EFMVFL: An Efficient and Flexible Multi-party Vertical Federated Learning without a Third Party

YIMIN HUANG, WANWAN WANG, XINGYING ZHAO, YUKUN WANG, XINYU FENG, HAO HE, and MING YAO, InsightOne Tech Co, Ltd, China

Federated learning (FL) is a machine learning setting which allows multiple participants collaboratively to train a model under the orchestration of a server without disclosing their local data. Vertical federated learning (VFL) is a special structure in FL. It handles the situation where participants have the same ID space but different feature spaces. In order to guarantee the security and privacy of the local data of each participant, homomorphic encryption (HE) is often used to transmit intermediate parameters or data during the training process. In most VFL frameworks, a trusted third-party server is necessary because the plaintexts of the parameters need to be revealed for the computation. However, it is hard to find such a credible entity in the real world. Existing methods for solving this problem are either communication-intensive or unsuitable for multi-party scenarios. By combining secret sharing (SS) and HE, we propose a novel VFL framework without any trusted third parties called EFMVFL. It allows intermediate parameters to be transmitted among multiple parties without revealing the plaintexts. EFMVFL is applicable to generalized linear models (GLMs) and supports flexible expansion to multiple participants. Extensive experiments under Logistic Regression and Poisson Regression show that our framework is outstanding in communication (reduced by 3.2×–6.8×) and efficiency (accelerated by 1.6×–3.1×).

CCS Concepts: • Computing methodologies → Machine learning; • Security and privacy → Privacy-preserving protocols;

Additional Key Words and Phrases: Federated learning, privacy protection, generalized linear models, secret sharing, homomorphic encryption

ACM Reference format:
Yimin Huang, Wanwan Wang, Xingying Zhao, Yukun Wang, Xinyu Feng, Hao He, and Ming Yao. 2023. EFMVFL: An Efficient and Flexible Multi-party Vertical Federated Learning without a Third Party. ACM Trans. Knowl. Discov. Data. 18, 3, Article 51 (December 2023), 20 pages. https://doi.org/10.1145/3627993

1 INTRODUCTION

With the improvement of data security and privacy protection laws, worldwide enterprises face the dual requirements of solving data silos problems and protecting data privacy. Federated learning (FL) was proposed by McMahan et al. [14] for collaboratively training machine learning models on decentralized data of separated parties. When FL was first proposed, it was mainly applied in
the scenario that the data is horizontal distributed on clients [11, 13, 14, 17]. Horizontal distributed data means that clients share the same feature space, but few sample IDs are overlapped among them. Then Qiang Yang et al. [23] extended the concept of FL to horizontal federated learning (HFL), vertical federated learning (VFL), and federated transfer learning (FTL). In VFL, two parties share the nearly overlapped sample ID space but differ in feature space, as shown in Figure 1. As VFL can improve the performance of machine learning models by expanding the feature dimension of the samples, it is applied in many scenarios such as medicine, finance, advertising, and marketing. We will focus on the efficiency and security of VFL in this article.

During the training process of FL, intermediate gradients or parameters are transmitted instead of original data among participants. However, it has been proved that the local data could be inferred by these intermediate results [30]. Therefore, cryptographic techniques were introduced into FL, e.g., Secret Sharing (SS) [29], Homomorphic Encryption (HE) [5], Differential privacy (DP) [19], and so on. HE allows encrypted data to be processed blindly with agnostic original data. This can protect the intermediate results from other clients. Therefore, HE has been widely used in privacy-preserving generalized linear models (GLMs) (e.g., Linear Regression [23], Logistic Regression (LR) [3, 7, 10]), tree-based models (e.g., Decision Tree [22], eXtreme Gradient Boosting (XGB) [4], Random Forest [1]) and deep learning models [12, 26, 27]. In most HE-based works, a third party is often needed to assist in generating HE key pairs and decrypting ciphertexts. In this way, the third party is able to get the plaintext information related to model parameters, which requires the third party to be fair and credible and can not collude with other parties. Such a credible entity is difficult to find in practice.

A series of VFL frameworks without a third party were proposed to tackle the above problem. They can be roughly classified into three categories, i.e., HE-based solutions [24, 28], SS-based solutions [8, 21, 29] and HE-SS-based solutions [3, 6]. However, there are still some issues:

1. HE-based methods usually allow one of the participants to generate HE key pairs and perform decryption operations. This participant may be the party with the data label [24], or it may be the party with the most features [28]. In fact, this is equivalent to the third party colluding with one of the other parties. It is not safe all the time. So in [24], some restrictions on the number of training iterations are proposed, which result in the loss of model effectiveness. A possible solution to this problem is that all participants generate HE key pairs and send the public key to each other, but it only supports two participants. Expanding to three or more participants will take a lot of work.

2. SS-based methods can support multi-party modeling easily. While this kind of method would introduce enormous communication overhead. During the training process, all of the training data and model parameters are shared through an SS algorithm.
(3) HE-SS-based methods combine SS and HE together in VFL. The works of [3] and [6] both focused on vertical federated LR. Ref. [3] first uses SS to protect model parameters and then combines with HE to complete the training. While [6] first uses HE and then combines it with SS. They both follow the idea of Secure Multi-Party Computation (MPC), which protects the model parameters in addition to intermediate results. Therefore, they also suffer from large communication overhead. These frameworks are both designed only for two parties. To the best of our knowledge, there have been no HE-SS-based solutions developed for multi-party situations.

**Our Contributions**

In this article, we propose an Efficient and Flexible Multi-party VFL without any third parties called EFMVFL. We follow the idea of FL that the model parameters are stored and updated locally. Instead of sharing the model parameters, only necessary intermediate results are transmitted online among the participants. We introduce SS and HE simultaneously to further protect the variables from being leaked. The contributions of our work can be summarized as follows.

*We propose a novel VFL framework without any trusted third-parties.* This framework contains four protocols and can complete the computation between two parties. We ensure the security of model parameters of all parties by SS and complete the model training task by HE. Through sophisticated structural design, our solution can complete the calculation without trusted third party assistance, and can ensure the security of data of all parties.

*Our framework is applicable to GLMs.* Our framework is flexible to be applied in GLMs. Meanwhile, it can greatly reduce the communication complexity by communicating intermediate variables instead of the original data, which improves the training efficiency of the GLMs.

*Our framework is easy to be extended to support multi-party scenarios.* We extend the framework of two parties to multiple parties and analyze the security of our multi-party model architecture. In multi-parties, we divide nodes into computing nodes and data-providing nodes. When adding nodes, the overall structure remains the same. In other words, the cost of adding nodes is very low.

*Our framework is more efficient with low communication overhead.* We implement our framework under LR and Poisson regression (PR), and prove that our VFL framework is more efficient and has a lower communication overhead compared with recent popular works.

**2 NOTATIONS**

We first introduce some notations here. As mentioned above, the focused scenario of our framework is VFL. In this scenario, data is vertically partitioned by parties, and there is only one party holding the labels. We use C to denote the party with the label (also named data demander) and use Bi (i ∈ [1, k]) to denote parties without the label (also known as data providers), where k is the size of data providers. As illustrated in Figure 2, C holds the label (Y) and feature (Xc) data, while Bi only holds feature (Xb) data.

