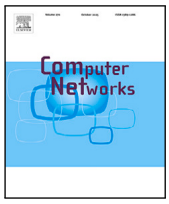
Computer Networks 290 (2026) 112734

Contents lists available at ScienceDirect

Computer Networks

journal homepage: www.elsevier.com/locate/comnet



PVFL-CPN: A privacy-preserving vertical federated learning method for computing power networks☆

Boyang Wang [a], Xiaolong Xu [b], Marcello Trovati [c], Nikolaos Polatidis [d], Francesco Palmieri [e],*

[a] School of Computer Science, Nanjing University of Posts and Telecommunications, China
[b] Jiangsu Key Laboratory of Big Data Security and Intelligent Processing, Nanjing University of Posts and Telecommunications, China
[c] School of Business, University of Lancashire, Preston, UK
[d] School of Arch, Tech and Eng at the University of Brighton, UK
[e] Department of Computer Science, University of Salerno, Italy

ARTICLE INFO

Keywords:
Computing power networks
Homomorphic encryption
Privacy preservation
Secret sharing
Vertical federated learning

ABSTRACT

With the rapid development of artificial intelligence, demand for computing power and training data has reached unprecedented levels. Vertical Federated Learning, a promising distributed training method, can effectively address data shortage problems within a single organization. In addition, the emergence of computing power networks (CPNs) provides an ideal platform to address both runtime resource limitations and distributed training issues, also in complex scenarios characterizing modern cyber–physical systems. However, native VFL is susceptible to label, gradient, and model inference attacks. Therefore, to address all of the above challenges, we propose PVFL-CPN, a privacy-preserving VFL method for computing power networks based on homomorphic encryption and secret sharing. This approach uses masking-based one-time pad encryption to secure the forward and backward propagation processes in VFL, as well as to protect communication between computing nodes in CPN. PVFL-CPN outperformed its VFL competitors in terms of accuracy, computational cost, and communication efficiency.

1. Introduction

With the rapid development of artificial intelligence (AI), the demand for computing power and training data is considerably increasing, also in scenarios characterized by complex organizational and resource-related constraints, such as advanced IoT and Cyber–Physical systems. Currently, only a few organizations have the resources to train and execute highly complex models. Although the computing power of modern edge and user terminal devices cannot address such a computational challenge, they can typically handle inference tasks. However, such computing resources are usually underutilized, which motivated the design of a novel network paradigm, the computing power networks, enabling distributed inference capabilities in next-generation cyber–physical systems. The core of the CPNs lies in the intelligent, on-demand, dynamic, and efficient scheduling of computing resources. A large number of computing nodes are deployed within the CPNs, enabling users to access computing resources anytime and anywhere. Consequently, computational tasks no longer require long-distance transmission, which significantly reduces transmission latency. Furthermore, the computing nodes in the network are capable of sensing information regarding computational tasks and network conditions, thereby endowing the network with perceptual capabilities. Finally, any Internet-connected device with processing capability can act as a computing power provider. As a result, the CPNs can break down computing-power silos and efficiently utilize isolated computing resources.

In this scenario, the introduction of privacy protection laws in various countries has made the collection of high-quality data increasingly difficult. To address this issue, Google proposed a distributed machine learning paradigm, Federated Learning (FL), which facilitates model training while protecting the data privacy of individual clients. Based on suitable data partitioning, Federated Learning is divided into horizontal Federated Learning (HFL), vertical Federated Learning (VFL), and Federated Transfer Learning (FTL), as shown in Fig. 1.

In this work, we focus on VFL, in which the training data is split according to the corresponding features. In other words, different clients may be associated with the same samples, but the corresponding features may differ, so that the labels for the samples are not available.

☆ This article is part of a Special issue entitled: 'Artificial Intelligence for Security in Networked CPS' published in Computer Networks.

* Corresponding author.
E-mail addresses: 2022040504@njupt.edu.cn (B. Wang), xuxl@njupt.edu.cn (X. Xu), MTrovati@lancashire.ac.uk (M. Trovati), N.Polatidis@brighton.ac.uk (N. Polatidis), fpalmieri@unisa.it (F. Palmieri).

https://doi.org/10.1016/j.comnet.2026.112734
Received 17 February 2026; Received in revised form 30 July 2026; Accepted 30 August 2026
Available online 15 September 2026

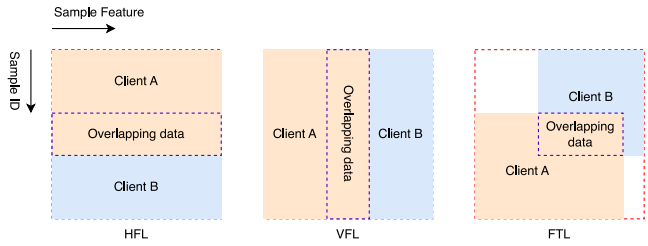


Fig. 1. HFL vs. VFL vs. FTL.

The goal of VFL is to train a *feature extractor* on each client and a *fusion model* on the server. The fusion model can collect the output of the client feature extractors (i.e., intermediate results) and predict the label corresponding to each sample. Subsequently, the server uses the true labels and the labels output by the fusion model to update the fusion model, and then calculates the gradients of the intermediate results and transmits them to the clients to update their feature extractors.

However, recent research has shown that VFL faces privacy and security challenges. For example, Li et al. [1] proposed an attack method to recover sample labels by calculating the cosine distance and the gradient norm between different samples' gradients. Fu et al. [2] used intermediate gradient results and labels held by malicious clients to perform label inference attacks. He et al. [3] proposed a model inversion attack to infer the privacy of feature data. Specifically, the authors used a regularized maximum likelihood estimation method to infer features under a white-box setting, a network inversion technique to recover features under a black-box setting, and a query-free shadow model reconstruction technique to recover features under a query-free setting. In conclusion, privacy and security issues in VFL have to be addressed urgently.

Furthermore, some privacy-preserving methods for VFL mainly focus on models such as Logistic Regression [4–9] or XGBoost [10–13]. Some other works focus on privacy protection methods for neural network models [14–18]. Among these, PrADA [15] only protects the intermediate results of the clients, while it does not protect the gradients in the backward propagation phase. However, malicious clients can infer server-side label information solely through gradient information. Fu et al. [16] use homomorphic encryption (HE) and secret sharing (SS) to protect both the client's features and the server's labels, but their high communication and computational complexity make the application difficult when the number of clients is greater than one. Gai et al. [19] proposed an adaptive differentially private vertical federated learning framework named Ada-VFed. Although the differential privacy technology used in this scheme can protect privacy, the model performance degrades sharply as the number of passive nodes increases. The remaining privacy-preserving methods [14,17,18] only support a system with one client and one server, making them difficult to apply to complex federated learning environments. Furthermore, the aforementioned works do not seem to address the issue of collusion between multiple passive nodes and the active node. This also remains an urgent problem to be solved in the privacy protection of vertical federated learning.

Motivated by privacy concerns in VFL and the limitations of existing methods, we propose *PVFL-CPN*, a privacy-preserving Vertical Federated Learning method for Computing Power Networks. Specifically, we refer to a computing node that is only associated with sample features as a passive node and a computing node that has sample features and labels as an active node. We design a VFL framework based on homomorphic encryption and secret sharing, with the assistance of a trusted third party. This framework achieves the protection of passive nodes' sample features and of the active node's labels while training the passive nodes' feature extractors and the active node's fusion model. Meanwhile, our method can prevent collusion among multiple semi-honest passive nodes and the active node. Compared to the state-of-the-art (SOTA) methods, namely BlindFL and AdaVFed, our method achieves higher performance, computational efficiency, and communication efficiency. Moreover, the performance of our method does not degrade as the number of nodes increases. Furthermore, since our method protects the output of the feature extractors, it can be extended to more complex models (e.g., CNN, AlexNet, etc.). The contributions of this work can be summarized as follows.

1. PVFL-CPN achieves the protection of passive nodes' features and the active node's labels while training the model. Even when multiple passive nodes and an active node collude, our method can still protect the features of honest passive nodes. Furthermore, compared to SOTA methods, our approach shows improvements in both communication and computational efficiency. The model performance of our method does not degrade as the number of nodes increases.
2. We propose the *PForward* and *AForward* algorithms, using homomorphic encryption and secret sharing techniques to encrypt the output of the feature extractors during the forward propagation of training and inference, thereby protecting the passive nodes' sample features.
3. We propose algorithms *PBackward* and *ABackward*, which also use homomorphic encryption and secret-sharing techniques to encrypt the gradients of the intermediate results during backward propagation, achieving model updates while protecting the active node labels.
4. We provide a comprehensive security analysis of our method and experimentally validate its effectiveness in protecting the features of passive nodes and the labels of the active node. Furthermore, we demonstrate through experiments that our method outperforms the SOTA methods in terms of performance, computational efficiency, and communication efficiency.

The rest of this article is organized as follows: In Section 2, existing works on VFL and privacy protection are discussed, while analyzing their advantages and disadvantages. Section 3 presents the definition of VFL, the cryptographic primitives used in our method, the system model, and the threat model. In Section 4, the specific details of PVFL-CPN are provided. Section 5 offers a comprehensive security analysis of the PVFL-CPN framework. Section 6 presents all the experiments conducted to assess the PVFL-CPN framework and compares it with the SOTA methods. Finally, Section 7 concludes this work and proposes future directions.

2. Related work

2.1. Vertical federated learning

Existing vertical federated learning works can be categorized into two types based on the fusion model: aggVFL and splitVFL, as shown in Fig. 2.

The fusion model used in aggVFL [20] is a loss function. The output of both the active and passive nodes' feature extraction models is a prediction result (intermediate result) for the samples. The active node "fuses" its own and the passive nodes' intermediate results into a complete prediction result using an algorithm, and then calculates the loss. For example, [21–26] proposed asynchronous collaborative methods for logistic regression models. [21] employed a backup-based straggler mitigation scheme, which uses old backup data to fill in the missing parameter positions in the current round. [22–26] used a tree-structured asynchronous communication scheme to improve communication efficiency. The advantage of aggVFL is its high computational and communication efficiency during aggregation. However, aggVFL requires a reasonable algorithm to aggregate the prediction results and ensure the accuracy of the model. Secondly, in

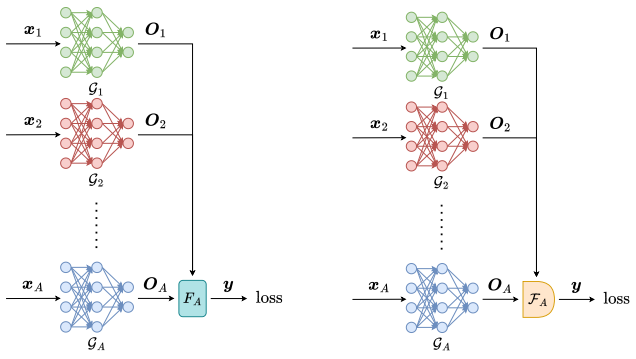


Fig. 2. aggVFL vs. splitVFL, where $\{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_A\}$ denotes the feature vectors, $\{\mathcal{G}_1, \ldots, \mathcal{G}_A\}$ denotes the feature extraction models, F_A and $\mathcal{F}_A$ denote the fusion models, and $\boldsymbol{y}$ denotes the labels corresponding to the samples. In aggVFL, F_A is a loss function, $\boldsymbol{O}_1, \ldots, \boldsymbol{O}_A$ represent the prediction results of the feature extraction models for the samples. In splitVFL, $\mathcal{F}_A$ is a prediction model, $\boldsymbol{O}_1, \ldots, \boldsymbol{O}_A$ represent the compact features extracted by the feature extraction models.

the backpropagation process, aggVFL transmits the gradients of the passive nodes' prediction results. Through these gradients, the active node's label information can be inferred.

The fusion model used in splitVFL is a prediction model. This model is typically stored on the active node, and it calculates the prediction results for samples by collecting the outputs (intermediate results) of all nodes' feature extraction models. For example, FedBCD [27] and Flex-VFL [28] proposed a training method based on federated stochastic block coordinate descent. [29] proposed an asynchronous query response vertical federated learning strategy that allows intermittent or even strategic client participation and uncoordinated training data selection. [30,31] proposed that each passive node learns the representation of each sample through unsupervised learning methods and sends this representation to the active node. The entire training process transmits the representations only once, greatly reducing communication overhead. [32–34] proposed methods for sampling and feature selection to reduce communication overhead. [32] employed group lasso regularization to filter out unimportant features. [33,34] used a trainable transformation matrix to filter out unimportant features. The advantage of splitVFL is that it can fully utilize the output of each node's feature extraction on samples to improve accuracy. However, splitVFL is inferior to aggVFL in terms of communication and computation efficiency because the dimensionality of the transmitted feature extraction outputs can vary greatly, leading to uncertainty in communication and computational efficiency during the training process.

2.2. Privacy-preservation in VFL

In privacy-preserving vertical federated learning, the protected models are categorized into three types: logistic regression, XGBoost models, and neural network models.

Privacy-preserving vertical federated learning based on logistic regression and XGBoost models includes the following representative works. Hardy et al. [5] employed a homomorphic encryption algorithm to encrypt the output of passive and active nodes and used an approximate loss function to obtain encrypted gradients on a coordinator. Yang et al. [6] used homomorphic encryption on the active node to encrypt the loss, allowing passive nodes to compute the gradients of their models. [7,8] employed homomorphic encryption and secret sharing to encrypt the intermediate results of the passive nodes and their corresponding gradients. Furthermore, [8] also protects the gradients used for model updates derived from the intermediate result gradients. However, [5–7] do not protect the gradients used for model updates derived from the intermediate result gradients, which may leak the active node's labels during batch updates. Although [8] protects these model update gradients, it only supports one active node and one passive node, limiting its applicability. [10,11] both used homomorphic encryption to encrypt the intermediate results of the passive nodes and their corresponding gradients, but similarly, they do not protect the model gradients derived from the intermediate result gradients. [12] protected the intermediate results of the passive nodes and their corresponding gradients, as well as the model gradients derived from the intermediate result gradients, by leveraging homomorphic encryption and secret-sharing techniques. However, its method supports only one active node and one passive node, restricting its scope. [13] utilized secret sharing techniques to protect not only the intermediate results of the passive nodes and their corresponding gradients, but also the model gradients derived from the intermediate result gradients, and it also supports training with multiple passive nodes. Xu et al. [35] proposed a novel vertical federated learning framework named ELXGB. Addressing the pain point where traditional federated gradient boosting trees struggle to balance computational overhead and privacy protection, this framework innovatively combines homomorphic encryption with differential privacy, achieving secure and low-latency tree model training and inference while maintaining high accuracy. Xu et al. [36] proposed a new federated learning scheme named SGBoost+. Targeting the challenges of privacy leakage and low computational/communication efficiency faced by vertical gradient boosting trees during federated outsourced training and inference, this scheme introduces a lossless and secure method during the training phase to construct models while protecting the training data, data distribution, split features, and labels. It also proposes a secure inference algorithm to complete inference at low cost, and optimizes the ciphertext to reduce both computational and communication overhead. Wang et al. [37] proposed a novel decentralized vertical federated learning framework named EDVFL, specifically designed to address the security, privacy, and incentive challenges during multi-agency collaborative data sharing in air traffic management. By constructing a secret-sharing-based tree gradient aggregation on the blockchain, and combining it with a differential-privacy-based Shapley value contribution evaluation alongside a local model reweighting strategy, this scheme achieves decentralized vertical federated learning that is resilient to reconstruction and poisoning attacks while ensuring fair incentives. However, the differential privacy scheme inevitably degrades model performance.

Privacy-preserving vertical federated learning based on neural network models includes the following representative works. In [14,15], they use homomorphic encryption to protect intermediate results and add noise to the fusion model in order to protect model predictions and fusion model gradients. However, neither of these methods protects the gradients of the passive nodes' intermediate results, allowing semi-honest passive nodes to infer the active node's label information through these gradients. Fu et al. [16] proposed a multiplication algorithm for the matrix of the source layer using homomorphic encryption and secret sharing to protect the information sent to each node in forward and backpropagation. However, this method only supports a system with one passive node and one active node, and it encounters communication and computational efficiency bottlenecks when extended to systems with multiple passive nodes. Additionally, this method cannot prevent collusion among multiple nodes. Wang et al. [38] proposed a vertical federated learning framework named PraVFed, aiming to address the core challenges of heterogeneous participant models (different architectures), massive communication overhead, and label privacy leakage. By allowing the passive parties to utilize blinding factors and perturbed labels from the active party for multiple rounds of local pre-training, and combining this with weighted representation learning to aggregate feature embeddings, this scheme

achieves vertical federated learning that supports heterogeneous models and exhibits low communication overhead while protecting privacy. Wang et al. [39] proposed a novel vertical federated learning framework named VFGCN, specifically designed to process complex graph data in vertically distributed scenarios under privacy guarantees. By designing four privacy-preserving sub-protocols—re-encryption, secure private set intersection, secure weight, and secure aggregation based on dual-threshold homomorphic encryption—this scheme constructs a vertical federated graph convolutional network framework capable of handling heterogeneous nodes and features without leaking intermediate information. However, this method does not seem to protect the gradients of intermediate results, so a certain risk of privacy leakage remains. Gai et al. [19] proposed an adaptive differentially private vertical federated learning framework named Ada-VFed. This framework provides an innovative solution to the sharp decline in model accuracy (low accuracy issues) caused by gradient clipping and noise addition in traditional differential privacy within vertical federated learning. By introducing a regularization term in the objective function to adaptively constrain the norm of intermediate results, combining this with a random gate at the input layer to filter core features, and finally utilizing Laplace scores to dynamically allocate the differential privacy budget across dimensions of varying importance, this scheme achieves high-accuracy vertical federated learning. However, the performance of this model drops significantly as the number of passive nodes increases. Liu et al. [40] proposed a new vertical federated learning framework named PVF-FD, aimed at effectively identifying "free-riders" (malicious participants that contribute no effort but obtain the results) while protecting privacy. This scheme is the first to propose using unsupervised auxiliary tasks for free-rider detection in vertical federated learning, widening the gap between normal and fraudulent parties through distribution alignment. Regarding privacy protection, it successfully constructs a ciphertext-level verifiable embedding learning scheme to achieve precise detection without exposing raw data. However, this method does not protect the active node's labels; that is, their scheme does not protect the gradients of the intermediate results. Wang et al. [41] proposed an additive ensemble vertical federated learning framework named AE-VFL. Following the core idea of boosting to improve overall model performance by sequentially generating models, this framework leverages each participant to compensate for the deficiencies of preceding ones. Specifically, AE-VFL constructs a weak model for each participant to complement the weaknesses of prior weak models. The objective of AE-VFL is to train a stronger model by integrating multiple weak models derived from different participants, thereby effectively enhancing overall model performance. On the other hand, each participant has its own independent training target—namely, the residual of the previous models rather than the server-side target label. Consequently, malicious participants can hardly infer the target label from gradient information. Hou et al. [42] proposed VerifyVFL, a scheme that enables data owners to verify the correctness of the received gradients. Specifically, they generated secure signatures for local model outputs and performed verification using the received proofs. Notably, the sizes of the signature and proof remain constant regardless of the batch size, thereby ensuring extremely low computational and communication overheads. To thwart direct label inference attacks, this method incorporates random masks, effectively preventing passive parties from accessing sample-level gradients.

