FedV: Privacy-Preserving Federated Learning over Vertically Partitioned Data

Runhua Xu†, Nathalie Baracaldo†, Yi Zhou†, Ali Anwar †, James Joshi‡, Heiko Ludwig†

{runhua,yi.zhou,ali.anwar2}@ibm.com,{baracald,hludwig}@us.ibm.com,jjoshi@pitt.edu
† IBM Research, San Jose, CA, USA
‡ University of Pittsburgh, Pittsburgh, PA, USA

ABSTRACT
Federated learning (FL) has been proposed to allow collaborative training of machine learning (ML) models among multiple parties where each party can keep its data private. In this paradigm, only model updates, such as model weights or gradients, are shared. Many existing approaches have focused on horizontal FL, where each party has the entire feature set and labels in the training data set. However, many real scenarios follow a vertically-partitioned FL setup, where a complete feature set is formed only when all the datasets from the parties are combined, and the labels are only available to a single party. Privacy-preserving vertical FL is challenging because complete sets of labels and features are not owned by one entity. Existing approaches for vertical FL require multiple peer-to-peer communications among parties, leading to lengthy training times, and are restricted to (approximated) linear models and just two parties. To close this gap, we propose FedV, a framework for secure gradient computation in vertical settings for several widely used ML models such as linear models, logistic regression, and support vector machines. FedV removes the need for peer-to-peer communication among parties by using functional encryption schemes; this allows FedV to achieve faster training times. It also works for larger and changing sets of parties. We empirically demonstrate the applicability for multiple types of ML models and show a reduction of 10%-70% of training time and 80% to 90% in data transfer with respect to the state-of-the-art approaches.

CCS CONCEPTS
• Security and privacy → Privacy-preserving protocols; • Computing methodologies → Distributed artificial intelligence; Cooperation and coordination.

KEYWORDS
secure aggregation, functional encryption, privacy-preserving protocol, federated learning, privacy-preserving federated learning

1 INTRODUCTION
Machine learning (ML) has become ubiquitous and instrumental in many applications such as predictive maintenance, recommendation systems, self-driving vehicles, and healthcare. The creation of ML models requires training data that is often subject to privacy or regulatory constraints, restricting the way data can be shared, used and transmitted. Examples of such regulations include the European General Data Protection Regulation (GDPR), California Consumer Privacy Act (CCPA) and Health Insurance Portability and Accountability Act (HIPAA), among others.

There is great benefit in building a predictive ML model over datasets from multiple sources. This is because a single entity, henceforth referred to as a party, may not have enough data to build an accurate ML model. However, regulatory requirements and privacy concerns may make pooling such data from multiple sources infeasible. Federated learning (FL) [31, 37] has recently been shown to be very promising for enabling a collaborative training of models among multiple parties - under the orchestration of an aggregator - without having to share any of their raw training data. In this paradigm, only model updates, such as model weights or gradients, need to be exchanged.

There are two types of FL approaches, horizontal and vertical FL, which mainly differ in the data available to each party. In horizontal FL, each party has access to the entire feature set and labels; thus, each party can train its local model based on its own dataset. All the parties then share their model updates with an aggregator and the aggregator then creates a global model by combining, e.g., averaging, the model weights received from individual parties. In contrast, vertical FL (VFL) refers to collaborative scenarios where individual parties do not have the complete set of features and labels and, therefore, cannot train a model using their own datasets locally. In particular, parties’ datasets need to be aligned to create the complete feature vector without exposing their respective training data, and the model training needs to be done in a privacy-preserving way.

Existing approaches, as shown in Table 1, to train ML models in vertical FL or vertical setting, are model-specific and rely on general (garbled circuit based) secure multi-party computation (SMC), differential privacy noise perturbation, or partially additive homomorphic encryption (HE) (i.e., Paillier cryptosystem [17]). These approaches have several limitations: First, they apply only to limited models. They require the use of Taylor series approximation to train non-linear ML models, such as logistic regression, that cannot be generalized to solve classification problems. Furthermore, the prediction and inference phases of these vertical FL solutions rely on approximation-based secure computation or noise perturbation. As such, these solutions cannot predict as accurately as a centralized ML model can. Secondly, using such cryptosystems as part of the training process substantially increases the training time. Thirdly, these protocols require a large number of peer-to-peer communication rounds among parties, making it difficult to deploy them in systems that have poor connectivity or where communication is limited to a few specific entities due to regulation such as HIPAA. Finally, other approaches such as the one proposed in [58] require sharing class distributions, which may lead to potential leakage of private information of each party.
To address these limitations, we propose FedV. This framework substantially reduces the amount of communication required to train ML models in a vertical FL setting. FedV does not require any peer-to-peer communication among parties and can work with gradient-based training algorithms, such as stochastic gradient descent and its variants, to train a variety of ML models, e.g., logistic regression, support vector machine (SVM), etc. To achieve these benefits, FedV orchestrates multiple functional encryption techniques [2, 3] - which are non-interactive in nature - speeding up the training process compared to the state-of-the-art approaches. Additionally, FedV supports more than two parties and allows parties to dynamically leave and re-join without a need for re-keying. This feature is not provided by garbled-circuit or HE based techniques utilized by state-of-the-art approaches.

To the best of our knowledge, this is the first generic and efficient privacy-preserving vertical federated learning (VFL) framework that drastically reduces the number of communication rounds required during model training while supporting a wide range of widely used ML models. The main contributions of this paper are as follows:

We propose FedV, a generic and efficient privacy-preserving vertical FL framework, which only requires communication between parties and the aggregator as a one-way interaction and does not need any peer-to-peer communication among parties.

FedV enables the creation of highly accurate models as it does not require the use of Taylor series approximation to address non-linear ML models. In particular, FedV supports stochastic gradient-based algorithms to train many classical ML models, such as, linear regression, logistic regression and support vector machines, among others, without requiring linear approximation for non-linear ML objectives as a mandatory step, as in the existing solutions. FedV supports both lossless training and lossless prediction.

We have implemented and evaluated the performance of FedV. Our results show that compared to existing approaches FedV achieves significant improvements both in training time and communication cost without compromising privacy. We show that these results hold for a range of widely used ML models including linear regression, logistic regression and support vector machines. Our experimental results show a reduction of 10%-70% of training time and 80%-90% of data transfer when compared to state-of-the-art approaches.

## 2 BACKGROUND

### 2.1 Vertical Federated Learning

VFL is a powerful approach that can help create ML models for many real-world problems where a single entity does not have access to all the training features or labels. Consider a set of banks and a regulator. These banks may want to collaboratively create an ML model using their datasets to flag accounts involved in money laundering. Such a collaboration is important as criminals typically use multiple banks to avoid detection. However, if several banks join together to find a common vector for each client and a regulator provides the labels, showing which clients have committed money laundering, such fraud can be identified and mitigated. However, each bank may not want to share its clients’ account details and in some cases it is even prevented to do so.

One of the requirements for privacy-preserving VFL is thus to ensure that the dataset of each party is kept confidential. VFL requires two different processes: entity resolution and vertical training. Both of them are orchestrated by an Aggregator that acts as a third semi-trusted party interacting with each party. Before we present the detailed description of each process, we introduce the notation used throughout the rest of the paper.

**Notation:** Let \( P = \{ p_i \}_{i \in [n]} \) be the set of \( n \) parties in VFL. Let \( D^{[X,Y]} \) be the training dataset across the set of parties \( P \), where \( X \in \mathbb{R}^d \) represents the feature set and \( Y \in \mathbb{R} \) denotes the labels. We assume that except for the identifier features, there are no overlapping training features between any two parties’ local datasets, and these datasets can form the “global” dataset \( D \). As it is commonly done in VFL settings, we assume that only one party has the class labels, and we call it the active party, while other parties are passive parties. For simplicity, in the rest of the paper, let \( p_1 \) be the active party. The goal of FedV is to train a ML model \( M \) over the dataset \( D \) from the party set \( P \) without leaking any party’s data.

**Private Entity Resolution (PER):** In VFL, unlike in a centralized ML scenario, \( D \) is distributed across multiple parties. Before training takes place, it is necessary to ‘align’ the records of each party without revealing its data. This process is known as entity resolution [15]. Figure 1 presents a simple example of how \( D \) can be vertically partitioned among two parties. After the entity resolution step, records from all parties are linked to form the complete set of training samples.

