma}{\sqrt{Var[\mathbf{x}]+\epsilon}}\bar{\mathbf{x}} + (\beta - \frac{\gamma\mathbb{E}[\mathbf{x}]}{\sqrt{Var[\mathbf{x}]+\epsilon}})$, where $\mathbb{E}[\mathbf{x}] = \mathbb{E}_{\mathcal{B}_{inf}}[\mu_{\mathcal{B}_{inf}}]$ and $Var[\mathbf{x}] = \frac{m}{m-1}\mathbb{E}_{\mathcal{B}_{inf}}[\sigma^2_{\mathcal{B}_{inf}}]$.

C. Hash Layer

Learning to hash is an NP-hard binary optimization problem [50]. Therefore, a line of work [46] adopts $tanh$ or $sigmoid$ for approximation, which ensures the differentiability. However, these attempts still leave the risks of leakage in training. Therefore, we use the $Sign$ function for binarization, which provides protection from training to inference. Formally, the $Sign$ function is defined as follows:

$$h = Sign(v) = \begin{cases} +1 & if\ v \geq 0 \\ -1 & otherwise, \end{cases}$$

where h is the binary value of the input v. For vectors, the function operates element-wise.

To solve the vanishment of gradients by using $Sign$, we combine the *Straight-Through Estimator* (STE) in back-propagation. In [29], to solve the challenge of estimating the gradients when stochastic or hard non-linearity neurons are used in neural networks, Bengio et al. proposed four estimators. STE is the most efficient solution, as it behaves like the identity function. Specifically, given a vector $\mathbf{v}$ and its hash code $\mathbf{h}$, which is obtained through the $Sign$ function, the gradients $\mathbf{g}$ of $\mathbf{v}$ can be estimated as:

$$\mathbf{g} = \frac{\partial\mathcal{L}}{\partial\mathbf{v}} = \frac{\partial\mathcal{L}}{\partial\mathbf{h}} \cdot \frac{\partial\mathbf{h}}{\partial\mathbf{v}} \approx \frac{\partial\mathcal{L}}{\partial\mathbf{h}},$$

where $\mathcal{L}$ is the calculated loss of $\mathbf{v}$. With such an approximation, the model's training can thus continue.

D. Generation of Pre-defined Binary Codes

The pre-defined binary codes are used to reduce the computation complexity in keeping the hash codes' consistency. Moreover, they are also crucial for the classification task.

According to [51], the probability of two vectors $\mathbf{v}_i$ and $\mathbf{v}_j$ having the same hash code under a family of hash functions using random hyperplane techniques is $1 - \frac{\theta_{ij}}{\pi}$, where θ_{ij} denotes the angle between $\mathbf{v}_i$ and $\mathbf{v}_j$. Therefore, to make the sample's hash codes discriminative for the task, we should let the pre-defined binary codes be as independent as possible, i.e., orthogonal to each other.

To achieve the orthogonality, we follow the practice in [33], i.e., randomly generating the binary codes according to the Bernoulli distribution with $p = \frac{1}{2}$, where p denotes the probability of signing $+1$ on each bit. The derivation of p's value is as follows.

First, for two randomly generated binary codes $\mathbf{o}_1$ and $\mathbf{o}_2$ with size n, we have $Pr(cos(\mathbf{o}_1, \mathbf{o}_2) = 0) = Pr(\Sigma_{i=1}^{n} o_{1i} \cdot o_{2i} = 0)$. Since o_{1i} and o_{2i} both obey the Bernoulli distribution with p, the probability q of that o_{1i} and o_{2i} have the same value is $p^2 + (1-p)^2$.

Then, $\Sigma_{i=1}^{n} o_{1i} \cdot o_{2i}$ becomes the binomial distribution with q, i.e., n consecutive Bernoulli trials. Hence, $Pr(\Sigma_{i=1}^{n} o_{1i} \cdot o_{2i} = 0) = \binom{n}{\frac{n}{2}} q^{\frac{n}{2}}(1-q)^{\frac{n}{2}}$. For a specific n, $\binom{n}{\frac{n}{2}}$ is a constant. With inequality $q(1-q) \leq (\frac{q+(1-q)}{2})^2$, where the equal sign is obtained when $q = 1-q$, we have $q = \frac{1}{2}$ to maximize $Pr(cos(\mathbf{o}_1, \mathbf{o}_2) = 0)$. Therefore, $p^2 + (1-p)^2 = \frac{1}{2}$, where we finally prove that $p = \frac{1}{2}$ is the best.

E. Metrics of Distance

Hamming distance [52] is commonly used in binary codes' distance calculation, while in this paper, we mainly discuss cosine similarity. The reason is that they are literally equal for binary codes [33].

For example, given a sample's hash code $\mathbf{h}$ and its corresponding target binary code $\mathbf{o}$, the hamming distance between them is: $H(\mathbf{h}, \mathbf{o}) = \frac{\tilde{d} - \mathbf{h}^T\mathbf{o}}{2}$, where $H(\cdot)$ is the hamming distance calculation function and $\tilde{d}$ is the length of codes.

Then, since $\mathbf{h}^T\mathbf{o} = \|\mathbf{h}\|\,\|\mathbf{o}\|\,cos\theta$, where $\|\cdot\|$ is the Euclidean norm and θ is the angle between $\mathbf{h}$ and $\mathbf{o}$, and both $\|\mathbf{h}\|$ and $\|\mathbf{o}\|$ equal to $\sqrt{\tilde{d}}$, we have: $H(\mathbf{h}, \mathbf{o}) = \frac{\tilde{d} - \tilde{d}cos\theta}{2} = \frac{\tilde{d}}{2}(1 - cos\theta)$. Therefore, minimizing the hamming distance equals to minimizing the angle between the two binary codes, which also means maximizing the cosine similarity between them.

IV. Experimental Setup

In our evaluation of the HashVFL, we use various datasets, models, and training details to assess its performance. We primarily consider the two-party VFL scenario following [18]–[21], as it is the most popular scenario in the industry due to the consideration of communication cost and computation overhead [14]. However, the framework can be easily extended to multi-party scenarios, and we also evaluate the impact of the number of parties in Section VI-A. The details of the datasets, models, and training procedures are presented in the following.

A. Datasets

Real-world VFL datasets [53]–[55] are proprietary and cannot be publicly accessed. Therefore, we choose to evaluate on public datasets instead. Specifically, we pick up six datasets, including three image datasets, two tabular datasets, and one text dataset: 1) MNIST [56] is the most popular benchmark for evaluation, which has a training set of 60,000 examples and a test set of 10,000 examples; 2) CIFAR10 [57] is another public dataset for image classification, which consists of 60,000 images with 10 classes; 3) FER [58] is used for facial expression recognition, which consists of a training set of 28,709 examples and a test set of 7178 examples; 4) Company Bankruptcy Prediction Dataset (denoted by CBPD) [59] was collected from the Taiwan Economic Journal for the years from 1999 to 2009, which consists of 6,819 instances with 95 attributes and 2 classes; 5) CRITEO [60] is used for Click-Through-Rate (CTR) prediction, which consists of 100,000 instances; 6) IMDb [61] is widely used in text analysis,

consisting of a training set of 25,000 reviews and a test set of 25,000 reviews.

We remove the categorical features in CBPD and CRITEO, as removing them helps improve the performance from the practices of Kaggle [1] (a community hosting competitions on data science) and the results of our experiments. Furthermore, since CBPD and CRITEO are quite imbalanced in different classes, we use the over-sampling method to balance the number of samples in each category. Then, we split the training and test dataset with a ratio of $7:3$.

To simulate the VFL scenario, we follow the method described in [18], [20], [21]. For image data, if two parties hold the same ratio of features, we split the features from the center to ensure each party has an equal number of pixel columns. The same was done for tabular data, but with a different number of attributes for each party based on the feature ratio. In the case of text data, the difference between parties was the length of sentences.

B. Models

In our evaluation, the bottom model for image processing is ResNet [48], for text processing is BERT [62], and for tabular data is MLP [63]. The ResNet and BERT are downloaded from PyTorch [2] and Hugging Face [3], respectively. The output dimension of the models is modified through a single linear layer, and their parameters are fine-tuned. The top model is a simple MLP with 1 hidden layer used to calculate the posteriors of the aggregated hash codes.

C. Hyperparameters

We set the training epochs for 30 times and an Adam [64] optimizer with a batch size of 256 for images and tabular data, and 8 for texts (due to the limitation of memory). The Adam optimizer has the learning rate of $1e-3$, the weight decay of $5e-4$, and a momentum by default in PyTorch's implementation. In addition, we shrink the learning rate by 10% every 10 epochs. We store the last epoch's model, which are then used to measure the main task's performance on the test set.