The remaining privacy-preserving works focus on implementing private data alignment or defending against poisoning attacks. For instance, Ye et al. [43] presented a secure and efficient framework against model poisoning attacks for hybrid federated learning (Hybrid FL). While ensuring training efficiency (reducing time and energy consumption), this framework efficiently screens for benign devices via malicious device detection before training, directly excluding malicious nodes to prevent them from uploading malicious updates. It also analyzes the model update process on the server side through poisoned model detection to identify and eliminate sophisticated/advanced poisoning models, neutralizing their impact on the global aggregation results. However, this method only investigates defense mechanisms against poisoning attacks and does not study privacy protection. Xu et al. [44] proposed an architecture-agnostic privacy-stealing attack framework named PISTE. An attacker only needs a small amount of auxiliary query data as test inputs to obtain "raw feature-pairwise embedding" data, thereby enabling model stealing, attribute inference, and data reconstruction attacks. Xi et al. [45] proposed an efficient and reliable Private Sample Alignment (PSA) protocol specifically designed for vertical federated learning. By introducing a lightweight delegated multi-party private set intersection (MPSI) mechanism and a threshold-triggering mechanism, this scheme addresses the privacy, efficiency, and fault-tolerance robustness issues faced by multiple clients during sample alignment. Yang et al. [46] proposed a novel vertical federated learning framework named OpenVFL. The core breakthrough of this framework lies in utilizing its innovative NCLPSI protocol to completely conceal the "intersection" (i.e., which samples are mutually owned) during the sample ID alignment phase, achieving a higher level of privacy protection and supporting direct model training on fully encrypted datasets. However, this method appears to support training for only two nodes, indicating poor system scalability. Liao et al. [47] proposed a novel vertical federated learning framework named MFVFL, aiming to solve the problem of missing features across multiple clients using tensor completion technology, and further proposed a noise-resilient extended version integrated with differential privacy, named MFVFL-DP. This scheme initially fills missing features at the client side using a learnable matrix encoding, stacks the feature embeddings into a high-dimensional tensor at the server side, and leverages tensor decomposition based on low-rank constraints alongside robust tensor principal component analysis to deeply repair missing features and decouple differential privacy noise.

Table 1
Definition of the symbols used in this work.

Symbol	Definition
M	The number of passive nodes.
A	The active node.
T	Trusted third party (TTP).
i	Indices for passive nodes, active node, and TTP, i.e. $i \in \{1, \dots, M, A, T\}$
$\mathcal{X}_i$	The feature vector.
$\mathcal{W}_i$	Feature extraction model.
$\boldsymbol{U}_i, \boldsymbol{V}_i$	The partial fusion model.
$\boldsymbol{W}_i$	The complete fusion model, satisfy $\boldsymbol{W}_i = \boldsymbol{U}_i + \boldsymbol{V}_i$
$\boldsymbol{X}_i$	The output of the feature extraction model.
$\boldsymbol{\phi}_i, \boldsymbol{\psi}_i$	The output of the algorithm HE2SS.
$\boldsymbol{O}_i^U, \boldsymbol{O}_i^V$	The output of the partial fusion model.
$[\![\cdot]\!]_i$	Tensor encrypted with HE.
$[\![\cdot]\!]_i^*$	Tensor encrypted with HE and One-Time Pad.
$\boldsymbol{\Phi}_i, \boldsymbol{\Psi}_i$	The masking value.
$\boldsymbol{\Gamma}_T, \boldsymbol{\Theta}_T$	The aggregated masking value computed by T.
$\boldsymbol{Z}_i$	The output of the complete fusion model.
$\boldsymbol{Z}$	The complete output.
$\nabla \boldsymbol{Z}$	Gradient of the complete output.
$\nabla \boldsymbol{V}_i, \nabla \boldsymbol{U}_i$	The gradient of the partial fusion model.
$\nabla \boldsymbol{W}_A$	The gradient of the fusion model.
$\nabla \boldsymbol{X}_i$	The gradient of the output of the feature extraction model.

3. Preliminaries

This section introduces the VFL and cryptographic primitives used in our method, then presents the system and threat models. The symbols used in this section and subsequent sections are shown in Table 1.

3.1. Vertical federated learning

VFL is a type of federated learning where clients share the same samples, but each client has different features for the same samples, with some overlap between the features. Clients can have labels for each sample, or only labels for a subset of samples, or no labels for any samples. We call a client without labels or with partial labels a passive client, and a client with full labels an active client. During training, passive and active clients feed their local features into their local models to obtain intermediate results. Passive clients then send their intermediate results to an active client for aggregation, and the active client calculates the loss based on the labels and performs backward propagation. The formal definition of VFL is as follows.

3.1.1. Formal definition

VFL aims to jointly train a machine learning model $\boldsymbol{\Theta}$ using a dataset $\mathcal{D} \triangleq \{(\boldsymbol{x}_i, y_i)\}_{i=1}^{N}$, while preserving the privacy of each client model and dataset. The loss function of VFL is defined as (1):

$$\min_{\boldsymbol{\Theta}} l(\boldsymbol{\Theta}; D) \triangleq \frac{1}{N} \sum_{i=1}^{N} f(\boldsymbol{\Theta}; \boldsymbol{x}_i, y_i) + \lambda \sum_{i=1}^{K} \gamma(\boldsymbol{\Theta}), \tag{1}$$

where N represents the number of samples in the dataset, K represents the number of clients, $f(\cdot)$ represents the loss function, $\gamma(\cdot)$ represents the regularization term, and λ is the hyperparameter that controls the regularization term.

VFL partitions the dataset based on sample features. The features $\boldsymbol{x}_i \in \mathbb{R}^{1\times d}$ of each data sample are divided into K clients $\{\boldsymbol{x}_{i,k} \in \mathbb{R}^{1\times d_k}\}_{k=1}^{K}$, where d_k is the number of feature dimensions owned by client k. The Kth client has the label information, i.e., $y_i = y_{i,K}$. We call the Kth client, which has the labels for all samples, the active client, and other client $k, k \in [0, K-1]$, denoted as the passive client. Therefore, passive client k has the dataset $\mathcal{D}_k \triangleq \{\boldsymbol{x}_{i,k}\}_{i=1}^{N}$, and the active client has the dataset $\mathcal{D}_K \triangleq \{\boldsymbol{x}_{i,K}, y_{i,K}\}_{i=1}^{N}$.

In model training, VFL divides the model into two parts. One part is the local model $\mathcal{G}_k$, represented by parameters $\theta_k, k \in \{1, \ldots, K\}$, and $\mathcal{G}_k$ is responsible for processing local sample features $\boldsymbol{x}_{i,k}$. The other part of the model is the global module $\mathcal{F}_K$, which is represented by parameters δ_K and is only calculated on the Kth client. Therefore, we can rewrite the loss function as (2):

$$f\left(\boldsymbol{\Theta}; \boldsymbol{x}_i, y_i\right) = \mathcal{L}\left(\mathcal{F}_K\left(\delta_K; \mathcal{G}_1(\boldsymbol{x}_{i,1}, \theta_1), \ldots, \mathcal{G}_K(\boldsymbol{x}_{i,K}, \theta_K),\right) y_{i,K}\right). \tag{2}$$

During collaborative training, the local data of each client is not exchanged. The local model $\mathcal{G}_k$ can be any model, including linear and logistic regression, support vector machines, multi-layer perceptrons, and convolutional neural networks. The global module $\mathcal{F}_K$ can be either trainable (single-layer neural network) or non-trainable (single aggregation function, such as the Sigmoid function). If $\mathcal{F}_K$ is trainable, we call this type of VFL splitVFL, and if $\mathcal{F}_K$ is non-trainable, we call this type of VFL aggVFL. The difference between splitVFL and aggVFL is shown in Fig. 2.

3.1.2. Training strategy

Here, we introduce a general training strategy for VFL. We assume that our data samples are already aligned, i.e., we do not consider data alignment methods here. Each client k calculates the local model output $O_k = \mathcal{G}_i\left(\boldsymbol{x}_k, \theta_k\right)$ on a mini-batch of data samples $\boldsymbol{x}_k$ and then sends O_k to the active client. After collecting all $\{O_k\}_{k=1}^{K}$, the active client calculates the training loss according to (2). Then, the active client calculates the gradient $\partial l/\partial \delta_K$ of the global module $\mathcal{F}_k$ and updates δ_K using $\partial l/\partial \delta_K$. Next, the active client calculates the gradient $\partial l/\partial O_k$ for each client and sends it back to the client. Finally, each client calculates its local model gradient according to (3):

$$\nabla_{\theta_k} l = \frac{\partial l}{\partial \theta_k} = \frac{\partial l}{\partial O_k} \frac{\partial O_k}{\partial \theta_k} \tag{3}$$

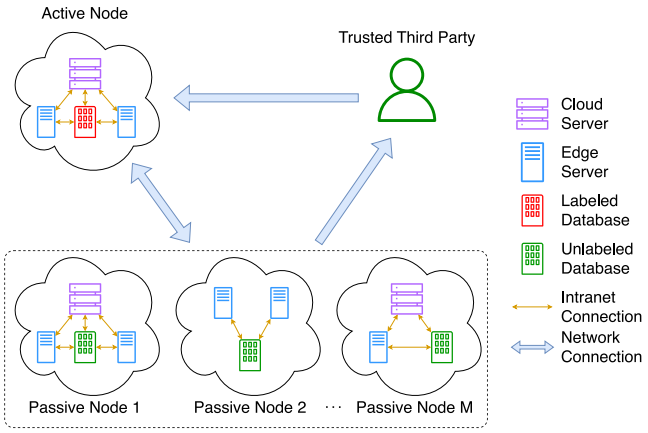


Fig. 3. The system model as described in Section 3.3.

Algorithm 1 The HE2SS Algorithm.

```
function HE2SS([[M]]_a, b)
    if a = b then
        M ← Pai.Dec([[M]]_a, sk_a)
        return M
    else
        b randomly generates a vector φ_b with the same shape as [[M]]_a
        [[M − φ_b]]_a ← [[M]]_a − φ_b
        return [[M − φ_b]]_a, φ_b
```

3.2. Cryptographic primitives

The Paillier HE algorithm consists of five parts: key generation, encryption, decryption, homomorphic addition, and homomorphic multiplication. We denote the key length as κ, and the five algorithms are described as follows:

- Pai.KeyGen(κ): randomly selects two large primes p and q of length $\kappa/2$ and then calculates their product $n = pq$. Next, the algorithm randomly selects g from $\mathcal{B}$, where $\mathcal{B} = \cup_{\alpha=1}^{\lambda} \mathcal{B}_\alpha$, and $\mathcal{B}_\alpha \subset \mathbb{Z}_{n^2}^*$, i.e. the set of elements in $\mathbb{Z}_{n^2}^*$ of order $n\alpha$. The parameter λ represents the Carmichael function, defined as: $\lambda(n) = \mathrm{lcm}(p-1, q-1)$. Finally, the algorithm checks whether the following condition is satisfied:

  $$\gcd(L(g^\lambda \bmod n^2), n) = 1,$$

  where $L(x) = (x-1)/n$. If the condition is satisfied, we represent $pk = (n, g)$ as the public key and $sk = (p, q)$ as the private key.
- Pai.Enc(pk, m): computes the ciphertext through $c = g^m r^n \bmod n^2 (r < n, m < n)$.
- Pai.Dec(sk, c): computes the plaintext through $m = \frac{L(c^\lambda \bmod n^2)}{L(g^\lambda \bmod n^2)} \bmod n (c < n^2)$.
- Pai.Add(pk, m_1, m_2): the additive homomorphism property of the algorithm can be expressed as $\mathrm{Enc}(pk, m_1 + m_2) = \mathrm{Enc}(pk, m_1) \cdot \mathrm{Enc}(pk, m_2)$.
- Pai.Mul(pk, m_1, m_2): the multiplicative homomorphism property of the algorithm can be expressed as $\mathrm{Enc}(pk, m_1 m_2) = \mathrm{Enc}(pk, m_1)^{m_2} = \mathrm{Enc}(pk, m_2)^{m_1}$.

An important aspect is *Secret Sharing (SS)*, which allows a secret to be divided into multiple shares distributed to different clients, each client not knowing the exact value of the secret. In this work, we employ Alg. 1 [16] to transform a HE variable $[\![s]\!]$ into a SS variable $\langle \phi, [\![s-\phi]\!] \rangle$, where s can be either a scalar or a vector.

The *One-Time Pad (OTP)* scheme presented in [48] encrypts the vector $\boldsymbol{x}_u$ for $u \in \mathcal{U}$ as follows (4)

$$\boldsymbol{y}_u = \boldsymbol{x}_u + \boldsymbol{s}_u \pmod{R}, \tag{4}$$

where $\boldsymbol{s}_u$ is a vector of pseudo-random numbers generated by client u. The client u then sends $\boldsymbol{y}_u$ to the aggregator (server), and subsequently all clients $u \in \mathcal{U}$ inform the aggregator in a secret way of $\sum_{u\in\mathcal{U}} \boldsymbol{s}_u \pmod{R}$. The aggregator then computes:

$$\boldsymbol{z} = \sum_{u\in\mathcal{U}} \boldsymbol{y}_u - \sum_{u\in\mathcal{U}} \boldsymbol{s}_u \pmod{R} = \sum_{u\in\mathcal{U}} \boldsymbol{x}_u \pmod{R}. \tag{5}$$

3.3. The system model

In this work, we consider a federated learning environment with M passive nodes and an active node. Each of them has a different infrastructure, such as cloud servers or edge servers, which will be considered as computing nodes. Each passive node only has feature vectors $\mathcal{X}_i \in \mathbb{R}^{d_i}, i = 1, \ldots, M$. Note that $\mathcal{X}_i$ can overlap with $\mathcal{X}_j$, for $i \neq j$. The active node A has feature vectors $\mathcal{X}_A$ and the corresponding labels $\boldsymbol{y}_A$. Both passive and active nodes have a feature extraction model $\mathcal{G}_i$ with $i \in \{i, \ldots, M, A\}$. The feature extraction model weights are represented by $\mathcal{W}_i$, for $i \in \{1, \ldots, M, A\}$, and the active node A has a fusion model $\mathcal{F}_A$ to aggregate the intermediate results of all nodes. In this work, the VFL paradigm considered is *splitVFL*, so that the fusion model is a single-layer neural network, used to output the prediction results. In addition, each computing node holds a part of the fusion model (that is, $\boldsymbol{U}_i$ and $\boldsymbol{V}_i$). Finally, a TTP has been introduced, which is responsible for calculating the masking value sum sent from the passive and active nodes. The overall system model is shown in Fig. 3.

3.4. Threat model

In the system model, the active node can be honest or honest-but-curious. When the active node is honest-but-curious, it attempts to infer the feature information of passive nodes from the intermediate results and their gradients sent by the passive nodes. When a passive node is honest-but-curious, it attempts to infer the active node's label information from the partial fusion model weights and the gradients used to update these weights that it possesses. Furthermore, an honest-but-curious active node attempts to collude with honest-but-curious passive nodes to steal the feature information of honest passive nodes. The proposed method can protect the feature information of the remaining honest passive nodes even when one active node and up to $M-2$ passive nodes collude.

4. Description of the method

This section presents the specific details of our method. The overview of our method is shown in Fig. 4.

4.1. System initialization

First, the public parameter κ is set, which will be used to generate the public–private key pair for each node. The parameters for federated learning, such as e (epoch), lr (learning rate), bs (batch size), etc., will also be defined.

Then, the passive node P_i initializes the public–private key pair (pk_i, sk_i), and the active node initializes the public–private key pair (pk_A, sk_A).

Next, each node initializes its feature extraction model $\mathcal{W}_i$ for $i \in \{1, \ldots, M, A\}$. For the active node, A randomly initializes a partial fusion model $\boldsymbol{V}_i$ for each P_i. Then, A encrypts $\boldsymbol{V}_i$ using its public key to get $[\![\boldsymbol{V}_i]\!]_A$ and send $[\![\boldsymbol{V}_i]\!]_A$ to P_i. For passive nodes, if $M > 1$, P_i randomly initializes another part of its fusion model $\boldsymbol{U}_i$, such that $\boldsymbol{U}_i + \boldsymbol{V}_i = \boldsymbol{W}_i$. Otherwise, P_i randomly initializes another part of its fusion model $\boldsymbol{U}_i$, such that $\boldsymbol{U}_i + \boldsymbol{V}_i = \boldsymbol{W}_i$. Then, P_i will follow the same approach to the partial fusion models $\boldsymbol{U}_A$ and $\boldsymbol{V}_A$ of A and sends $\boldsymbol{U}_A$ to A.

Algorithm 2 SumPhi.

```
▷ Parameters in "[]" are input only when the condition is met. ◁
function SumPhi(M, {Φ_i}_{i=1}^M, [Φ_A, V_A])
    if M > 1 then
        Γ_T = Σ_{i=1}^M Φ_i
    else
        Γ_T = Φ_A V_A + Φ_1
    return Γ_T
```

Algorithm 3 The PForward Algorithm.

```
function PForward(M, X_i, [[V_i]]_A, U_i, [V_A, X_A + Φ_A])
    [[X_i V_i]]_A ← X_i [[V_i]]_A
    [[X_i V_i − φ_i]]_A, φ_i ← HE2SS([[X_i V_i]]_A, i)
    [[O_i^V]]_A ← [[X_i V_i − φ_i]]_A
    P_i randomly generates Φ_i with the same shape as [[O_i^V]]_A
    [[O_i^V]]_A^* ← [[O_i^V]]_A + Φ_i
    if M > 1 then
        O_i^U ← X_i U_i + φ_i
    else
        O_i^U ← X_i U_i + φ_i + (X_A + Φ_A) V_A
    return [[O_i^V]]_A^*, O_i^U, Φ_i
```

4.2. The aggregation of the masking value

The *SumPhi* algorithm aggregates the masking values used for encryption by passive nodes during forward propagation, as well as for encrypting intermediate results by the active node during backward propagation. If $M > 1$, the TTP aggregates the masking values of the passive nodes, i.e.

$$\boldsymbol{\Gamma}_T = \sum_{i=1}^{M} \boldsymbol{\Phi}_i,$$

if $M = 1$, the TTP aggregates the masking values from P_1 and A, i.e.

$$\boldsymbol{\Gamma}_T = \boldsymbol{\Phi}_A \boldsymbol{V}_A + \boldsymbol{\Phi}_1,$$

where $\boldsymbol{\Gamma}_T$ represents the masking values aggregated by the TTP, and $\boldsymbol{\Phi}_i$ represents the masking value of P_i. The pseudocode is shown in Alg. 2. Finally, the TTP sends $\boldsymbol{\Gamma}_T$ to A.

4.3. Passive node forward propagation

In the *PForward* algorithm, each P_i uses the output of its feature extraction model (intermediate result) and a portion of the weights of the fusion model to compute a partial prediction output, which is then sent to A for aggregation. Throughout this process, HE and SS ensure the security of the intermediate results. The specific steps of the algorithm are as follows.

First, P_i computes $[\![\boldsymbol{X}_i \boldsymbol{V}_i]\!]_A$ using the intermediate result $\boldsymbol{X}_i$ and the encrypted fusion model $[\![\boldsymbol{V}_i]\!]_A$, i.e. $[\![\boldsymbol{X}_i \boldsymbol{V}_i]\!]_A \leftarrow \boldsymbol{X}_i [\![\boldsymbol{V}_i]\!]_A$. Then, P_i converts $[\![\boldsymbol{X}_i \boldsymbol{V}_i]\!]_A$ into a SS variable using Alg. 1 and sets it as output $[\![\boldsymbol{O}_i^{\boldsymbol{V}}]\!]_A$, i.e. $[\![\boldsymbol{X}_i \boldsymbol{V}_i - \boldsymbol{\phi}_i]\!]_A, \boldsymbol{\phi}_i \leftarrow \text{HE2SS}([\![\boldsymbol{X}_i \boldsymbol{V}_i]\!]_A, i)$, $[\![\boldsymbol{O}_i^{\boldsymbol{V}}]\!]_A \leftarrow [\![\boldsymbol{X}_i \boldsymbol{V}_i - \boldsymbol{\phi}_i]\!]_A$.