Correspondingly, P is the party that can be C or Bi. In federated learning models, Y denotes the label of data of party C, Xp denotes the features of party P, xi denotes a column of the features, Wp denotes the linear model coefficients, WpXp denotes the point-wise product of model coefficients Wp and data Xp, {pkp, skp} denotes the HE key pairs, and [[x]]p denotes the ciphertext of x which is encrypted by public key pkp. ⟨x⟩p is the secret share of x in party P.
3 PRELIMINARIES

3.1 Secret-Sharing-based MPC

Secure MPC was first proposed in [25], which allows multiple parties to jointly perform privacy-preserving computing tasks. SS [18] is one of the main techniques to achieve this goal by splitting local data into shares. SS is divided into two processes: sharing and reconstruction. In the sharing phase, secrets are divided into several shares and stored in different parties. During the reconstruction phase, the secret can be calculated from the holdings of some or all of the parties.

SS has properties that allow some operations to be performed directly on shares. The operation on shares is equivalent to the shares of operation on secret. Such operations include addition, multiplication, division, power, and so on. In our study, addition and multiplication are necessary. Assuming that party C holds data $X_c$ and party B holds data $X_b$. Firstly, they both need to share their data using Protocol 1 introduced in the next section. Then they can calculate the shares of $X_c + X_b$ and $X_c * X_b$ as follows:

- **Addition**: The share of $X_c + X_b$ in party C (i.e., $(X_c + X_b)_c$) can be calculated by $(X_c)_c + (X_b)_c$, similarly, $(X_c + X_b)_b = (X_c)_b + (X_b)_b$.
- **Multiplication**: Besides the secret shares of original data, additional Beaver’s triplet $((\mu, \nu, \omega)$ and $\omega = \mu * \nu$) [2] is required for calculating shares of $X_c * X_b$. Specifically, each party can use the shares it holds to compute its own share of the multiplication, i.e., $(X_c * X_b)_c = (\omega) + ((X_c) - \langle \mu \rangle) * (\langle \nu \rangle + (X_b) - \langle \nu \rangle) * (\langle \mu \rangle) * ((X_b) - \langle \nu \rangle)$. The sum of $(X_c * X_b)_c$ from all parties can get the value of $X_c * X_b$.

There have been many protocols to complete the above calculations, such as secureML [16], ABY3 [15], secureNN [20], SPDZ [9], and so on.

3.2 Homomorphic Encryption

HE methods support computation over ciphertexts. The decrypted result of operating on ciphertexts is the same as the direct operations on plaintexts. As in SS, only addition and multiplication are needed in our framework. Specifically, Calculating $X_c + X_b$ and $X_c * X_b$ with HE mainly consists of the following steps:

- **Key generation**: One party, e.g., C, generates HE key pairs $\{(pk_c, sk_c) = \text{Gen}(1^\lambda), \text{where } \lambda \text{ is a security parameter}\}$, and can send public key $pk_c$ to the other party.

- **Encryption**: C uses the $pk_c$ to encrypt data $X_c([[X_c]])_c = \text{Enc}(X_c, pk_c))$. In the same way, the other party (i.e., B) can get $[[X_b]]_c$.

- **Addition**: Given $[[X_c]]_c$ and $[[X_b]]_c$, the addition between the two ciphertexts satisfies $[[X_c]]_c \oplus [[X_b]]_c = [[X_c + X_b]]_c$.

- **Scalar Addition**: Given $[[X_c]]_c$ and $X_b$, the addition between the ciphertext and plaintext satisfies $[[X_c]]_c \oplus X_b = [[X_c]]_c \oplus [[X_b]]_c = [[X_c + X_b]]_c$.

- **Scalar multiplication**: Given $[[X_c]]_c$ and $X_b$, the multiplication between the ciphertext and plaintext satisfies $[[X_c]]_c \otimes X_b = [[X_c * X_b]]_c$.

- **Decryption**: C uses $sk_c$ to decrypt the ciphertext and can get the plaintext $X_c (X_c = \text{Dec}([[X_c]], sk_c))$.

We call this probabilistic asymmetric encryption scheme for restricted computation (addition or multiplication) over ciphertexts **partially homomorphic encryption (PHE)**.

3.3 Generalized Linear Models

GLMs are flexible generalizations of linear regression. The GLM generalizes linear regression by allowing the linear model to be related to the response variable (Y) via a link function and by
allowing the magnitude of the variance of each measurement to be a function of its predicted value. Each kind of GLM consists of the following three elements:

1. An exponential family of probability distributions that \( E(Y|X) \) is assumed to satisfy.
2. A linear predictor \( \eta = WX \).
3. A link function \( g \) such that \( E(Y|X) = \mu = g^{-1}(\eta) \).

The **maximum likelihood estimation (MLE)** using iteratively gradient descent method is commonly applied to solve the weight parameters \( W \) of GLMs. For Example:

**Logistic regression.** (LR) can be regarded as a binary classification model and is widely used in industry because of its simplicity and interpretability. Assuming \( E(Y|X) \) satisfies Bernoulli distribution, and the link function is \( g = \ln\left(\frac{\eta}{1-\eta}\right) \), LR can find a direct relationship between the classification probability and the input vector \( (WX) \) by the Sigmoid function. The loss of LR can be calculated through MLE,

\[
\text{loss}_{LR} = \frac{1}{m} \sum_{i=1}^{m} \ln(1 + e^{-YW_iX}),
\]

where \( m \) is the sample size, and \( Y \in \{-1,1\} \) is the data label.

With formula (1), we can calculate its gradient, and approximate the gradient with MacLaurin expansion to avoid non-linear calculations:

\[
g_{LR} = X^T * \frac{1}{m} (0.25WX - 0.5Y),
\]

where \( T \) means the transposition of a matrix.

**Poisson regression.** (PR) assumes that \( E(Y|X) \) has a Poisson distribution and usually adopts Log function \( (g = \ln(\eta)) \) as its link function. PR is used to represent counts of rare independent events which happen at random but at a fixed rate, such as the number of claims in insurance policies in a certain period of time, and the number of purchases a user makes after being shown online advertisements, and so on. Also with the MLE, the loss function of PR is formed as

\[
\text{loss}_{PR} = \frac{1}{m} \sum_{i=1}^{m} (YW_iX - e^{WX} - \ln(Y!))
\]

and the gradient can be calculated by the following equation:

\[
g_{PR} = X^T * \frac{1}{m} (e^{WX} - Y).
\]

**Gamma regression.** (GR) assumes that \( E(Y|X) \) follows a Gamma distribution, which is a common distribution for positive and skewed variables. To transform the mean of the Gamma distribution into a linear predictor, the model typically uses the canonical link function \( g = -\frac{1}{\eta} \). The formula for its loss and gradient are as follows:

\[
\text{loss}_{GR} = \frac{1}{m} \sum_{i=1}^{m} (WX + Ye^{-WX}),
\]

\[
g_{GR} = X^T * \frac{1}{m} (1 - Ye^{-WX}).
\]

**Tweedie regression.** (TR) assumes that \( E(Y|X) \) follows a Tweedie distribution, which is a composite distribution of Poisson and Gamma distributions. Its canonical link function is defined as
Fig. 2. Data and parameters partition of VFL.

\( g = \frac{\eta^{1-p}}{1-p}, \) where \( p \in (1, 2) \) is a hyperparameter that controls the ratio of the Gamma and Poisson distributions. The formula for its loss and gradient are as follows:

\[
\text{loss}_{TR} = \frac{1}{m} \sum_{i=1}^{m} \left( -\frac{Y}{1-p} e^{(1-p)WX} + \frac{1}{2-p} e^{(2-p)WX} \right), \tag{7}
\]

\[
g_{TR} = X^T + \frac{1}{m} (-Ye^{(1-p)WX} + e^{(2-p)WX}). \tag{8}
\]

4 PROPOSED FRAMEWORK

In this section, we first introduce the structure of VFL. Then we state the core of the ideology and the main architecture of our proposed framework. Then we show how to apply our framework into GLMs (like LR, PR, GR, and TR) in federated learning scenarios. Finally, we show that it’s easy for our framework to be extended to multiple parties (three or more).