Ensuring that the entity resolution process does not lead to inference of private data of each party is crucial in VFL. A curious party should not be able to infer the presence or absence of a record. Existing approaches, such as [28, 41], use a bloom filter and random
where $L$ computation setting, since $D$ ters (a.k.a. the weights) of a ML model based on of the locally steepest descent as defined by the negative of the GD finds a solution of (1) by iteratively moving in the direction $R$ loss function; for example, in machine learning domain, a typical class of optimization algorithms that find the minimum of a target prediction function, and $R$ is regularization term with coefficient $\lambda$. GD finds a solution of (1) by iteratively moving in the direction of the locally steepest descent as defined by the negative of the gradient, i.e., $\mathbf{w} \leftarrow \mathbf{w} - \alpha \nabla E_D(\mathbf{w})$, where $\alpha$ is the learning rate, and $\nabla E_D(\mathbf{w})$ is the gradient computed at the current iteration. Due to their simple algorithmic schemes, GD and its variants, like SGD, have become the common approaches to find the optimal parameters (a.k.a. the weights) of a ML model based on $D$. In a VFL setting, since $D$ is vertically partitioned among parties, the gradient computation $\nabla E_D(\mathbf{w})$ is more computationally involved than in a centralized ML setting.

Considering the simplest case where there are only two parties, $P_A, P_B$, in a VFL system as illustrated in Figure 1, and MSE (Mean Squared Loss) is used as the target loss function, i.e., $E_D(\mathbf{w}) = \frac{1}{n} \sum_{i=1}^{n} (y(i) - f(x(i); \mathbf{w})^2$, we have

$$\nabla E_D(\mathbf{w}) = -\frac{2}{n} \sum_{i=1}^{n} (y(i) - f(x(i); \mathbf{w})) \nabla f(x(i); \mathbf{w}).$$

If we expand (2) and compute the result of the summation, we need to compute $-y(i) \nabla f(x(i); \mathbf{w})$ for $i = 1,...,n$, which requires feature information from both $P_A$ and $P_B$, and labels from $P_B$. And, clearly, $\nabla f(x(i); \mathbf{w}) = [\partial_w f(x(i); \mathbf{w}); \partial_{ww} f(x(i); \mathbf{w})]$ does not always hold for any function $f$, since $f$ may not be well-separable w.r.t. $\mathbf{w}$. Even when it holds for linear functions like $f(x(i); \mathbf{w}) = \mathbf{x}(i) \cdot \mathbf{w}_A + \mathbf{x}(i) \cdot \mathbf{w}_B$, (2) will be reduced as follows:

$$\nabla E_D(\mathbf{w}) = -\frac{2}{n} \sum_{i=1}^{n} [(y(i) - x(i) \cdot \mathbf{w}_A) \cdot \mathbf{x}(i); x(i) \cdot \mathbf{w}_B \cdot \mathbf{x}(i); \partial_{ww} f(x(i); \mathbf{w})];$$

$$= -\frac{2}{n} \sum_{i=1}^{n} [(y(i) - x(i) \cdot \mathbf{w}_A - x(i) \cdot \mathbf{w}_B) \cdot \mathbf{x}(i); \partial_{ww} f(x(i); \mathbf{w})].$$

This may lead to exposure of training data between two parties due to the computation of some terms (colored in red) in (3). Under the VFL setting, the gradient computation at each training epoch relies on (i) the parties’ collaboration to exchange their “partial model” with each other, or (ii) exposing their data to the aggregator to compute the final gradient update. Therefore, any naive solutions will lead to a significant risk of privacy leakage, which will counter the initial goal of the FL to protect data privacy. Before presenting our approach, we first overview the basics of functional encryption.

2.3 Functional Encryption

Our proposed FedV makes use of encryption (FE) a cryptosystem that allows computing a specific function over a set of ciphertexts without revealing the inputs. FE belongs to a public-key encryption family [9, 33], where possessing a secret key called a functionally derived key enables the computation of a function $f$ that takes as input ciphertexts, without revealing the ciphertexts. The functionally derived key is provided by a trusted third-party authority (TPA) which also responsible for initially setting up the cryptosystem. For VFL, we require the computation of inner products. For that reason, we adopt functional encryption for inner product (FEIP), which allows the computation of the inner product between two vectors $x$ containing encrypted private data, and $y$ containing public plaintext data. To compute the inner product $(x, y)$ the decrypting entity (e.g., aggregator) needs to obtain a functionally derived key from the TPA. To produce this key, the TPA requires access to the public plaintext vector $y$. Note that the TPA does not need access to the private encrypted vector $x$.

We adopt two types of inner product FE schemes: single-input functional encryption ($E_{SIFE}$) proposed in [2] and multi-input functional encryption ($E_{MIFE}$) introduced in [3], which we explain in detail below.

$SIFE(E_{SIFE})$. To explain this crypto system, consider the simple example. A party wants to keep $x$ private but wants an entity (aggregator) to be able to compute the inner product $(x, y)$. Here $x$ is secret and encrypted and $y$ is public and provided by the aggregator to compute the inner product. During set up, the TPA provides the public key $pk_{SIFE}$ to a party. Then, the party encrypts $x$ with that key, denoted as $ct_x = E_{SIFE}(pk_{SIFE}; x)$; and sends $ct_x$ to the aggregator with a vector $y$ in plaintext. The TPA generates a functionally derived key that depends on $y$, denoted as $dk_y$. The aggregator decrypts $ct_x$ using the received key denoted as $dk_y$. As a result of the decryption, the aggregator obtains the result inner...
We now introduce our proposed approach, FedV, which contains a subset of features and wants to collaboratively train a global model. We name the supported function as follows: (i) one active party that has training samples with partial features and the class labels, represented as \( p_1 \) in Figure 2; and (ii) multiple passive parties who have training samples with only partial features.

### 3.1 Threat Model and Assumptions

The main goal of FedV is to train an ML model protecting the privacy of the features provided by each party without revealing beyond what is revealed by the model itself. That is, FedV enables privacy of the input. The goal of the adversary is to infer party’s features. We now present the assumptions for each entity in the system.

We assume an honest-but-curious aggregator who correctly follows the algorithms and protocols, but may try to learn private information from the aggregated model updates. The aggregator is often times run by large companies, where adversaries may have a hard time modifying the protocol without been noticed by others.

With respect to the parties in the system, we assume a limited number of dishonest parties who may try to infer the honest parties’ private information. Dishonest parties may collude with each other to try to obtain features from other participants. In FedV, the number of such parties is bounded by \( m - 1 \) out of \( m \) parties. We also assume that the aggregator and parties do not collude.

To enable functional encryption, a TPA may be used. At the time of completion of this work, new and promising cryptosystems that remove the TPA have been proposed [1, 14]. These cryptosystems do not require a trusted TPA. If a cryptosystem that uses a TPA is used, this entity needs to be fully trusted by other entities in the system to provide functional derived keys uniquely to the aggregator.

In real-world scenarios, different sectors already have entities that can take the role of a TPA. For example, central banks of the banking industry often play a role of a fully trusted entity. In other sectors third-party companies such as consultant firms can run the TPA.

We assume that secure channels are in place; hence, man-in-the-middle and snooping attacks are not feasible. Finally, denial of service attacks and backdoor attacks [5, 11] where parties try to cause the final model to create a targeted misclassification are outside the scope of this paper.

### 3.2 Overview of FedV

FedV enables VFL without a need for any peer-to-peer communication resulting in a drastic reduction in training time and amounts of data that need to be transferred. We first overview the entities in the system and explain how they interact under our proposed two-phase secure aggregation technique that makes these results possible.

Algorithm 1 shows the operations followed by FedV. First crypto keys are obtained by all entities in the system. After that, to align the samples of each parties, a private entity resolution process as defined in [26, 42] (see section 2.1) takes place. Here, each party receives an entity resolution vector, \( \nu_i \), and shuffles its local data samples under the aggregator’s orchestration. This results in parties having all records appropriately aligned before the training phase starts.

The training process by executing the Federated Vertical Secure Gradient Descent (FedV-SecGrad) procedure, which is the core novelty of this paper. FedV-SecGrad is called at the start of each epoch to securely compute the gradient of the loss function \( E \) based on \( D \). FedV-SecGrad consists of a two-phased secure aggregation operation that enables the computation of gradients and requires the parties to perform a sample-dimension and feature-dimension encryption (see Section 4). The resulting cyphertexts are then sent to the aggregator.
with TPA-free FE schemes [1, 14], where parties collaboratively depend on the adopted FE schemes. In the TPA-free FE setting, the inference prevention module can be deployed at encryption $p$.

FedV-SecGrad update step of the ML model. We present uses stochastic gradient descent (SGD) method to illustrate the gradient-based step to update the ML model. Line 6 of Algorithm 1 aggregator can obtain the exact gradients that can be used for any aggregator then obtains the result of the corresponding inner can also be deployed at each training party. Using decryption key, generate the functional decryption key. In that scenario, the IPM decryption key to the aggregator. Notice that aggregation vectors are valid, the TPA provides the functional making sure the vectors are adequate. If the IPM concludes that Section 4.3, once the TPA gets an aggregation vector, it is inspected parties.