Then, to protect the output $\boldsymbol{O}_i^{\boldsymbol{V}}$ of P_i from collusion attacks by passive nodes and the active node, P_i generates a masking value $\boldsymbol{\Phi}_i$ and encrypts $\boldsymbol{O}_i^{V}$ with it, i.e., $[\![\boldsymbol{O}_i^{\boldsymbol{V}}]\!]_A^* \leftarrow [\![\boldsymbol{O}_i^{\boldsymbol{V}}]\!]_A + \boldsymbol{\Phi}_i$. P_i sends $\boldsymbol{\Phi}_i$ to T.

Finally, P_i computes the output $\boldsymbol{O}_i^{\boldsymbol{U}}$ of the intermediate result $\boldsymbol{X}_i$ on the fusion model $\boldsymbol{U}_i$, i.e. $\boldsymbol{O}_i^{\boldsymbol{U}} \leftarrow \boldsymbol{X}_i \boldsymbol{U}_i + \boldsymbol{\phi}_i$. If $M = 1$, to correctly aggregate the prediction results, P_i needs to add the output of $\boldsymbol{X}_A + \boldsymbol{\Phi}_A$ on the fusion model $\boldsymbol{V}_A$, i.e. $\boldsymbol{O}_i^{\boldsymbol{U}} \leftarrow \boldsymbol{X}_i \boldsymbol{U}_i + \boldsymbol{\phi}_i + (\boldsymbol{X}_A + \boldsymbol{\Phi}_A)\boldsymbol{V}_A$. Here, $\boldsymbol{X}_A + \boldsymbol{\Phi}_A$ is the intermediate result $\boldsymbol{X}_A$ encrypted by the active node A using a masking value $\boldsymbol{\Phi}_A$. The pseudocode is shown in Alg. 3.

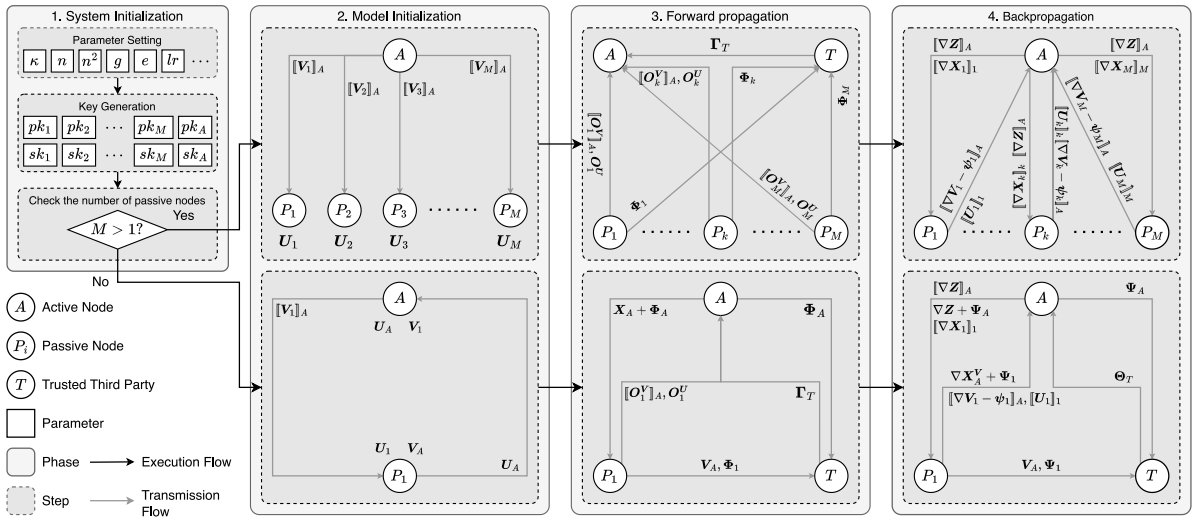


Fig. 4. The overview of the PVFL-CPN method.

Algorithm 4 The AForward Algorithm.

function AForward(M, $\boldsymbol{X}_A$, $\boldsymbol{W}_A$ or $\boldsymbol{U}_A$, $\{[\![\boldsymbol{O}_i^V]\!]_A^*\}_{i=1}^M$, $\{\boldsymbol{O}_i^U\}_{i=1}^M$, $\boldsymbol{\Gamma}_T$, $\boldsymbol{y}$)
  for $i = 1$ to M **do**
    $\boldsymbol{O}_i^{V,*} \leftarrow$ HE2SS($[\![\boldsymbol{O}_i^V]\!]_A^*, A$)
  if $M > 1$ **then**
    $\boldsymbol{Z}_A \leftarrow \boldsymbol{X}_A \boldsymbol{W}_A$
    $\boldsymbol{Z} \leftarrow \boldsymbol{Z}_A + \left(\sum_{i=1}^M \boldsymbol{O}_i^{V,*} + \boldsymbol{O}_i^U\right) - \boldsymbol{\Gamma}_T$
  else
    $\boldsymbol{Z}_A^U \leftarrow \boldsymbol{X}_A \boldsymbol{U}_A$
    $\boldsymbol{Z} \leftarrow \boldsymbol{Z}_A^U + \boldsymbol{O}_1^{V,*} + \boldsymbol{O}_1^U - \boldsymbol{\Gamma}_T$
  loss $\leftarrow$ loss_fun($\boldsymbol{Z}, \boldsymbol{y}$)
  return loss

4.4. Active node forward propagation

In the AForward algorithm, A collects the output of all P_i and the masking value aggregated by T. Then, it decrypts the passive nodes' prediction results using the HE2SS algorithm and the masking value. Subsequently, A computes its own prediction result based on its intermediate result and the fusion model. Finally, A aggregates all the results and calculates the loss based on the labels. The specific steps are as follows.

First, the active node A invokes Alg. 1 to decrypt the prediction output $[\![\boldsymbol{O}_i^V]\!]_A^*$ sent by each passive node P_i, that is, $\boldsymbol{O}_i^{V,*} \leftarrow$ HE2SS($[\![\boldsymbol{O}_i^V]\!]_A^*, A$). Subsequently, A computes the prediction result $\boldsymbol{Z}_A$ using its intermediate result $\boldsymbol{X}_A$ and the fusion model $\boldsymbol{W}_A$, i.e. $\boldsymbol{Z}_A \leftarrow \boldsymbol{X}_A \boldsymbol{W}_A$. If $M = 1$, then $\boldsymbol{W}_A$ is replaced by $\boldsymbol{U}_A$, i.e. $\boldsymbol{Z}_A^U \leftarrow \boldsymbol{X}_A \boldsymbol{U}_A$.

After that, A aggregates all the prediction results and decrypts them using the masking value $\boldsymbol{\Gamma}_T$ to obtain the complete prediction result $\boldsymbol{Z}$, which is

$$\begin{aligned}\boldsymbol{Z} &= \boldsymbol{Z}_A + \left(\sum_{i=1}^M \boldsymbol{O}_i^{V,*} + \boldsymbol{O}_i^U\right) - \boldsymbol{\Gamma}_T = \boldsymbol{Z}_A + \sum_{i=1}^M \boldsymbol{X}_i \boldsymbol{W}_i \\ &= \boldsymbol{Z}_A + \sum_{i=1}^M \boldsymbol{X}_i \boldsymbol{V}_i - \boldsymbol{\phi}_i + \boldsymbol{\Phi}_i + \boldsymbol{X}_i \boldsymbol{U}_i + \boldsymbol{\phi}_i - \boldsymbol{\Gamma}_T \\ &= \boldsymbol{Z}_A + \sum_{i=1}^M \boldsymbol{X}_i \boldsymbol{W}_i\end{aligned}$$

If $M = 1$, the above expression becomes

$$\boldsymbol{Z} \leftarrow \boldsymbol{Z}_A^U + \boldsymbol{O}_1^{V,*} + \boldsymbol{O}_1^U - \boldsymbol{\Gamma}_T.$$

Algorithm 5 The ABackward Algorithm.

function ABackward(loss, $\boldsymbol{X}_A$, $\boldsymbol{W}_A$)
  $\nabla \boldsymbol{Z} \leftarrow$ loss.backward()
  $[\![\nabla \boldsymbol{Z}]\!]_A \leftarrow$ Pai.Enc($pk_A, \nabla \boldsymbol{Z}$)
  A sends $[\![\nabla \boldsymbol{Z}]\!]_A$ to P_i
  for $P_i, (i = 1, \ldots, M)$ **parallel do**
    $[\![\nabla \boldsymbol{V}_i - \boldsymbol{\psi}_i]\!]_A \leftarrow$ PCalVGrad($[\![\nabla \boldsymbol{Z}]\!]_A$)
    P_i sends $[\![\nabla \boldsymbol{V}_i - \boldsymbol{\psi}_i]\!]_A$ to A
  for $i = 1, \ldots, M$ **do**
    $\nabla \boldsymbol{V}_i - \boldsymbol{\psi}_i \leftarrow$ HE2SS($[\![\nabla \boldsymbol{V}_i - \boldsymbol{\psi}_i]\!]_A, A$)
  $\nabla \boldsymbol{W}_A \leftarrow \boldsymbol{X}_A^T \nabla \boldsymbol{Z}$
  if $M > 1$ **then**
    $\nabla \boldsymbol{X}_A \leftarrow \nabla \boldsymbol{Z} \boldsymbol{W}_A^T$
  else
    A generates a masking value $\boldsymbol{\Psi}_A$
    A calculates $\nabla \boldsymbol{Z} + \boldsymbol{\Psi}_A$ and sends it to P_1
    $\nabla \boldsymbol{X}_A^{V,*} + \boldsymbol{\Psi}_1 \leftarrow$ PCalXGrad($\nabla \boldsymbol{Z} + \boldsymbol{\Psi}_A$)
    A sends $\boldsymbol{\Psi}_A$ to T
    $\boldsymbol{\Theta}_T \leftarrow$ SumPhi($M, \boldsymbol{V}_A, \boldsymbol{\Psi}_A, \boldsymbol{\Psi}_1$)
    $\nabla \boldsymbol{X}_A \leftarrow \nabla \boldsymbol{Z} \boldsymbol{U}_A^T + \nabla \boldsymbol{X}_A^{V,*} + \boldsymbol{\Psi}_1 - \boldsymbol{\Theta}_T$
  return $\{\nabla \boldsymbol{V}_i - \boldsymbol{\psi}_i\}_{i=1}^M, \nabla \boldsymbol{W}_A, \nabla \boldsymbol{X}_A$
function PCalVGrad($[\![\nabla \boldsymbol{Z}]\!]_A$)
  $[\![\nabla \boldsymbol{V}_i]\!]_A \leftarrow \boldsymbol{X}_i^T [\![\nabla \boldsymbol{Z}]\!]_A$
  $[\![\nabla \boldsymbol{V}_i - \boldsymbol{\psi}_i]\!]_A, \boldsymbol{\psi}_i \leftarrow$ HE2SS($[\![\nabla \boldsymbol{V}_i]\!]_A, i$)
  return $[\![\boldsymbol{V}_i - \boldsymbol{\psi}_i]\!]_A$
function PCalXGrad($\nabla \boldsymbol{Z} + \boldsymbol{\Psi}_A$)
  $\nabla \boldsymbol{X}_A^{V,*} \leftarrow (\nabla \boldsymbol{Z} + \boldsymbol{\Psi}_A) \boldsymbol{V}_A^T$
  P_1 generates a masking value $\boldsymbol{\Psi}_1$
  P_1 calculates $\nabla \boldsymbol{X}_A^{V,*} + \boldsymbol{\Psi}_1$
  P_1 sends $\boldsymbol{V}_A$ and $\boldsymbol{\Psi}_1$ to T
  return $\nabla \boldsymbol{X}_A^{V,*} + \boldsymbol{\Psi}_1$

Finally, A computes the loss using the prediction result $\boldsymbol{Z}$ and the labels $\boldsymbol{y}$ via the loss function loss_fun. The pseudocode is shown in Alg. 4.

4.5. Active node backward propagation

In the ABackward algorithm, A performs backward propagation to obtain the gradient of the prediction result $\boldsymbol{Z}$. Then, A encrypts this gradient and interacts with each P_i to obtain the gradient of the fusion model $\boldsymbol{V}_i$. Finally, A computes the gradients of its own fusion model $\boldsymbol{W}_A$ or $\boldsymbol{U}_A$ and intermediate result $\boldsymbol{X}_A$. The specific steps are as follows.

Algorithm 6 The PBackward Algorithm.

function PBackward($\boldsymbol{U}_i$, pk_i, sk_i)
  $[\![\boldsymbol{U}_i]\!]_i \leftarrow$ Pai.Enc($pk_i, \boldsymbol{U}_i$)
  P_i sends $[\![\boldsymbol{U}_i]\!]_i$ to A
  $[\![\nabla \boldsymbol{X}_i]\!]_i \leftarrow$ ACalGrad($[\![\boldsymbol{U}_i]\!]_i$)
  A sends $[\![\nabla \boldsymbol{X}_i]\!]_i$ to P_i
  $\nabla \boldsymbol{X}_i \leftarrow$ Pai.Dec($sk_i, [\![\nabla \boldsymbol{X}_i]\!]_i$)
  return $\nabla \boldsymbol{X}_i$
function ACalGrad($[\![\boldsymbol{U}_i]\!]_i$)
  $[\![\boldsymbol{W}_i]\!]_i \leftarrow [\![\boldsymbol{U}_i]\!]_i + \boldsymbol{V}_i$
  $[\![\nabla \boldsymbol{X}_i]\!]_i \leftarrow \nabla \boldsymbol{Z} [\![\boldsymbol{W}_i]\!]_i^T$
  return $[\![\nabla \boldsymbol{X}_i]\!]_i$

Algorithm 7 The Training Algorithm.

function Train()
  ▷ *Forward propagation* ◁
  $\boldsymbol{X}_A \leftarrow \mathcal{W}_A^e(\mathcal{X}_A)$
  for $P_i, i = 1, \ldots, M$ **parallel do**
    $\boldsymbol{X}_i \leftarrow \mathcal{W}_i^e(\mathcal{X}_i)$
    if $M = 1$ **then**
      A generates a masking value $\boldsymbol{\Phi}_A$ and sends it to T
      A sends $\boldsymbol{X}_A + \boldsymbol{\Phi}_A$ to P_i
      $[\![\boldsymbol{O}_i^V]\!]_A^*, \boldsymbol{O}_i^U, \boldsymbol{\Phi}_i \leftarrow$ PForward($\boldsymbol{X}_i, [\![\boldsymbol{V}_i^e]\!]_A, \boldsymbol{U}_i^e, \boldsymbol{V}_A^e, \boldsymbol{X}_A + \boldsymbol{\Phi}_A$)
      P_i sends $\boldsymbol{V}_A^e, \boldsymbol{\Phi}_i$ to T
    else
      $[\![\boldsymbol{O}_i^V]\!]_A^*, \boldsymbol{O}_i^U, \boldsymbol{\Phi}_i \leftarrow$ PForward($\boldsymbol{X}_i, [\![\boldsymbol{V}_i^e]\!]_A, \boldsymbol{U}_i^e$)
      P_i sends $\boldsymbol{\Phi}_i$ to T
    P_i sends $[\![\boldsymbol{O}_i^V]\!]_A^*, \boldsymbol{O}_i^U$ to A
  if $M > 1$ **then**
    $\boldsymbol{\Gamma}_T \leftarrow$ SumPhi($\{\boldsymbol{\Phi}_i\}_{i=1}^T$), T sends $\boldsymbol{\Gamma}_T$ to A
    loss $\leftarrow$ AForward($\boldsymbol{X}_A, \boldsymbol{W}_A, \{[\![\boldsymbol{O}_i^V]\!]_A^*\}_{i=1}^M, \{\boldsymbol{O}_i^U\}_{i=1}^M, \boldsymbol{\Gamma}_T, \boldsymbol{y}$)
  else
    $\boldsymbol{\Gamma}_T \leftarrow$ SumPhi($\boldsymbol{V}_A^e, \boldsymbol{\Phi}_i, \boldsymbol{\Phi}_A$), T sends $\boldsymbol{\Gamma}_T$ to A
    loss $\leftarrow$ AForward($\boldsymbol{X}_A, \boldsymbol{U}_A, [\![\boldsymbol{O}_1^V]\!]_A^*, \boldsymbol{O}_1^U, \boldsymbol{\Gamma}_T, \boldsymbol{y}$)
  ▷ *Backward propagation* ◁
  if $M > 1$ **then**
    $\{\nabla \boldsymbol{V}_i - \boldsymbol{\psi}_i\}_{i=1}^M, \nabla \boldsymbol{W}_A, \nabla \boldsymbol{X}_A \leftarrow$ ABackward(loss, $\boldsymbol{X}_A, \boldsymbol{W}_A^e$)
  else
    $\nabla \boldsymbol{V}_1 - \boldsymbol{\psi}_1, \nabla \boldsymbol{W}_A, \nabla \boldsymbol{X}_A \leftarrow$ ABackward(loss, $\boldsymbol{X}_A, \boldsymbol{U}_A^e$)
  for $P_i, i = 1, \ldots, M$ **parallel do**
    $\nabla \boldsymbol{X}_i \leftarrow$ PBackward($\boldsymbol{U}_i, pk_i, sk_i$)
  ▷ *Model update* ◁
  for $P_i, i = 1, \ldots, M$ **do**
    $\boldsymbol{V}_i^{e+1} \leftarrow \boldsymbol{V}_i^e - \eta(\nabla \boldsymbol{V}_i - \boldsymbol{\psi}_i)$
    $[\![\boldsymbol{V}_i^{e+1}]\!]_A \leftarrow$ Pai.Enc($pk_A, \boldsymbol{V}_i^{e+1}$)
    A sends $[\![\boldsymbol{V}_i^{e+1}]\!]_A$ to P_i
  if $M > 1$ **then**
    $\boldsymbol{W}_A^{e+1} \leftarrow \boldsymbol{W}_A^e - \eta \nabla \boldsymbol{W}_A$
  else
    $\boldsymbol{U}_1^{e+1} \leftarrow \boldsymbol{U}_1^e - \eta \boldsymbol{\psi}_1$
  A performs backward propagation using $\nabla \boldsymbol{X}_A$ and updates $\mathcal{W}_A^e$
  for $P_i, i = 1, \ldots, M$ **parallel do**
    $\boldsymbol{U}_i^{e+1} \leftarrow \boldsymbol{U}_i^e - \eta \boldsymbol{\psi}_i$
    P_i performs backward propagation using $\nabla \boldsymbol{X}_i$, updating $\mathcal{W}_i^e$

First, A obtains the gradient $\nabla \boldsymbol{Z}$ of the prediction result $\boldsymbol{Z}$ through the backward propagation algorithm. Subsequently, A encrypts $\nabla \boldsymbol{Z}$ and sends it to P_i, i.e. $[\![\nabla \boldsymbol{Z}]\!]_A \leftarrow$ Pai.Enc($pk_A, \nabla \boldsymbol{Z}$).

After receiving the encrypted gradient $[\![\nabla \boldsymbol{Z}]\!]_A$, P_i uses its own intermediate result $\boldsymbol{X}_i$ to calculate the encrypted gradient $[\![\nabla \boldsymbol{V}_i]\!]_A$ of the fusion model $\boldsymbol{V}_i$, that is, $[\![\nabla \boldsymbol{V}_i]\!]_A \leftarrow \boldsymbol{X}_i^T [\![\nabla \boldsymbol{Z}]\!]_A$. Then, P_i converts $[\![\nabla \boldsymbol{V}_i]\!]_A$ into an SS variable using Alg. 1 and sends it to A, i.e., $[\![\nabla \boldsymbol{V}_i - \boldsymbol{\psi}_i]\!]_A, \boldsymbol{\psi}_i \leftarrow$ HE2SS($[\![\nabla \boldsymbol{V}_i]\!]_A, i$). This part is represented by the *PCalVGrad* function.

Algorithm 8 The PVFL-CPN Algorithm.