4.1 Overview

In VFL, data and parameters are vertically partitioned by parties. The Figure 2 shows the relationship between a centralized dataset and vertically distributed data. \( x_i \) is a feature of the dataset, and there are \( n \) features in total. The dimension of parameter \( W \) is same as the number of features. The features are held separately by different entities denoted as \( X_c, X_{b_1}, \ldots, X_{b_i} \), so the vector \( W \) is partitioned into \( W_c, W_{b_1}, \ldots, W_{b_i} \) to match the data.

Gradient descent is one of the core optimization methods in machine learning. As introduced in Section 3.3, the gradient of GLMs can be formalized as follows:

\[
g = X^T d, \tag{9}
\]

where \( g \) is the gradient, \( X \) is the feature data, and \( d \) is the gradient operator. The value of \( d \) varies in different models. For example, in LR, \( d = \frac{1}{m} (0.25WX - 0.5Y) \). When \( g \) is known, the model coefficients can be updated with the following equation.

\[
W = W - \alpha g, \tag{10}
\]
where $\alpha$ is the learning rate. In the case of LR, $X = [X_c \ X_b_1 \ldots X_b_i]$, $W = [W_c \ W_b_1 \ldots W_b_i]^T$, so Equation (10) can be expressed as

$$
\begin{align*}
\mathbf{g} &= \left[ \begin{array}{c}
X_c \\
\vdots \\
X_b_i \\
\end{array} \right] \cdot \frac{1}{m} \left( 0.25 [X_c \ X_b_1 \ldots X_b_i] \cdot \left[ \begin{array}{c}
W_c \\
W_b_1 \\
\vdots \\
W_b_i \\
\end{array} \right] - 0.5 Y \right) \\
&= \left[ \begin{array}{c}
X_c \\
\vdots \\
X_b_i \\
\end{array} \right] \cdot \frac{1}{m} \left( 0.25 (X_c W_c + \sum_{i=1}^{l} X_b_i W_b_i - 0.5 Y) \right) = \left[ \begin{array}{c}
X_c d \\
X_b_1 d \\
\vdots \\
X_b_i d \\
\end{array} \right].
\end{align*}
$$

The gradients of party $P$ can be denoted as $g_p = X_p d$, where $P \in \{C, B_1, \ldots, B_l\}$. The update of parameters can be implemented separately in multiple parties using $W_p = W_p - \alpha g_p$ if we can gather $X_p W_p$ from all parties to get the gradient-operator $d$. However, transferring data among multiple parties without a trusted third party will lead to the leakage of data.

### 4.2 Ideology and Architecture

We introduce a framework to locally compute gradient $g_p$ by communicating intermediate result $d$ among parties securely. Under this framework, we can complete vertical federated training in privacy without a trusted third party.

In our schemes, instead of sharing original data, we split intermediate results of GLMs training (e.g., $W_p X_p$) only, which leads to a dramatic drop in communication overhead.

There are four main protocols in our framework, i.e., SS protocol, Secure gradient-operator computing protocol, Secure gradient computing protocol, and Secure loss computing protocol. In this section, we only consider two-party situation (party $C$ and $B_1$) and will explain how to extend our framework to support multiple participants in Section 4.4.

**Secret sharing protocol.** We use SS technique to generate two shares of the data held by one party. As shown in Figure 3, party $C$ holds the plaintext data $Z_c$ (e.g., labels $Y$) and splits it into two shares $\langle Z_c \rangle_c$ and $\langle Z_c \rangle_b_1$. $\langle Z_c \rangle_c$ is generated randomly and kept locally by party $C$, $\langle Z_c \rangle_b_1 = Z_c - \langle Z_c \rangle_c$. Then party $C$ sends $\langle Z_c \rangle_b_1$ to party $B_1$. $Z_c$ can be recovered if and only if two shares of it are added. So the party $B_1$ cannot deduce the data on $C$. Similarly, $B_1$ generates two shares of its own data and sends one of the shares to party $C$. Protocol 1 is an example of SS.
Protocol 1: Secret sharing protocol

**Input**: a vector data $Z_{p_0}$, party $P_0$, $Z_{p_0}$ is held by $P_0$, $P_0$ may be $C$ or $B_1$

**Output**: $\langle Z_{p_0} \rangle_c$ for $C$ and $\langle Z_{p_0} \rangle_b$ for $B_1$, and $\langle Z_{p_0} \rangle_c + \langle Z_{p_0} \rangle_b = Z_{p_0}$

1. $P_1 = \{C, B_1\} - \{P_0\}$
2. $P_0$ locally generates a share $\langle Z_{p_0} \rangle_p$
3. $P_0$ calculates $\langle Z_{p_0} \rangle_c = Z_{p_0} - \langle Z_{p_0} \rangle_p$
4. $P_0$ sends $\langle Z_{p_0} \rangle_p$ to $P_1$
5. **return** $\langle Z_{p_0} \rangle_c$ for $C$ and $\langle Z_{p_0} \rangle_b$ for $B_1$

Protocol 2: Secure gradient-operator computing protocol

**Input**: $\langle Z_p \rangle_c$'s and $\langle Z_p \rangle_b$'s calculated by Protocol 1, $Z_p$ could be $WX$, $Y$, etc.

**Output**: $\langle d \rangle_c$ for $C$ and $\langle d \rangle_b$ for $B_1$

1. According to Equation (2), Equation (4), Equation (6), or Equation (8), $C$ and $B_1$ calculate $\langle d \rangle_c$ and $\langle d \rangle_b$, base on MPC method separately
2. **return** $\langle d \rangle_c$ for $C$ and $\langle d \rangle_b$ for $B_1$

Secure gradient-operator computing protocol. Section 4.1 has shown that the gradient-operator $d$ is necessary to update parameters. As shown in Figure 4 corresponding to Protocol 2, we can calculate shares of $d$ for different parties based on SS. In more detail, we can use shares $\langle Z_c \rangle_c$ and $\langle Z_b \rangle_c$ to calculate $\langle d \rangle_c$. $\langle d \rangle_b$ can be calculated in the same way. In the case of LR, $C$ holds shares $\langle X_c W_c \rangle_c, \langle X_b W_b \rangle_c$, and $\langle Y \rangle_c$, while $B_1$ holds shares $\langle X_c W_c \rangle_{b_1}, \langle X_b W_b \rangle_{b_1}$, and $\langle Y \rangle_{b_1}$. According to Equation (2), we can get $\langle d \rangle_c = \frac{1}{m}(0.25(\langle X_c W_c \rangle_c + \langle X_b W_b \rangle_{b_1}) - 0.5\langle Y \rangle_c)$, $\langle d \rangle_{b_1} = \frac{1}{m}(0.25(\langle X_c W_c \rangle_{b_1} + \langle X_b W_b \rangle_{b_1}) - 0.5\langle Y \rangle_{b_1})$. Eventually party $C$ gets $\langle d \rangle_c$ and party $B_1$ gets $\langle d \rangle_{b_1}$ (Protocol 2).
Protocol 3: Secure gradient computing protocol

Input: \( \langle d \rangle_c \) and \( \langle d \rangle_b \), calculated by Protocol 2, feature data matrix \( X_{p_0} \), party \( P_0 \), \( X_{p_0} \) is held by \( P_0 \), \( P_0 \) may be \( C \) or \( B_1 \).