We observe that the gradient computations over vertically partitioned data,

\[ \loss = \frac{1}{n} \sum_{i=1}^{n} (y(i) - w^T x(i))^2, \]

where $y(i)$ is the identity function, the ML objective $f$ reduces to a linear model, which will be discussed in Section 4.1. When $g$ is not the identity function, Claim 1 covers a special class of nonlinear ML model; for example, when $g$ is the sigmoid function, our defined ML objective is a logistic classification/regression model. We demonstrate how FedV-SecGrad is extended to nonlinear models in Section 4.2. Note that in Claim 1, we deliberately omit the regularizer $R$ commonly used in an ML (see equation (1)), because common regularizers only depend on model weights $w$; it can be computed by the aggregator independently. We provide details of how logistic regression models are covered by Claim 1 in Appendix A.

4 VERTICAL TRAINING PROCESS: FEDV-SECGRAD

We now present in detail our federated vertical secure gradient descent (FedV-SecGrad) and its supported ML models, as captured in the following claim.

**Claim 1.** FedV-SecGrad is a generic approach to securely compute gradients of an ML objective with a prediction function that can be written as $f(x; w) = g(w^T x)$, where $g : \mathbb{R} \to \mathbb{R}$ is a differentiable function, $x$ and $w$ denote the feature vector and the model weights vector, respectively.

ML objective defined in Claim 1 covers many classical ML models including nonlinear models, such as logistic regression, SVMs, etc. When $g$ is the identity function, the ML objective $f$ reduces to a linear model, which will be discussed in Section 4.1. When $g$ is not the identity function, Claim 1 covers a special class of nonlinear ML model; for example, when $g$ is the sigmoid function, our defined ML objective is a logistic classification/regression model. We demonstrate how FedV-SecGrad is extended to nonlinear models in Section 4.2. Note that in Claim 1, we deliberately omit the regularizer $R$ commonly used in an ML (see equation (1)), because common regularizers only depend on model weights $w$; it can be computed by the aggregator independently. We provide details of how logistic regression models are covered by Claim 1 in Appendix A.

**4.1 FedV-SecGrad for Linear Models**

We first present FedV-SecGrad for linear models, where $g$ is the identity function and the loss is the mean-squared loss. The target loss function then becomes $\loss = \frac{1}{n} \sum_{i=1}^{n} (y(i) - x(i))^2$, where $y(i)$ is the identity function, the ML objective $f$ reduces to a linear model, which will be discussed in Section 4.1. We observe that the gradient computations over vertically partitioned data, $\nabla \loss(w)$, can be reduced to two types of operations: (i) feature-dimension aggregation and (ii) sample/batch-dimension aggregation. To perform these two operations, FedV-SecGrad follows a two-phased secure aggregation (2Phased-SA) process. Specifically, the feature dimension SA securely aggregates several batches of training data that belong to different parties in feature-dimension to acquire the value of $y(i) - x(i)$ for each data sample as illustrated in (3), while the sample dimension SA can securely aggregate one batch of training data owned by one party in sample-dimension with the weight of $y(i) - x(i)$ for each sample, to obtain the batch gradient $\nabla \loss_B(w)$. The communication between the parties and the aggregator is a one-way interaction requiring a single message.
We use a simple case of two parties to illustrate the proposed protocols, where $p_1$ is the active party and $p_2$ is a passive party. Recall that the training batch size is $s$ and the total number of features is $d$. Then the current training batch samples for $p_1$ and $p_2$ can be denoted as $y^{(1)}_{p_1} x^{(m)}_{p_1}$ and $y^{(1)}_{p_2} x^{(d-m)}_{p_2}$ as follows:

\[
\begin{bmatrix}
    y^{(1)}_{p_1} & x^{(1)}_{p_1} & \cdots & x^{(m)}_{p_1} \\
    \vdots & \vdots & \ddots & \vdots \\
    y^{(s)}_{p_1} & x^{(s)}_{p_1} & \cdots & x^{(d)}_{p_1}
\end{bmatrix}
\]

\[
\begin{bmatrix}
    y^{(1)}_{p_2} & x^{(1)}_{p_2} & \cdots & x^{(m)}_{p_2} \\
    \vdots & \vdots & \ddots & \vdots \\
    y^{(s)}_{p_2} & x^{(s)}_{p_2} & \cdots & x^{(d)}_{p_2}
\end{bmatrix}
\]

**Feature dimension SA.** The goal of feature dimension SA is to securely aggregate the sum of a group of ‘partial models’ $x^{(i)}_{p_i}$ from multiple parties without disclosing the inputs to the aggregator. Taking the $s^{th}$ data sample in the batch as an example, the aggregator is able to securely aggregate $\sum_{k=1}^{m} w_k x^{(s)}_{k} = y^{(s)} + \sum_{k=m+1}^{d} w_k x^{(s)}_{k}$. For this purpose, the active process and all other passive parties perform slightly different pre-processing steps before invoking FedV-SecGrad. The active party, $p_1$, appends a vector with labels $y$ to obtain $x^{(i)}_{p_1} w_{p_1} = y^{(i)}$ as its ‘partial model’. For the passive party $p_2$, its ‘partial model’ is defined by $x^{(i)}_{p_2} w_{p_2}$. Each party $p_1$ encrypts its ‘partial model’ using the MIFE encryption algorithm with its public key $pk_{p_1}^MIFE$, and sends it to the aggregator.

Once the aggregator receives the partial models, it prepares a fusion vector $\mathbf{v}$ of size equal to the number of parties to perform the aggregation and sends it to the TPA to request a function key $sk_{MIFE}^\mathbf{v}$. With the received key $sk_{MIFE}^\mathbf{v}$, the aggregator can obtain the aggregated sum of the elements of $w_{p_1}^{x^{(m)}_{p_1}} w_{p_2}^{x^{(d-m)}_{p_2}} - y^{x^{(s)}}$ and $w_{p_2}^{x^{(d-m)}_{p_2}}$ in the feature dimension.

It is easy to extend the above protocol to a general case with $n$ parties. In this case, the fusion vector $\mathbf{v}$ can be set as a binary vector with $n$ elements, where one indicates that the aggregator has received the replies from the corresponding party, and zero indicates otherwise. In this case, the aggregator gives equal fusion weights to all the replies for the feature dimension aggregation. We discuss the case where only a subset of parties replies in detail in Section 4.3.3.

**Sample dimension SA.** The goal of the sample dimension SA is to securely aggregate the batch gradient. For example, considering the first feature weight $w_1$ for data sample owned by $p_1$, the aggregator is able to securely aggregate $\mathbf{V}(w_1) = \sum_{k=1}^{d} x^{(k)}_{1} u_k$ via sample dimension SA where $\mathbf{u}$ is the aggregation result of feature dimension SA discussed above. This SA protocol requires the party to encrypt its batch samples using the SIFE cryptosystem with its public key $pk_{MIFE}^\mathbf{u}$. Then, the aggregator exploits the results of the feature dimension SA, i.e., an element-related weight vector $\mathbf{u}$ to request a function key $sk_{MIFE}^\mathbf{u}$ from the TPA. With the function key $sk_{MIFE}^\mathbf{u}$, the aggregator is able to decrypt the ciphertext and acquire the batch gradient $\mathbf{V}(\mathbf{u})$.

**Detailed Execution of the FedV-SecGrad Process.** As shown in Algorithm 1, the general FedV adopts a mini-batch based SGD algorithm to train a ML model in a VFL setting. After system setup, all parties use the random seed provided by the TPA to generate a one-time-password sequence [25] that will be used to generate batches during the training process. Then, the training process can begin.

At each training epoch, the FedV-SecGrad approach specified in Procedure 2 is invoked in line 5 of Algorithm 1. The aggregator queries the parties with the current model weights, $w_{p_i}$. To reduce data transfer and protect against inference attacks, the aggregator only sends each party the weights that pertain to its partial feature set. We denote these partial model weights as $w_{p_i}$ in line 2.

In Algorithm 1, each party uses a random seed $r$ to generate its one-time password chain. For each training epoch, each party uses the one-time-password chain associated with the training epoch to randomly select the samples that are going to be included in a batch for the given batch index, as shown in line 20. In this way, the aggregator never gets to know what samples are included in a batch, thus preventing inference attacks (see Section 5).

Then each party follows the feature-dimension and sample-dimension encryption process shown in lines 22, 23 and 24 of Procedure 2, respectively. As a result, each party’s local ‘partial model’ is encrypted and the two ciphertexts, $c_{id}$ and $c_{ed}$, are sent back to the aggregator. The aggregator waits for a pre-defined duration for parties’ replies, denoted as two sets of corresponding ciphertexts $c_{id}$ and $c_{ed}$. Once this duration has elapsed, it continues the training process by performing the following secure aggregation steps. First, the feature dimension SA, is performed. For this purpose, in line 4, vector $\mathbf{v}$ is initialized with all-one vector and is updated to zeros for not responding parties, as in line 10. This vector provides the weights for the inputs of the received encrypted ‘partial models’. Vector $\mathbf{v}$ is sent to the TPA that verifies the suitability of the vector (see Section 4.3). If $\mathbf{v}$ is suitable, the TPA returns the private key $dk_{\mathbf{v}}$ to perform the decryption. The feature dimension SA, is completed in line 13, where the MIFE-based decryption takes place resulting in $\mathbf{u}$ that contains the aggregated weighted feature values of $s$-th batch samples. Then the sample dimension SA, is taken place, where the aggregator uses $\mathbf{u}$ as an aggregation vector and sends it to the TPA to obtain a functional key $dk_{\mathbf{u}}$. The TPA verifies the validity of $\mathbf{u}$ and returns the key if appropriate (see Section 4.3). Finally, the aggregated gradient update $\mathbf{V}(\mathbf{e}_{fitness}(\mathbf{w}))$ is computed as in lines 16 and 17 by performing a SIFE decryption using $dk_{\mathbf{u}}$.