Input: $\kappa, \eta, M, epoch, \mathcal{X}_i (i = 1, \ldots, M), \mathcal{X}_A, \boldsymbol{y}$
for $P_i, i = 1, \ldots, M$ **parallel do**
  $pk_i, sk_i \leftarrow$ Pai.KeyGen(κ)
  P_i randomly initializes the feature extraction model $\mathcal{W}_i$
  P_i randomly initializes the fusion model $\boldsymbol{U}_i$
$pk_A, sk_A \leftarrow$ Pai.KeyGen(κ)
A randomly initializes the feature extraction model $\mathcal{W}_A$.
if $M = 1$ **then**
  P_1 randomly initializes the fusion models $\boldsymbol{U}_A$ and $\boldsymbol{V}_A$
  P_1 sends $\boldsymbol{U}_A$ to A
else
  A randomly initializes the fusion model $\boldsymbol{W}_A$
for $i = 1, \ldots, M$ **do**
  A randomly generates $\boldsymbol{V}_i$
  $[\![\boldsymbol{V}_i]\!]_A \leftarrow$ Pai.Enc($pk_A, \boldsymbol{V}_i$)
  A sends $[\![\boldsymbol{V}_i]\!]_A$ to P_i
for $e = 1, \ldots, epoch$ **do**
  Train()

After A collects all the SS variables $[\![\nabla \boldsymbol{V}_i - \boldsymbol{\psi}_i]\!]_A$, it invokes Alg. 1 to decrypt them, i.e.:

$$\nabla \boldsymbol{V}_i - \boldsymbol{\psi}_i \leftarrow \text{HE2SS}([\![\nabla \boldsymbol{V}_i - \boldsymbol{\psi}_i]\!]_A, A).$$

Finally, A computes the gradient $\nabla \boldsymbol{W}_A$ of the fusion model $\boldsymbol{W}_A$ or $\boldsymbol{U}_A$ and the gradient $\nabla \boldsymbol{X}_A$ of its intermediate result $\boldsymbol{X}_A$. If $M > 1$, A can compute the gradient $\nabla \boldsymbol{W}_A$ of the fusion model $\boldsymbol{W}_A$ using its intermediate result $\boldsymbol{X}_A$ and the gradient of the prediction result $\nabla \boldsymbol{Z}$, i.e. $\nabla \boldsymbol{W}_A \leftarrow \boldsymbol{X}_A^T \nabla \boldsymbol{Z}$. Similarly, A can compute the gradient $\nabla \boldsymbol{X}_A$ of its intermediate result $\boldsymbol{X}_A$ using the gradient of the prediction result $\nabla \boldsymbol{Z}$ and the fusion model $\boldsymbol{W}_A$, i.e. $\nabla \boldsymbol{X}_A \leftarrow \nabla \boldsymbol{Z} \boldsymbol{W}_A^T$.

If $M = 1$, to protect the gradient $\nabla \boldsymbol{Z}$, A encrypts $\nabla \boldsymbol{Z}$ using the masking value $\boldsymbol{\Psi}_A$, and then sends $\nabla \boldsymbol{Z} + \boldsymbol{\Psi}_A$ and $\boldsymbol{\Psi}_A$ to P_1 and T respectively. After receiving $\nabla \boldsymbol{Z} + \boldsymbol{\Psi}_A$, P_1 uses the fusion model $\boldsymbol{V}_A$ to calculate the gradient of the intermediate result $\boldsymbol{X}_A^{\boldsymbol{V}}$, i.e., $\nabla \boldsymbol{X}_A^{\boldsymbol{V},*} \leftarrow (\nabla \boldsymbol{Z} + \boldsymbol{\Psi}_A) \boldsymbol{V}_A^T$. To protect $\boldsymbol{V}_A$, P_1 encrypts $\nabla \boldsymbol{X}_A^{\boldsymbol{V},*}$ using the masking value $\boldsymbol{\Psi}_1$. Subsequently, P_1 sends $\nabla \boldsymbol{X}_A^{\boldsymbol{V},*} + \boldsymbol{\Psi}_1$ and $(\boldsymbol{V}_A, \boldsymbol{\Psi}_1)$ to A and T. The part on P_1 is represented by the *PCalXGrad* function. T aggregates the masking value from P_1 and A using Alg. 2 and sends the result $\boldsymbol{\Theta}_T$ to A. Finally, A computes the gradient $\nabla \boldsymbol{X}_A$ of the intermediate result $\boldsymbol{X}_A$ using $\nabla \boldsymbol{Z}$, $\boldsymbol{U}_A$, $\nabla \boldsymbol{X}_A^{\boldsymbol{V},*} + \boldsymbol{\Psi}_1$ and $\boldsymbol{\Theta}_T$, i.e.:

$$\nabla \boldsymbol{X}_A = \nabla \boldsymbol{Z} \boldsymbol{U}_A^T + \nabla \boldsymbol{X}_A^{\boldsymbol{V},*} + \boldsymbol{\Psi}_1 - \boldsymbol{\Theta}_T.$$

The gradient $\nabla \boldsymbol{W}_A$ of the fusion model $\nabla \boldsymbol{U}_A$ is computed in the same way as in the $M > 1$ case. The pseudocode is shown in Alg. 5.

4.6. Passive node backward propagation

In the PBackward algorithm, P_i encrypts the fusion model $\boldsymbol{U}_i$ and sends it to A. A can then calculate the gradient of $\boldsymbol{X}_i$ using the fusion model $\boldsymbol{V}_i$ and the gradient $\nabla \boldsymbol{Z}$ of the prediction result $\boldsymbol{Z}$. The specific steps are as follows.

First, P_i encrypts the fusion model $\boldsymbol{U}_i$ and sends it to A, i.e., $[\![\boldsymbol{U}_i]\!]_i \leftarrow$ Pai.Enc($pk_i, \boldsymbol{U}_i$). Subsequently, A adds $[\![\boldsymbol{U}_i]\!]_i$ to the fusion model $\boldsymbol{V}_i$ to obtain the encrypted complete fusion model $[\![\boldsymbol{W}_i]\!]_i$, i.e., $[\![\boldsymbol{W}_i]\!]_i \leftarrow [\![\boldsymbol{U}_i]\!]_i + \boldsymbol{V}_i$. Then, A multiplies it by the gradient $\nabla \boldsymbol{Z}$ of the prediction result $\boldsymbol{Z}$ to get the encrypted gradient $[\![\nabla \boldsymbol{X}_i]\!]_i$ of the intermediate result, i.e., $[\![\nabla \boldsymbol{X}_i]\!]_i \leftarrow \nabla \boldsymbol{Z} [\![\boldsymbol{W}_i]\!]_i^T$. Lastly, A sends $[\![\nabla \boldsymbol{X}_i]\!]_i$ to P_i. The part in A is represented by the function *ACalGrad*.

Finally, P_i decrypts $[\![\nabla \boldsymbol{X}_i]\!]_i$ to get the gradient $\nabla \boldsymbol{X}_i$ of the intermediate result $\boldsymbol{X}_i$, i.e. $\nabla \boldsymbol{X}_i \leftarrow$ Pai.Dec($sk_i, [\![\nabla \boldsymbol{X}_i]\!]_i$). The same algorithm is used for both $M > 1$ and $M = 1$. The pseudocode is shown in Alg. 6.

4.7. PVFL-CPN

In Sections 4.3, 4.4, 4.5, and 4.6, we have detailed the four core algorithms of PVFL-CPN. Therefore, the pseudocodes for the complete training process and the complete PVFL-CPN algorithm are shown in Alg. 7 and Alg. 8.

5. Security analysis

In this section, some theoretical results related to the security analysis will be presented and discussed. To prove the security of our PVFL-CPN algorithm, let $\mathcal{P}$ be the set of passive nodes, and $\mathcal{N} = \mathcal{P} \cup \{A\}$ be the set of all nodes. Let the set of passive nodes' outputs be denoted as $\mathcal{O}_{\mathcal{P}}^{\boldsymbol{V}} = \{[\![\boldsymbol{O}_i^{\boldsymbol{V}}]\!]_A^*\}_{P_i \in \mathcal{P}}$ and $\mathcal{O}_{\mathcal{P}}^{\boldsymbol{U}} = \{\boldsymbol{O}_i^{\boldsymbol{U}}\}_{P_i \in \mathcal{P}}$. We denote a subset of passive nodes as $\mathcal{C} \subseteq \mathcal{P}$, and the union of this subset of passive nodes and the active node A as $\mathcal{C}^* = \mathcal{C} \cup \{A\}$. The view of a passive node P_i or the active node A includes its internal state and all the messages it receives.

We define a random variable $\mathsf{REAL}_{C^*}^{\mathcal{N},\kappa,F}(\mathcal{O}_{\mathcal{P}}^{\boldsymbol{V}}, \mathcal{O}_{\mathcal{P}}^{\boldsymbol{U}})$, where κ denotes the security parameter and F denotes forward propagation. This random variable represents the combined view of all nodes in the set C^*.

Lemma 1 (*Honest-but-Curious Security*). *There exists a probabilistic polynomial-time (PPT) simulator* SIM *such that for all $\mathcal{N}, \kappa, \mathcal{P}$ and $C^* \subseteq \mathcal{N}$, the output of the simulator* SIM *is computationally indistinguishable from* $\mathsf{REAL}_{C^*}^{\mathcal{N},\kappa,F}(\mathcal{O}_{\mathcal{P}}^{\boldsymbol{V}}, \mathcal{O}_{\mathcal{P}}^{\boldsymbol{U}})$, *i.e.:*

$$\mathsf{REAL}_{C^*}^{\mathcal{N},\kappa,F}(\mathcal{O}_{\mathcal{P}}^{\boldsymbol{V}}, \mathcal{O}_{\mathcal{P}}^{\boldsymbol{U}}) \approx_c \mathsf{SIM}_{C^*}^{\mathcal{N},\kappa,F}(\mathcal{O}_{\mathcal{C}}^{\boldsymbol{V}}, \mathcal{O}_{\mathcal{C}}^{\boldsymbol{U}}, \mathcal{O}_{\mathcal{P}\backslash\mathcal{C}}^{\boldsymbol{V}}, \mathcal{O}_{\mathcal{P}\backslash\mathcal{C}}^{\boldsymbol{U}})$$

Proof. A standard hybrid argument method is adopted, where the simulator SIM is constructed by gradually modifying the random variable REAL in a sequence of steps. We prove that any two consecutive modifications result in computationally indistinguishable distributions, denoted as Hyb-i.

Hyb-0. In this hybrid, the distribution of the random variable is identical to that of REAL, which represents the combined view of C^* in the real execution.

Hyb-1. In this hybrid, for each honest node $P \in \mathcal{P}\backslash C$, when invoking Alg. 1, it does not use $\boldsymbol{\phi}_i$ to convert $[\![\boldsymbol{X}_i\boldsymbol{V}_i]\!]_A$ into a SS variable, but instead uses a uniformly random variable $\boldsymbol{\phi}_i^\star$. Since $\boldsymbol{\phi}_i^\star$ and $\boldsymbol{\phi}_i$ are independent and identically distributed, $[\![\boldsymbol{X}_i\boldsymbol{V}_i - \boldsymbol{\phi}_i]\!]_A$ and $[\![\boldsymbol{X}_i\boldsymbol{V}_i - \boldsymbol{\phi}_i^\star]\!]_A$ are also independent and identically distributed. Therefore, this hybrid is computationally indistinguishable from the previous one.

Hyb-2. In this hybrid, for each honest node $P \in \mathcal{P}\backslash C$, it does not use $\boldsymbol{\Phi}_i$ to encrypt $[\![\boldsymbol{O}_i^{\boldsymbol{V}}]\!]_A$, but instead uses a uniform random variable $\boldsymbol{\Phi}_i^\star$. Since $\boldsymbol{\Phi}_i^\star$ and $\boldsymbol{\Phi}_i$ are independent and identically distributed, the encrypted output $[\![\boldsymbol{O}_i^{\boldsymbol{V}}]\!]_A^\star \leftarrow [\![\boldsymbol{O}_i^{\boldsymbol{V}}]\!]_A + \boldsymbol{\Phi}_i^\star$ is also independent and identically distributed with $[\![\boldsymbol{O}_i^{\boldsymbol{V}}]\!]_A^*$. Therefore, this hybrid is computationally indistinguishable from the previous one.

Hyb-3. In this hybrid, for each honest node $P \in \mathcal{P}\backslash C$, it does not use $\boldsymbol{\phi}_i$ to encrypt $\boldsymbol{X}_i\boldsymbol{U}_i$, but instead uses a uniform random variable $\boldsymbol{\phi}_i^\star$. Since $\boldsymbol{\phi}_i^\star$ and $\boldsymbol{\phi}_i$ are independent and identically distributed, $\boldsymbol{O}_i^{\boldsymbol{U},*}$ and $\boldsymbol{O}_i^{\boldsymbol{U},\star} \leftarrow \boldsymbol{X}_i\boldsymbol{U}_i + \boldsymbol{\phi}_i^\star$ are also independent and identically distributed. Therefore, this hybrid is computationally indistinguishable from the previous one. When $M = 1$, the simulation process is similar to this hybrid, so we will not elaborate further.

Therefore, we can define a PPT simulator SIM that samples from the distribution described in the previous hybrid. The above arguments demonstrate that the output of SIM is computationally indistinguishable from the output of REAL, thus completing the proof.

In backward propagation, we define the set of gradients output by the active node A as $\mathcal{G} = \{\nabla\boldsymbol{V}_i - \boldsymbol{\psi}_i\}_{i=1}^M$, and its subset as $\mathcal{G}_{\mathcal{C}} = \{\nabla\boldsymbol{V}_i - \boldsymbol{\psi}_i\}_{P_i \in \mathcal{C}}$. Then, we define a random variable $\mathsf{REAL}_{C^*}^{\mathcal{N},\kappa,B}(\mathcal{G}, \nabla\boldsymbol{W}_A, \nabla\boldsymbol{X}_A)$, where B denotes the backward propagation. This random variable represents the combined view of all nodes in the set $C^* \subseteq \mathcal{N}$.

Lemma 2 (*Honest-but-Curious Security*). *There exists a probabilistic polynomial-time simulator* SIM *such that for all $\mathcal{N}, \kappa, \mathcal{P}$ and $C^* \subseteq \mathcal{N}$, the output of the simulator* SIM *is computationally indistinguishable from* $\mathsf{REAL}_{C^*}^{\mathcal{N},\kappa,B}(\mathcal{G}, \nabla\boldsymbol{W}_A, \nabla\boldsymbol{X}_A)$, *i.e.:*

$$\mathsf{REAL}_{C^*}^{\mathcal{N},\kappa,B}(\mathcal{G}, \nabla\boldsymbol{W}_A, \nabla\boldsymbol{X}_A) \approx_c \mathsf{SIM}_{C^*}^{\mathcal{N},\kappa,B}(\mathcal{G}_{\mathcal{C}}, \nabla\boldsymbol{W}_A, \nabla\boldsymbol{X}_A, \mathcal{G}_{\mathcal{P}\backslash\mathcal{C}})$$

Proof. A standard hybrid argument method is adopted, where the simulator SIM is constructed by gradually modifying the random variable REAL in a sequence of steps. We prove that any two consecutive modifications result in computationally indistinguishable distributions, denoted as Hyb-i.

Hyb-0. In this hybrid, the distribution of the random variable is identical to that of REAL, which represents the combined view of C^* in the real execution.

Hyb-1. In this hybrid, when encrypting $\nabla\boldsymbol{Z}$, A does not use pk_A, but instead uses a randomly-chosen $pk_A^\star$ by the simulator, resulting in $[\![\nabla\boldsymbol{Z}]\!]_A^\star \leftarrow \text{Pai.Enc}(pk_A^\star, \nabla\boldsymbol{Z})$. The security of the homomorphic encryption algorithm guarantees that this hybrid is computationally indistinguishable from the previous one.

Hyb-2. In this hybrid, for each honest node $P_i \in \mathcal{P}\backslash C$, when invoking Alg. 1 to transform $[\![\nabla\boldsymbol{V}_i]\!]_A$ into a SS variable, it does not use $\boldsymbol{\psi}_i$, but instead uses a uniformly random variable $\boldsymbol{\psi}_i^\star$. Since $\boldsymbol{\psi}_i^\star$ and $\boldsymbol{\psi}_i$ are independent and identically distributed, $[\![\nabla\boldsymbol{V}_i - \boldsymbol{\psi}_i^\star]\!]_A$ and $[\![\nabla\boldsymbol{V}_i - \boldsymbol{\psi}_i]\!]_A$ are also independent and identically distributed. Therefore, this hybrid is computationally indistinguishable from the previous one.

Hyb-3. In this hybrid, the honest node A (in this case $M = 1$, and the passive node P_1 is an honest-but-curious node) does not use $\boldsymbol{\Psi}_A$ when encrypting $\nabla\boldsymbol{Z}$, but instead uses a uniformly random variable $\boldsymbol{\Psi}_A^\star$. Since $\boldsymbol{\Psi}_A^\star$ and $\boldsymbol{\Psi}_A$ are independent and identically distributed, $\nabla\boldsymbol{Z} + \boldsymbol{\Psi}_A^\star$ and $\nabla\boldsymbol{Z} + \boldsymbol{\Psi}_A$ are also independent and identically distributed. Therefore, this hybrid is computationally indistinguishable from the previous one.

Hyb-4. In this hybrid, the honest node P_1 (in this case $M = 1$, and the active node A is an honest but cautious node) does not use $\boldsymbol{\Psi}_1$ when encrypting $\nabla\boldsymbol{X}_A^{\boldsymbol{V},*}$, but instead uses a uniformly random variable $\boldsymbol{\Psi}_1^\star$. Since $\boldsymbol{\Psi}_1$ and $\boldsymbol{\Psi}_1^\star$ are independent and identically distributed, $\nabla\boldsymbol{X}_A^{\boldsymbol{V},*} + \boldsymbol{\Psi}_1^\star$ and $\nabla\boldsymbol{X}_A^{\boldsymbol{V},*} + \boldsymbol{\Psi}_1$ are also independent and identically distributed. Therefore, this hybrid is computationally indistinguishable from the previous one.

Therefore, we can define a PPT simulator SIM that samples from the distribution described in the previous hybrid. The above arguments demonstrate that the output of SIM is computationally indistinguishable from the output of REAL, thus completing the proof.

Lemma 3. *Suppose $M > 1$ and the active node A knows $\boldsymbol{Z}$ and $\nabla\boldsymbol{Z}$. Then, A cannot infer $\boldsymbol{X}_i, \boldsymbol{W}_i$, and $\boldsymbol{X}_i\boldsymbol{W}_i$.*

Proof. Let us consider the equation

$$\boldsymbol{Z}_i = \sum_{i=1}^{M} \boldsymbol{O}_i^{\boldsymbol{V}} + \boldsymbol{O}_i^{\boldsymbol{U}} = \sum_{i=1}^{M} \boldsymbol{X}_i\boldsymbol{W}_i + \boldsymbol{\Phi}_i.$$

Since the active node knows $\boldsymbol{Z}_i$ and $\boldsymbol{\Gamma}_T = \sum_{i=1}^M \boldsymbol{\Phi}_i$, but does not know any individual $\boldsymbol{\Phi}_i$, it cannot infer $\boldsymbol{X}_i\boldsymbol{W}_i$ and consequently cannot deduce $\boldsymbol{X}_i$ or $\boldsymbol{W}_i$.

During backward propagation, let us consider the equation $\nabla\boldsymbol{W}_i = \nabla\boldsymbol{V}_i - \boldsymbol{\psi}_i$. Since A does not know $\boldsymbol{\psi}_i$, it cannot infer $\nabla\boldsymbol{W}_i$. Therefore,

even if the active node A knows $\boldsymbol{Z}$ and $\nabla\boldsymbol{Z}$, it cannot infer $\boldsymbol{X}_i$, $\boldsymbol{W}_i$, and $\boldsymbol{X}_i\boldsymbol{W}_i$.