D. Environment

We implement the attacks in Python and conduct all experiments on a workstation equipped with AMD Ryzen 9 3950X and an NVIDIA GTX 3090 GPU card. We use PyTorch to implement the models used in the experiments, and pandas [4] and sklearn [5] for data pre-processing.

V. Evaluation

This section evaluates the proposed framework, HashVFL, on specific tasks and its defensive performance against data reconstruction attacks. Additionally, the extra benefit of HashVFL in detecting abnormal inputs and the need for combining differential privacy (DP) [65], [66] are discussed.

[1]https://www.kaggle.com
[2]https://pytorch.org
[3]https://huggingface.co
[4]https://pandas.pydata.org
[5]https://scikit-learn.org/stable

TABLE II
Accuracy (%) on test set of HashVFL on various datasets.

Dataset	MNIST	CIFAR10	FER	CBPD	CRITEO	IMDb
Without Defense	98.99	76.22	55.93	50.30	70.72	73.64
With Defense	97.75	70.83	51.11	69.34	72.94	69.72

A. Performance Evaluation

We evaluate the performance of the HashVFL model under strict conditions by setting the length of the hash code to just enough to cover the number of classes, as redundant bits may leak information inadvertently [67]. It is achieved by setting the length of the hash code to $\lceil \log_2 C \rceil$, where C is the number of classes and $\lceil \cdot \rceil$ is the ceiling function. For example, on CIFAR10, the length of the hash code is set to 4 bits (2^4) to cover 10 classes.

TABLE II summarizes the performance of HashVFL on different datasets. The results on MNIST, CIFAR10, and IMDb, show that HashVFL maintains the performance compared to results without defense, with the largest loss of accuracy on the test set being 5.39% on CIFAR10.

The results obtained on CBPD and CRITEO datasets demonstrate that HashVFL can even improve performance, as evidenced by the accuracy of 72.94% on CRITEO with defense compared to 70.72% without defense. We speculate that this improvement may be attributed to the nature of tabular data, where the sign of values can be more informative for classification compared to floating-point numbers [68]. Specifically, in our experimental design, we set the length of the embedding as 1 for binary classification tasks. This design choice makes the classification task more challenging since the server only receives two floating-point numbers as input. However, when applying the hash code, it effectively forces the bottom model to map the floating-point number to a binary class (e.g., +1/-1) first, and then enables the server to make the final judgment based on the submitted codes. We speculate that this operation transforms the problem into a voting scenario for the top model, making it easier to learn and contribute to the observed performance improvement.

In conclusion, although the length of the hash code is limited to a minimum value, HashVFL satisfies the requirement of maintaining main task performance and can be applied to various data types.

B. Defending against Reconstruction Attacks

In this section, we assess the defensive capability of HashVFL against data reconstruction attacks. Our analysis is based on the threat model outlined in Section II-D, which assumes that the adversary has complete knowledge of the victim's bottom model and the target sample's hash code. We examine HashVFL's protection of privacy from three perspectives:

- Recovery of the intermediate result from its hash code;
- Recovery of the target sample from its hash code;
- Revealing common features among a group of samples sharing the same code.

TABLE III
LABEL LEAKAGE MEASUREMENT WITH DIFFERENT ARCHITECTURES UNDER PLA.

Dataset	CBPD	CRITEO	MNIST	CIFAR10	FER	IMDb
Base	62.41±4.38	72.31±0.82	95.05±0.13	70.13±0.09	52.09±0.22	55.64±0.85
Ours	63.17±0.43	69.17±0.10	85.24±0.06	69.93±0.07	51.20±0.10	50.48±0.24

Reconstructing the intermediate result from its hash code is intractable. Previous works such as [27], [44], [46] imposed a constraint on the embeddings to be close to their binary codes using a loss term of Euclidean distance, in addition to hashing. This constraint ideally results in the intermediate results being equal to the hash codes. However, our proposed design, HashVFL, does not impose such a constraint. Consequently, it is not possible to reconstruct a sample's intermediate results using its hash codes. This fundamental difference sets HashVFL apart from previous works.

Reconstructing the target sample from its hash code is impossible. Since the intermediate results are not accessible, the semi-honest adversary can only obtain a sample's hash code. Then, we further conclude that the adversary cannot reconstruct a specific target sample's raw data, even with complete knowledge.

On the one hand, *Sign* function discards a significant amount of information. On the other hand, the strict limitation on the length of the hash code, preventing the assignment of a unique hash code to a target sample. These factors reinforce the privacy guarantee of HashVFL.

The same code shared by a group of samples may leak common features. A shorter hash code leads to more hash collisions. In our case, samples belonging to the same class are expected to have the same hash code. This raises the question of whether the adversary can uncover common features among samples of the same class.

1) Measurement of Label Leakage: It is evident that if our model achieves perfect classification performance, the hash code will also accurately represent the corresponding label. Therefore, there exists a correlation between performance and label leakage in VFL. In particular, in [21], Fu et al. highlighted that an adversary possessing a bottom model could deduce a sample's label based on the local embedding, with an accuracy that is proportional to the quality of the bottom model. Hence, we further investigate whether the combination of hashing can reduce the degree of label leakage.

Specifically, we compare the label inference accuracy between the original embeddings and the binarized embeddings on the selected datasets, following the methodology of the passive label inference attack (PLA) outlined in [21]. The default length of the hash code is set to 16. The summarized results are presented in TABLE III.

The results demonstrate that the inference performance of PLA decreases after the binarization process. This reduction in performance can be attributed to the loss of a significant amount of information during the binarization step. For example, on the MNIST dataset, the accuracy decreases from 95.05% to 85.24%, which is the highest reduction observed among all the datasets evaluated.

In conclusion, the introduction of the hashing mechanism not only effectively mitigates the data reconstruction attack but also helps alleviate the degree of label leakage to a certain extent.

2) Visual Results: In addition to considering label leakage, it is essential to examine what other common features the hash code may potentially reveal. To address this question, we assess the defense performance of HashVFL on the image datasets. It is because the reconstructed images provides an intuitive measure of the level of information leakage, specifically looking for any blurred contours in the reconstruction results.

Our approach is based on the method described in [26]. In this work, He et al. proposed a novel attack that could precisely recover images in a black-box scenario, achieving state-of-the-art results. Our study relaxes their assumption by allowing the adversary to have knowledge of the bottom model, making the attack stronger than in the black-box scenario. Specifically, we reconstruct the features according to the following formula:

$$\tilde{\mathbf{x}}^* = \arg\min_{\tilde{\mathbf{x}}} MSE(f_\theta(\tilde{\mathbf{x}}), \mathbf{o}_y) + \lambda TV(\tilde{\mathbf{x}}),$$

where $\tilde{\mathbf{x}}^*$ is the generated sample; $f_\theta(\cdot)$ denotes the bottom model; $MSE(\cdot)$ calculates two vectors' mean square error; $\mathbf{o}_y$ denotes the target class y's corresponding binary code; $TV(\cdot)$ denotes the Total Variation (TV) term [42]; λ is a coefficient to modify the weight of TV term.

The interpretation of this formula is simple. If the adversary is aware of the model and the output of the target sample, he/she can reconstruct the target sample by creating $\tilde{\mathbf{x}}$, whose output is similar to that of the target sample. Similar idea can be seen in [39], where Zhu et al. used the target sample's gradients instead of the output. In our case, since the hash code is the only information that the adversary can access, the formula is a reasonable one to choose.

Since the generation process may produce noise, the TV term is introduced to smooth the output. The TV term is calculated as follows:

$$TV(\mathbf{x}) = \Sigma_{i,j} \sqrt{\|\mathbf{x}_{i+1,j} - \mathbf{x}_{i,j}\|^2 + \|\mathbf{x}_{i,j+1} - \mathbf{x}_{i,j}\|^2},$$

where i and j denote the pixel indices.

We conducted 3,000 rounds of reconstruction for each class code and varied the hash code length from 4 bits to 16 bits to examine the influence of code length on information leakage. The reconstructed results for MNIST are displayed in Fig. 3. The results demonstrate that even with the strongest assumption, the adversary cannot recover meaningful information.

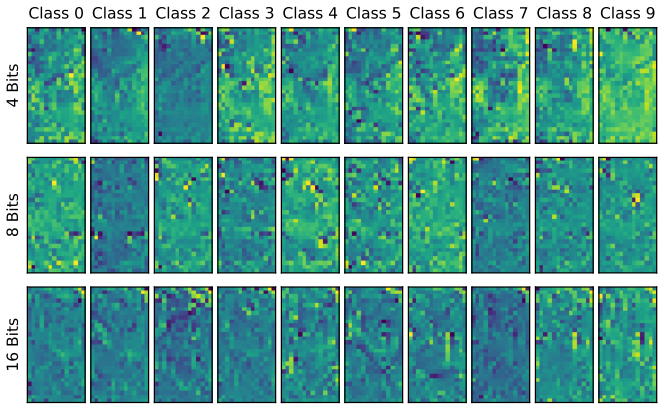

Fig. 3. Revealing common features on MNIST. The first row shows the reconstructed results with 4 bits hash code on different classes. The second and third rows show 8 and 16 bits, respectively.