**4.2 FedV for Non-Linear Models**

In this section, we extend FedV-SecGrad to compute gradients of non-linear models, i.e., when $g$ is not the identity function in Claim 1, without the help of Taylor approximation. For non-linear models, FedV-SecGrad requires the active party to share labels with the aggregator in plaintext. Since $g$ is not the identity function and may be non-linear, the corresponding gradient computation does not consist only linear operations. We present the differences between Procedure 2 and FedV-SecGrad for non-linear models in Procedure 3. Here, we briefly analyze the extension on logistic models and SVM models. More details can be found in Appendix A.

**Logistic Models.** We now rewrite the prediction function $f(x;\mathbf{w}) = \frac{1}{1+e^{-\mathbf{w}^T x}}$ as $g(\mathbf{w}^T x)$, where $g(\cdot)$ is the sigmoid function, i.e., $g(z) = \frac{1}{1+e^{-z}}$. If we consider classification problem and hence use cross-entropy loss, the gradient computation over a mini-batch $B$ of size

\[\text{In this type of attack, a party may try to find out if its features are more important than those of other parties. This can be easily inferred in linear models.} \]
FedV: Privacy-Preserving Federated Learning over Vertically Partitioned Data

4.3 Enabling Dynamic Participation in FedV and Inference Prevention

In some applications, parties may have glitches in their connectivity that momentarily inhibit their communication with the aggregator. The ability to easily recover from such disruptions, ideally without losing the computations from all other parties, would help reduce the training time. FedV allows a limited number of non-active parties to dynamically drop out and re-join during the training phase. This is possible because FedV requires neither sequential peer-to-peer communication among parties nor re-keying operations when restricting label sharing, where the logic construction is transferred to linear computation via Taylor approximation, as used in existing VFL solutions [26]. Detailed specifications of the above approaches are provided in Appendix A.

SVMs with Kernels. SVM with kernel is usually used when data is not linearly separable. We first discuss linear SVM model. When it uses squared hinge loss function and its objective is to minimize \[ \frac{1}{n} \sum_{i=1}^{n} \max(0, 1 - y_i (\mathbf{w}^T \mathbf{x}_i) + \theta) \]. The gradient computation over a mini-batch B of size s can be described as \( \nabla E_B(\mathbf{w}) = \frac{1}{s} \sum_{i \in B} (y_i - \mathbf{w}^T \mathbf{x}_i) \mathbf{x}_i \). With the provided labels and acquired \( \mathbf{w}(\mathbf{x}_i) \), Line 14 of Procedure 3 can be updated so that the aggregator computes \( u_i = -2y_i (\max(0, 1 - y_i (\mathbf{w}^T \mathbf{x}_i))) \mathbf{x}^T \mathbf{x}_i \). Now let us consider the case where SVM uses non-linear kernels. Suppose the prediction function is \( f(x; w) = \sum_{i=1}^{n} w_i k(x_i, x) \), where \( k(\cdot) \) denotes the corresponding kernel function. As non-linear kernel functions, such as polynomial kernel \( (x^T y)^d \), sigmoid kernel \( \tanh(\beta x^T y + \theta) \) (\( \beta \) and \( \theta \) are kernel coefficients), are based on inner-product computation which is supported by our feature dimension SA and sample dimension SA protocols, these kernel matrices can be computed before the training process begins. And the aforementioned objective for SVM with non-linear kernels will be reduced to SVM with linear kernel case with the pre-computed kernel matrix. Then the gradient computation process for these SVM models will be reduced to a gradient computation of a standard linear SVM, which can clearly be supported by FedV-SecGrad.

s can be described as \( \nabla E_B(\mathbf{w}) = \frac{1}{s} \sum_{i \in B} (y_i - \mathbf{w}^T \mathbf{x}_i) \mathbf{x}_i \). The aggregator is able to acquire \( z^{(i)} = \mathbf{w}^T \mathbf{x}^{(i)} \) following the feature dimension SA process. With the provided labels, it can then compute \( u_i = g(z) - y^{(i)} \) as in line 14 of Procedure 3. Note that line 14 is specific for the adopted cross-entropy loss function. If another loss function is used, we need to update line 14 accordingly. Finally, sample dimension SA is applied to compute \( \nabla E_B(\mathbf{w}) = \sum_{i \in B} z^{(i)} \). FedV-SecGrad also provides an alternative approach for the case of
a party drops. To overcome missing replies, FedV allows the aggregator to set the corresponding element in $u$ as zero (Procedure 2, line 10).

**Inference Threats and Prevention Mechanisms.** The dynamic nature of the inner product aggregation vector in Procedure 2, line 10, may enable the inference attacks below, where the aggregator is able to isolate the inputs from a particular party. We analyze two potential inference threats and show how FedV design is resilient against them.

An honest-but-curious aggregator may be able to analyze the traces where some parties drop off; in this case, the resulting aggregated results will uniquely include a subset of replies making it easier to infer the input of a party. This attack is defined as follows:

**Definition 4.1 (Inference Attack).** An inference attack carried by an adversary to infer party $p_i$ input $w_{p_i,x^{(i)}}$ or party’s local features $x^{(i)}_{p_i}$ without directly accessing them.

Here, we briefly analyze this threat from the feature and sample dimensions separately, and show how to prevent this type of attack even under the case of an actively curious aggregator. We formally prove the privacy guarantee of FedV in Section 5.

**Feature dimension aggregation inference:** To better understand this threat, let’s consider an active attack where a curious aggregator obtains a function key, $dk_{\text{exploited}}$, by a manipulated vector such as $d = (0, ..., 0, 1)$ to infer the last party’s input that corresponds to a target vector $w_{\text{exploited}}^T x^{(i)}_{p_i}$ because the inner-product $u = (w_{p_i,x^{(i)}}, a_{\text{exploited}})$ is known to the aggregator.

**Sample dimension aggregation inference:** An actively curious aggregator may decide to isolate a single sample by requesting a key that has fewer samples. In particular, rather than requesting a key for $u$ of size $s$ (Procedure 2 line 14), the curious aggregator may select a subset of $s$ samples, and in the worst case, a single sample. After the aggregation of this subset of samples, the aggregator may infer one feature value of a target data sample.

To mitigate the previous threats, the Inference Prevention Module (IPM) takes two parameters: $t$, a scalar that represents the minimum number of parties for which the aggregation is required, and $s$, which is the number of batch samples to be included in a sample aggregation. For a feature aggregation, the IPM verifies that the vector’s size is $n = |a|$, to ensure it is well formed according to Procedure 2, line 31. Additionally, it verifies that the sum of its elements is greater than or equal to $t$ to ensure that at least the minimum tolerable number of parties’ replies are aggregated. If these conditions hold, the TPA can return the associated functional key to the aggregator. Finally, to prevent sample based inference threats, the aggregator needs to verify that vector $u$ in Procedure 2, line 34 needs to always be equal to the predefined batch size $s$. By following this procedure the IPM ensures that the described active and passive inference attacks are thwarted so as to ensure the data of each party is kept private throughout the training phase.

Another potential attack to infer the same target sample $x^{(\text{target})}$ is to utilize two manipulated vectors in subsequent training batch iterations, for example, $d^\text{exploited}_{\text{batch i}} = (1, ..., 1, 1)$ and $d^\text{exploited}_{\text{batch i+1}} = (1, ..., 1, 0)$ in training batch iteration $i$ and $i+1$, respectively. Given results of $(w_{x^{(\text{target})}}, p^\text{batch i}_{\text{exploited}})$ and $(w_{x^{(\text{target})}}, p^\text{batch i+1}_{\text{exploited}})$, in theory the curious aggregator could subtract the latter one from the first to infer the target sample. The IPM cannot prevent this attack, hence, we incorporate a random-batch selection process to address it.

FedV incorporates a random-batch selection process that makes it resilient against this threat. In particular, we incorporate randomness in the process of selecting data samples ensuring that the aggregator does not know if one sample is part of a batch or not. Samples in each mini-batch are selected by parties according to a one-time password. Due to this randomness, data samples included in each batch can be different. Even if a curious aggregator computes the difference between two batches as described above, it cannot tell if the result corresponds to the same data sample or not, and no inference can be performed. As long as the aggregator does not know the one-time password chain used to generate batches, the aforementioned attack is not possible. In summary, it is important for the one-time password to be kept secret by all parties from the aggregator.