Lemma 4. *Suppose $M = 1$ and the active node A knows $\boldsymbol{Z}$ and $\nabla\boldsymbol{Z}$. Then, A cannot infer $\boldsymbol{X}_1$, $\boldsymbol{W}_1$, and $\boldsymbol{X}_1\boldsymbol{W}_1$.*

Proof. Consider the equation

$$\boldsymbol{Z} = \boldsymbol{X}_A\boldsymbol{W}_A + \boldsymbol{X}_1\boldsymbol{W}_1 + \boldsymbol{\Phi}_1 + \boldsymbol{\Phi}_A\boldsymbol{V}_A - \boldsymbol{\Gamma}_T.$$

While active node A knows both $\boldsymbol{Z}$ and $\boldsymbol{\Gamma}_T$, it cannot isolate $\boldsymbol{\Phi}_1$ and $\boldsymbol{\Phi}_A\boldsymbol{V}_A$ from $\boldsymbol{\Gamma}_T$. Since $\boldsymbol{\Phi}_T$ and $\boldsymbol{Z}$ have the same dimension (matrix addition cannot be performed if they do not have the same dimension), let $\boldsymbol{\Phi}_T \in \mathbb{R}^{BS\times classes}$, where BS represents the batch size and $classes$ represents the number of classes. Let $\boldsymbol{X}_A \in \mathbb{R}^{BS\times in}$, then $\boldsymbol{\Phi}_A \in \mathbb{R}^{BS\times in}$. Then let $\boldsymbol{V}_A \in \mathbb{R}^{in\times classes}$ and $\boldsymbol{\Phi}_1 \in \mathbb{R}^{BS\times classes}$. For node A, both $\boldsymbol{V}_A$ and $\boldsymbol{\Phi}_1$ are unknown, which means there are $(BS+in)\times classes$ unknowns. Since the result matrix $\boldsymbol{\Gamma}_T \in \mathbb{R}^{BS\times classes}$, we can list $BS \times classes$ equations to form an equation system. However, the equation system has $(BS + in) \times classes$ unknowns, so it must have infinitely many solutions. Therefore, the active node A cannot uniquely determine $\boldsymbol{\Phi}_1$ and $\boldsymbol{V}_A$. Similarly, when calculating the gradient of $\nabla\boldsymbol{X}_A$, the active node cannot isolate $\boldsymbol{\Psi}_A\boldsymbol{V}_A^T$ and $\boldsymbol{\Psi}_1$.

6. Performance and evaluation experiments

6.1. Dataset and model settings

We conducted our performance evaluation experiments on five datasets: MNIST, a9a, w8a, cifar-10, and EMNIST. During the experiments, each node receives all the data, but the features of the received data are different. Only the active node has labels, while the passive nodes do not.

The MNIST dataset consists of a training set of 60,000 samples and a test set of 10,000 samples. Each image in the dataset is a single-channel image of size 28 × 28 pixels, and there are a total of 10 classes.

The a9a dataset consists of a training set of 32,561 samples and a test set of 16,281 samples. Each sample in the dataset contains 123 features, of which 14 are valid features, and there are a total of 2 classes.

The w8a dataset consists of a training set of 49,749 samples and a test set of 14,951 samples. Each sample in the dataset contains 300 features, of which 12 are valid features, and there are a total of 2 classes.

The cifar-10 dataset consists of a training set of 50,000 samples and a test set of 10,000 samples. Each sample in the dataset is a 3-channel image of size 32 × 32 pixels, and there are a total of 10 classes.

The EMNIST dataset consists of a training set of 697,932 samples and a test set of 116,323 samples. Each sample in the dataset is a single-channel image of size 28 × 28 pixels, and there are a total of 62 classes.

In this experiment, we used a total of three models: logistic regression (LR), multilayer perceptron (MLP), and Convolutional Neural Networks (CNN). The specific model parameters are shown in Table 2, where "numC" represents the number of classes, "fc" represents a fully connected layer, and x represents the output dimension of the fully connected layer. The four parameters in the convolutional layer represent the output channels, kernel size, padding, and stride, respectively. "MP" represents a max-pooling layer.

Table 2
The model settings.

Model	Parameters
LR	numC-d fc
MLP-x	x-d fc → x-d fc → numC-d fc
CNN	$(32, 5\times5, 2, 1)$ conv → MP → $(64, 5\times5, 2, 1)$ conv → MP → 256-d fc → numC-d fc

6.2. Baseline

In this work, three state-of-the-art methods were selected, namely the Naïve VFL [28], BlindFL [16] and AdaVFed [19], as our comparison methods.

Naïve VFL: Naïve VFL is a VFL method without privacy preservation. In this method, each passive node extracts the features of the local data from the local model, which is the intermediate result. Then, the passive nodes send the intermediate results to the active node, which aggregates the intermediate results and outputs the prediction results through the global module. Subsequently, the active node calculates the loss of the total output using the loss function and backpropagates to calculate the gradient of the global module. At the same time, it backpropagates to calculate the gradients of the intermediate results of all passive nodes and itself. Finally, the active node sends the gradients of the intermediate results of the passive nodes to them, and all nodes update the model according to the gradients of the intermediate results. This method is mainly used as our benchmark when comparing accuracy.

BlindFL: BlindFL is a vertical federated learning privacy-preserving algorithm that utilizes homomorphic encryption and secret sharing to aggregate federated source layers. Firstly, the passive node takes the intermediate result as input and outputs the encrypted result on the encrypted part of the global module encrypted by the active node, and then sends it to the active node. At the same time, the active node takes the intermediate result as input and outputs the encrypted result on the encrypted part of the global module encrypted by the passive node, and then sends it to the passive node. The passive node decrypts the output sent by the active node to itself, adds it to the output of its intermediate result in the other part of the global module, and sends it to the active node. The active node also performs the same operation as the passive node. In this way, each node can complete the aggregation of intermediate results and the subsequent backward propagation process without leaking its privacy. However, this method will encounter a large computational and communication bottleneck, because there are a large number of homomorphic encryption operations in it, and all the information sent is encrypted. Therefore, when the total number of nodes increases, it will cause a communication bottleneck on the active node side.

AdaVFed: This scheme provides a solution to the low-accuracy issue in traditional vertical federated learning, where model accuracy drops significantly due to gradient clipping and noise injection operations. First, AdaVFed introduces a regularization term into the objective function (loss function) of vertical federated learning. This term actively constrains the L_2 norm of the intermediate results within the clipping threshold. Through this adaptive constraint, the vast majority of data information is preserved intact during subsequent clipping, thereby enhancing the robustness of the clipping operation. Second, AdaVFed utilizes a stochastic gate technique at the input layer of the network to screen and identify the most essential and core features, reducing the loss of critical information from the source. Finally, it employs Laplacian scores to evaluate the importance of different dimensions (features) of the intermediate results, and dynamically allocates the privacy budget based on these importance levels—allocating more budget to important dimensions and injecting more noise into minor dimensions, thus achieving adaptive differential privacy. However, the performance of this method degrades sharply as the number of nodes increases, making it unsuitable for multi-node environments.

6.3. Experiments setup

We conducted a total of 6 experiments to evaluate the performance of our method PVFL-CPN. Specifically, the experimental setup is shown in Table 3.

Table 3
Experimental settings.

Exp	Model	Dataset	Nodes Num	LR	BS
LI/Perf	CNN	MNIST	2/4	0.001	32
	MLP-128	MNIST	2/4	0.001	32
	LR	a9a	2/4	0.05	64
	LR	w8a	2/4	0.05	64
	CNN	cifar-10	2/4	0.01	32
	CNN	EMNIST	2/4	0.001	64
GI	MLP-128	MNIST	2/4	0.001	32
	CNN	cifar-10	2/4	0.01	32
	LR	a9a	2/4	0.05	64
MP	MLP-128	MNIST	4	0.001	32
Comp	MLP-128	MNIST	10–100	0.001	32
Comm	MLP-128	MNIST	10–100	0.001	32

Table 4
Experiment environment settings.

CPU	Intel(R) Xeon(R) Silver 4214R CPU @ 2.40 GHz
Number of CPU cores	48
RAM	64G
Storage	1T
System	Ubuntu 20.04
gcc/g++	Version 9.3.0
OpenMP	Version 4.5
NTL	Version 11.5.1
GMP	Version 6.2.0
PyTorch	Version 1.12.1

Table 5
Data partitioning settings.

Dataset	Partition size
MNIST	(16, 16)
cifar-10	(20, 20)
a9a	70
w8a	240
EMNIST	(16, 16)

Label Inference (LI). This experiment evaluates the accuracy of the passive node's prediction of the active node's label when $\boldsymbol{U}_P$ and $\boldsymbol{W}_P$ are known. This experiment reflects the extent to which our method protects the active node's label.

Gradient Inference (GI). This experiment evaluates the accuracy of the passive node's prediction of the active node's label when $\nabla \boldsymbol{W}_P$ is known. This experiment reflects the extent to which the passive node can infer the active node's label if the active node leaks the gradient to the passive node.

Model Protection (MP). This experiment evaluates the protection of the global module by comparing the passive node $\boldsymbol{U}_i$ and $\boldsymbol{W}_i$.

Performance Evaluation (Perf). This experiment evaluates the performance of our method compared to SOTA methods on different datasets, showing whether our method is inferior to other methods in terms of performance.

Computational Efficiency Evaluation (Comp). This experiment evaluates the computational time consumed by our method and SOTA methods per batch, showing whether our method outperforms SOTA methods in terms of computational performance.

Communication Efficiency Evaluation (Comm). This experiment evaluates the amount of data that needs to be transmitted between the passive node and the active node for our method and SOTA methods, showing whether our method outperforms SOTA methods in terms of communication performance.

6.4. Implementation details

We simulated all passive nodes and one active node on a single server, and the server configuration is shown in Table 4. We set a key length $\kappa = 2048$. In order to accelerate the encryption and calculation of matrices, we used the OpenMP library to perform parallel computing. When implementing Paillier, we use the NTL number theory and the GMP high-precision computing libraries. In the implementation of VFL, we divided the feature vectors according to Table 5, and this partition ensured that each node possesses a portion of the original features, with some overlap between features held by different nodes. In VFL, PyTorch was utilized to implement all the training and testing processes. In the experiments for AdaVFed, we uniformly set the batch size to 128 and the learning rate to 0.005, while the remaining settings are identical to our experiments.

6.5. Experimental results analysis

In our experiments, we used two evaluation metrics: accuracy and area under the ROC curve (AUC). Accuracy is calculated as (6):

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}. \tag{6}$$

where TP, TN, FP, and FN represent True Positives, True Negatives, False Positives, and False Negatives, respectively.

Accuracy reflects the percentage of correctly predicted results among all samples. Two other important metrics are the true (TPR) and false positive (FPR) rates, defined in (7).

$$TPR = \frac{TP}{TP + FN}, \qquad FPR = \frac{FP}{TN + FP}. \tag{7}$$

By plotting FPR on the x-axis and TPR on the y-axis, we obtain the ROC curve. The area under the ROC curve (AUC) measures the performance of the model, with a larger AUC representing better performance.

We employed Accuracy as the evaluation metric for the MNIST, cifar-10, and EMNIST datasets, while AUC was used for the a9a and w8a datasets. Subsequently, we introduced three statistical metrics: mean, standard deviation, and confidence intervals. Specifically, at the end of each training epoch, we performed bootstrap sampling (random sampling with replacement) on the test set, with the sample size equal to the original test set size. This sampled set was used for evaluation to obtain an Accuracy or AUC value. We repeated this process 1000 times to generate a distribution of 1000 Accuracy/AUC scores. Finally, we calculated the mean and standard deviation of these 1000 values, and determined the 95% confidence interval by taking the 2.5%-th and 97.5%-th percentiles.

6.5.1. Label inference results analysis

We conducted label inference experiments across five datasets, with the results presented in Fig. 5 and Table 6. The **baseline** refers to the performance of the model on the test set when all nodes participate in training, **w/ ModelSS** refers to the performance of a passive node predicting the labels of the active node on the test set using its partial global module $\boldsymbol{U}_i$. This experiment verifies whether a semi-honest passive node can predict the active node's labels when only the partial global module $\boldsymbol{U}_i$ is known. **w/o ModelSS** refers to the performance of a passive node predicting the labels of the active node on the test set if it knows its global module $\boldsymbol{W}_i$. This experiment confirms that the accuracy of predicting the active node's labels improves once a semi-honest passive node obtains the complete global module, thereby demonstrating the necessity of splitting the global module in our method. As we can see from Fig. 5 and Table 6, the accuracy of predicting the labels of the active node is significantly improved when the passive node knows $\boldsymbol{W}_i$, particularly in experiments where the number of passive nodes $M = 1$. This is because when calculating the gradient of the global module $\boldsymbol{W}_i$, the intermediate results aggregated from only one passive node and one active node are used for gradient

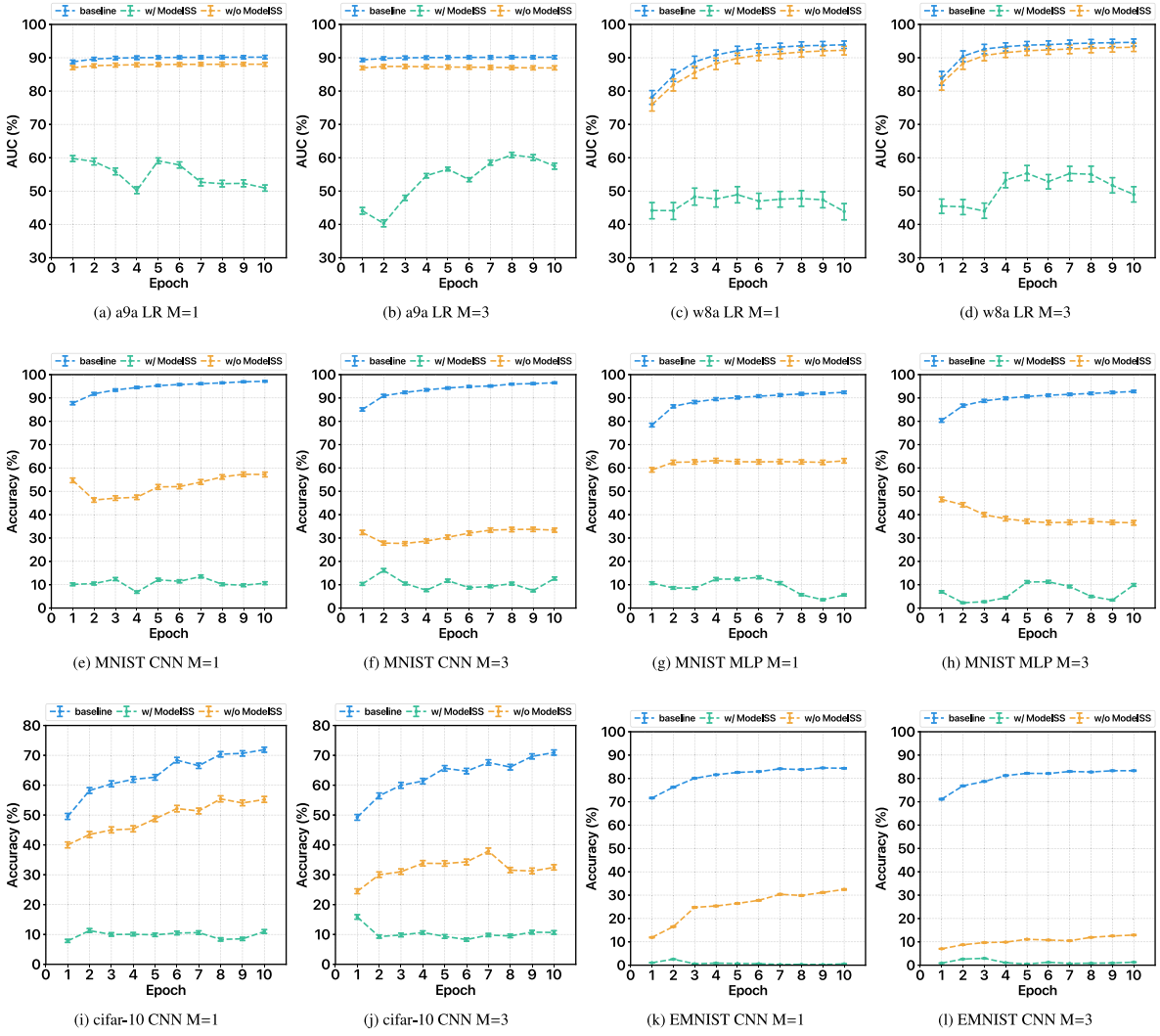


Fig. 5. Label inference results.

computation. In other words, this passive node's "contribution" to the gradient is substantial. Therefore, once the passive node knows $\boldsymbol{W}_i$, it can infer the active node's labels with high accuracy. As the number of passive nodes M increases, the label inference accuracy of a single passive node decreases. However, this does not imply that $\boldsymbol{W}_i$ does not require protection. Through the $M = 3$ experiments, we found that although the inference accuracy decreases, it remains higher than the probability of random guessing and gradually increases as the number of training epochs grows. Contrary to the above, if the passive node only possesses $\boldsymbol{U}_i$, its accuracy in predicting the labels of the active node is approximately equal to $1/classes$, which is the reciprocal of the number of classes. However, such prediction performance is equivalent to random guessing. Therefore, these results demonstrate that our method can effectively protect the labels of the active node.

6.5.2. Gradient inference results analysis

We tested the accuracy of passive nodes in predicting active node labels when they know $\nabla \boldsymbol{W}_i$ on three datasets using three models. Specifically, we construct a model $\widehat{\boldsymbol{W}}_i$ at the passive node that is identical to the global module, update $\widehat{\boldsymbol{W}}_i$ using $\nabla \boldsymbol{W}_i$, and test its accuracy in predicting the active node's labels, as shown in Fig. 6 and Table 7. Here **baseline** refers to the baseline accuracy of the method. **w/ GradSS** indicates the passive node's performance in inferring active node labels through existing information without knowing $\nabla \boldsymbol{W}_i$. **w/o GradSS** represents the passive node's performance in inferring active node labels by maintaining a global module when $\nabla \boldsymbol{W}_i$ is known. The experimental results indicate that when the active node leaks $\nabla \boldsymbol{W}_i$ to the passive node, the accuracy of the passive node in predicting the active node label increases significantly. From the results on the MNIST dataset, we observe that the label inference accuracy increases as the number of nodes grows, which differs slightly from the results on the cifar-10 and a9a datasets. A possible reason is that the global module is initialized randomly for each experiment; although they follow the same distribution, this may affect label prediction after updating $\widehat{\boldsymbol{W}}_i$ with $\nabla \boldsymbol{W}_i$. Additionally, the learning rate for the MNIST dataset is lower, which may cause $\widehat{\boldsymbol{W}}_i$ to converge faster when the number of passive nodes increases (as the gradients contain more information). In summary, once a passive node knows $\nabla \boldsymbol{W}_i$, it can infer the active node's labels to a certain extent. Through the **w/ GradSS** experimental results, we can see that if $\nabla \boldsymbol{W}_i$ is protected, the passive node will not be able to infer the active node label.

6.5.3. Model protection result analysis

Fig. 7 illustrates the range of all parameter values covered by the global module $\boldsymbol{W}_i$ and the partial global module $\boldsymbol{U}_i$. The blue regions represent the distribution of $\boldsymbol{W}_i$, while the green regions denote the

Table 6
Accuracy/AUC (%) of the label inference experiments.