TABLE IV
STATISTICAL ANALYSIS OF LEAKAGE WITH THREE DIFFERENT METRICS.

Dataset	Measure		
	KLD (≥ 0)	SSIM ($[0,1]$)	DCOR ($[0,1]$)
MNIST	11.5757±7.2420	0.0157±0.0137	0.8757±0.0487
CIFAR10	340.5571±270.2468	0.0620±0.0366	0.8540±0.0916
FER	813.7469±536.4684	0.0948±0.0511	0.9004±0.0693

Moreover, the length of the code does not have a significant impact on the reconstruction process.

3) Statistical Analysis: In order to conduct a comprehensive analysis of the information leakage resulting from the reconstruction attack on the hash code, we employ three additional metrics: Kullback–Leibler divergence (KLD) [69], distance correlation (DCOR) [70], and structural similarity index measure (SSIM) [25], as suggested by Pham et al. [71]. These metrics enable us to provide a statistical evaluation of the leakage. The results are summarized in TABLE IV.

The results indicate that the KLD values are considerably large across all three datasets. This means significant differences in the distributions of pixel values between the raw data and the reconstructed data. Moreover, the SSIM scores for all three datasets are notably small, indicating a reduced semantic similarity between the reconstructed data and the raw data. A similar trend is observed in the DCOR scores.

Based on these findings, we can conclude that the hashing operation in HashVFL effectively defends against the reconstruction attack and minimizes the leakage of common features.

C. Defending against Adversarial Attack

The purpose of this section is to investigate whether the employed hashing mechanism results in a loss of robustness in VFL models against adversarial attacks. To thoroughly assess the robustness, we enhance the capabilities of the adversary based on the previous threat model, enabling them to fully execute adversarial attacks. For example, we provide the adversary with complete knowledge of both the bottom models and the top model, allowing them to successfully carry out the projected gradient descent (PGD) attack [72].

TABLE V
DEFENSE PERFORMANCE OF DIFFERENT FRAMEWORK WITH DIFFERENT HYPERPARAMETERS ON MNIST AND CIFAR10.

Dataset	Threshold	Step Size	Attack Success Rate	
			Base	Hashing
MNIST	$\omega = 1$	$\eta = 0.1$	85	0
		$\eta = 1$	74	64
	$\omega = 2$	$\eta = 0.1$	1	0
		$\eta = 1$	1	1
CIFAR10	$\omega = 1$	$\eta = 0.1$	76	0
		$\eta = 1$	66	85
	$\omega = 2$	$\eta = 0.1$	98	0
		$\eta = 1$	95	64

PGD attack is one standard white-box adversarial attack widely used in the field. Formally, it can be described as follows:

$$\begin{cases} \mathbf{x}_m = \mathbf{x}_{m-1} - \eta * sign(\nabla_{\mathbf{x}_{m-1}} \mathcal{L}(\mathbf{x}_{m-1}, y_t; \theta)) \\ \boldsymbol{\phi}_m = clip(\mathbf{x}_m - \mathbf{x}_0, \omega) \\ \mathbf{x}_m = \mathbf{x}_0 + \boldsymbol{\phi}_m, \end{cases} \tag{1}$$

where $\mathbf{x}_0$ denotes the original intermediate results, $\mathbf{x}_m$ denotes the perturbed adversarial results at m-th optimization, η is the step size, y_t denotes the target class, and $clip(\mathbf{x}_m - \mathbf{x}_0, \omega)$ denotes the restriction that clips the perturbation $\boldsymbol{\phi}_m$ to a given threshold, which is $(-\omega, \omega)$.

To evaluate the performance of the proposed method, we select MNIST and CIFAR10 datasets for experimentation. These two datasets are suitable for evaluating robustness against adversarial attacks as they both consist of 10 classes and are widely used benchmarks. We conduct evaluations using different combinations of the threshold ω and step size η, while keeping the hash code fixed at 16 bits. For the experiments, we randomly select 100 samples from all classes except class '0' and calculate the success rate of the attack when these samples are misclassified as class '0'. The results are summarized in TABLE V.

The results reveal that the hashing mechanism employed can actually enhance the robustness of VFL models. This improvement can be attributed to the binarization operation, which requires that the values of the submitted codes from each party be limited to $\{-1, 1\}$. This limitation, in turn, restricts the performance of the PGD attack when its step size and threshold are smaller than 1. In other words, our hashing-based framework can expand the robust radius of each sample by at least 1 compared to the base VFL model without any defense mechanism. Unfortunately, when the threshold and

step size exceed 1, both frameworks fail to defend against the adversarial attack.

In conclusion, our HashVFL approach enhances the robustness of VFL models by forcing the embeddings to map to a fixed set of $\{-1, 1\}$, thereby expanding the robust radius of the samples.

D. Detecting Abnormal Inputs

Adversarial attacks essentially exploit abnormal inputs to alter outcomes. Therefore, when we return to the multi-party computing framework of VFL, it means that the adversary's embedding should be inconsistent with other normal parties' embeddings. From this intuition, we explore an additional advantage of HashVFL: its ability to efficiently detect abnormal inputs, as consistency requires identical code from each party. If the hash codes of one sample from two parties differ significantly, it may indicate abnormal inputs. For instance, if the hamming distance between the two codes is larger than half the length of the code, it may indicate cheating.

The detection capability of HashVFL is evaluated in a two-party scenario, where each party holds half the features. To verify our speculation, all combinations of different hash codes are detected. For each class, the code from one party, $P_{initiator}$, is set as the corresponding pre-defined binary code, and the other party's code, $P_{participant}$, is varied to observe differences between correct and incorrect predictions. Ideally, the hamming distance between the codes of correct predictions should be less than half the length, while it should be greater for incorrect predictions. This means that a malicious party must change at least half the bits to alter the prediction. The length of the hash code is 4 bits.

TABLE VI summarizes the results. In TABLE VI, the '-' symbol denotes the absence of a wrong prediction, and '/' denotes the absence of the case in the dataset. The results in the 'Average' column confirm our speculation that if the hamming distance between two parties' codes is greater than half the length (2 in this case), the prediction is probably incorrect. This conclusion is also supported by most of the detailed results for each class. However, on CIFAR10, there are exceptions in the results for class '1' and class '2'. These exceptions are due to a concentration of wrong predictions on specific classes and a relatively small hamming distance between the pre-defined codes of the classes. For example, many wrong predictions on class '1' give class '5', and on class '2' give class '8'. The pre-defined codes of class '1' and class '5' are $[-1, 1, 1, 1]$ and $[1, 1, 1, 1]$, whose hamming distance is relatively small. Therefore, the average hamming distance of wrong predictions is smaller in such two cases. The results on CBPD, CRITEO, and IMDb are consistent with the above conclusion.

In summary, the HashVFL method efficiently detects abnormal inputs by computing the hamming distance between hash codes submitted by different parties. If the hamming distance is greater than half the length of the code, it suggests the possibility of cheating during inference.

TABLE VI
ANALYSIS OF 4-BIT HASH CODE DETECTION PERFORMANCE. THE 'CLASS' COLUMN SHOWS THE AVERAGE HAMMING DISTANCE BETWEEN THE TWO HASH CODES. THE 'AVERAGE' COLUMN REPRESENTS THE AVERAGE RESULTS FOR EACH CLASS.

Dataset		Class										Average
		0	1	2	3	4	5	6	7	8	9	
MNIST	correct	0.67	1.57	1.50	1.20	1.00	1.20	1.17	1.75	1.33	1.70	1.31
	error	2.31	2.33	2.50	2.36	2.23	2.36	2.50	2.25	2.40	2.50	2.38
CIFAR10	correct	1.00	**2.10**	**3.00**	1.43	1.00	1.67	1.43	1.50	1.20	1.62	1.59
	error	2.33	**1.83**	**1.67**	2.44	2.33	2.43	2.44	2.50	2.36	2.38	2.27
FER	correct	**2.00**	**2.00**	1.90	0.67	1.29	1.93	1.00	/	/	/	1.35
	error	**2.00**	**2.00**	2.17	2.31	2.56	2.50	2.33	/	/	/	2.27
CBPD	correct	1.87	**2.00**	/	/	/	/	/	/	/	/	1.94
	error	**4.00**	-	/	/	/	/	/	/	/	/	4.00
CRITEO	correct	1.87	**2.00**	/	/	/	/	/	/	/	/	1.94
	error	**4.00**	-	/	/	/	/	/	/	/	/	4.00
IMDb	correct	1.87	**2.00**	/	/	/	/	/	/	/	/	1.94
	error	**4.00**	-	/	/	/	/	/	/	/	/	4.00

E. Analysis of Combining Differential Privacy

This section explores the need for further incorporating differential privacy (DP) [65], [66] into our existing scheme. DP is a widely used privacy-enhancing technique in DNNs and has been integrated into frameworks such as FATE and TF Encrypted for data protection.

a) Theoretical Analysis: In [71], Pham et al. proposed a method to integrate DP with binary code:

$$\mathbf{h} = Sign(Sign(f(\mathbf{x})) + Lap(\frac{s}{\epsilon})),$$

where $\mathbf{h}$ is the hash code; $\mathbf{x}$ is the feature vector; f is a DNN model; s is the sensitivity of $Sign(\cdot)$, actually 2 for binary codes; ϵ is the privacy budget [65]; and $Lap(\cdot)$ is the Laplace distribution sampling function.