**5 SECURITY AND PRIVACY ANALYSIS**

Recall that the goal of FedV is to train an ML model protecting the privacy of the features provided by each party without revealing beyond what is revealed by the model itself. In other words, FedV protects the privacy of the input. In this section, we formally prove the security and privacy guarantees of FedV with respect to this goal. First, we introduce the following lemmas with respects to the security of party input in the secure aggregation, security and randomness of one-time password (OTP) based seed generation, and solution of the non-homogeneous system to assist the proof of privacy guarantee of FedV as shown in Theorem 5.4.

**Lemma 5.1 (Security of Party Input).** The encrypted party’s input in the secure aggregation of FedV has ciphertext indistinguishability and is secure against adaptive corruptions under the classical DDH assumption.

The formal proof of Lemma 5.1 is presented in the functional encryption schemes [2, 3]. Under the DDH assumption, given encrypted input $\mathcal{E}_{FE}.Enc(w_{x^{(s)}})$ and $\mathcal{E}_{FE}.Enc(x)$, there is adversary has non-negligible advantage to break the $\mathcal{E}_{FE}.Enc(w_{x^{(s)}})$ and $\mathcal{E}_{FE}.Enc(x)$ to directly obtain $w_{x^{(s)}}$ and $x$, respectively.

**Lemma 5.2 (Solution of a Non-Homogeneous System).** A non-homogeneous system is a linear system of equations $Ax = b$ s.t. $b \neq 0$, where $A \in \mathbb{R}^{m \times n}$, $b, x \in \mathbb{R}^n$. $Ax = b$ is consistent if and only if the rank of the coefficient matrix $\text{rank}(A)$ is equal to the rank of the augmented matrix $\text{rank}(A; b)$, while $Ax = b$ has only one solution if and only if $\text{rank}(A) = \text{rank}(A; b) = n$.

**Lemma 5.3 (Security and Randomness of OTP-based Seed Generation).** Given a predefined party group $\mathcal{P}$ with OTP setting, we have the following claims: SECURITY-except for the released seeds, $\forall p \notin \mathcal{P}$, $p$ cannot infer the next seed based on released seeds; RANDOMNESS-$\forall p_1, p_2 \in \mathcal{P}, p_1$ can obtain a synchronized and sequence-related one-time seed without peer-to-peer communication with other parties.

The Lemma 5.2 is derived from the conclusion of the Rouché-Capelli theorem [43]. Hence, we do not present the specific proof.
Theorem 5.4 (Privacy Guarantee of FedV). Under the threat models defined in Section 3.1, FedV can protect the privacy of the parties’ input under the inference attack in Definition 4.1.

Proof of Theorem 5.4. We prove the theorem by hybrid games to simulate the inference activities of a PPT adversary $A$.

**Game G0:** $A$ obtains an encrypted input $Enc(x)$ to infer $x$;

**Game G1:** $A$ observes the randomness of one round of batch selection to infer the next round of batch selection;

**Game G2:** $A$ collects a triad of encrypted input, aggregation weight and inner-product, $(Enc(x), w, ⟨w, x⟩)$, to infer $x$, $w, x \in \mathbb{R}^n$;

**Game G3:** $A$ collects a set $S(Enc(x_{\text{target}}), w(x_{\text{target}}))$ to infer $x_{\text{target}}$.

Here, we analyze each inference game and the hybrid cases. According to Lemma 5.1, $A$ does not have non-negligible advantage to infer $x$ by breaking $Enc(x)$. As we have proved in Lemma 5.3, in game $G1$, $A$ also does not have non-negligible advantage to infer the next round of batch selection. Here, the combination of game $G0$ or $G1$ with other games does not increase the advantage of $A$.

In game $G2$, suppose that $A$ has a negligible advantage to infer $x$. Then, $G2$ can be reduced to that $A$ has a negligible advantage to solve a non-homogeneous system, $w^T x = b$. Here we consider three cases:

**Case C1:** Except for directly solving the $w^T x = b$ system, $A$ has no extra advantage. According to Lemma 5.2, if $w^T x = b$ has one confirmed solution, it requires that $n = 1$. In FedV, the number of features and the batch size setting are greater than one. Thus, $A$ cannot solve the non-homogeneous system.

**Case C2:** Based on $C1$, $A$ could be an aggregator, where $A$ can manipulate a weight vector $w_{\text{exploited}}$ s.t. $w_i = 1$, $\forall j \in [n], j \neq i$, $w_j = 0$ to infer $x_i \in x$. However, FedV does not allow the functional key generation using $w_{\text{exploited}}$ due to IPM setting. Without functional decryption key, $A$ cannot acquire the inner-product, i.e., $b$, in the non-homogeneous system. In this case, $w_{\text{exploited}}^T x = b$ has multiple solutions, and hence $x_i$ cannot be confirmed.

**Case C3:** Based on $C1$, $A$ could be a group of colluding parties, where $A$ also have learned part of information of $x$. Then, the inference task is reduced to solve $w^T x = b'$ system. According to Lemma 5.2, it requires that $|x| = 1$ to have one solution if and only if colluding parties learn $w$. In the threat model of FedV, aggregator is assumed not colluding with parties in the aggregation process, and hence such a condition is not satisfied. Thus, $A$ cannot solve $w^T x = b'$ system.

In short, $A$ cannot solve the non-homogeneous system and hence $A$ does not have a non-negligible advantage to infer $x$ in game $G2$.

Game $G3$ is a variant of game $G2$, where $A$ collects a set of triads as shown in game $G2$ for a target data sample $x_{\text{target}}$. With enough triads, $A$ can reduce the inference task to the task of constructing a non-homogeneous system of $Wx_{\text{target}} = b$, s.t., $\text{rank}(W) = \text{rank}(W, b) = n$ as illustrated in Lemma 5.2. Here we also consider two cases: (i) $A$ could be the aggregator, however, FedV employs the OTP-based seed generation mechanism to chose the samples for each training batch. According to game $G1$, $A$ does not have a non-negligible advantage to observer and infer random batch selection. (ii) $A$ could be the colluding parties, then it is reduced to $G2$ case $C3$. As a result, $A$ still cannot construct a non-homogeneous system to solve $x_{\text{target}}$.

Based on the above simulation games, $A$ does not have the non-negligible advantage to infer the private information defined in Definition 4.1. Thus, the privacy guarantee of FedV is proved. □

Remark. According to our threat model and FedV design, labels are kept fully private for linear models by encrypting them during the feature dimension secure aggregation (Procedure 2 line 22). For non-linear models, a slightly different process is involved. In this case, the active party shares the label with the aggregator to avoid costly peer-to-peer communication. Sharing labels, in this case, does not compromise the privacy of the features of other parties for two reasons. First, all the features are still encrypted using the feature dimension scheme. Secondly, because the aggregator does not know what samples are involved in each batch (OTP-based seed generation induced randomness and security), it cannot perform either of the previous inference attacks.

In conclusion, FedV protects the privacy of the features provided by all parties.

6 EVALUATION

To evaluate the performance of our proposed framework, we compare FedV with the following baselines:

(i) **Hardy**: we use the VFL proposed in [26] as the baseline because it is the closest state-of-the-art approach. In [26], the trained ML model is a logistic regression (LR) and its secure protocols are built using additive homomorphic encryption (HE). Like most of the additive HE based privacy-preserving ML solutions, the SGD and loss computation in [26] relies on the Taylor series expansion to approximately compute the logistic function.

(ii) **Centralized baselines**: we refer to the training of different ML models in a centralized manner as the centralized baselines. We train multiple models including an LR model with and without Taylor approximation, a basic linear regression model with mean squared loss and a linear Support Vector Machine (SVM).

Theoretical Communication Comparison. Before presenting the experimental evaluation, we first theoretically compare the number of communications between the proposed FedV with respect to Hardy. Suppose that there are $n$ parties and one aggregator in the VFL framework. As shown in Table 2, in total, FedV reduces the number of communications during the training process from $4n - 2$ for [26] to $n$, while reducing the number of communications during the loss computation (see Appendix B for details) from $(n^2 - 3n)/2$ to $n$. In FedV, the number of communications and loss computation phase is linear to the number of parties.