Experiment	Method	Epoch				Experiment	Method	Epoch			
		1	4	7	10			1	4	7	10
a9a LR M=1	baseline	88.76 ± 0.27	90 ± 0.26	90.13 ± 0.25	90.18 ± 0.26	a9a LR M=3	baseline	89.35 ± 0.26	90.05 ± 0.25	90.13 ± 0.27	90.15 ± 0.25
	w/ ModelSS	59.79 ± 0.46	50.2 ± 0.5	52.66 ± 0.54	50.92 ± 0.5		w/ ModelSS	44.09 ± 0.51	54.63 ± 0.34	58.53 ± 0.4	57.49 ± 0.46
	w/o ModelSS	87.03 ± 0.3	87.9 ± 0.3	88.03 ± 0.28	88.05 ± 0.3		w/o ModelSS	86.96 ± 0.29	87.29 ± 0.29	87.12 ± 0.31	86.94 ± 0.3
w8a LR M=1	baseline	78.18 ± 1.07	90.83 ± 0.79	93.2 ± 0.66	93.89 ± 0.57	w8a LR M=3	baseline	83.79 ± 1.05	93.31 ± 0.62	94.21 ± 0.59	94.63 ± 0.56
	w/ ModelSS	44.18 ± 1.25	47.64 ± 1.29	47.57 ± 1.22	43.88 ± 1.29		w/ ModelSS	45.5 ± 1.1	53.28 ± 1.14	55.28 ± 1.14	48.98 ± 1.18
	w/o ModelSS	75.99 ± 1.04	88.25 ± 0.88	91.19 ± 0.76	92.25 ± 0.68		w/o ModelSS	82.33 ± 1.03	91.55 ± 0.7	92.62 ± 0.68	93.17 ± 0.63
MNIST CNN M=1	baseline	87.75 ± 0.34	94.48 ± 0.23	96.1 ± 0.19	97.13 ± 0.16	MNIST CNN M=3	baseline	85.05 ± 0.34	93.46 ± 0.25	95.11 ± 0.22	96.49 ± 0.18
	w/ ModelSS	10.14 ± 0.3	6.81 ± 0.26	13.52 ± 0.34	10.63 ± 0.3		w/ ModelSS	10.31 ± 0.31	7.62 ± 0.26	9.27 ± 0.29	12.65 ± 0.33
	w/o ModelSS	54.69 ± 0.52	47.43 ± 0.48	53.98 ± 0.52	57.21 ± 0.51		w/o ModelSS	32.32 ± 0.46	28.68 ± 0.45	33.38 ± 0.47	33.36 ± 0.46
MNIST MLP M=1	baseline	78.41 ± 0.4	89.48 ± 0.31	91.26 ± 0.29	92.39 ± 0.26	MNIST MLP M=3	baseline	80.35 ± 0.39	89.86 ± 0.3	91.53 ± 0.28	92.77 ± 0.26
	w/ ModelSS	10.65 ± 0.3	12.38 ± 0.34	10.7 ± 0.31	5.59 ± 0.22		w/ ModelSS	6.92 ± 0.25	4.4 ± 0.21	9.22 ± 0.29	9.9 ± 0.3
	w/o ModelSS	59.18 ± 0.49	63.16 ± 0.49	62.72 ± 0.49	63.05 ± 0.48		w/o ModelSS	46.56 ± 0.5	38.26 ± 0.48	36.73 ± 0.49	36.53 ± 0.48
cifar-10 CNN M=1	baseline	49.51 ± 0.52	61.96 ± 0.48	66.52 ± 0.45	71.88 ± 0.44	cifar-10 CNN M=3	baseline	49.26 ± 0.49	61.38 ± 0.49	67.6 ± 0.46	70.92 ± 0.45
	w/ ModelSS	7.84 ± 0.26	10.06 ± 0.3	10.61 ± 0.31	10.97 ± 0.32		w/ ModelSS	15.84 ± 0.37	10.66 ± 0.3	9.8 ± 0.28	10.66 ± 0.3
	w/o ModelSS	39.98 ± 0.48	45.36 ± 0.51	51.36 ± 0.49	55.2 ± 0.5		w/o ModelSS	24.44 ± 0.43	33.86 ± 0.47	37.92 ± 0.49	32.46 ± 0.47
EMNIST CNN M=1	baseline	71.61 ± 0.13	81.58 ± 0.11	84.14 ± 0.1	84.35 ± 0.1	EMNIST CNN M=3	baseline	71.1 ± 0.13	81.21 ± 0.12	82.98 ± 0.11	83.31 ± 0.11
	w/ ModelSS	1.05 ± 0.03	0.88 ± 0.03	0.31 ± 0.02	0.53 ± 0.02		w/ ModelSS	0.87 ± 0.03	1.07 ± 0.03	0.74 ± 0.02	1.28 ± 0.03
	w/o ModelSS	11.91 ± 0.1	25.36 ± 0.12	30.35 ± 0.14	32.45 ± 0.13		w/o ModelSS	7.01 ± 0.08	9.87 ± 0.09	10.48 ± 0.09	12.9 ± 0.1

Table 7
Accuracy/AUC (%) of the gradient inference experiments.

Experiment	Method	Epoch				Experiment	Method	Epoch			
		1	4	7	10			1	4	7	10
a9a LR M=1	baseline	88.76 ± 0.27	90 ± 0.26	90.13 ± 0.25	90.18 ± 0.26	a9a LR M=3	baseline	89.35 ± 0.26	90.05 ± 0.25	90.13 ± 0.27	90.15 ± 0.25
	w/ GradSS	59.79 ± 0.46	50.2 ± 0.5	52.66 ± 0.54	50.92 ± 0.5		w/ GradSS	44.09 ± 0.51	54.63 ± 0.34	58.53 ± 0.4	57.49 ± 0.46
	w/o GradSS	86.09 ± 0.32	87.55 ± 0.3	87.8 ± 0.29	87.82 ± 0.28		w/o GradSS	86.03 ± 0.31	86.21 ± 0.32	85.99 ± 0.32	85.86 ± 0.32
cifar-10 CNN M=1	baseline	49.51 ± 0.52	61.96 ± 0.48	66.52 ± 0.45	71.88 ± 0.44	cifar-10 CNN M=3	baseline	49.26 ± 0.49	61.38 ± 0.49	67.6 ± 0.46	70.92 ± 0.45
	w/ GradSS	7.84 ± 0.26	10.06 ± 0.3	10.61 ± 0.31	10.97 ± 0.32		w/ GradSS	15.84 ± 0.37	10.66 ± 0.3	9.8 ± 0.28	10.66 ± 0.3
	w/o GradSS	26.64 ± 0.43	45.28 ± 0.48	52.76 ± 0.5	55.58 ± 0.49		w/o GradSS	22.02 ± 0.42	24.88 ± 0.44	25.84 ± 0.44	25.33 ± 0.42
MNIST MLP M=1	baseline	78.41 ± 0.4	89.48 ± 0.31	91.26 ± 0.29	92.39 ± 0.26	MNIST MLP M=3	baseline	80.35 ± 0.39	89.86 ± 0.3	91.53 ± 0.28	92.77 ± 0.26
	w/ GradSS	10.65 ± 0.3	12.38 ± 0.34	10.7 ± 0.31	5.59 ± 0.22		w/ GradSS	6.92 ± 0.25	4.4 ± 0.21	9.22 ± 0.29	9.9 ± 0.3
	w/o GradSS	13.27 ± 0.33	27.3 ± 0.46	27.04 ± 0.44	26.46 ± 0.46		w/o GradSS	31.6 ± 0.46	27.78 ± 0.46	30.46 ± 0.44	30.93 ± 0.45

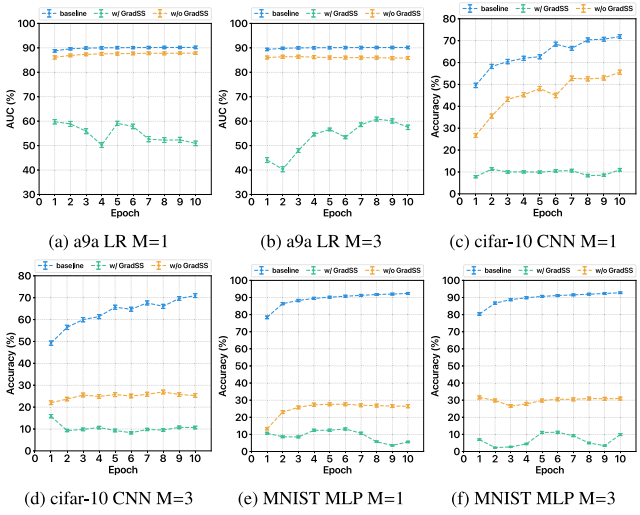
distribution of $\boldsymbol{U}_i$. It can be seen from the figure that our method divides the global module into two parts, $\boldsymbol{U}_i$ and $\boldsymbol{V}_i$, and the $\boldsymbol{U}_i$ owned by the passive node has a large difference with $\boldsymbol{W}_i$, so the passive node cannot rely on $\boldsymbol{U}_i$ to infer the label of the active node. Therefore, our method can effectively protect the global module of each passive node.

6.5.4. Performance analysis

Fig. 8 and Table 8 present the experimental results comparing our method with other SOTA methods. From the results, we observe that our method consistently ranks first or second in terms of Accuracy or AUC for both $M = 1$ and $M = 3$ configurations. This performance is close to that of NaïveVFL and superior to the existing BlindFL and AdaVFed methods. First, regarding BlindFL, its performance limitations likely stem from the fact that it performs operations directly between original feature vectors and encrypted model parameters. This process necessitates the quantization of original features, which may lead to the loss of subtle feature information. For $M = 3$, we observe an improvement in BlindFL's performance; a possible reason is that while quantization loses some information, the overlapping features across multiple nodes allow the model to learn more comprehensive information, thereby mitigating the performance degradation caused by quantization loss. In contrast, our method quantifies intermediate results (a portion of the predicted values) rather than the raw inputs, thus preserving the integrity of the original features. This is reflected in our experimental results, where our method outperforms BlindFL in both $M = 1$ and $M = 3$ scenarios. Regarding the AdaVFed method, we find that its performance is relatively high when $M = 1$ but declines when $M = 3$. This is because their approach introduces noise to protect privacy; as M increases, the cumulative noise injected during

Table 8
Accuracy/AUC (%) of performance comparison experiments (Bold and *italic* values denote the best and second-best Accuracy/AUC, respectively).

Experiment	Method	Epoch				Experiment	Method	Epoch			
		1	4	7	10			1	4	7	10
a9a LR M=1	NaïveVFL	88.7 ± 0.28	90 ± 0.25	**90.14 ± 0.25**	90.19 ± 0.24	a9a LR M=3	NaïveVFL	**89.58 ± 0.25**	**90.1 ± 0.25**	**90.16 ± 0.25**	**90.17 ± 0.25**
	BlindFL	84.94 ± 0.34	89 ± 0.27	89.59 ± 0.25	89.82 ± 0.26		BlindFL	88.92 ± 0.27	89.88 ± 0.26	90.02 ± 0.25	90.07 ± 0.25
	AdaVFed	83.88 ± 0.33	84.17 ± 0.33	84.43 ± 0.3	84.65 ± 0.32		AdaVFed	53.5 ± 0.49	53.8 ± 0.47	54.06 ± 0.5	54.36 ± 0.49
	Ours	**88.76 ± 0.27**	**90 ± 0.26**	90.13 ± 0.25	**90.18 ± 0.26**		Ours	89.35 ± 0.26	90.05 ± 0.25	90.13 ± 0.27	90.15 ± 0.25
w8a LR M=1	NaïveVFL	76.52 ± 1.11	89.1 ± 0.86	92.72 ± 0.67	93.73 ± 0.61	w8a LR M=3	NaïveVFL	79.91 ± 1.1	92.13 ± 0.71	93.94 ± 0.62	**94.63 ± 0.56**
	BlindFL	67.15 ± 1.25	75 ± 1.25	80.03 ± 1.15	83.57 ± 1.08		BlindFL	74.78 ± 1.28	84.61 ± 1.07	88.67 ± 0.93	90.88 ± 0.83
	AdaVFed	76.51 ± 1.13	77.91 ± 1.13	79.24 ± 1.1	80.36 ± 1.06		AdaVFed	51.83 ± 1.37	52.06 ± 1.37	52.06 ± 1.42	51.98 ± 1.39
	Ours	**78.18 ± 1.07**	**90.83 ± 0.79**	**93.2 ± 0.66**	**93.89 ± 0.57**		Ours	**83.79 ± 1.05**	**93.31 ± 0.62**	**94.21 ± 0.59**	**94.63 ± 0.56**
MNIST CNN M=1	NaïveVFL	83.64 ± 0.38	93.55 ± 0.25	95.69 ± 0.19	96.88 ± 0.17	MNIST CNN M=3	NaïveVFL	80.97 ± 0.39	93.09 ± 0.26	94.8 ± 0.22	95.87 ± 0.2
	BlindFL	77.75 ± 0.42	91.09 ± 0.28	93.93 ± 0.24	94.92 ± 0.22		BlindFL	81.84 ± 0.39	93.09 ± 0.26	95.1 ± 0.21	96.22 ± 0.19
	AdaVFed	78.86 ± 0.41	91.27 ± 0.28	94.1 ± 0.24	95.2 ± 0.22		AdaVFed	58.73 ± 0.52	79.52 ± 0.42	82.42 ± 0.38	85.61 ± 0.34
	Ours	**87.75 ± 0.34**	**94.48 ± 0.23**	**96.1 ± 0.19**	**97.13 ± 0.16**		Ours	**85.05 ± 0.34**	**93.46 ± 0.25**	**95.11 ± 0.22**	**96.49 ± 0.18**
MNIST MLP M=1	NaïveVFL	73.14 ± 0.45	88.71 ± 0.32	90.51 ± 0.29	91.76 ± 0.28	MNIST MLP M=3	NaïveVFL	72.36 ± 0.46	87.88 ± 0.32	90.23 ± 0.3	91.59 ± 0.29
	BlindFL	40.9 ± 0.48	80.69 ± 0.41	84.97 ± 0.36	86.82 ± 0.33		BlindFL	51.15 ± 0.49	85.57 ± 0.35	88.93 ± 0.32	90.36 ± 0.3
	AdaVFed	56.45 ± 0.52	84.53 ± 0.36	87.71 ± 0.34	89.06 ± 0.31		AdaVFed	47.6 ± 0.48	71.48 ± 0.47	77.5 ± 0.4	80.12 ± 0.41
	Ours	**78.41 ± 0.4**	**89.48 ± 0.31**	**91.26 ± 0.29**	**92.39 ± 0.26**		Ours	**80.35 ± 0.39**	**89.86 ± 0.3**	**91.53 ± 0.28**	**92.77 ± 0.26**
cifar-10 CNN M=1	NaïveVFL	45.55 ± 0.51	59.8 ± 0.48	**67.32 ± 0.47**	70.06 ± 0.46	cifar-10 CNN M=3	NaïveVFL	46.68 ± 0.49	**61.43 ± 0.46**	**67.65 ± 0.48**	68.31 ± 0.45
	BlindFL	43.42 ± 0.49	59.64 ± 0.49	63.45 ± 0.47	65.6 ± 0.46		BlindFL	43.52 ± 0.49	59.07 ± 0.48	65.07 ± 0.5	68.04 ± 0.49
	AdaVFed	28.91 ± 0.45	41.96 ± 0.48	46.51 ± 0.49	51.14 ± 0.5		AdaVFed	19.65 ± 0.38	33.43 ± 0.49	39.3 ± 0.47	44.69 ± 0.48
	Ours	**49.51 ± 0.52**	**61.96 ± 0.48**	66.52 ± 0.45	**71.88 ± 0.44**		Ours	**49.26 ± 0.49**	61.38 ± 0.49	67.6 ± 0.46	**70.92 ± 0.45**
EMNIST CNN M=1	NaïveVFL	**73.35 ± 0.13**	81.27 ± 0.11	83.61 ± 0.11	84.24 ± 0.1	EMNIST CNN M=3	NaïveVFL	69.52 ± 0.14	79.29 ± 0.12	82.18 ± 0.11	82.11 ± 0.11
	BlindFL	60.69 ± 0.14	73.71 ± 0.13	75.4 ± 0.12	77.33 ± 0.12		BlindFL	66.94 ± 0.13	78.55 ± 0.13	81.2 ± 0.11	82.21 ± 0.11
	AdaVFed	66.82 ± 0.14	74.51 ± 0.13	76.35 ± 0.13	76.88 ± 0.12		AdaVFed	53.74 ± 0.15	62.33 ± 0.13	64.92 ± 0.14	65.76 ± 0.14
	Ours	71.61 ± 0.13	**81.58 ± 0.11**	**84.14 ± 0.1**	**84.35 ± 0.1**		Ours	**71.1 ± 0.13**	**81.21 ± 0.12**	**82.98 ± 0.11**	**83.31 ± 0.11**



(a) a9a LR M=1 (b) a9a LR M=3 (c) cifar-10 CNN M=1

(d) cifar-10 CNN M=3 (e) MNIST MLP M=1 (f) MNIST MLP M=3

Fig. 6. Gradient inference results.

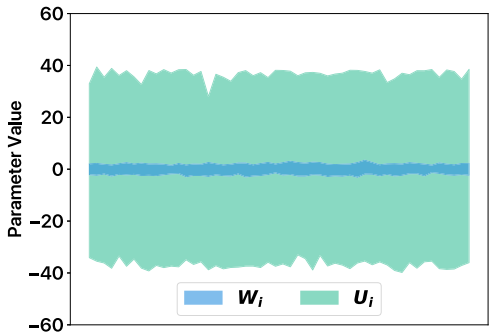


Fig. 7. Model protection result. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

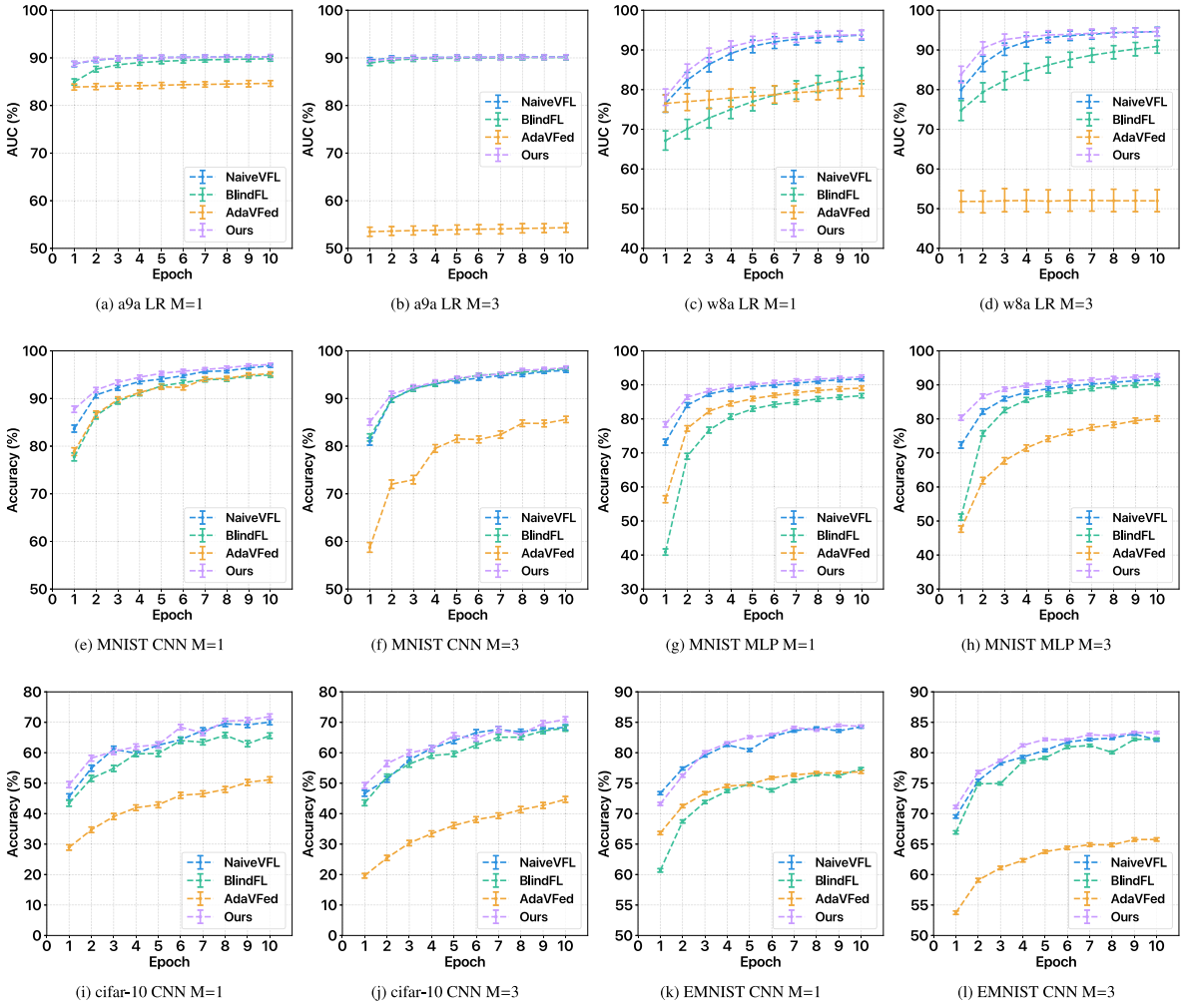
Table 9
Evaluation of extra computation cost (in seconds, PP: Privacy Preservation).