From this design, if $|Lap(\frac{2}{\epsilon})| < 1$, then $\mathbf{h} = Sign(f(\mathbf{x}))$. The probability of $|Lap(\frac{2}{\epsilon})| < 1$ can be calculated as:

$$Pr[|Lap(\frac{2}{\epsilon})| < 1] = 1 - [cdf(1) - cdf(-1)] = 1 - e^{-\frac{\epsilon}{2}},$$

where $cdf(\cdot)$ denotes the cumulative distribution function of Laplace. According to the definition of Approximate DP [73], the analysis also indicates that the hashing operation satisfies (ϵ, δ)-DP, where $\delta = 1 - e^{-\frac{\epsilon}{2}}$.

However, the derivation also reveals a problem that the added noise cannot change the value of one bit when the privacy budget ϵ is large (indicating a weak privacy protection level). Specifically, following the above calculation, we have $Pr[|Lap(\frac{2}{\epsilon})| \geq 1] = e^{-\frac{\epsilon}{2}}$. Consider that there is half chance that the sign of the noise is the same as the bit, the probability of flipping one bit's sign is $\frac{1}{2}e^{-\frac{\epsilon}{2}}$. When we set $\epsilon = 10$, the probability decreases rapidly to 0.33%, which almost does not provide any privacy protection. In such a situation, the adversary can use the received hash code as the real value, and there is almost no error.

The analysis suggests that integrating DP in HashVFL is not necessary, if there is a large privacy budget. In addition, the added noise may cause the performance of the main task to degrade. Therefore, we do not recommend integrating DP in HashVFL as it has a limited defensive effect.

b) Experimental Demonstration: To better understand the necessity of combining DP in HashVFL, we conduct experiments with different privacy budgets to evaluate its impact on main task's performance. We still evaluated the two-party scenario, where each party holds half of the features. The results are summarized in TABLE VII.

The results show that the impact of noise on accuracy is significant when $\epsilon = 1$ or $\epsilon = 2$. However, when $\epsilon \geq 10$, the loss of accuracy is minimal. This is because the probability of flipping one bit's sign is 0.33% when $\epsilon = 10$, meaning that accurate information can be maintained, leading to better performance.

Additionally, the loss of accuracy decreases with the increase in hash code length. This is expected as adding noise to each bit can be regarded as a Bernoulli trial with a probability of $\frac{1}{2}e^{-\frac{\epsilon}{2}}$. Given an expected number k of flipped bits and the length n, the probability can be calculated using the formula

$$Pr[H(\mathbf{h}', \mathbf{h}) = k] = \binom{n}{k}(\frac{1}{2}e^{-\frac{\epsilon}{2}})^k(1 - \frac{1}{2}e^{-\frac{\epsilon}{2}})^{(n-k)},$$

where $H(\cdot)$ calculates the Hamming distance and $\mathbf{h}$ is the perturbed code of $\mathbf{h}$. For example, when $\epsilon = 1$, and taking the case of 16 bits and $k = 4$, the corresponding probability is approximately 20%. This means that a longer code can retain most of the valid bits, maintaining performance.

In conclusion, we believe that incorporating DP in HashVFL is unnecessary as it would significantly reduce model performance with a small privacy budget and offer limited privacy protection with a large privacy budget. If DP is deemed necessary, the length of the hash code should be increased accordingly to reduce performance loss.

TABLE VII
PERFORMANCE COMPARISON WITH DIFFERENT PRIVACY BUDGETS. THE CELLS REPORT THE ACCURACY ON THE TEST SET.

Dataset	Code Length	$\epsilon = 1$	$\epsilon = 2$	$\epsilon = 10$	$\epsilon = \infty$
MNIST	4 Bits	34.28	61.73	97.26	**97.75**
	8 Bits	45.65	80.34	97.95	**98.34**
	16 Bits	65.62	93.35	98.21	**98.42**
CIFAR10	4 Bits	26.89	44.76	69.11	**70.83**
	8 Bits	33.54	57.40	72.99	**73.65**
	16 Bits	47.08	68.73	74.23	**74.96**
FER	4 Bits	27.57	39.08	50.57	**52.73**
	8 Bits	33.81	56.38	53.15	**54.50**
	16 Bits	40.32	51.56	**55.06**	55.00
CBPD	4 Bits	58.56	66.06	70.38	**70.98**
	8 Bits	61.99	67.85	75.68	**76.62**
	16 Bits	63.96	69.44	76.64	**77.35**
CRITEO	4 Bits	64.04	70.18	**73.82**	73.70
	8 Bits	64.55	71.03	72.74	**73.13**
	16 Bits	66.81	72.31	**74.03**	73.32
IMDb	4 Bits	64.03	68.51	**69.35**	68.59
	8 Bits	66.61	69.66	71.44	**72.10**
	16 Bits	68.26	70.21	72.42	**72.66**

VI. SENSITIVITY ANALYSIS

This section delves deeper into the effects of the default setting in previous experiments. It addresses the following research questions:

- **Q1:** How does the *number of parties* affect HashVFL?
- **Q2:** How does the *length of hash codes* affect HashVFL?
- **Q3:** How does different *feature ratios* affect HashVFL?
- **Q4:** How does the *number of classes* affect HashVFL?

A. Number of Parties (Q1)

This section explores the impact of the number of parties on performance. We split the features of samples in a dataset to simulate multi-party scenarios, resulting in each party having fewer features as the number of parties increases. Considering that many columns in images of MNIST are black, we decide to exclude it from the datasets used in the experiment. We conduct experiments on CIFAR10, FER, CBPD, and CRITEO, but not on IMDb as it requires too much memory to run BERTs simultaneously.

Each party holds the same feature ratio, rounded to the nearest whole number. For example, with 3 parties, the ratios are 30%, 30%, and 40%. To mitigate the loss in accuracy, we set the length of hash codes to 16 bits. The results are shown in Fig. 4.

On CIFAR10 and FER, performance decreases with the increasing number of parties, as splitting useful features can destroy their integrity. In contrast, on CBPD and CRITEO, performance remains stable as tabular data features are more independent.

Despite the number of parties affecting main task performance, our proposed HashVFL framework maintains performance close to without defense, and even improves performance for tabular datasets as seen in Section V-A.

In conclusion, while the number of parties impacts performance, HashVFL can maintain close to the performance without defense.

B. Length of Hash Codes (Q2)

In this section, we assess the impact of varying the length of hash codes on performance. By doubling the length of hash codes from 4 bits to 128 bits, we observe improved performance initially, which then converges. TABLE VIII summarizes the results. We speculate that the improvement in the previous stage is because the increased bits can compensate for the information loss caused by hashing, and when the information that the model can extract is saturated, more bits can only cause redundancy.

It is recommended to determine the appropriate hash code length according to the required security level, where longer hash codes result in improved performance and shorter hash codes offer stricter data protection.

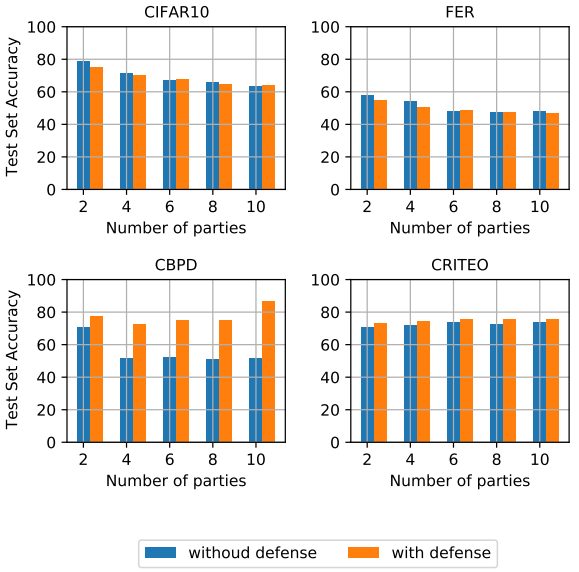

Fig. 4. Impact of number of parties on accuracy. X-axis is the number of parties involved, while the Y-axis is the accuracy achieved on the test set.