Output: \( g_c \) for \( C \) if \( P_0 \) is \( C \) or \( g_{b_1} \) for \( B_1 \) if \( P_0 \) is \( B_1 \)

1. \( P_1 = \{C, B_1\} - \{P_0\} \)
2. \( P_0 \) locally calculates the share of gradient using Equation (9), i.e., \( \langle g_{p_0} \rangle_{p_0} = X_{p_0}^T \ast \langle d \rangle_{p_0} \)
3. \( P_1 \) encrypts \( \langle d \rangle_{p_1} \) using \( pk_{p_1} \) and sends \( \[\langle \langle d \rangle_{p_1} \rangle_{p_1} \]_{p_1} \) to \( P_0 \)
4. \( P_0 \) locally calculates encrypted share of gradient using Equation (9), i.e., \( \[\langle \langle g_{p_0} \rangle_{p_1} \rangle_{p_1} \]_{p_1} = X_{p_0}^T \ast \[\langle \langle d \rangle_{p_1} \rangle_{p_1} \]_{p_1} \)
5. \( P_0 \) locally generates random noise \( R_{p_0} \)
6. \( P_0 \) locally masks encrypted share of \( g_{p_0} \) with \( \[\langle \langle g_{p_0} \rangle_{p_1} \rangle_{p_1} \]_{p_1} = \[\langle \langle g_{p_0} \rangle_{p_1} \rangle_{p_1} \]_{p_1} - R_{p_0} \) and sends \( \[\langle \langle g_{p_0} \rangle_{p_1} \rangle_{p_1} \]_{p_1} \) to \( P_1 \)
7. \( P_1 \) gets \( \langle g_{p_0} \rangle_{p_1} \) by decrypting \( \[\langle \langle g_{p_0} \rangle_{p_1} \rangle_{p_1} \]_{p_1} \) with \( sk_{p_1} \), and sends \( \langle g_{p_0} \rangle_{p_1} \) to \( P_0 \)
8. \( P_0 \) calculates the gradient of its coefficients with \( g_{p_0} = \langle g_{p_0} \rangle_{p_1} + \langle g_{p_0} \rangle_{p_0} + R_{p_0} \)
9. return \( g_c \) for \( C \) or \( g_{b_1} \) for \( B_1 \)

Secure gradient computing protocol. The gradient of GLMs can be calculated by Equation (9). And the local gradient \( g_p \) used to update local parameter \( W_p, p \in \{c, b_1\} \), can be calculated by the product of local data \( X_p \) and gradient-operator \( d \). However, \( d \) is shared over party \( C \) and \( B_1 \), denoted as \( \langle d \rangle_c \) and \( \langle d \rangle_{b_1} \), respectively. In regards to privacy concerns, we stipulate that data \( X_p \) and the share of gradient-operator \( \langle d \rangle_p \) cannot be sent to other parties. In order to securely compute the gradient, we introduce HE in Protocol 3. After computing the share of gradient-operator in Protocol 2, we need three rounds of communication to get \( g_p \). As shown in Figure 4 corresponding Protocol 3: ① Party \( C \) and \( B_1 \) encrypt \( \langle d \rangle_c \) and \( \langle d \rangle_{b_1} \), respectively, using public key \( pk_c \) and \( pk_{b_1} \), and send the encrypted value \( \[\langle \langle d \rangle_c \rangle_c \] \) and \( \[\langle \langle d \rangle_{b_1} \rangle_{b_1} \] \) to each other. As line 4 in Protocol 3, this is used to compute the encrypted share of gradient, for instance \( \[\langle \langle g_c \rangle_{b_1} \] \), it denotes the value that the share of \( g_c \) on party \( B_1 \) is encrypted by \( pk_{b_1} \). ② In party \( C \), masking the encrypted share of gradient with a random noise \( R_c \), and sending back to \( B_1 \). Similarly, so does party \( B_1 \). The masking step can prevent \( P_1 \) from computing \( X_{p_0} \) by \( \[\langle \langle g_{p_0} \rangle_{p_1} \] \) and \( \[\langle \langle d \rangle_{p_1} \] \). ③ Sending \( \langle g_c \rangle_{b_1} \) back to party \( C \) and \( \langle g_{b_1} \rangle_c \) to party \( B_1 \). Then, we can get \( g_c \) for \( C \) and \( g_{b_1} \) for \( B_1 \) by suming the shares and noise.

Secure loss computing protocol. After the model coefficients are updated, it is always necessary to see how well they fit the labels and whether the training process needs to stop. Loss value is the most direct and common indicator to quantify the model performance. There are different forms of loss for different GLMs. Protocol 4 shows how to calculate loss privately in LR/PR/GR/TR scenarios. Similar to Protocol 2, shares of loss can be calculated based on the SS intermediate results. For example, in the case of LR, loss value is dependent on \( YWX \) according to Equation (1). Based on the multiplication of SS, we can calculate shares of \( YWX \) on \( C \) and \( B_1 \). If there is a Beaver’s triplet \( (\mu, \nu, \omega) \) (described in Section 3.1), \( (YWX) \)’s share of party \( P \) can be calculated by

\[
\langle YWX \rangle_p = \langle \omega \rangle_p + \langle (WX) \rangle_p - \langle \mu \rangle_p \ast \langle (Y) \rangle_p - \langle \nu \rangle_p + \langle (WX) \rangle_p - \langle \mu \rangle_p \ast \langle (Y) \rangle_p - \langle \nu \rangle_p \ast \langle \mu \rangle_p \rho,
\]

(12)

where \( \langle WX \rangle_p = \langle W_c X_c \rangle_p + \langle W_{b_1} X_{b_1} \rangle_p \), \( P \) could be \( C \) or \( B_1 \). Then according to Equation (1), \( C \) and \( B_1 \) can calculate \( \langle \text{loss} \rangle_c \) and \( \langle \text{loss} \rangle_{b_1} \) of LR based on the MPC method separately. At last, the loss value needs to be revealed to party \( C \) for evaluating the model performance.
Protocol 4: Secure loss computing protocol

**Input:** $\langle Z \rangle_c$’s and $\langle Z \rangle_{b_1}$’s calculated by Protocol 1, $Z$ could be the share of $WX$, share of $Y$, etc.

**Output:** loss for $C$.

1. According to Equation (1), Equation (3), Equation (5), or Equation (7), $C$ and $B_1$ calculate $\langle \text{loss} \rangle_c$ and $\langle \text{loss} \rangle_{b_1}$ based on MPC method separately.
2. $B_1$ sends $\langle \text{loss} \rangle_{b_1}$ to $C$.
3. $C$ reveals loss by $\text{loss} = \langle \text{loss} \rangle_c + \langle \text{loss} \rangle_{b_1}$.
4. **return** loss for $C$.

We have introduced the protocols involving two parties above. We refer to Algorithm 1 to illustrate how these two-party protocols can be extended to multi-party protocols.

### 4.3 GLMS in VFL

In this section, we introduce how to apply our framework to GLMs. Regarding Protocol 2 and Protocol 4, the differences among different GLMs only lie in the formula of the gradient-operator and loss, while Protocol 1 selects different vector data $Z$ for SS based on the specific formula. Protocol 3 remains unchanged. Specifically, we will take the implementations of LR, PR, GR, and TR as examples.

**Logistic regression.** (LR) According to Equation (2), the gradient-operator of LR in Protocol 2 is

$$d_{LR} = \frac{1}{m}(0.25WX - 0.5Y).$$

On the other hand, Protocol 4 can be implemented according to Equation (1). On the basis of the calculation formulas of loss and gradient, party $C$ and $B_1$ both need to share $WX$ in Protocol 1, and $C$ needs to share label $Y$ additionally.