	Without PP	With PP
$Comp_F$	0.0195 ± 0.0001	3.1227 ± 0.0851
$Comp_T$	0.0346 ± 0.0004	21.5704 ± 0.4978

each training round grows, which subsequently degrades the overall model performance. Finally, we compare our method with NaïveVFL. As shown in Fig. 8 and Table 8, the performance of our approach closely matches that of NaïveVFL. Given that NaïveVFL operates without any privacy preservation, these results demonstrate that incorporating privacy protection into our method causes no noticeable degradation in model training and inference performance.

6.5.5. Computation cost analysis

Fig. 9 shows the time cost of our method and BlindFL. In the experiments, we recorded the time required for PVFL-CPN and BlindFL to complete training on a batch of data. Specifically, we measured the total computation time of both methods across all computational nodes for the forward propagation, backward propagation, and model update phases on a single batch of data (this recorded time is the sum of the computation times across all nodes, not the computation time of a single node). From the graph, we can observe that our method saves nearly half the time compared to BlindFL (our method consumes approximately 58% of the time taken by BlindFL). This improvement stems from the fact that all homomorphic encryption operations in BlindFL are performed at the source layer; specifically, BlindFL executes homomorphic encrypted multiplications directly on the original features. If the dimension of the original features is D_{feat} and the dimension of the first layer's output is D_{layer}, the time complexity of the homomorphic operations performed by a single node is $O\left(D_{feat}D_{layer}\right)$. In contrast, our method performs homomorphic encryption calculations on intermediate results. Given that the dimension of the intermediate results is D_{IR} and the number of classes is C, the time complexity of our method is $O(D_{IR}C)$. In practice, $D_{feat}D_{layer} > D_{IR}C$. Although our approach also incorporates single-mask encryption, the total overhead remains lower than that of BlindFL. During backward propagation,



(a) a9a LR M=1 (b) a9a LR M=3 (c) w8a LR M=1 (d) w8a LR M=3

(e) MNIST CNN M=1 (f) MNIST CNN M=3 (g) MNIST MLP M=1 (h) MNIST MLP M=3

(i) cifar-10 CNN M=1 (j) cifar-10 CNN M=3 (k) EMNIST CNN M=1 (l) EMNIST CNN M=3

Fig. 8. Performance comparison results.

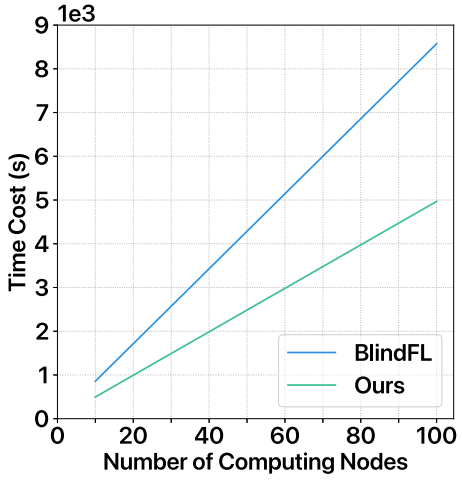


Fig. 9. Computation cost results.

our method requires the calculation of $[\![\nabla \boldsymbol{V}_i - \boldsymbol{\psi}_i]\!]_A$ in addition to $[\![\boldsymbol{U}_i]\!]_i$, which is not required in BlindFL. This part of the computation involves homomorphic encryption and incurs a complexity of $O(D_{IR}C)$. Therefore, by integrating the aforementioned advantages, our method saves nearly half the time compared to BlindFL.

Here, we further analyze the additional computational overhead. During forward propagation, the primary increase in computational cost stems from homomorphic encryption operations. In homomorphic encryption, the computational complexity of a single encryption or decryption operation is approximately $O\left(L^{2.585}\right)$, a homomorphic multiplication costs approximately $O\left(b \cdot L^{1.585}\right)$ (where b denotes the average binary bit length of the plaintext, and L represents the public key length), and a homomorphic addition costs approximately $O\left(L^{1.585}\right)$ (here, the computational complexity is derived using Karatsuba algorithm). In forward propagation, each passive node P_i needs to perform one homomorphic matrix multiplication and two homomorphic matrix additions, while the active node A performs M decryptions on the transmitted $[\![\boldsymbol{O}_i^{\boldsymbol{V}}]\!]_A^*$ from M nodes. Assuming the dimension of the fusion model (prediction layer) is $N \cdot C$ (where C is the number of classes) and its input $\boldsymbol{X}_i$ has a dimension of $B \cdot N$, the matrix multiplication requires $B \cdot N \cdot C$ homomorphic multiplications and $B \cdot C \cdot (N-1)$ homomorphic additions. Subsequently, the two matrix additions require $2BC$ homomorphic additions. Finally, the decryption process is executed M times, yielding a total of MBC decryptions. Consequently, we obtain the total additional computational overhead across all nodes during forward propagation, denoted as $Comp_F$ (note that, except for the active node A's operations, the operations of all

Table 10
Evaluation of extra communication cost (in MB, PP: Privacy Preservation).

	Without PP	With PP
$Comm_F$	0.0938	0.4688
$Comm_B$	0.0938	23.1788

passive nodes are executed in parallel, and thus are calculated only once), i.e.

$$Comp_F = \left(BNC \cdot bL^{1.585} + BC(N-1) \cdot L^{1.585}\right) + 2BC \cdot L^{1.585} + MBC \cdot L^{2.585}.$$

Similarly, we can obtain the total additional computational overhead across all nodes during backpropagation. During backpropagation, when computing $\nabla \boldsymbol{V}_i - \boldsymbol{\psi}_i$, it requires one encryption on $\nabla \boldsymbol{Z}$, one homomorphic matrix multiplication, and one homomorphic matrix addition on $[\![\nabla \boldsymbol{V}_i - \boldsymbol{\psi}_i]\!]_A$, followed by M decryptions. We denote the computational overhead of this part as $Comp_B^A$, which is detailed as follows:

$$Comp_B^A = BC \cdot L^{2.585} + \left(NBC \cdot bL^{1.585} + NC(B-1) \cdot L^{1.585}\right) + NC \cdot L^{1.585} + MNC \cdot L^{2.585}.$$

Each term in the above equation corresponds directly to the descriptions. Subsequently, to compute $\nabla \boldsymbol{X}_i$, each passive node performs one encryption on $\boldsymbol{U}_i$, and the active node A executes M homomorphic matrix additions and homomorphic matrix multiplications to obtain $[\![\boldsymbol{X}_i]\!]_i$. Finally, each passive node performs one decryption on $[\![\boldsymbol{X}_i]\!]_i$. We denote the computational overhead of this process as $Comp_B^P$, which is detailed as follows:

$$Comp_B^P = NC \cdot L^{2.585} + MNC \cdot L^{1.585} + M(BCN + BN(C-1)) \cdot bL^{1.585} + BN \cdot L^{2.585}.$$

Finally, during the model update following backpropagation, the active node A performs M encryptions on the updated parameters $\boldsymbol{V}_i^{e+1}$, incurring a cost of $MNC \cdot L^{2.585}$. Therefore, within a single training batch, the total additional computational overhead is given by:

$$Comp_T = Comp_F + Comp_B^A + Comp_B^P + MNC \cdot L^{2.585}.$$

Here, we also experimentally measure the computational time required to train a single batch with and without privacy preservation. We conduct this evaluation on the "MNIST CNN M=3" configuration, and the results are detailed in Table 9.

As observed from the results in Table 9, the empirical measurements are largely consistent with our analysis. During forward propagation, significant computational overhead is introduced by homomorphic encryption and secret sharing. The backpropagation phase under privacy preservation incurs a substantially longer computation time than the forward pass, which aligns with the aforementioned analysis because it involves additional homomorphic matrix multiplications, homomorphic matrix additions, as well as extra encryption and decryption operations. Although the computational overhead is higher than that of non-privacy-preserving methods, this trade-off is necessary to achieve robust privacy safeguards. In our previous performance comparison, we benchmarked against a differential privacy-based method, AdaVFed, which achieves much lower computational overhead. However, its model performance suffers significantly, deteriorating further as the number of nodes increases. Given that a primary goal of privacy-preserving machine learning is to safeguard privacy without substantially compromising model performance, we contend that sacrificing a reasonable degree of computational efficiency is worthwhile to maintain model accuracy while protecting privacy.

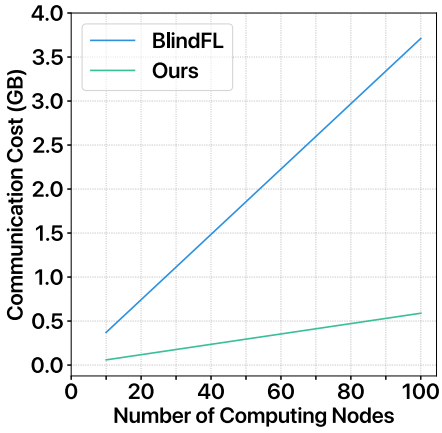


Fig. 10. Communication cost results.

6.5.6. Communication cost analysis

Fig. 10 shows the communication cost of our method in forward propagation and backward propagation. In the experiments, we measured the total amount of data exchanged (sent and received) by the active node A during the training of one data batch for both PVFL-CPN and BlindFL. Specifically, we summed the data volume transmitted from the passive nodes P_i to the active node A and vice versa during the forward propagation, backward propagation, and model update phases. As shown in the figure, our method achieves a communication overhead that is more than 5 times lower than BlindFL (approximately 15.9% of BlindFL's overhead). The reason our method incurs lower communication overhead than BlindFL is that, during forward propagation, the communication complexity of BlindFL is at least $O(D_{layer}L)$, where L is the length of a homomorphically encrypted ciphertext. In contrast, the communication complexity of our method during forward propagation is $O(CL)$. Since $C < D_{layer}$, even though we employ mask encryption, its communication overhead is far smaller than that of homomorphically encrypted ciphertexts; thus, our method saves a significant amount of communication overhead during forward propagation. Although our method incurs an additional communication overhead of $O(D_{IR}CL)$ when calculating $[\![\nabla \boldsymbol{X}_i]\!]_i$ during the backward process, BlindFL incurs a communication overhead of $O(D_{feat}D_{layer}L)$ when calculating the gradient $\nabla \boldsymbol{V}_i$. This remains substantially larger than the $O(D_{IR}CL)$ communication overhead incurred by our method to compute $\nabla \boldsymbol{V}_i$, where we utilize mask encryption for the calculation of $\nabla \boldsymbol{V}_i$, yet its overhead is still far less than that of homomorphically encrypted ciphertexts. Therefore, after synthesizing the aforementioned analysis, our method is superior to BlindFL in terms of communication overhead.

Here, we further analyze the additional communication overhead. During forward propagation, the additional communication overhead primarily stems from the ciphertext matrices $[\![\boldsymbol{O}_i^{\boldsymbol{V}}]\!]_A^*$ transmitted from the passive nodes to the active node, alongside the masks $\boldsymbol{\Phi}_i$ used for secret sharing. The masks are unencrypted, with each element's size corresponding to the bit length of a standard integer. In contrast, the size of each ciphertext in the matrix depends on the public key length L, where the bit length of a single ciphertext is $2L$. Consequently, the forward propagation communication overhead equals the cumulative size of the M ciphertext matrices $[\![\boldsymbol{O}_i^{\boldsymbol{V}}]\!]_A^*$, as the size of $\boldsymbol{\Phi}_i$ is negligible by comparison. Denoting the communication overhead during forward propagation as $Comm_F$, we obtain:

$$Comm_F = MBC \cdot 2L.$$

Similarly, during backpropagation and model updating, when computing $\nabla \boldsymbol{V}_i - \boldsymbol{\psi}_i$, the active node transmits M ciphertext matrices $[\![\nabla \boldsymbol{Z}]\!]_A$ and receives M matrices $[\![\nabla \boldsymbol{V}_i - \boldsymbol{\psi}_i]\!]_A$. When computing $\nabla \boldsymbol{X}_i$,

each passive node transmits $[\![\boldsymbol{U}_i]\!]_i$ to the active node and receives $[\![\nabla \boldsymbol{X}_i]\!]_i$. Finally, during model updating, the active node transmits the newly encrypted parameters $[\![\boldsymbol{V}_i^{e+1}]\!]_A$ to all passive nodes. Based on the above description, the additional communication overhead for backpropagation and model updating is derived as:

$$\begin{aligned} Comm_B = & MBC \cdot 2L + MNC \cdot 2L + MNC \cdot 2L \\ & MBN \cdot 2L + MNC \cdot 2L. \end{aligned}$$

where each term corresponds directly to the descriptions above. Thus, the total additional communication overhead is given by $Comm_T = Comm_F + Comm_B$. Here, we also evaluate the communication overhead required to train a single batch on the "MNIST CNN M=3" experimental setup, with the results presented in Table 10.

As observed from the results in Table 10, the experimental measurements are largely consistent with our analysis, showing that the communication overhead during backpropagation is significantly higher than that during forward propagation. Therefore, to safeguard privacy while maintaining model performance, sacrificing a degree of communication efficiency is a necessary trade-off to achieve these objectives.

6.5.7. Scalability analysis

In this section, we analyze the scalability of our method. First, regarding accuracy, based on the experimental results in Fig. 8 and Table 8, we observe that accuracy does not decrease significantly as the number of passive nodes increases. The accuracy for all experiments with $M = 3$ differs from the $M = 1$ results by only approximately 1%. We attribute this minor decline to potential errors during initialization or the training process, which stands in contrast to the sharp decline in accuracy observed in AdaVFed as the number of passive nodes grows.

Regarding computational overhead, we focus our analysis on the most time-consuming components of our method. In our approach, the dominant operations are homomorphic encryption, addition, and decryption. Other encryption operations (e.g., mask encryption) incur negligible overhead compared to homomorphic encryption; thus, we only quantify the homomorphic operations. Let D_{IR} denote the dimension of the intermediate results output by each node, B denote the batch size, and C denote the number of classes. Each passive node performs one homomorphic encrypted matrix multiplication and two encrypted matrix additions during forward propagation, resulting in a total number of homomorphic operations $HO_{PF} = BD_{IR}C + 2BC$. This count remains invariant regardless of the number of nodes. The active node only performs decryption during forward propagation, thus, $HO_{AF} = MBC$. In backward propagation, the active node performs one encryption on the prediction gradient $\nabla \boldsymbol{Z}$, one decryption on the gradient $\nabla \boldsymbol{V}_i - \boldsymbol{\psi}_i$, one matrix addition and multiplication to compute $\nabla \boldsymbol{X}_i$, and an encryption of $\boldsymbol{V}_{i+1}^e$ during model updates. Therefore, the total homomorphic operations for the active node in backward propagation is $HO_{AB} = BC + MD_{IR}C + \left(MD_{IR}C + MBD_{IR}C\right) + MD_{IR}C$, where each term corresponds to the aforementioned operations, respectively. Each passive node performs one matrix multiplication and one addition to calculate $\nabla \boldsymbol{V}_i - \boldsymbol{\psi}_i$, and one encryption and one decryption to calculate $\nabla \boldsymbol{X}_i$; thus, $HO_{PB} = BD_{IR}C + D_{IR}C + D_{IR}C + BD_{IR}$. From this analysis, we find that if the number of passive nodes M increases by a factor of k, only HO_{AF} and HO_{AB} are affected. Since these terms are linear with respect to M, the computational overhead is linearly correlated with the number of nodes.

Regarding communication overhead, we let L denote the length of a ciphertext. Since the transmission of homomorphically encrypted ciphertexts constitutes the largest portion of communication throughout the entire training process, our analysis focuses specifically on these ciphertexts. During the forward propagation phase, the active node receives $[\![\boldsymbol{O}_i^{\boldsymbol{V}}]\!]_A^*$ sent by the passive nodes, resulting in a communication overhead of $CCost_F = MBCL$. In the backward propagation phase, the active node needs to send $[\![\nabla \boldsymbol{Z}]\!]_A$, receive $[\![\nabla \boldsymbol{V}_i - \boldsymbol{\psi}_i]\!]$, receive $[\![\boldsymbol{U}_i]\!]_i$, and send $[\![\boldsymbol{X}_i]\!]_i$ and $[\![\boldsymbol{V}_i^{e+1}]\!]_A$. Consequently, $CCost_B = MBCL + MD_{IR}CL + MD_{IR}CL + MBD_{IR}L + MD_{IR}CL$ is ultimately obtained. It is evident from $CCost_F$ and $CCost_B$ that the communication overhead is linearly proportional to the number of passive nodes M.

7. Conclusion

In this article, we propose a privacy-preserving vertical federated learning method for computing power networks, PVFL-CPN. Our method employs homomorphic encryption, secret sharing, and single-mask encryption technologies to protect the intermediate results of passive nodes during forward propagation, and the intermediate result gradients calculated by the active node during backward propagation, thereby safeguarding the private data of passive nodes and the labels of the active node. Furthermore, our method performs encryption and aggregation on only a portion of the intermediate results, and conducts backward propagation directly on the encrypted intermediate results, which reduces both computational and communication overhead. Subsequently, the introduction of single-mask encryption and a Trusted Third Party (TTP) prevents collusion among multiple passive nodes and the active node. Rigorous security analysis theoretically proves the security of our method. Experimental results demonstrate that our method outperforms SOTA methods, namely BlindFL and AdaVFed, in terms of model performance. Additionally, our method surpasses BlindFL in both computational and communication efficiency. As the number of passive nodes increases, the performance of our method does not degrade, successfully avoiding the performance degradation issue faced by AdaVFed when scaling up the number of passive nodes.

In future work, homomorphic encryption operations will be replaced with secret sharing, as they consume a large amount of computing resources, which is not conducive to our extension of the method to large models. Secondly, a TTP is an ideal participant, but in real applications, such an ideal participant does not exist. Therefore, we will also consider how to achieve secure vertical federated learning without a TTP. Finally, we will explore more complex threat models, such as model poisoning, data poisoning, and backdoor attacks, to devise corresponding defensive countermeasures. Furthermore, we aim to integrate these defense mechanisms with privacy-preserving techniques to construct a more robust vertical federated learning system.