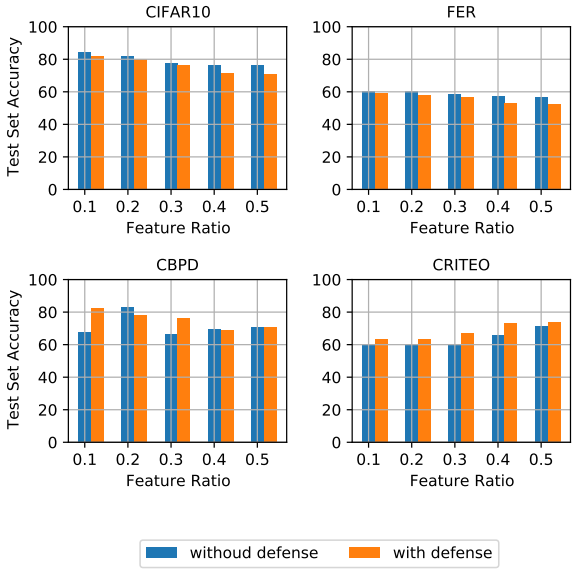

Fig. 5. Impact of feature ratio on accuracy. X-axis is the ratio of features held by one party, while the Y-axis is the accuracy achieved on the test set.

TABLE VIII
PERFORMANCE COMPARISON WITH DIFFERENT LENGTHS. THE CELL REPORTS THE ACCURACY ON THE TEST SET.

Dataset	4 Bits	8 Bits	16 Bits	32 Bits	64 Bits	128 Bits
MNIST	97.75	98.34	98.42	98.59	98.42	98.57
CIFAR10	70.83	73.65	74.96	76.14	75.34	75.03
FER	52.73	54.50	55.00	55.66	55.34	55.61
CBPD	70.98	76.62	77.35	77.95	81.64	85.00
CRITEO	73.70	73.13	73.32	74.08	74.48	74.67
IMDb	68.59	72.10	72.66	73.10	73.46	73.59

C. Feature Ratio (Q3)

This section evaluates the effect of the feature ratio, which is defined as the proportion of features owned by a single party in the whole set of features. Experiments were conducted in a two-party scenario, where one party's feature ratio varied from 10% to 50%. The symmetric scenario of varying feature ratio from 60% to 90% was omitted as it was expected to be the same.

The results, shown in Fig. 5, indicated that performance decreases with increasing feature ratio on CIFAR10 and FER. This is because as the feature ratio increases, the other party's image completeness decreases, and important features become concentrated in the middle region, making inference difficult. However, on CBPD and CRITEO, where features are more independent, the effect of the feature ratio was less pronounced with some fluctuations.

The results showed that HashVFL kept the accuracy loss within an acceptable range on CIFAR10 and CRITEO and even improved performance on CBPD and CRITEO, as analyzed in Section V-A. In conclusion, complete and valid features are crucial for reducing accuracy loss in HashVFL.

TABLE IX
PERFORMANCE COMPARISON WITH DIFFERENT NUMBERS OF CLASSES. WE PLOT THE RESULTS OF CIFAR10 WITH DEFENSE FOR REFERENCE.

Dataset		8 Bits	16 Bits	32 Bits	64 Bits	128 Bits
CIFAR100	without defense	33.99	34.09	34.75	33.39	34.41
	with defense	24.23	29.51	33.31	33.77	31.96
CIFAR10	with defense	73.65	74.96	76.14	75.34	75.03

D. Number of Classes (Q4)

The results of our experiments on CIFAR10 and CIFAR100, which share the same data but have different number of classes, showed that the number of classes does make the task more challenging. As seen in TABLE IX, the highest accuracy achieved by our models on the CIFAR100 test set was 33.77% when the length was 64 bits with defense, while the accuracy on CIFAR10 was 75.34% with the same length.

When the length was short, the accuracy loss was higher on CIFAR100 compared to CIFAR10. For instance, when the length was 8 bits, the accuracy loss on CIFAR100 was 10%, while it was 6% on CIFAR10 (as shown in TABLE II). However, when the length was increased to 32 or 64 bits, the loss of accuracy was less than 1%.

In conclusion, a larger number of classes increases the complexity of the task and causes a significant accuracy loss when the length of the hash codes is short. But, when there are enough bits, our method, HashVFL, can maintain acceptable performance.

VII. ABLATION STUDY

Three challenges were introduced in Section I. While previous works like [27], [44], [46], [71] have addressed the challenge of learnability in deep hashing using different approaches, we find that the combination of the $Sign$ function and the STE is the most efficient and completely irreversible

during training. Other functions like $tanh$ and $sigmoid$ preserve the values during training, leaving the risk of information leakage.

However, while this technique has been proven effective in retrieval systems [27] and split learning [71], there are two additional challenges, namely bit balance and consistency, that need to be addressed in the context of VFL. In this section, our primary objective is to answer two important questions: the necessity of the incorporated modules in addressing the challenges we encountered, and the effectiveness of these modules compared to Greedy Hash [27] and B-SL [71]. We summarize these questions as follows:

- **AS1:** What is the role of *bit balance* in HashVFL?
- **AS2:** What benefit does *consistency* bring to HashVFL?

A. Bit Balance (AS1)

This section provides experimental evidence for the importance of the BN layer in HashVFL to address the challenge of bit balance.

Baseline: We compare our HashVFL approach with Greedy Hash. Greedy Hash also uses the $Sign$ function and STE for gradient estimation, but its design focuses on improving retrieval performance and reducing the impact of hash collisions. Consequently, it does not consider the need to maximize the leverage of each bit, as its hash code's length can exceed 128 bits. Additionally, Greedy Hash introduces a penalty term based on the Euclidean distance between the embeddings and their corresponding binary codes.

Experimental Setup: By default, we conduct experiments in a two-party scenario, where each party holds half of the features. We vary the length of the hash code from 4 bits to 16 bits for comparison.

The results, summarized in TABLE X, demonstrate that our method with a BN layer outperforms Greedy Hash on all datasets, while performing similarly without it. On CBPD, CRITEO, and IMDb datasets, our method without a BN layer exhibits a significant loss of performance, but adding a BN layer mitigates this issue.

We speculate that the BN layer reduces the impact of large rotations caused by the cosine similarity loss during optimization. For instance, in a binary classification task with a 4-bit hash code length, there are $2^4 \cdot 2 = 32$ rotation angles involved in the optimization process. Compared to the changes in gradients resulting from the classification loss, the rotation caused by the similarity loss may be too large, causing the hash code of a sample to flip at every optimization step. Consequently, it becomes difficult for the top model to learn a stable function for accurate prediction. However, the addition of a BN layer can alleviate this issue by reducing the impact of rotations. The BN layer evenly divides the distribution of each bit for every batch, allowing for a larger range for each bit to vary without flipping its sign.

In conclusion, simply applying the approach used in Greedy Hash cannot address the performance degradation in VFL when using limited-length hash codes. The BN layer is essential in HashVFL as it evenly divides the distribution of each bit, thereby maximizing the leverage of each bit.

TABLE X
ABLATION STUDY OF BATCH NORMALIZATION'S IMPACT. 'GREEDY HASH' DENOTES THE BASELINE FOR REFERENCE. 'OURS' REFERS TO OUR DESIGN. THE CELL REPORTS THE ACCURACY ON THE TEST SET.

Dataset	Method		4 Bits	8 Bits	16 Bits
MNIST	Greedy Hash		96.24	97.17	97.82
	Ours	without BN	96.75	97.24	97.02
		with BN	97.75	98.34	98.42
CIFAR10	Greedy Hash		55.53	63.57	61.52
	Ours	without BN	60.44	63.24	60.26
		with BN	70.83	73.65	74.96
FER	Greedy Hash		40.42	42.24	44.65
	Ours	without BN	37.95	45.74	48.68
		with BN	52.73	54.50	55.00
CBPD	Greedy Hash		61.57	63.13	62.42
	Ours	without BN	48.74	49.89	48.56
		with BN	70.98	76.62	77.35
CRITEO	Greedy Hash		63.21	66.94	68.63
	Ours	without BN	49.76	49.93	49.94
		with BN	73.70	73.13	73.32
IMDb	Greedy Hash		70.71	70.63	70.55
	Ours	without BN	50.59	50.34	50.26
		with BN	68.59	72.10	72.66

B. Consistency (AS2)

When comparing VFL to split learning, it is important to consider the computational costs associated with an increasing number of parties. In Section I, we introduced the challenge of consistency in VFL and proposed the use of predefined target binary codes to reduce the complexity of calculating distances between parties from $O(N^2)$ to $O(N)$. This approach saves computational resources and accelerates training.

In addition to the computational benefits, we are also interested in exploring the additional advantages of consistency in HashVFL. We speculate that consistency can simplify the task by reducing the number of combinations of multi-party hash codes.