**Poisson regression.** (PR) Similarly, the gradient-operator of PR can be calculated by following according to Equation (4):

$$d_{PR} = \frac{1}{m}(e^{WX} - Y).$$

To avoid non-linear operations when calculating gradient-operator and loss of PR based on MPC, shares of $e^{WX}$ are also required in Protocol 1 in addition to $WX$ and $Y$.

**Gamma regression.** (GR) According to Equation (6), the gradient-operator of GR in Protocol 2 is

$$d_{GR} = \frac{1}{m}(1 - Ye^{-WX}).$$

In addition to the loss function Equation (5), it is necessary to secret share $WX$, $e^{-WX}$, and $Y$ through Protocol 1, just like in GR.

**Tweedie regression.** (TR) According to Equation (8), the gradient-operator of TR in Protocol 2 is

$$d_{TR} = \frac{1}{m}(-Ye^{(1-p)WX} + e^{(2-p)WX}).$$

So in Protocol 1, $e^{(1-p)WX}$, $e^{(2-p)WX}$ and $Y$ need to be shared.

Note that although we introduce our framework in the situations of LR, PR, GR, and TR, it is also suitable for other GLMs, such as Linear regression, Inverse Gaussian regression, Negative binomial regression, and so on.

ACM Transactions on Knowledge Discovery from Data, Vol. 18, No. 3, Article 51. Publication date: December 2023.
4.4 Multi-party VFL

In this section, we will explain how to extend our framework to support multiple participants. When it comes to multiple parties, we select two parties, C and B_1, as the computing parties (hereafter, CPs).

From Algorithm 1 we can see that Protocol 1 does not need to be modified for CPs. Other parties just need to first locally generate two shares of their own data and then send shares to the two CPs separately. For example, for party B_2, which is not selected as CPs, it generates two shares \( \langle Z_{b_2} \rangle_c \) and \( \langle Z_{b_2} \rangle_b \). Then, party B_2 sends them to C and B_1 separately.

Protocols 2 and 4 even don’t need to change because only the CPs hold shares of gradient-operator and label. They can complete the gradient-operator and loss computing tasks by themselves. The difference is only in the terms that participate in the computation. Each CP needs to deal with all shares received from other parties, which including the another CP and other extra parties. At last, CPs have to send the encrypted gradient-operator to other participants after the computing is finished in Protocol 2.

Protocol 3 needs some modifications. Figure 5 is a communication and computation process of three-party situation. When there are three parties, B_3 that is not selected as CPs needs to get the two encrypted shares of gradient-operator from CPs first, since these shares are calculated in CPs. Shown as the step \( \text{(1)} \) in Figure 5, C and B_1 separately send \( \langle \langle d \rangle_c \rangle \rangle_c \) and \( \langle \langle d \rangle_{b_1} \rangle \rangle_{b_1} \) to B_3. Then each party calculates the shares of gradient or the encrypted shares of gradient as step \( \text{(2)} \).

At step \( \text{(3)} \) and \( \text{(4)} \), C and B_1 exchange information with each other and decrypt \( \langle \langle g_{p_1} \rangle \rangle_{p_1} \) like Protocol 3. B_3 needs to send back encrypted and masked gradient shares back to C and B_1 to get a decrypted version at step \( \text{(5)} \). Finally, each party sums all the shares of gradients and noises of their own to get local gradients, then they update the local parameter separately. When there all more than three parties, the parties not in CPs only need to do the same way as party B_2.

5 SECURITY ANALYSIS

Our study is based on the idea of local computation in FL, so we focus on whether either party in the training process can infer the original data through the delivered intermediate variables, which causes data leakage. The security of our multi-party model architecture only depends on the security of two-party model architecture. For the two-party model, we use the same security model and architecture as [3]. The following is the detailed security analysis.

5.1 Security of Protocols

We prove two necessary lemmas first, which are prepared for the proof of Algorithm 1. These two lemmas describe properties of solutions to system of linear equations.

**Lemma 1.** Let \( X \in \mathbb{Z}_q^{n \times m}, \{w_i \in \mathbb{Z}_q^m\}_{i \in [T]} \), where \( n, m \) and \( q \) are integers and \( T \) is the number of iterations. For any probabilistic polynomial-time adversary \( A \), given some \( \{y_i \in \mathbb{Z}_q^m\}_{i \in [T]} \) which satisfy that \( X \cdot w = y \) for \( i \in [T] \), the necessary condition for that \( X \) and \( \{w_i\}_{i \in [T]} \) can not be accurately calculated is \( n \leq m \) or \( n > m \) and \( T \leq \frac{n \times m}{n - m} \).

**Proof.** When the adversary \( A \) gets a vector \( y_i \), which means that \( A \) gets \( n \) equations and \( n \times m + m \) unknowns. Similarly, through \( T \) iterations, \( A \) can obtain \( n \times T \) equations and \( (n + T) \times m \) unknowns. There are two cases in this situation.

**case 1:** \( n \leq m \). In this case, we can get \( nT < (n + T)m \) which means that the number of unknowns is larger than the number of equations. At this time, \( A \) cannot calculate the precise information of \( X \) and \( \{w_i\}_{i \in [T]} \).

ACM Transactions on Knowledge Discovery from Data, Vol. 18, No. 3, Article 51. Publication date: December 2023.
Fig. 5. A variant of Protocol 3. Calculating gradient in three party VFL.

case 2: \( n > m \). In this case, we just need to set \( T \leq \frac{n \times m}{n - m} \), then we can get \( nT < (n + T)m \). which means that the number of unknowns is large than the number of equations. At this time, \( \mathcal{A} \) cannot calculate the precise information of \( X \) and \( \{ w_i \}_{i \in [T]} \).

In conclusion, the necessary condition for that \( X \) and \( \{ w_i \}_{i \in [T]} \) cannot be accurately calculated is \( n \leq m \) or \( n > m \) and \( T \leq \frac{n \times m}{n - m} \). \( \square \)

Lemma 2. Let \( X \in \mathbb{Z}_{q}^{n \times m}, d \in \mathbb{Z}_q^n \) where \( n, m \) and \( q \) are integers. For any probabilistic polynomial-time adversary \( \mathcal{A} \), given \( y \in \mathbb{Z}_q^m \) and \( X \in \mathbb{Z}_q^{n \times m} \) which satisfy that \( y = X^T d \), the necessary condition for \( d \) to be accurately calculated is \( n \leq m \).

Proof. When the adversary \( \mathcal{A} \) gets \( y \in \mathbb{Z}_q^m \) and \( X \in \mathbb{Z}_q^{n \times m} \), which means that \( \mathcal{A} \) gets \( m \) equations and \( n \) unknowns. The necessary condition for \( d \) to be accurately calculated is \( n \leq m \). \( \square \)

Now we give a formal proof of Theorem 1.

Theorem 1. Let \( \{ g_i \} \in \mathbb{Z}_q^{m_1} \}_{i \in [T]}, X_1 \in \mathbb{Z}_q^{n \times m_1}, X_2 \in \mathbb{Z}_q^{n \times m_2}, \{ d_i \} \in \mathbb{Z}_q^n \}_{i \in [T]}, \{ w_i \} \in \mathbb{Z}_q^{m_2} \}_{i \in [T]}, \) where \( n, m \) and \( q \) are integers and \( T \) is the number of iterations. For any probabilistic polynomial-time adversary \( \mathcal{A} \), given some \( \{ g_i \} \in \mathbb{Z}_q^{m_1} \}_{i \in [T]} \) and \( X_1 \in \mathbb{Z}_q^{n \times m_1} \) which satisfy that \( g_i = X_1^T d_i \) for all \( i \in [T] \), the necessary condition for that \( X_2 \) and \( \{ w_i \} \in \mathbb{Z}_q^{m_2} \) which satisfy that \( d_i = X_2 w_i \) for all \( i \in [T] \) can not be accurately calculate if \( n > m_1 \) or \( n \leq \min \{ m_1, m_2 \} \) or \( m_2 < n \leq m_1, T \leq \frac{n \times m_2}{n - m_2} \).