CRediT authorship contribution statement

Boyang Wang: Writing – original draft, Validation, Methodology, Formal analysis, Conceptualization. **Xiaolong Xu:** Writing – review & editing, Supervision, Methodology, Funding acquisition, Conceptualization. **Marcello Trovati:** Writing – review & editing, Formal analysis. **Nikolaos Polatidis:** Writing – review & editing. **Francesco Palmieri:** Writing – review & editing, Supervision, Methodology, Funding acquisition, Conceptualization.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

This work was supported by the National Natural Science Foundation of China under Grant U23A20307, by the Postgraduate Research & Practice Innovation Program of Jiangsu Province under Grant KYCX23_1080 and by project SERICS (PE00000014) under the NRRP-MUR program funded by the EU-NGEU.

Data availability

Data will be made available on request.

References

[1] O. Li, J. Sun, X. Yang, W. Gao, H. Zhang, J. Xie, V. Smith, C. Wang, Label leakage and protection in two-party split learning, 2022, arXiv:2102.08504. URL https://arxiv.org/abs/2102.08504.

[2] C. Fu, X. Zhang, S. Ji, J. Chen, J. Wu, S. Guo, J. Zhou, A.X. Liu, T. Wang, Label inference attacks against vertical federated learning, in: Proc. 31st USENIX Secur. Symp., USENIX Association, Boston, MA, 2022, pp. 1397–1414, URL https://www.usenix.org/conference/usenixsecurity22/presentation/fu-chong.

[3] Z. He, T. Zhang, R.B. Lee, Model inversion attacks against collaborative inference, in: Proc. 35th Annu. Comput. Secur. Appl. Conf., Association for Computing Machinery, New York, NY, USA, 2019, pp. 148–162, http://dx.doi.org/10.1145/3359789.3359824.

[4] A. Gascon, P. Schoppmann, B. Balle, M. Raykova, J. Doerner, S. Zahur, D. Evans, Privacy-preserving distributed linear regression on high-dimensional data, 2016, http://dx.doi.org/10.1515/popets-2017-0053, URL https://eprint.iacr.org/2016/892. Cryptology ePrint Archive, Paper 2016/892.

[5] S. Hardy, W. Henecka, H. Ivey-Law, R. Nock, G. Patrini, G. Smith, B. Thorne, Private federated learning on vertically partitioned data via entity resolution and additively homomorphic encryption, 2017, arXiv:1711.10677. URL https://arxiv.org/abs/1711.10677.

[6] S. Yang, B. Ren, X. Zhou, L. Liu, Parallel distributed logistic regression for vertical federated learning without third-party coordinator, 2019, arXiv:1911.09824. URL http://arxiv.org/abs/1911.09824.

[7] D. He, R. Du, S. Zhu, M. Zhang, K. Liang, S. Chan, Secure logistic regression for vertical federated learning, IEEE Internet Comput. 26 (2) (2022) 61–68, http://dx.doi.org/10.1109/MIC.2021.3138853.

[8] C. Chen, J. Zhou, L. Wang, X. Wu, W. Fang, J. Tan, L. Wang, A.X. Liu, H. Wang, C. Hong, When homomorphic encryption marries secret sharing: Secure large-scale sparse logistic regression and applications in risk control, in: Proc. 27th ACM SIGKDD Conf. Knowl. Discovery Data Mining, Association for Computing Machinery, New York, NY, USA, 2021, pp. 2652–2662, http://dx.doi.org/10.1145/3447548.3467210.

[9] R. Xu, N. Baracaldo, Y. Zhou, A. Anwar, J. Joshi, H. Ludwig, FedV: Privacy-preserving federated learning over vertically partitioned data, in: Proc. 14th ACM Workshop Artif. Intell. Secur., Association for Computing Machinery, New York, NY, USA, 2021, pp. 181–192, http://dx.doi.org/10.1145/3474369.3486872.

[10] K. Cheng, T. Fan, Y. Jin, Y. Liu, T. Chen, D. Papadopoulos, Q. Yang, SecureBoost: A lossless federated learning framework, IEEE Intell. Syst. 36 (6) (2021) 87–98, http://dx.doi.org/10.1109/MIS.2021.3082561.

[11] T. Fan, W. Chen, G. Ma, Y. Kang, L. Fan, Q. Yang, SecureBoost+: Large scale and high-performance vertical federated gradient boosting decision tree, in: Proc. Advances Knowl. Discovery Data Mining, vol. 14647, Springer, 2024, pp. 237–249, http://dx.doi.org/10.1007/978-981-97-2259-4_18.

[12] W. Fang, D. Zhao, J. Tan, C. Chen, C. Yu, L. Wang, L. Wang, J. Zhou, B. Zhang, Large-scale secure XGB for vertical federated learning, in: Proc. 30th ACM Int. Conf. Inf. Knowl. Manage., Association for Computing Machinery, New York, NY, USA, 2021, pp. 443–452, http://dx.doi.org/10.1145/3459637.3482361.

[13] L. Xie, J. Liu, S. Lu, T.-H. Chang, Q. Shi, An efficient learning framework for federated xgboost using secret sharing and distributed optimization, ACM Trans. Intell. Syst. Technol. 13 (5) (2022) 1–28, http://dx.doi.org/10.1145/3523061.

[14] Y. Zhang, H. Zhu, Additively homomorphical encryption based deep neural network for asymmetrically collaborative machine learning, 2020, arXiv:2007.06849. URL https://arxiv.org/abs/2007.06849.

[15] Y. Kang, Y. He, J. Luo, T. Fan, Y. Liu, Q. Yang, Privacy-preserving federated adversarial domain adaptation over feature groups for interpretability, IEEE Trans. Big Data 10 (6) (2024) 879–890, http://dx.doi.org/10.1109/TBDATA.2022.3188292.

[16] F. Fu, H. Xue, Y. Cheng, Y. Tao, B. Cui, BlindFL: Vertical federated machine learning without peeking into your data, in: Proc. 2022 Int. Conf. Manage. Data, Association for Computing Machinery, New York, NY, USA, 2022, pp. 1316–1330, http://dx.doi.org/10.1145/3514221.3526127.

[17] Y. Liu, Y. Kang, C. Xing, T. Chen, Q. Yang, A secure federated transfer learning framework, IEEE Intell. Syst. 35 (4) (2020) 70–82, http://dx.doi.org/10.1109/MIS.2020.2988525.

[18] S. Sharma, C. Xing, Y. Liu, Y. Kang, Secure and efficient federated transfer learning, in: Proc. 2019 IEEE Int. Conf. Big Data, IEEE, 2019, pp. 2569–2576, http://dx.doi.org/10.1109/BigData47090.2019.9006280.

[19] K. Gai, M. Wang, J. Yu, L. Xu, P. Jiang, L. Zhu, B. Xiao, Differentially private vertical federated learning with adaptive constraints and dynamic noise, IEEE Trans. Inf. Forensics Secur. 20 (2025) 11150–11164, http://dx.doi.org/10.1109/TIFS.2025.3620213.

[20] J. Zhang, S. Guo, Z. Qu, D. Zeng, H. Wang, Q. Liu, A.Y. Zomaya, Adaptive vertical federated learning on unbalanced features, IEEE Trans. Parallel Distrib. Syst. 33 (12) (2022) 4006–4018, http://dx.doi.org/10.1109/TPDS.2022.3178443.

[21] D. Cai, T. Fan, Y. Kang, L. Fan, M. Xu, S. Wang, Q. Yang, Accelerating vertical federated learning, IEEE Trans. Big Data 10 (6) (2024) 752–760, http://dx.doi.org/10.1109/TBDATA.2022.3192898.

[22] Y. Hu, D. Niu, J. Yang, S. Zhou, FDML: A collaborative machine learning framework for distributed features, in: Proc. 25th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining, Association for Computing Machinery, New York, NY, USA, 2019, pp. 2232–2240, http://dx.doi.org/10.1145/3292500.3330765.

[23] B. Gu, A. Xu, Z. Huo, C. Deng, H. Huang, Privacy-preserving asynchronous vertical federated learning algorithms for multiparty collaborative learning, IEEE Trans. Neural Netw. Learn. Syst. 33 (11) (2022) 6103–6115, http://dx.doi.org/10.1109/TNNLS.2021.3072238.

[24] Q. Zhang, B. Gu, C. Deng, S. Gu, L. Bo, J. Pei, H. Huang, AsySQN: Faster vertical federated learning algorithms with better computation resource utilization, in: Proc. 27th ACM SIGKDD Conf. Knowl. Discovery Data Mining, Association for Computing Machinery, New York, NY, USA, 2021, pp. 3917–3927, http://dx.doi.org/10.1145/3447548.3467169.

[25] Q. Zhang, B. Gu, C. Deng, H. Huang, Secure Bilevel Asynchronous Vertical Federated Learning with Backward Updating, vol. 35, AAAI Press, Palo Alto, California, USA, 2021, pp. 10896–10904, http://dx.doi.org/10.1609/aaai.v35i12.17301, URL https://ojs.aaai.org/index.php/AAAI/article/view/17301.

[26] B. Gu, Z. Dang, X. Li, H. Huang, Federated doubly stochastic kernel learning for vertically partitioned data, in: Proc. 26th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining, Association for Computing Machinery, New York, NY, USA, 2020, pp. 2483–2493, http://dx.doi.org/10.1145/3394486.3403298.

[27] Y. Liu, X. Zhang, Y. Kang, L. Li, T. Chen, M. Hong, Q. Yang, FedBCD: A communication-efficient collaborative learning framework for distributed features, IEEE Trans. Signal Process. 70 (2022) 4277–4290, http://dx.doi.org/10.1109/TSP.2022.3198176.

[28] T. Castiglia, S. Wang, S. Patterson, Flexible vertical federated learning with heterogeneous parties, IEEE Trans. Neural Netw. Learn. Syst. 35 (12) (2024) 17878–17892, http://dx.doi.org/10.1109/TNNLS.2023.3309701.

[29] T. Chen, X. Jin, Y. Sun, W. Yin, VAFL: a method of vertical asynchronous federated learning, 2020, arXiv:2007.06081. URL https://arxiv.org/abs/2007.06081.

[30] Z. Wu, Q. Li, B. He, Practical vertical federated learning with unsupervised representation learning, IEEE Trans. Big Data 10 (6) (2024) 864–878, http://dx.doi.org/10.1109/TBDATA.2022.3180117.

[31] A. Khan, M. ten Thij, A. Wilbik, Communication-efficient vertical federated learning, Algorithms 15 (8) (2022) 273, http://dx.doi.org/10.3390/a15080273.

[32] A. Li, H. Peng, L. Zhang, J. Huang, Q. Guo, H. Yu, Y. Liu, FedSDG-FS: Efficient and secure feature selection for vertical federated learning, in: Proc. 2023 IEEE Conf. Comput. Commun., 2023, pp. 1–10, http://dx.doi.org/10.1109/INFOCOM53939.2023.10228895.

[33] T. Castiglia, Y. Zhou, S. Wang, S. Kadhe, N. Baracaldo, S. Patterson, LESS-VFL: Communication-efficient feature selection for vertical federated learning, in: Proc. 40th Int. Conf. Mach. Learn., vol. 202, PMLR, 2023, pp. 3757–3781, URL https://proceedings.mlr.press/v202/castiglia23a.html.

[34] S. Feng, Vertical federated learning-based feature selection with non-overlapping sample utilization, Expert Syst. Appl. 208 (2022) 118097, http://dx.doi.org/10.1016/j.eswa.2022.118097, URL https://www.sciencedirect.com/science/article/pii/S095741742201291X.

[35] W. Xu, H. Zhu, Y. Zheng, F. Wang, J. Zhao, Z. Liu, H. Li, ELXGB: An efficient and privacy-preserving XGBoost for vertical federated learning, IEEE Tran. Serv. Comput. 17 (3) (2024) 878–892, http://dx.doi.org/10.1109/TSC.2024.3394706.

[36] W. Xu, H. Zhu, J. Zhao, Y. Zheng, F. Wang, B. Sun, S. Zhang, D. Feng, SGBoost+: Efficient and privacy-preserving vertical boosting trees for federated outsourced training and inference, IEEE Trans. Inf. Forensics Secur. 20 (2025) 8701–8714, http://dx.doi.org/10.1109/TIFS.2025.3597728.

[37] Q. Wang, Z. Wu, Y. Lu, EDVFL: An evaluable and decentralized privacy-preserving VFL for secure data sharing in ATM, Comput. Netw. 271 (2025) 111555, http://dx.doi.org/10.1016/j.comnet.2025.111555, URL https://www.sciencedirect.com/science/article/pii/S1389128625005225.

[38] S. Wang, K. Gai, J. Yu, Z. Zhang, L. Zhu, PraVFed: Practical heterogeneous vertical federated learning via representation learning, IEEE Trans. Inf. Forensics Secur. 20 (2025) 2693–2705, http://dx.doi.org/10.1109/TIFS.2025.3530700.

[39] G. Wang, Q. Li, X. Liu, X. Yan, Q. Dong, H. Wu, X. Kong, L. Zhou, VFGCN: A vertical federated learning framework with privacy preserving for graph convolutional network, IEEE Trans. Dependable Secur. Comput. 22 (5) (2025) 5617–5631, http://dx.doi.org/10.1109/TDSC.2025.3570626.

[40] X. Liu, Z. Hua, M. Song, Y. Zheng, G. Xu, X. Jia, PVF-FD: Free-rider detection in privacy-preserving vertical federated learning, IEEE Trans. Dependable Secur. Comput. 23 (3) (2026) 4799–4813, http://dx.doi.org/10.1109/TDSC.2025.3649071.

[41] S. Wang, Z. Chen, J. Liu, W. Xu, Towards efficient and secure vertical federated learning with additive ensemble, Int. J. Mach. Learn. Cybern. 17 (3) (2026) 112, http://dx.doi.org/10.1007/s13042-025-02897-2, URL https://link.springer.com/10.1007/s13042-025-02897-2.

[42] J. Hou, L. Zhang, VerifyVFL: Practical verifiable vertical federated learning, in: Proc. 55th Annu. IEEE/IFIP Int. Conf. Dependable Syst. Networks, 2025, pp. 75–87, http://dx.doi.org/10.1109/DSN64029.2025.00022.

[43] C. Yu, Z. Meng, W. Zhang, L. Lei, J. Ni, K. Zhang, H. Zhao, Secure and efficient federated learning against model poisoning attacks in horizontal and vertical data partitioning, IEEE Trans. Neural Netw. Learn. Syst. 36 (6) (2025) 10913–10927, http://dx.doi.org/10.1109/TNNLS.2024.3486028.

[44] X. Xu, W. Wang, Z. Chen, B. Wang, C. Li, L. Duan, Z. Han, Y. Han, Finding the PISTE: Towards understanding privacy leaks in vertical federated learning systems, IEEE Trans. Dependable Secur. Comput. 22 (2) (2025) 1537–1550, http://dx.doi.org/10.1109/TDSC.2024.3445600.

[45] Y. Xi, Y. Guo, S. Xu, C. Cai, X. Jia, Private sample alignment for vertical federated learning: An efficient and reliable realization, IEEE Trans. Inf. Forensics Secur. 20 (2025) 3834–3848, http://dx.doi.org/10.1109/TIFS.2025.3555794.

[46] Y. Yang, X. Chen, Y. Pan, J. Shen, Z. Cao, X. Dong, X. Li, J. Sun, G. Yang, R. Deng, OpenVFL: A vertical federated learning framework with stronger privacy-preserving, IEEE Trans. Inf. Forensics Secur. 19 (2024) 9670–9681, http://dx.doi.org/10.1109/TIFS.2024.3477924.

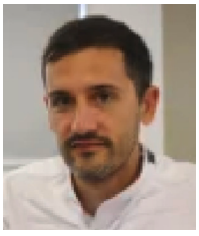
[47] T. Liao, L. Fu, L. Zhang, L. Yang, C. Chen, M.K. Ng, H. Huang, Z. Zheng, Privacy-preserving vertical federated learning with tensor decomposition for data missing features, IEEE Trans. Inf. Forensics Secur. 20 (2025) 3445–3460, http://dx.doi.org/10.1109/TIFS.2025.3552033.

[48] K. Bonawitz, V. Ivanov, B. Kreuter, A. Marcedone, H.B. McMahan, S. Patel, D. Ramage, A. Segal, K. Seth, Practical secure aggregation for privacy-preserving machine learning, in: Proc. 2017 ACM SIGSAC Conf. Comput. Commun. Secur., Association for Computing Machinery, New York, NY, USA, 2017, pp. 1175–1191, http://dx.doi.org/10.1145/3133956.3133982.

Boyang Wang is currently working as a researcher for Jiangsu Key Laboratory of Big Data Security & Intelligent Processing, Nanjing, China. He received his B.E degree in computer science and technology in 2022 from Nanjing University of Posts and Telecommunications, Nanjing, China, where he is pursuing a Ph.D. degree with the School of Computer Science at Nanjing University of Posts and Telecommunications. During his undergraduate years, he was granted a patent for "An Arithmetic Network Privacy-Preserving Federated Learning Method, Apparatus, and Storage Medium" during his Ph.D. studies. His research interests include federated learning and secure multi-party computation.

Xiaolong Xu is currently a professor in the School of Computer Science, Nanjing University of Posts & Telecommunications, Nanjing, China. He is also working at Jiangsu Key Laboratory of Big Data Security & Intelligent Processing. He received the B.E. in computer and its applications, the M.E. in computer software and theories, and the Ph.D. degree in communications and information systems from Nanjing University of Posts and Telecommunications, in 1999, 2002 and 2008, respectively. He is a senior member of the China Computer Federation. He teaches graduate courses and conducts research in areas of Cloud Computing, Big Data, Information Security and Novel Network Computing Technologies. As the leader of project teams, he has successfully completed a number of high-level research projects, including projects sponsored by the National Science Fund of China. He has published more than 100 Journal and conference papers as the first or corresponding author and 5 books. He has 52 patents authorized by the State Intellectual Property Office of China as the first inventor. He was rated an excellent young professor of Jiangsu province in 2014, selected as the high-level creative talent of Jiangsu province in 2015, and won the title of outstanding expert in the area of computer science and technology.

Marcello Trovati is a Professor of Digital Transformation at the School of Business, University of Lancashire, as well as the Editor in Chief of Applied Artificial Intelligence, Taylor&Francis. After having obtained his Ph.D. in Mathematics at the University of Exeter in 2007, specializing in theoretical dynamical systems with singularities, he has worked both in R&D and academic institutions. In 2025 Marcello joined the School of Business at the University of Lancashire, where he is involved in several research themes and projects.



Nikolaos Polatidis is a Principal Lecturer in Computer Science in the School of Architecture, Technology and Engineering at the University of Brighton. Prior to this, I was a Senior Lecturer from February 2019 to January 2022, a Lecturer from May 2018 to January 2019 and a Research Fellow from March 2017 to March 2018. He obtained a B.Sc. (Hons) from Heriot-Watt University in 2008, an M.Sc. in Internet Software Systems from the University of Birmingham in 2011 and a Ph.D. in Applied Informatics from the University of Macedonia (Greece) in 2017. He has published more than 50 peer-reviewed articles, served the wider community via various chairing roles and committee memberships for conferences and workshops and by reviewing for a wide range of journals for publishers such as IEEE, Springer, Elsevier, Wiley, Taylor & Francis and others.

Francesco Palmieri is a professor at the University of Salerno, Italy. His major research interests concern high-performance networking protocols and architectures, routing algorithms and network security. Previously, he has been an associate professor at the University of Salerno, an assistant professor at the Second University of Naples, and the Director of the telecommunication and networking division of the Federico II University, in Naples, Italy. At the start of his career, he also worked for several international companies on networking-related projects. He has been closely involved with the development of the Internet in Italy as a senior member of the Technical-Scientific Advisory Committee and of the CSIRT of the Italian NREN GARR. He has published a large number of papers in leading technical journals, books and conferences and currently serves as the editor-in-chief of an international journal and is part of the editorial board or associate editor of several other well-reputed ones.