Baseline: To establish a baseline comparison, we choose an extension of B-SL as our baseline, which does not utilize predefined binary codes. Additionally, we introduce an Euclidean distance penalty in B-SL to ensure model convergence without consistency guarantees.

Experimental Setup: We conduct the experiments in a two-party scenario, with each party holding half of the features. Furthermore, we still vary the length of the hash code from 4 bits to 16 bits for comparison.

The results, as depicted in Fig. 6, demonstrate that the curves with consistency requirements outperformed those without on all datasets, with a gap of nearly 15% on CRITEO and around 1% on other datasets, except for CBPD and IMDb with 4-bit codes. Moreover, the inclusion of the cosine similarity loss also accelerated training, resulting in a faster improvement in accuracy during the initial 10 epochs. However, after 10 epochs, the accuracy on the training set continued to increase while the accuracy on the test set decreased, potentially indicating overfitting. By the 30-th epoch, the

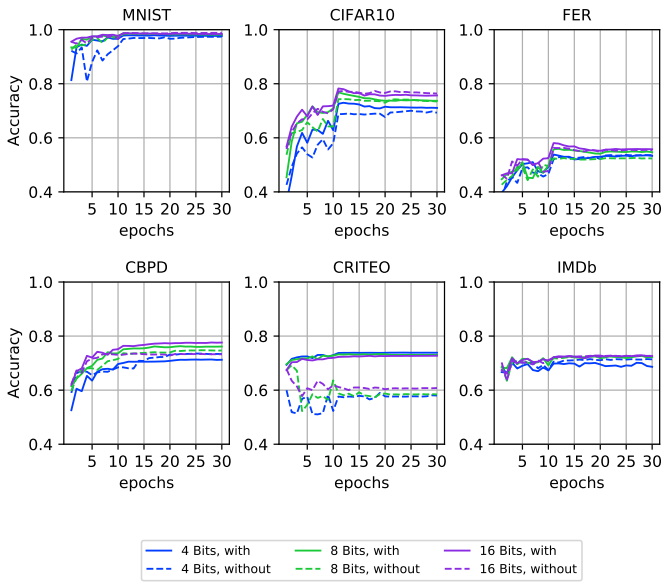

Fig. 6. Comparison of performance with and without addressing 'consistency'. The X-axis represents the training epoch, and the Y-axis shows the accuracy. The legends 'with' and 'without' indicate the presence or absence of cosine similarity.

accuracy on the test set decreased for training with cosine similarity loss, while it improved for training without it.

In conclusion, addressing consistency in HashVFL not only provides additional benefits such as detecting abnormal inputs and reducing computational costs, but also improves and accelerates training. These findings underscore the significance of consistency in HashVFL.

VIII. Related Work

In this section, we supplement the related work and compare the work we refer to with our method.

A. Learning to Hash

Nearest neighbor search is a problem that seeks to find the samples in a database with the smallest distance to the query. Hashing is a commonly used solution due to its computational and storage efficiency. With the advancement of deep neural networks (DNNs), deep hashing has emerged as a more effective solution than traditional methods. Deep supervised hashing [27], [32], [33], [44]–[46], [74]–[76] is a subfield of deep hashing, where the goal is to solve binary optimization and address vanishing gradients in DNNs.

Different approaches have been proposed in the literature for deep supervised hashing. A line of work focuses on binarizing the activations in DNNs. For example, Cao et al. [46] uses a combination of $Tanh$ and $Sigmoid$ functions, while Li et al. [75] designed a penalty function to generate binary features. In [27], Su et al. proposed the $Sign$ function to directly binarize the features.

Another line of work focuses on a different approach to learning hashing. For example, Fan et al. [32] used a random assignment scheme to generate target vectors with maximal inter-class distance. Then, they optimized the distance between the embeddings and the vectors. Yuan et al. [77], however, used the Hadamard matrix as target centers. Hoe et al. [33] integrated category information into one loss by revealing the connection between cosine similarity and Hamming distance.

B. Attacks in Vertical Federated Learning

Recently, several studies have explored the security of VFL. These studies mainly focus on two aspects: data privacy and security.

Regarding privacy, Luo et al. [18] proposed a DNN-based method for reconstructing data in VFL. Weng et al. [19] also studied the privacy risks of VFL using machine learning methods such as logistic regression and XGBoost. Qiu et al. [20] investigated the privacy risks of graph data in VFL, and Fu et al. [21] looked into the leakage of labels.

For security, Liu et al. [78] found that the party that owns the label can easily carry out a backdoor attack. They also explored the possibility of a backdoor attack when the adversary has no access to labels and found that replacing gradients can be effective.

C. Defenses in Vertical Federated Learning

VFL is a relatively new field and there have been limited studies on defenses against attacks in VFL. Two lines of research have been proposed to address different types of attacks.

In [24] and [25], Sun et al. and Vepakomma et al. proposed schemes to reduce data reconstruction attacks by incorporating the correlation distance between extracted embeddings and raw inputs into the penalty function. Sun et al. proposed a method to defend against label inference attacks by integrating DP into the forward process in [79]. Defenses against label inference attacks were also discussed in [21] by using gradient compression [80]. Pham et al. proposed a defense against feature reconstruction attacks by integrating binary neural networks (BNNs) [28] into the first few layers in [71].

In [81], Liu et al. used feature reconstruction to defend against backdoor attacks, which applied an attack for good.

D. Remark

Our proposed HashVFL aims to defend against feature reconstruction attacks through the use of hashing. Unlike prior defenses such as [24], [25], hashing can eliminate the connection between the binarized embedding and the input even when the adversary has complete knowledge of the model and the hash code.

Comparing to the defense proposed by Pham et al. [71], who only binarizes the first few layers of the feature maps, HashVFL is capable of handling more complex scenarios, such as different types of data, and can easily be integrated into existing frameworks.

Research in learning to hash has provided valuable insights for integrating hashing into our design. The GreedyHash method proposed by Su et al. [27] offers scalability suitable for VFL model. However, it does not fully address the challenges of balancing bits and consistency.

Methods such as [46], [75] are effective in learning to hash, but they still entail the risk of leakage ($Tanh$ and $Sigmoid$ leave the risk of reversibility). Hence, we do not adopt these methods in HashVFL's design.

IX. Discussion

A. Adaptive Attack

Our evaluation demonstrates that HashVFL effectively mitigates the privacy issues stemming from reconstruction attacks. However, as identified in the analysis of label leakage, it is possible to infer certain attributes with a specific classifier. Considering the remarkable performance of generative models like ChatGPT [82] and Stable Diffusion models [83], we speculate that a feasible adaptive attack to bypass HashVFL could follow the following steps:

1) Extracting relevant attributes through PLA using a set of classifiers.
2) Utilizing the extracted attributes to construct a prompt.
3) Employing a suitable generative model to reconstruct the target sample using the earlier prompt.

The intuition behind this adaptive attack is to maximize the extraction of common features of a particular class of samples that are retained in HashVFL.

In the future, it is crucial to further decouple data utilization and data distribution based on these findings. This can help strengthen the defense mechanism against adaptive attacks and ensure that data privacy is upheld effectively.

B. Bias Between Parties

Section VI-C reveals that when one party possesses a majority of features, the performance improves compared to that both parties hold equal portions. Our analysis attributes this improvement to feature completeness in the former scenario, enabling the top model to effectively use the information. However, this also raises a concern about the party with more features having greater influence on VFL's final predictions. The use of HashVFL may exacerbate this bias by discarding much of the information for all parties.

The presence of the bias in the dominant party raises the concern of malicious manipulation of the final prediction during inference in HashVFL. Addressing this bias and promoting equal feature importance in the prediction is a key challenge for its practical implementation.

Our HashVFL design has been proven effective in detecting malicious code through consistency checks. However, how to mitigate such bias in training remains an open area for future research with promising potential.

X. Conclusion

This work introduces HashVFL, a new VFL framework that leverages hashing to defend against data reconstruction attacks. As far as we know, this is the first VFL framework that incorporates hashing. We address three challenges in integrating hashing into VFL and provide effective solutions. Our evaluation results show that HashVFL retains the performance of the main task while effectively protecting against data reconstruction attacks. Additionally, we show experimentally that HashVFL can reduce the degree of label leakage, mitigate the adversarial attack, and detect abnormal inputs. We anticipate that this work will spark further investigations into the practical applications of HashVFL.

Acknowledgments

This work was partly supported by the National Key Research and Development Program of China under No. 2022YFB3102100 and NSFC under No. 62102360.