6.1 Experimental Setup

To evaluate the performance of FedV, we train several popular ML models including linear regression, logistic regression, Taylor approximation based logistic regression, and linear SVM to classify several publicly available datasets from UCI Machine Learning Repository [20], including website phishing, ionosphere, landsat satellite, optical recognition of handwritten digits (optdigits), and MNIST...
which is a performance intensive computation, we use the same Hardy as for each iteration in the VFL. Performance of FedV for Logistic Regression. We trained two FedV to Procedure 3, referred as FedV models with FedV on a 2.3 GHz 8-Core Intel Core i9 platform with 32 GB of RAM. Both 

Table 2: Number of required crypto-related communication for each iteration in the VFL.

| Communication          | Hardy et al [26] | FedV |
|------------------------|------------------|------|
| Secure Stochastic Gradient Descent | $2n$ | $n$ |
| aggregator ↔ parties   |                  |      |
| parties ↔ parties      | $2(n - 1)$ | $0$  |
| TOTAL                  | $2(n - 1)$ | $n$  |

[32]. Each dataset is partitioned vertically and equally according to the numbers of parties in all experiments. The number of attributes of these datasets is between 10 and 784, while the total number of sample instances is between 351 and 70000, and the details can be found in Table 3 of Appendix D. Note that we use the same underlying logic used by the popular Scikit-learn ML library to handle multi-class classification models, we convert the multi-label datasets into binary label datasets, which is also the strategy used in the comparable literature [26].

Implementation. We implemented Hardy, our proposed FedV and several centralized baseline ML models in Python. To achieve the integer group computation that is required by both the additive homomorphic encryption and the functional encryption, we employ the gmpy2 library 5. We implement the Paillier cryptosystem for the construction of an additive HE scheme; this is the same as the one used in [26]. The constructions of MIFE and SIFE are from [2] and [3], respectively. As these constructions do not provide the solution to address the discrete logarithm problem in the decryption phases, which is a performance intensive computation, we use the same hybrid approach that was used in [56]. Specifically, to compute $f$ in $h = g^f$, we setup a hash table $T_{h,g,b}$ to store $(h, f)$ with a specified $g$ and a bound $b$, where $-b \leq f \leq b$, when the system initializes. When computing discrete logarithms, the algorithm first looks up $T_{h,g,b}$ to find $f$, the complexity for which is $O(1)$. If there is no result in $T_{h,g,b}$, the algorithm employs the traditional baby-step giant-step algorithm [44] to compute $f$, the complexity for which is $O(n^2)$.

Experimental Environment. All the experiments are performed on a 2.3 GHz 8-Core Intel Core i9 platform with 32 GB of RAM. Both Hardy and our FedV frameworks are distributed among multiple processes, where each process represents a party. The parties and the aggregator communicate using local sockets; hence the network latency is not measured in our experiment.

6.2 Experimental Results

As Hardy only supports two parties to train a logistic regression model, we first present the comparison results for that setting. Then, we explore the performance of FedV using different ML models. Lastly, we study the impact of varying number of parties in FedV.

Performance of FedV for Logistic Regression. We trained two models with FedV: 1) a logistic regression model trained according to Procedure 3, referred as FedV; and 2) a logistic regression model with Taylor series approximation, which reduces the logistic regression model to a linear model, trained according to Procedure 2 and referred as FedV with approximation. We also trained a centralized version (non-FL setting) of a logistic regression with and without Taylor series approximation, referred as centralized LR and centralized LR (approx.), respectively. We also present the results for Hardy.

Figure 3 shows the test accuracy and training time of each approach to train the logistic regression on different datasets. Results show that both of our FedV and FedV with approximation can achieve a test accuracy comparable to those of the Hardy and the centralized baselines for all four datasets. With regards to the training time, FedV and FedV with approximation efficiently reduce the training time by 10% to 70% for the chosen datasets with 360 total training epochs. For instance, as depicted in Figure 3, FedV can reduce around 70% training time for the ionosphere dataset while reducing around 10% training time for the sat dataset. The variation in training time reduction among different datasets is caused by different data sample sizes and model convergence speed.

We decompose the training time required to train the LR model to understand the exact reason for such reduction. These results are shown for the ionosphere dataset. In Figure 4, we can observe that Hardy requires communication between parties and the aggregator (phase 1) and peer-to-peer communication (phase 2). In contrast, FedV does not require peer-to-peer communication, resulting in savings in training times. Additionally, it can be seen that the computational time for phase 1 of the aggregator and phase 2 of each party are significantly higher for Hardy than for FedV. We also compare and decompose the total size of data transmitted for the LR model over various datasets. As shown in Figure 5, compared to Hardy, FedV can reduce the total amount of data transmitted by 80% to 90%; this is possible because FedV only relies on non-interactive secure aggregation protocols and does not need the frequent rounds of communications used by the contrasted VFL baseline.

Performance of FedV with Different ML Models. We explore the performance of FedV using various popular ML models including linear regression and linear SVM.

The first row of Figure 6 shows the test accuracy while the second row shows the training time for a total of 360 training epochs. In general, our proposed FedV achieves comparable test accuracy for all types of ML models for the chosen datasets. Note that our FedV is based on cryptosystems that compute over integers instead of floating-point numbers, so as expected, FedV will lose a portion of fractional parts of a floating-point numbers. This is responsible for the differences in accuracy with respect to the central baselines. As expected, compared with our centralized baselines, FedV requires more training time. This is due to the distributed nature of the vertical training process.

Impact of Increasing the Number of Parties. We explore the impact of increasing number of parties in FedV. Recall that Hardy does not support more than two parties, and hence we cannot report its performance in this experiment. Figure 7a shows the accuracy and training time of FedV for collaborations varying from two to 15 parties. The results are shown for the OptDigits dataset and the trained model is a Logistic Regression. As shown in Figure 7a, the number of parties does not impact the model accuracy and finally all test cases reach the 100% accuracy. Importantly, the training time shows a linear relation to the number of parties. As reported in Figure 3, the training time of FedV in logistic regression model is very close to that of the normal non-FL logistic regression. For instance, for 100 iterations, the training time

5https://pypi.org/project/gmpy2/
for FedV with 14 parties is around 10 seconds, while the training time for normal non-FL logistic regression is about 9.5 seconds. We expect this time will increase in a fully distributed setting depending on the latency of the network.

**Performance on Image Dataset.** Figure 7b reports the training time and model accuracy for training a linear SVM model on MNIST dataset using a batch size of 8 for 100 epochs. Note that Hardy is not reported here because that approach was proposed for approximated logistic regression model, but not for linear SVM. Compared to the centralized linear SVM model, FedV can achieve comparable model accuracy. While FedV provides a strong security guarantee, the training time is still acceptable.

Overall, our experiments show reductions of 10%-70% of training time and 80%-90% transmitted data size compared to Hardy. We also showed that FedV is able to train machine learning models that the baseline cannot train (see Figure 7b). FedV final model accuracy was comparable to central baselines showing the advantages of not requiring Taylor approximation techniques used by Hardy.

### 7 RELATED WORK

FL was proposed in [31, 37] to allow a group of collaborating parties to jointly learn a global model without sharing their data [34]. Most of the existing work in the literature focus on horizontal FL while these papers address issues related to privacy and security [5, 8, 21, 23, 38, 39, 45, 49, 56], system architecture [7, 31, 36, 37], and new learning algorithms e.g., [16, 48, 60].

A few existing approaches have been proposed for distributed data mining [47, 50, 51, 58]. A survey of vertical data mining methods is presented in [50], where these methods are proposed to train specific ML models such as support vector machine [58], logistic regression [47] and decision tree [51]. These solutions are not designed to prevent inference/privacy attacks. For instance, in [58], the parties form a ring where a first party adds a random number to its input and sends it to the following one; each party adds its value and sends to the next one; and the last party sends the accumulated value to the first one. Finally, the first party removes the random number and broadcasts the aggregated results. Here, it is possible to infer private information given that each party knows intermediate and final results. The privacy-preserving decision tree model in [51] has to reveal class distribution over the given attributes, and thus may have privacy leakage. Split learning [46, 52], a new type of distributed deep learning, was recently proposed to train neural networks without sharing raw data. Although it is mentioned that secure aggregation may be incorporated during the method, no discussion on the possible cryptographic techniques were provided. For instance, it is not clear if the an applicable cryptosystem would require Taylor approximation. None of these approaches provide strong privacy protection against inference threats.

Some proposed approaches have incorporated privacy into vertical FL [12, 13, 22, 24, 26, 53, 57, 59]. These approaches are either limited to a specific model type: a procedure to train secure linear regression was presented in [22], a private logistic regression process was presented in [26], and [13] presented an approach to train XGBoost models, or suffered from the privacy, utility and communication efficiency concerns: differential privacy based noise perturbation solutions were presented in [12, 53], random noise perturbation with tree-based communication were presented in [24, 59]. There are several differences between these approaches and FedV. First, these solutions either rely on the hybrid general (garbled circuit based) secure multi-party computation approach or are built on partially additive homomorphic encryption (i.e., Paillier cryptosystem [17]). In these approaches, the secure aggregation process is inefficient in terms of communication and computation.
Finally, multiple cryptographic approaches have been proposed for secure aggregation, including (i) general secure multi-party computation techniques [10, 27, 54, 55] that are built on the garbled circuits and oblivious transfer techniques; (ii) secure computation using more recent cryptographic approaches such as homomorphic encryption and its variants [4, 6, 18, 29, 35]. However, these two kinds of secure computation solutions have limitations with regards to either the large volumes of ciphertexts that need to be transferred or the inefficiency of computations involved (i.e., unacceptable computation time). Furthermore, to lower communication overhead and computation cost, customized secure aggregation approaches such as the one proposed in [8] are mainly based on secret sharing techniques and they use authenticated encryption to securely compute sums of vectors in horizontal FL. In [56], Xu et al. proposed the use of functional encryption [9, 33] to enable horizontal FL. However, this approach cannot be used to handle the secure aggregation requirements in vertical FL.