Proof. If we want to accurately calculate \( X_2 \) and \( \{ w_i \} \in \mathbb{Z}_q^{m_2} \), then we must first accurately calculate the value of \( d_i \). According to this condition, we will discuss this issue in three situations below.
ALGORITHM 1: EFMVFL: An Efficient and Flexible Multi-party Vertical Federated Learning without a Third Party

Input: feature data $X_p$, label data $Y$, HE key pairs $\{pk_p, sk_p\}$, party $P$ may be $C$ or $B_1$, learning rate $\alpha$, max iteration $T$, loss threshold $L$

Output: $W_p$ for $P$

1: Let $t = 0$
2: Initialize $W_p$ as zero vector
3: Select $\{C, B_1\}$ as the computing parties (CPs)
4: while $t < T$ and loss $< L$ do
5: $P$ locally calculates $Z$'s, $Z$ could be $W_p X_p$, $e^{W_p Y_p}$, $Y$, etc., and $Y$ is held by $C$.
6: if $P$ is in CPs then
7: Do secret sharing with Protocol 1.
8: Recive shares from other parties
9: Do secure gradient-operator computing with Protocol 2.
10: Do secure gradient computing with Protocol 3, get gradients $g_p$.
11: Send encrypted gradient-operator to other parties
12: Recive encrypted gradient from other parties and decrypt and send back to other parties
13: Do secure loss computing with Protocol 4.
14: else
15: Generate two random noise $\langle d_c \rangle_c, \langle d_{b_i} \rangle_{b_i}$'s as $\langle Z \rangle_c$, $\langle Z \rangle_{b_i}$'s = $Z$'s
16: Send shares $\langle Z \rangle_c$'s to $C$, $\langle Z \rangle_{b_i}$'s to $B_1$.
17: Recive encrypted gradient-operator [[\langle d_c \rangle_c], [[\langle d_{b_i} \rangle_{b_i}]]_{b_i}} form $C$, $B_1$ separately.
18: Calculate two encrypted shares of $P$'s gradient [[\langle g_p \rangle_c], [[\langle g_{p} \rangle_{b_i}]]_{b_i}} using Equation (9)
19: Generate two random noise $R_{b_i}^1, R_{b_i}^2$, Send [[\langle g_p \rangle_c] - R_{b_i}^1, [[\langle g_p \rangle_{b_i}]]_{b_i} - R_{b_i}^2$ to $B_1$.
20: Recive $\langle g_p \rangle_c, \langle g_p \rangle_{b_i}$ form $C, B_1$ separately
21: Calculate the gradient of $P$'s coefficients with $g_p = \langle g_p \rangle_c + \langle g_p \rangle_{b_i} + R_{b_i}^1 + R_{b_i}^2$
22: end if
23: Update model coefficients $W_p$, $W_p^{t+1} = W_p^t - \alpha g_p$
24: $t = t + 1$
25: end while
26: return $W_p$ for $P$

case 1: $n > m_1$. By Lemma 2, the necessary condition for $d_i$ to be accurately calculated is $n \leq m_1$. In this case, $d_i$ can not be accurately calculated, and then $X_2$ and $\{w_i\}_{i \in [T]}$ can not be accurately calculated.

case 2: $n \leq \min(m_1, m_2)$. In this case, we can accurately calculate $d_i$ by Lemma 2. However, in this situation, through Lemma 1, any probabilistic polynomial-time adversary $\mathcal{A}$ can not accurately calculate $X_2$ and $\{w_i\}_{i \in [T]}$.

case 3: $m_2 < n \leq m_1$, $T \leq \frac{n \times m_2}{n - m_2}$. In this case, we can accurately calculate $d_i$ by Lemma 2. However, in this situation, through Lemma 1, any probabilistic polynomial-time adversary $\mathcal{A}$ can not accurately calculate $X_2$ and $\{w_i\}_{i \in [T]}$.

\(\square\)

Then, we can analyze the security of Protocols 1–4. In the following, Theorem 2 shows that Protocol 1 is secure against semi-honest adversaries.
Theorem 2. Assume that $\langle Z \rangle_{p_0}$ is generated by a secure \textit{pseudo-random number generator} (PRNG). Then Protocol 1 is secure in semi-honest model.

Proof. We prove that $Z - \langle Z \rangle_{p_0}$ is computationally indistinguishability with random number. $\langle Z \rangle_{p_0}$ is computationally indistinguishability with random number because it is generated by a secure PRNG. Then $Z - \langle Z \rangle_{p_0}$ is computationally indistinguishability with random number. □

For Protocols 2 and 4, the calculation is based on MPC method, so they are secure against semi-honest adversaries.

Theorem 3. Assume that there is a secure MPC protocol under semi-honest adversary model. Then Protocols 2 and 4 are secure in semi-honest model.

Proof. Obviously, the security of Protocols 2 and 4 is directly dependent on the security of MPC protocol. There are many MPC protocols which are secure under semi-honest adversary, just like [15, 20]. □

In the following, we will give a formal theorem that Protocol 3 is secure against semi-honest adversaries.

Assumption 1. For party C, we have $g_c = X_c^T d$, where $X_c \in \mathbb{Z}_q^{n \times m_1}$ is the data on C and $d \in \mathbb{Z}_q^n$ is the gradient operator. $X_{b_1} \in \mathbb{Z}_q^{n \times m_2}$ is the data on $B_1$. $T$ is the number of iterations to compute the parameter. We assume that $n > m_1$ or $n \leq \min\{m_1, m_2\}$ or $m_2 < n \leq m_1$, $T \leq \frac{n \times m_2}{n - m_2}$.

Theorem 4. Assume that the additively HE protocol $\Pi = (\text{KeyGen}, \text{Enc}, \text{Dec})$ is indistinguishable under chosen-plaintext attacks. Then Protocol 3 is secure in semi-honest model if Assumption 1 is true.

Proof. In lines 1–4, the data is calculated locally and sent to other participants in the form of ciphertext. The security of these lines is dependent on the HE protocol $\Pi$. In lines 5–7, we use the same technology which has been proposed in Protocol 2 of [3]. They give a detailed proof in Appendix B.1, which we will not repeat it here. In lines 8–9, $P_0$ will get gradient $g$, which is the point-wise product of data $X_{p_0}$ and gradient-operator $d$. According to Theorem 1, we can know that $P_0$ could not calculate other party’s feature data matrix $X_{p_i}$ and model parameter $W_{p_i}$ under Assumption 1. □

In the following, we will give a formal theorem that Algorithm 1 is secure against semi-honest adversaries.

Theorem 5. Assume that the additively HE protocol $\Pi = (\text{KeyGen}, \text{Enc}, \text{Dec})$ is indistinguishable under chosen-plaintext attacks and there is a secure MPC protocol under semi-honest adversary model. Then Algorithm 1 is secure in semi-honest model.