References

[1] L. Breiman, "Random forest," *Machine Learning*, vol. 45, no. 1, pp. 5–32, 2001.
[2] C.-C. Chang and C.-J. Lin, "LIBSVM," *ACM Transactions on Intelligent Systems and Technology*, vol. 2, no. 3, pp. 1–27, apr 2011.
[3] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, may 2015.
[4] A. J. Dautel, W. K. Härdle, S. Lessmann, and H.-V. Seow, "Forex exchange rate forecasting using deep recurrent neural networks," *Digital Finance*, vol. 2, no. 1-2, pp. 69–96, mar 2020.
[5] M. Dixon, D. Klabjan, and J. H. Bang, "Classification-based financial markets prediction using deep neural networks," *Algorithmic Finance*, vol. 6, no. 3-4, pp. 67–77, dec 2017.
[6] J. Jumper, R. Evans, A. Pritzel, T. Green, M. Figurnov, O. Ronneberger, K. Tunyasuvunakool, R. Bates, A. Žídek, A. Potapenko, A. Bridgland, C. Meyer, S. A. A. Kohl, A. J. Ballard, A. Cowie, B. Romera-Paredes, S. Nikolov, R. Jain, J. Adler, T. Back, S. Petersen, D. Reiman, E. Clancy, M. Zielinski, M. Steinegger, M. Pacholska, T. Berghammer, S. Bodenstein, D. Silver, O. Vinyals, A. W. Senior, K. Kavukcuoglu, P. Kohli, and D. Hassabis, "Highly accurate protein structure prediction with AlphaFold," *Nature*, vol. 596, no. 7873, pp. 583–589, jul 2021.
[7] Q. Wang, Y. Zhou, T. Ruan, D. Gao, Y. Xia, and P. He, "Incorporating dictionaries into deep neural networks for the chinese clinical named entity recognition," *Journal of Biomedical Informatics*, vol. 92, p. 103133, apr 2019.
[8] S. Krebs, B. Duraisamy, and F. Flohr, "A survey on leveraging deep neural networks for object tracking," in *2017 IEEE 20th International Conference on Intelligent Transportation Systems (ITSC)*. IEEE, oct 2017.
[9] I. F. Kupryashkin, "Impact of the radar image resolution of military objects on the accuracy of their classification by a deep convolutional neural network," *Journal of the Russian Universities. Radioelectronics*, vol. 25, no. 1, pp. 36–46, feb 2022.
[10] P. Voigt and A. Von dem Bussche, "The eu general data protection regulation (gdpr)," *A Practical Guide, 1st Ed., Cham: Springer International Publishing*, 2017.
[11] D. Kiselbach and C. E. Joern, "New consumer product safety laws in canada and the united states: Business on the border," *Global Trade and Customs Journal*, vol. 7, no. 1, 2012.
[12] J. Xu, B. S. Glicksberg, C. Su, P. Walker, J. Bian, and F. Wang, "Federated learning for healthcare informatics," 2020.
[13] A. Hard, K. Rao, R. Mathews, S. Ramaswamy, F. Beaufays, S. Augenstein, H. Eichner, C. Kiddon, and D. Ramage, "Federated learning for mobile keyboard prediction," 2018.
[14] 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, 2019.
[15] T. S. Brisimi, R. Chen, T. Mela, A. Olshevsky, I. C. Paschalidis, and W. Shi, "Federated learning of predictive models from federated electronic health records," *International journal of medical informatics*, vol. 112, pp. 59–67, 2018.
[16] 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.
[17] C. Zhang, S. Li, J. Xia, W. Wang, F. Yan, and Y. Liu, "Batchcrypt: Efficient homomorphic encryption for cross-silo federated learning," in *2020 USENIX Annual Technical Conference*, 2020, pp. 493–506.
[18] X. Luo, Y. Wu, X. Xiao, and B. C. Ooi, "Feature inference attack on model predictions in vertical federated learning," *CoRR*, vol. abs/2010.10152, 2020.

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

[20] P. Qiu, X. Zhang, S. Ji, T. Du, Y. Pu, J. Zhou, and T. Wang, "Your labels are selling you out: Relation leaks in vertical federated learning," *IEEE Transactions on Dependable and Secure Computing*, pp. 1–16, 2022.

[21] 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, pp. 1397–1414. [Online]. Available: https://www.usenix.org/conference/usenixsecurity22/presentation/fu-chong

[22] A. Creswell, T. White, V. Dumoulin, K. Arulkumaran, B. Sengupta, and A. A. Bharath, "Generative adversarial networks: An overview," *IEEE Signal Processing Magazine*, vol. 35, no. 1, pp. 53–65, jan 2018.

[23] T. Karras, S. Laine, M. Aittala, J. Hellsten, J. Lehtinen, and T. Aila, "Analyzing and improving the image quality of StyleGAN," in *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, jun 2020.

[24] J. Sun, Y. Yao, W. Gao, J. Xie, and C. Wang, "Defending against reconstruction attack in vertical federated learning," *CoRR*, vol. abs/2107.09898, 2021.

[25] P. Vepakomma, A. Singh, O. Gupta, and R. Raskar, "Nopeek: Information leakage reduction to share activations in distributed deep learning," *2020 International Conference on Data Mining Workshops (ICDMW)*, pp. 933–942, 2020.

[26] Z. He, T. Zhang, and R. B. Lee, "Model inversion attacks against collaborative inference," *Proceedings of the 35th Annual Computer Security Applications Conference*, 2019.

[27] S. Su, C. Zhang, K. Han, and Y. Tian, "Greedy hash: Towards fast optimization for accurate hash coding in cnn," in *Advances in Neural Information Processing Systems*, S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, Eds., vol. 31. Curran Associates, Inc., 2018.

[28] H. Qin, R. Gong, X. Liu, X. Bai, J. Song, and N. Sebe, "Binary neural networks: A survey," *ArXiv*, vol. abs/2004.03333, 2020.

[29] Y. Bengio, N. Léonard, and A. C. Courville, "Estimating or propagating gradients through stochastic neurons for conditional computation," *ArXiv*, vol. abs/1308.3432, 2013.

[30] P. Yin, J. Lyu, S. Zhang, S. J. Osher, Y. Qi, and J. Xin, "Understanding straight-through estimator in training activation quantized neural nets," in *International Conference on Learning Representations*, 2019.

[31] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *Proceedings of the 32nd International Conference on International Conference on Machine Learning - Volume 37*, ser. ICML'15. JMLR.org, 2015, p. 448–456.

[32] L. Fan, K. W. Ng, C. Ju, T. Zhang, and C. S. Chan, "Deep polarized network for supervised learning of accurate binary hashing codes," in *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI-20*, C. Bessiere, Ed. International Joint Conferences on Artificial Intelligence Organization, 7 2020, pp. 825–831, main track.

[33] J. T. Hoe, K. W. Ng, T. Zhang, C. S. Chan, Y.-Z. Song, and T. Xiang, "One loss for all: Deep hashing with a single cosine similarity based learning objective," in *Advances in Neural Information Processing Systems*, M. Ranzato, A. Beygelzimer, Y. Dauphin, P. Liang, and J. W. Vaughan, Eds., vol. 34. Curran Associates, Inc., 2021, pp. 24 286–24 298.

[34] "FATE," https://fedai.org.

[35] "PySyft," https://www.openmined.org.

[36] "TF Encrypted," https://tf-encrypted.io.

[37] B. Knott, S. Venkataraman, A. Hannun, S. Sengupta, M. Ibrahim, and L. van der Maaten, "Crypten: Secure multi-party computation meets machine learning," in *arXiv 2109.00984*, 2021.

[38] C. Gentry, "Fully homomorphic encryption using ideal lattices," in *Proceedings of the forty-first annual ACM symposium on Theory of computing*, 2009, pp. 169–178.

[39] L. Zhu, Z. Liu, and S. Han, "Deep leakage from gradients," in *NeurIPS*, 2019.

[40] D. Pasquini, G. Ateniese, and M. Bernaschi, "Unleashing the tiger: Inference attacks on split learning," *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, 2021.

[41] E. Erdogan, A. Kupcu, and A. E. Cicek, "Unsplit: Data-oblivious model inversion, model stealing, and label inference attacks against split learning," *Proceedings of the 21st Workshop on Privacy in the Electronic Society*, 2021.

[42] L. I. Rudin, S. Osher, and E. Fatemi, "Nonlinear total variation based noise removal algorithms," *Physica D: Nonlinear Phenomena*, vol. 60, pp. 259–268, 1992.

[43] R. L. Rivest, "The md5 message-digest algorithm," in *RFC*, 1990.

[44] D. Wu, Q. Dai, J. Liu, B. Li, and W. Wang, "Deep incremental hashing network for efficient image retrieval," in *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, jun 2019.

[45] J. Wang, T. Zhang, J. Song, N. Sebe, and H. T. Shen, "A survey on learning to hash," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 40, no. 4, pp. 769–790, apr 2018.

[46] Z. Cao, M. Long, J. Wang, and P. S. Yu, "Hashnet: Deep learning to hash by continuation," *2017 IEEE International Conference on Computer Vision (ICCV)*, pp. 5609–5618, 2017.