8 CONCLUSIONS

Most of the existing privacy-preserving FL frameworks only focus on horizontally partitioned datasets. The few existing vertical federated learning solutions work only on a specific ML model and suffer from inefficiency with regards to secure computations and communications. To address the above-mentioned challenges, we have proposed FedV, an efficient and privacy-preserving VFL framework based on a two-phase non-interactive secure aggregation approach that makes use of functional encryption.

We have shown that FedV can be used to train a variety of ML models, without a need for any approximation, including logistic regression, SVMs, among others. FedV is the first VFL framework that supports parties to dynamically drop and re-join for all these models during a training phase, thus, it is applicable in challenging situations where a party may be unable to sustain connectivity throughout the training process. More importantly, FedV removes the need of peer-to-peer communications among parties, thus, reducing substantially the training time and making it applicable to applications where parties cannot connect with each other. Our experiments show reductions of 10%-70% of training time and 80%-90% transmitted data size compared to those in the state-of-the-art approaches.

REFERENCES

[1] Michel Abdalla, Fabrice Benhamouda, Markulf Kohlweiss, and Hendrik Waldner. Decentralizing inner-product functional encryption. In IACR International
A FORMAL ANALYSIS OF CLAIM 1

Here, we present our detailed proof of Claim 1. Note that we skip the discussion on how to compute $\nabla B$ in the rest of the analysis such as in equation (6) below, since the aggregator can compute it independently.

A.1 Linear Models in FedV

Here, we formally analyze the details of how our proposed FedV-SECGrad approach (also called 2Phase-SA) is applied in a vertical federated learning framework with underlying linear ML model. Suppose the a generic linear model is defined as:

$$f(\mathbf{x}; w) = w_0 x_0 + w_1 x_1 + \ldots + w_d x_d,$$

where $x_0^{(i)} = 1$ represents the bias term. For simplicity, we use the vector-format expression in the rest of the proof, described as:

$$f(\mathbf{w}^\top \mathbf{x}) = \mathbf{w}^\top \mathbf{x},$$

where $\mathbf{x} \in \mathbb{R}^{d+1}, \mathbf{w} \in \mathbb{R}^{d+1}, x_0 = 1$. Note that we omit the bias item $w_0 x_0 = 0$ in the rest of analysis as the aggregator can compute it independently. Suppose that the loss function here is least-squares function, defined as:

$$L(f(\mathbf{w}^\top \mathbf{x}), y) = \frac{1}{2} \left(f(\mathbf{w}^\top \mathbf{x}) - y\right)^2$$

and we use L2-norm as the regularization term, defined as $R(\mathbf{w}) = \frac{1}{2} \sum_{i=1}^{n} w_i^2$. According to equations (1), (4) and (5), the gradient of $E(\mathbf{w})$ computed over a mini-batch $B$ of $s$ data samples is as follows:

$$\nabla E_B(\mathbf{w}) = \nabla L_B(\mathbf{w}) + \nabla R_B(\mathbf{w})$$

$$= \frac{1}{s} \sum_{i=1}^{s} \left(w^\top x^{(i)} - y^{(i)} x^{(i)} + \nabla R_B(\mathbf{w})\right)$$

Suppose that $p_i$ is the active party with labels $y$, then secure gradient computation can be expressed as follows:

$$\nabla E = \frac{1}{s} \sum_{i=1}^{s} \left(w_i x^{(i)} - y^{(i)} x^{(i)} + \nabla R_B(\mathbf{w})\right)$$

Next, let $u^{(i)}$ be the intermediate value to represent the difference-loss for current $\mathbf{w}$ over one sample $x^{(i)}$, which is also the aggregation result of feature dimension $\mathbf{SA}$. Then, the updated gradient $\nabla E_B(\mathbf{w})$ is continually computed as follows:

$$\nabla E = \frac{1}{s} \sum_{i=1}^{s} u^{(i)} - y^{(i)} x^{(i)}$$

$$= \frac{1}{s} \sum_{i=1}^{s} u^{(i)} - y^{(i)} x^{(i)}$$

$\frac{1}{s} \sum_{i=1}^{s} u^{(i)}$ sample dimension $\mathbf{SA}$ $\frac{1}{s} \sum_{i=1}^{s} y^{(i)} x^{(i)}$ sample dimension $\mathbf{SA}$

To deal with the secure computation task of training loss as described in Algorithm 1, we only apply feature dimension $\mathbf{SA}$ approach. As the average loss function here is least-squares function, secure computation involved is as follows:

$$L_B(\mathbf{w}) = \frac{1}{s} \sum_{i=1}^{s} \left(\mathbf{w}^\top x^{(i)} - y^{(i)}\right)^2$$

Obviously, the feature dimension $\mathbf{SA}$ can satisfy the computation task in the above equation.

A.2 Generalized Linear Models in FedV

Here we formally analyze the details of applying our FedV-SECGrad approach to train generalized linear models in FedV.

We use logistic regression as an example, which has the following fitting (prediction) function:

$$f(\mathbf{x}; \mathbf{w}) = \frac{1}{1 + e^{-\mathbf{w}^\top \mathbf{x}}}$$

For binary label $y \in \{0, 1\}$, the loss function can be defined as:

$$L(f(\mathbf{x}; \mathbf{w}), y) = \begin{cases} -\log(f(\mathbf{x}; \mathbf{w})) & \text{if } y = 1 \\ -\log(1 - f(\mathbf{x}; \mathbf{w})) & \text{if } y = 0 \end{cases}$$

Then, the total loss over a mini-batch $B$ of size $s$ is computed as follows:

$$E(\mathbf{w}) = -\frac{1}{s} \sum_{i=1}^{s} \left(\log(1 + e^{\mathbf{w}^\top \mathbf{x}^{(i)}})\right)$$

The gradient is computed as follows:

$$\nabla E(\mathbf{w}) = \frac{1}{s} \sum_{i=1}^{s} \left(\frac{\partial}{\partial \mathbf{w}} \log(1 + e^{\mathbf{w}^\top \mathbf{x}^{(i)}})\right)$$

$$= \frac{1}{s} \sum_{i=1}^{s} \left(\frac{1}{1 + e^{\mathbf{w}^\top \mathbf{x}^{(i)}}} - y^{(i)} \mathbf{x}^{(i)}\right)$$

Note that we also do not include the regularization term $\lambda R(\mathbf{w})$ here for the same aforementioned reason. Here, we show two potential solutions in detail:
(i) FedV for nonlinear model (Procedure 3). Firstly, although the prediction function in (12) is a non-linear function, it can be decomposed as \( f(x; w) = g(h(x; w)) \), where:

\[
g(h(x; w)) = \frac{1}{1+e^{-h(x; w)}}, \text{ where } h(x; w) = w^T x
\]

We can see that the sigmoid function \( g(z) = \frac{1}{1+e^{-z}} \) is not a linear function, while \( h(x; w) \) is linear. We then apply our feature dimension and sample dimension secure aggregations on linear function \( h(x; w) \) instead. To be more specific, the formal expression of the secure gradient computation is as follows:

\[
\nabla E(w) = \frac{1}{2} \sum_{i=1}^{s} \left( \frac{1}{1+e^{-w^T x(i)}} - y(i) \right) x(i)
\]

Normal Computation — \( u(i) \)

\[
= \frac{1}{2} \left( \sum_{i=1}^{s} u(i) \cdot x(i) \right)_{\text{feature dim. SA}}
\]

Sample dimension SA

Note that the output of feature dimension SA is in plaintext, and hence, the aggregator is able to evaluate the sigmoid function \( g(\cdot) \) together with the labels. The secure loss can be computed as follows:

\[
E(w) = -\frac{1}{2} \sum_{i=1}^{s} \left[ y(i) - \frac{1}{2} \right] \sum_{i=1}^{s} u(i) \cdot x(i)
\]

\[
\log(1 + e^{-w^T x(i)}) \cdot \text{feature dim. SA}
\]

Normal Computation

Similar to secure gradient descent computation, however, we only have the feature dimension SA with subsequent normal computation.

(ii) Taylor approximation. FedV also supports the Taylor approximation approach as proposed in [26]. In this approach, the Taylor series expansion of function \( \log(1 + e^{-z}) = \log 2 - \frac{z}{2} + \frac{z^2}{4} + O(z^4) \) is applied to equation (13) to approximately compute the gradient as follows:

\[
\nabla E_B(w) = \frac{1}{2} \sum_{i=1}^{s} \left( -w^T x(i) - y(i) + \frac{1}{2} x(i) \right)
\]

As in equation (6), we are able to apply the 2Phase-SA approach in the secure computation of equation (20).