Proof. Algorithm 1 comprehensively uses the structure of Protocols 1–4. The proof of Theorem 5 can reuse the proofs of Theorems 2–4. □

5.2 Discussion

In this part, we discuss the security boundary of our algorithm. First, our method applies to the case of semi-honest adversaries on all parties. Second, during the iterations, we recover the gradient $g_p$ of each party. Supposing the gradient of party C at iteration $t (t < T)$ is $g^{(t)}_c$, then there are two systems of linear equations involved for C to compute the data of $B_1$, i.e., $g^{(t)}_c = X_c^T d^{(t)}$ and $d^{(t)} = X_c^T W_c + X_{b_1}^T W_{b_1}$. Therefore, C can potentially get the information of $B_1$. However, according to Theorem 1, we can protect the data and parameter of $B_1$ from C as long as some conditions...
are satisfied. If the number of samples is greater than the number of features of $X_c$ ($n > m_1$ in Assumption 1), $C$ cannot calculate $d^{(t)}$ accurately. The data on $B_1$ cannot be gotten any further. There are similar analyses for other cases. In general, if Assumption 1 is satisfied, any party cannot access the data of other parties.

6 IMPLEMENTATIONS AND ASSESSMENTS

In this section, we implement sufficient experiments to show that our framework is effective and efficient with less communication, applicable to GLMs, and easy to scale to multi-party modeling.

6.1 Dataset

Our experiments are based on the following open-source datasets. We vertically split both datasets into two parts as FATE\(^1\) does, corresponding to party $C$ and $B_1$. In the multi-party case, the dataset was split into six equal parts along the feature dimensions, with each party owning one of the parts, which enabled collaborative training without directly sharing data.

**Default of credit card clients Dataset**\(^2\) consists of 30 thousand samples with 24 attributes. This dataset contains information on default payments, demographic factors, credit data, history of payment, and bill statements of credit card clients in Taiwan. The study of this dataset is focused on predicting whether a customer will default on their payment or not, which is a binary classification problem. As such, we chose to use LR for this dataset.

**Dvisits Dataset**\(^3\) comes from the Australian Health Survey of 1977-1978 and consists of 5,190 single adults with 19 features, such as sex, age, income, and number of illnesses in past two weeks, and so on. The goal of this dataset is to predict the number of doctor visits made by patients, which is a regression problem for count data. So we trained the PR model on it.

6.2 Setting

All our experiments are run on Linux servers with 32 Intel(R) Xeon(R) CPU E5-2640 v2@2.00 GHz and 128 GB RAM. For each server, the CPU resources are 16 cores. We limit network bandwidth to 1,000 Mbps. In our experiments, we use the Paillier cryptosystem to implement HE. Any HE method that makes addition and multiplication properties in Section 3.2 valid can be used. To ensure computational accuracy, the integer in SS is represented by 128 bits. The Paillier HE key length, max iteration, threshold, and learning rate of LR and PR are set to 1,024, 30, 1e-4, 0.15, and 0.1, respectively, in the two-party case. We set the ratio between training and test set to 7:3.

For the multi-party scenario, we only tested our architecture on LR, as the performance of PR is expected to be similar. When there are only a few participants, the number of feature dimensions involved in the training process is relatively small, resulting in slower convergence of the model. To counteract this effect, we increased the learning rate to 0.5 for the multi-party scenario, which helped to speed up the convergence of the model.

6.3 Experiments and Result

We compare our framework on LR with those methods with a third party (TP-LR [10]) and without a third party (SS-LR [21], SS-HE-LR [3]). TP-LR is based on HE, while SS-LR is based on SS, and SS-HE-LR is based on both SS and HE methods. Since the SS and HE method proposed by [3] only supports two-party LR, we only compare our framework on PR with TP-PR (inspired by [7]) and SS-PR [8]. For the multi-party scenario, we only compare our framework with TP and SS baselines.

\(^1\)https://github.com/FederatedAI/FATE
\(^2\)https://archive.ics.uci.edu/ml/datasets/default+of+credit+card+clients
\(^3\)https://www.rdocumentation.org/packages/faraway/versions/1.0.7/topics/dvisits
Fig. 6. Loss curve of LR (upper) and PR (lower).

Fig. 7. Communication distribution of EFMVFL.

Table 1. LR Results on Test Set

| framework     | auc  | ks   | comm  | runtime |
|---------------|------|------|-------|---------|
| TP-LR         | 0.712| 0.371| **14.20 mb** | 34.79 s |
| SS-LR         | 0.719| 0.363| 181.8 mb  | 71.05 s |
| SS-HE-LR      | 0.702| 0.367| 85.30 mb  | 37.6 s  |
| EFMVFL-LR     | 0.712| 0.372| **26.45 mb** | **23.29 s** |

The bold values signify an experimental outcome that demonstrates a distinct advantage.

It’s worth mentioning that the SS-HE-LR has already been implemented by FATE, while we have implemented the other methods ourselves.

The training loss curve in Figure 6 shows that our proposed method (red solid lines) achieves a comparable convergence rate to the baseline methods that with a third-party (blue dashed lines). This suggests that our proposed method has a similar ability to converge during the training process to the other methods, despite being based on a different approach. Notably, the difference between the loss curves in the upper panel is because the loss used in TP-LR is a Taylor approximation of our method. Overall, the comparable convergence rate achieved by our proposed method and the baseline methods underscores the effectiveness and competitiveness of our approach.

Tables 1 and 2 provide more details about comparison results for these methods. Here, we use two key metrics—"comm" and "runtime"—to measure the efficiency of the model iteration. "Comm" refers to the average amount of data transferred among parties during a round, measured in MByte (mb). On the other hand, "runtime" measures the time taken in seconds (s). Additionally, we evaluate the performance of classification models (e.g., LR) using metrics such as "auc" and "ks". The "auc" metric represents the area under the Receiver Operating Characteristic (ROC) curve, and "ks" refers to the Kolmogorov-Smirnov statistic. Larger values of "auc" and "ks" indicate higher classification model performance. In contrast, regression models (e.g., PR) can be evaluated using metrics like "mae" and "rmse", where "mae" is mean absolute error and "rmse" is root mean squared error. Smaller values indicate improved regression model performance. Note that all these assessment results are measured on test set in the case of two parties. As shown in Tables 1 and 2, the metrics of our method were comparable to those obtained by other methods, which means our method can achieve comparable model performance with other methods.

Moreover, our framework only needs to communicate intermediate results to each other. All of these intermediate results are vectors (see Protocol 3), while SS-based methods have to share all of original data. Therefore, communication overhead of our framework is relatively low. The effectiveness of our method is further substantiated by the "comm" and "runtime" columns presented in Table 1. Comparing with the methods without a third party (SS-LR and SS-HE-LR), our method can
Table 2. PR Results on Test Set

| framework       | mae         | rmse        | comm      | runtime |
|-----------------|-------------|-------------|-----------|---------|
| TP-PR           | 0.571       | 0.834       | 4.27 mb   | 12.44 s |
| SS-PR           | 0.571       | 0.834       | 20.77 mb  | 11.54 s |
| EFMVFL-PR       | 0.571       | 0.834       | 5.60 mb   | 10.78 s |

The bold values signify an experimental outcome that demonstrates a distinct advantage.

reduce the communication overhead by $3.2 \times -6.8 \times$ and speed up by $1.6 \times -3.1 \times$. Besides, comparing with the method with a third party (TP-LR and TP-PR), our method spends less time.

We further analyze the communication overhead during vertical federated training. The protocols involved in communication are Protocols 1, 3, and 4. Protocol 4 transfers a scalar loss. It can be ignored compared with other communications. The variable that needs to communicate in Protocol 1 is vectors processed by SS, and its size is related to the number of samples. From Figure 7, we can see that HE accounts for a larger proportion in communication than SS in our method. This is because when encrypted by secret key, each number in the vector takes about 1,024 bits, which is the same length as secret key. While each number in SS takes 128 bits. Protocol 3 involves the communication with the HE. In Protocol 3, encrypted gradient-operator $d$ and gradient $g$ are sent between two parties. The size of encrypted $d$ and $g$ is related to the number of samples and the dimension of features, respectively. Therefore, the communication overhead depends on the number of samples and the dimension of features.