[47] A. J. Paverd and A. C. Martin, "Modelling and automatically analysing privacy properties for honest-but-curious adversaries," University of Oxford, Tech. Rep., 2014.

[48] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 770–778, 2016.

[49] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *CoRR*, vol. abs/1409.1556, 2015.

[50] Y. Weiss, A. Torralba, and R. Fergus, "Spectral hashing," in *NIPS*, 2008.

[51] M. Charikar, "Similarity estimation techniques from rounding algorithms," in *STOC '02*, 2002.

[52] M. Norouzi, D. J. Fleet, and R. Salakhutdinov, "Hamming distance metric learning," in *NIPS*, 2012.

[53] A. Hard, C. M. Kiddon, D. Ramage, F. Beaufays, H. Eichner, K. Rao, R. Mathews, and S. Augenstein, "Federated learning for mobile keyboard prediction," 2018.

[54] Webank, "A case of traffic violations insurance-using federated learning," 2020, https://www.fedai.org/cases.

[55] ——, "Utilization of FATE in risk management of credit in small and micro enterprises," 2020, https://www.fedai.org/cases.

[56] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proc. IEEE*, vol. 86, pp. 2278–2324, 1998.

[57] A. Krizhevsky, G. Hinton *et al.*, "Learning multiple layers of features from tiny images," Toronto, ON, Canada, Tech. Rep., 2009.

[58] E. Barsoum, C. Zhang, C. Canton Ferrer, and Z. Zhang, "Training deep networks for facial expression recognition with crowd-sourced label distribution," in *ACM International Conference on Multimodal Interaction (ICMI)*, 2016.

[59] D. Liang, C.-C. Lu, C.-F. Tsai, and G.-A. Shih, "Financial ratios and corporate governance indicators in bankruptcy prediction: A comprehensive study," *European Journal of Operational Research*, vol. 252, no. 2, pp. 561–572, 2016.

[60] H. Guo, R. Tang, Y. Ye, Z. Li, and X. He, "Deepfm: A factorization-machine based neural network for ctr prediction," in *IJCAI*, 2017.

[61] A. L. Maas, R. E. Daly, P. T. Pham, D. Huang, A. Y. Ng, and C. Potts, "Learning word vectors for sentiment analysis," in *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*. Portland, Oregon, USA: Association for Computational Linguistics, June 2011, pp. 142–150.

[62] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," in *NAACL*, 2019.

[63] J. Tang, C. Deng, and G. Huang, "Extreme learning machine for multilayer perceptron," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 27, pp. 809–821, 2016.

[64] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *CoRR*, vol. abs/1412.6980, 2015.

[65] C. Dwork and A. Roth, "The algorithmic foundations of differential privacy," *Found. Trends Theor. Comput. Sci.*, vol. 9, pp. 211–407, 2014.

[66] M. Abadi, A. Chu, I. J. Goodfellow, H. B. McMahan, I. Mironov, K. Talwar, and L. Zhang, "Deep learning with differential privacy," *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, 2016.

[67] L. Melis, C. Song, E. De Cristofaro, and V. Shmatikov, "Exploiting unintended feature leakage in collaborative learning," in *2019 IEEE symposium on security and privacy (SP)*. IEEE, 2019, pp. 691–706.

[68] L. Grinsztajn, E. Oyallon, and G. Varoquaux, "Why do tree-based models still outperform deep learning on typical tabular data?" in *Thirty-sixth Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2022.

[69] H. Dong, C. Wu, Z. Wei, and Y. Guo, "Dropping activation outputs with localized first-layer deep network for enhancing user privacy and

data security," *IEEE Transactions on Information Forensics and Security*, vol. 13, pp. 662–670, 2017.
[70] S. Abuadbba, K. Kim, M. Kim, C. Thapa, S. A. Çamtepe, Y. Gao, H. Kim, and S. Nepal, "Can we use split learning on 1d cnn models for privacy preserving training?" *Proceedings of the 15th ACM Asia Conference on Computer and Communications Security*, 2020.
[71] N. D. Pham, A. Abuadbba, Y. Gao, T. D. K. Phan, and N. K. Chilamkurti, "Binarizing split learning for data privacy enhancement and computation reduction," *ArXiv*, vol. abs/2206.04864, 2022.
[72] A. Madry, A. Makelov, L. Schmidt, D. Tsipras, and A. Vladu, "Towards deep learning models resistant to adversarial attacks," *ArXiv*, vol. abs/1706.06083, 2017.
[73] C. Dwork, K. Kenthapadi, F. McSherry, I. Mironov, and M. Naor, "Our data, ourselves: Privacy via distributed noise generation," in *Advances in Cryptology-EUROCRYPT 2006: 24th Annual International Conference on the Theory and Applications of Cryptographic Techniques, St. Petersburg, Russia, May 28-June 1, 2006. Proceedings 25*. Springer, 2006, pp. 486–503.
[74] H. Liu, R. Wang, S. Shan, and X. Chen, "Deep supervised hashing for fast image retrieval," *International Journal of Computer Vision*, pp. 1–18, 2016.
[75] W.-J. Li, S. Wang, and W.-C. Kang, "Feature learning based deep supervised hashing with pairwise labels," *ArXiv*, vol. abs/1511.03855, 2016.
[76] Q.-Y. Jiang and W.-J. Li, "Asymmetric deep supervised hashing," in *AAAI*, 2018.
[77] L. Yuan, T. Wang, X. Zhang, F. E. H. Tay, Z. Jie, W. Liu, and J. Feng, "Central similarity quantization for efficient image and video retrieval," *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 3080–3089, 2020.
[78] Y. Liu, T. Zou, Y. Kang, W. Liu, Y. He, Z. qian Yi, and Q. Yang, "Batch label inference and replacement attacks in black-boxed vertical federated learning," 2021. [Online]. Available: https://arxiv.org/abs/2112.05409
[79] J. Sun, X. Yang, Y. Yao, and C. Wang, "Label leakage and protection from forward embedding in vertical federated learning," *ArXiv*, vol. abs/2203.01451, 2022.
[80] Y. Lin, S. Han, H. Mao, Y. Wang, and W. J. Dally, "Deep gradient compression: Reducing the communication bandwidth for distributed training," *ArXiv*, vol. abs/1712.01887, 2018.
[81] J. Liu, C. Xie, K. Kenthapadi, O. O. Koyejo, and B. Li, "RVFR: Robust vertical federated learning via feature subspace recovery," 2022. [Online]. Available: https://openreview.net/forum?id=a_ASZbWsQp_
[82] OpenAI, "Gpt-4 technical report," *ArXiv*, vol. abs/2303.08774, 2023.
[83] R. Rombach, A. Blattmann, D. Lorenz, P. Esser, and B. Ommer, "High-resolution image synthesis with latent diffusion models," 2021.

Pengyu Qiu is currently a Ph.D. student in the College of Computer Science and Technology at Zhejiang University. He received his Bachelor's degree from Zhejiang University. His current research interests include Federated Learning, AI security.

Xuhong Zhang is a ZJU 100-Young Professor with the School of Software Technology at Zhejiang University. He received his Ph.D. in Computer Engineering from University of Central Florida in 2017 . His research interests include distributed big data and AI systems, big data mining and analysis, data-driven security, AI and Security. He has authored over 20 publications in premier journals and conferences such as TDSC, TPDC, IEEE S&P, USENIX Security, ACM CCS, NDSS, VLDB, etc.

Shouling Ji is a Qiushi Distinguished Professor in the College of Computer Science and Technology at Zhejiang University. He received a Ph.D. degree in Electrical and Computer Engineering from Georgia Institute of Technology and a Ph.D. degree in Computer Science from Georgia State University. His current research interests include Data-driven Security and Privacy, AI Security and Software and System Security. He is a member of ACM and IEEE, and a senior member of CCF. He was a Research Intern at the IBM T. J. Watson Research Center. Shouling is the recipient of the 2012 Chinese Government Award for Outstanding Self-Financed Students Abroad and 10 Best/Outstanding Paper Awards, including ACM CCS 2021.

Chong Fu is currently a Ph.D. student in the College of Computer Science and Technology at Zhejiang University. He received his Bachelor's degree from Jilin University. His current research interests include federated learning and adversarial machine learning.

Xing Yang is a researcher at the State Key Laboratory of Pulsed Power Laser Technology, National University of Defense Technology. He received his BS, MS and Ph.D. degrees from Hefei Electronic Engineering Institute in 2006, 2009, and 2012 respectively. Currently, his research interests mainly focus on optoelectronic engineering, artificial intelligence, and cyberspace security.

Ting Wang is an assistant professor in the College of Information Sciences and Technology at Penn State. He received his Ph.D. degree from Georgia Tech. He conducts research at the intersection of data science and privacy & security. His ongoing work focuses on making machine learning systems more practically usable through improving their Security Assurance, Privacy Preservation and Decision-Making Transparency.