We also use another ML model, SVMs with Kernels, as an example. Here, we consider two cases:

(i) linear SVM model for data that is not linearly separable: Suppose that the linear SVM model uses squared hinge loss as the loss function, and hence, its objective is to minimize

\[
E(w) = \frac{1}{2} \sum_{i=1}^{s} \left( \max(0, 1 - y(i) w^T x(i)) \right)^2
\]

The gradient can be computed as follows:

\[
\nabla E = \frac{1}{2} \sum_{i=1}^{s} \left[ -2y(i) \cdot \max(0, 1 - y(i) w^T x(i)) \right] \cdot x(i)
\]

As we know, the aggregator can obtain \( w^T x(i) \) via feature dimension SA. With the provided labels and \( w(i) \cdot x(i) \), FedV-SecGrad can update Line 14 of Procedure 2 so that the aggregator computes \( u_i = -2y(i) \max(0, 1 - y(i) w(i) x(i)) \) instead.

(ii) the case where SVM uses nonlinear kernels: The prediction function is as follows:

\[
f(x; w) = \sum_{i=1}^{s} w_i y_i k(x; x_i),
\]

where \( k(\cdot) \) denotes the corresponding kernel function. As non-linear kernel functions, such as polynomial kernel \( k(x; x_i)^d \), sigmoid kernel \( \tanh(b x^T x_i + \theta) \) (\( b \) and \( \theta \) are kernel coefficients), are based on inner-product computation which is supported by our feature dimension SA and sample dimension SA protocols, these kernel matrices can be computed before the training process. And the aforementioned objective for SVM with nonlinear kernels will be reduced to SVM with linear kernel case with the pre-computed kernel matrix. Then the gradient computation process for these SVM models will be reduced to a gradient computation of a standard linear SVM, which can clearly be supported by FedV-SecGrad.

\section*{B \ SECURE LOSS COMPUTATION IN FEDV}

Unlike the secure loss computation (SLC) protocol in the contrasted VFL framework [26], the SLC approach in FedV is much simpler. Here, we use the logistic regression model as an example. As illustrated in Procedure 4, unlike the SLC in [26] that is separate and different from the secure gradient computation, the SLC here does not need additional operations for the parties. The loss result is computed by reusing the result of the feature dimension SA in the FedV-SecGrad.

\section*{C \ PROOF OF LEMMA 5.3}

Here, we present the specific proof for Lemma 5.3. Given a predefined party group \( \mathcal{P} \) with OTP setting, we have the following claims: SECURITY—except for the released seeds, \( \forall p \notin \mathcal{P}, p \) cannot infer the next seed based on released seeds; RANDOMNESS—\( \forall p_i \in \mathcal{P}, p_i \) can obtain a synchronized and sequence-related one-time seed without peer-to-peer communication with other parties.

Proof of Lemma 5.3. Since there exist various types of OPT, we adopt the hash chains-based OPT to prove the security and randomness of the OPT-based seed generation. Given the cryptographic hash function \( H \), the initial random seed \( r \) and a sequence index \( b_i \) (i.e., the batch index in the FedV training), the OTP-based seed for \( b_i \) is \( r_i = H^{(t-1)}(r) \). Note that \( H^{(t)}(r) = H(H^{(t-1)}(r)) = \ldots = H(H^{(t-1)}(r)) \). Next, the OTP-based seed for \( b_{i+1} \) is \( r_{i+1} = H^{(t-1)}(r) \), and hence we have \( r_i = H(r_{i+1}) \). Given the \( r_i \) at training index \( b_i \), suppose that an adversary has a non-negligible advantage to infer the next \( r_{i+1} \), the adversary needs to find a way of calculating the inverse function \( H^{-1} \). Since the cryptographic hash function should be one-way and proved to be computationally intractable according to the adopted schemes, the adversary does not have the
non-negligible advantage to infer the next seed. With respect to the randomness, the initial seed \( r \) is randomly selected, and the follow-up computation over \( r \) is a sequence of the hash function, which does not break the randomness of \( r \).

\[ \square \]

**D  DATASET DESCRIPTION**

As shown in Table 3, we present the dataset we used and the division of training set and test set.

**E  FUNCTIONAL ENCRYPTION SCHEMES**

**E.1 Single-input FEIP Construction**

The single-input functional encryption scheme for the inner-product function \( f(x, y) \) is defined as follows:

\[
E_{\text{SIFE}} = (E_{\text{SIFE}.\text{Setup}}, E_{\text{SIFE}.\text{DKGen}}, E_{\text{SIFE}.\text{Enc}}, E_{\text{SIFE}.\text{Dec}}).
\]

Each of the algorithms is constructed as follows:

- \( E_{\text{SIFE}.\text{Setup}}(1^\lambda, \eta) \): The algorithm first generates two samples as \((G, p, g) \leftarrow \text{GroupGen}(1^\lambda)\), and \( s = (s_1, ..., s_\eta) \leftarrow \mathcal{R}_{\mathbb{Z}_p}^\eta\) on the inputs of security parameters \( \lambda \) and \( \eta \), and then sets \( pp = (g, h_i = g^{s_i})_{i \in [1, ..., \eta]} \) and \( msk = s \). It returns the pair \((pp, msk)\).

- \( E_{\text{SIFE}.\text{DKGen}}(msk, y) \): The algorithm outputs the function derived key \( dk_y = (y, s) \) on the inputs of master secret key \( msk \) and vector \( y \).

- \( E_{\text{SIFE}.\text{Enc}}(pp, x) \): The algorithm first chooses a random \( r \leftarrow \mathcal{R}_{\mathbb{Z}_p} \) and computes \( ct_0 = g^r \). For each \( i \in [1, ..., \eta] \), it computes \( ct_i = h_i^{r_i} \cdot g^{s_i} \). Then the algorithm outputs the ciphertext \( ct = (ct_0, ct_i)_{i \in [1, ..., \eta]} \).

- \( E_{\text{SIFE}.\text{Dec}}(pp, ct, dk_y, y) \): The algorithm takes the ciphertext \( ct \), the public key \( msk \) and functional key \( dk_y \) for the vector \( y \), and returns the discrete logarithm in basis \( g \), i.e., \( g^{|y|} = \prod_{i \in [1, ..., n]} c_i^{\eta_i} / c_0^{dk_y} \).

**E.2 Multi-input FEIP Construction**

The multi-input functional encryption scheme for the inner-product function \( f_{\text{MIFE}}(x_1, ..., x_n, y) \) is defined as follows:

\[
E_{\text{MIFE}} = (E_{\text{MIFE}.\text{Setup}}, E_{\text{MIFE}.\text{SKDist}}, E_{\text{MIFE}.\text{DKGen}}, E_{\text{MIFE}.\text{Enc}}, E_{\text{MIFE}.\text{Dec}})
\]

The specific construction of each algorithm is as follows:

- \( E_{\text{MIFE}.\text{Setup}}(1^\lambda, \eta, n) \): The algorithm first generates secure parameters as \( G = (G, p, g) \leftarrow \text{GroupGen}(1^\lambda) \), and then generates several samples as \( a \leftarrow \mathcal{R}_{\mathbb{Z}_p}^n \), \( \forall i \in [1, ..., n] : W_i = s_i^{s_i^2} \), \( u_i = s_i^{2^\eta} \). Then, it generates the master public key and master private key as \( mpk = (G, g^a, g^W) \), \( msk = (W, (u_i)_{i \in [1, ..., n]}) \).

- \( E_{\text{MIFE}.\text{SKDist}}(mpk, msk, id_i) \): It looks up the existing keys via \( id_i \) and returns the party secret key as \( sk_i = (G, g^{a_i}, (W_a)_i, u_i) \).

- \( E_{\text{MIFE}.\text{DKGen}}(mpk, msk, y) \): The algorithm first partitions \( y \) into \((y_1, |y_2|, ..., |y_n|)\), where \( |y_i| \) is equal to \( \eta_i \). Then it generates the function derived key as follows: \( dk_y = (d_1^{y_i}, W_i, z \leftarrow \sum y_i u_i) \).

- \( E_{\text{MIFE}.\text{Enc}}(sk_i, x_i) \): The algorithm first generates a random nonce \( r_i \leftarrow \mathcal{R}_{\mathbb{Z}_p} \), and then computes the ciphertext as follows: \( ct_i = (t_i \leftarrow g^{ar_i}, c_i \leftarrow g^x \cdot g^y \cdot g^W a_i r_i) \).

- \( E_{\text{MIFE}.\text{Dec}}(ct, dk_y) \): The algorithm first calculates as follows: \( C = \prod_{i \in [1, ..., n]} (y_i |c_i| / d_i^{dk_y}) \), and then recovers the function result as \( f((x_1, x_2, ..., x_n), y) = \log_g(C) \).