In the lower panel of Figure 8, the communication overhead is almost constant as the features increase. This is because that the feature dimension is much smaller than the number of samples. The communication overhead mainly increases with the number of samples, as shown in the lower panel of Figure 9. The relationship between communication overhead and the number of samples is nearly linear. The increase in sample size and feature dimension will make the data computation more expensive, so the runtime will increase, as shown in the upper panel of Figures 8 and 9.

In the multi-party scenario, on the one hand, as shown in Figure 10, the convergence of loss is still maintained. However, the level of improvement in the model’s effectiveness varies with the increase in the number of participants due to the different impact of features contributed by different participants. On the other hand, our framework’s communication and runtime in the LR scenario were compared to TP-LR and SS-LR as a function of the number of participants, as depicted in Figure 11. It shows that our framework outperforms both TP-LR and SS-LR in terms of communication overhead and time consumption in the multi-party scenario as well. Specifically, compared to SS-LR, EFMVFL-LR demonstrates significant advantages in terms of communication.
and time consumption. Additionally, EFMVFL-LR is more efficient than TP-LR in terms of time consumption while maintaining comparable communication costs. In the lower panel, for clarity, we fit a straight line to show that all the framework communication increases linearly with the number of participants. In the upper panel, as the number of participants changes from 2 to 3, we find the runtime of our framework increases suddenly and then flattens out. This is because when it comes to multiple parties, there would be 2 cipher product operations for parties that are not the computing party (CPs) in Algorithm 1.

CONCLUSION
In this article, we present an Efficient and Flexible Multi-Party Vertical Federated Learning framework (EFVFL) that does not require a third party by combining SS and HE. The framework is applicable to many kinds of generalized linear regression models and has been shown in LR, PR, Gamma regression, and Tweedie regression scenarios. Through theoretical analysis and comparison with some recent popular FL works, we show that EFVFL is secure, effective, and more efficient with less communication overhead. Furthermore, our framework is scalable to multi-party modeling, and experiments show that runtime and communication both grow almost linearly as the number of participants increases. In the future, we will expand our framework to more machine learning algorithms, such as deep learning.

ACKNOWLEDGMENTS
We thank Tianxiang Mao and Renzhang Liu for their professional advice to this work.

REFERENCES
[1] Ibrahim Aliyu, Marco Carlo Feliciano, Sélinde van Engelenburg, Dong Ok Kim, and Chang Gyoon Lim. 2021. A blockchain-based federated forest for SDN-enabled in-vehicle network intrusion detection system. IEEE Access 9 (2021), 102593–102608. DOI: https://doi.org/10.1109/ACCESS.2021.3094365
[2] Donald Beaver. 1991. Efficient multiparty protocols using circuit randomization. In Proceedings of the Advances in Cryptology—CRYPTO ’91, 11th Annual International Cryptology Conference. Joan Feigenbaum (Ed.), Vol. 576, Lecture Notes in Computer Science, Springer, 420–432. DOI: https://doi.org/10.1007/3-540-46766-1_34
[3] Chaochao Chen, Jun Zhou, Li Wang, Xibin Wu, Wenjing Fang, Jin Tan, Lei Wang, Alex X. Liu, Hao Wang, and Cheng Hong. 2021. When homomorphic encryption marries secret sharing: Secure large-scale sparse logistic regression and applications in risk control. In Proceedings of the KDD ’21: The 27th ACM SIGKDD Conference on Knowledge Discovery and Data Mining. Feida Zhu, Beng Chin Ooi, and Chunyan Miao (Eds.), ACM, 2652–2662. DOI: https://doi.org/10.1145/3447548.3467210
[4] Kewei Cheng, Tao Fan, Yilun Jin, Yang Liu, Tianjian Chen, Dimitrios Papadopoulos, and Qiang Yang. 2021. SecureBoost: A lossless federated learning framework. IEEE Intelligent Systems 36, 6 (2021), 87–98. DOI: https://doi.org/10.1109/MIS.2021.3082561
[5] Haokun Fang and Quan Qian. 2021. Privacy preserving machine learning with homomorphic encryption and federated learning. Future Internet 13, 4 (2021), 94. DOI: https://doi.org/10.3390/fi13040094
An Efficient and Flexible Multi-party Vertical Federated Learning without a Third Party 51:19

[6] FU Fangcheng, LIU Shu, CHENG Yong, and TAO Yangyu. 2022. Vertical federated logistic regression via homomorphic encryption and secret sharing. Information and Communications Technology and Policy 48, 5 (2022), 34.

[7] Stephen Hardy, Wilko Henecka, Hamish Ivey-Law, Richard Nock, Giorgio Patrini, Guillaume Smith, and Brian Thorne. 2017. Private federated learning on vertically partitioned data via entity resolution and additively homomorphic encryption. arXiv:1711.10677. Retrieved from https://arxiv.org/abs/1711.10677

[8] Mahimna Kelkar, Phil Hung Le, Mariana Raykova, and Karn Seth. 2022. Secure Poisson Regression. In 31st USENIX Security Symposium (USENIX Security 22). USENIX Association, Boston, MA, 791–808. https://www.usenix.org/conference/usenixsecurity22/presentation/kelkar

[9] Marcel Keller. 2020. MP-SPDZ: A versatile framework for multi-party computation. In Proceedings of the CCS ’20: 2020 ACM SIGSAC Conference on Computer and Communications Security. Jay Ligatti, Xinning Ou, Jonathan Katz, and Giovanni Vigna (Eds.). ACM, 1575–1590. DOI: https://doi.org/10.1145/3372297.3417872

[10] Miran Kim, Yongsoo Song, Shuang Wang, Yuhou Xia, and Xiaojian Jiang. 2018. Secure Logistic Regression Based on Homomorphic Encryption: Design and Evaluation. JMLR 6, 2 (2018), e19. https://doi.org/10.2196/medinfor.8805

[11] Jakub Konečný, H. Brendan McMahan, Felix X. Yu, Peter Richtárik, Ananda Theertha Suresh, and Dave Bacon. 2016. Federated learning: Strategies for improving communication efficiency. arXiv:1610.05492. Retrieved from https://arxiv.org/abs/1610.05492

[12] Eunsang Lee, Joon-Woo Lee, Junghyun Lee, Young-Sik Kim, Yongjuimine Kim, Jong-Seon No, and Woosuk Choi. 2022. Low-Complexity Deep Convolutional Neural Networks on Fully Homomorphic Encryption Using Multiplexed Parallel Convolutions. In Proceedings of the 39th International Conference on Machine Learning (Proceedings of Machine Learning Research, Vol. 162), Kamalika Chaudhuri, Stefanie Jegelka, Le Song, Csaba Szepesvari, Gang Niu, and Sivan Sabato (Eds.). PMLR, 12403–12422. https://proceedings.mlr.press/v162/lee22e.html

[13] Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Agüera y Arcas. 2017. Communication-efficient learning of deep networks from decentralized data. In Proceedings of the 20th International Conference on Artificial Intelligence and Statistics. Aarti Singh and Xiaojin (Jerry) Zhu (Eds.), Proceedings of Machine Learning Research, Vol. 54, PMLR, 1273–1282. Retrieved from http://proceedings.mlr.press/v54/mcmahan17a.html

[14] H. Brendan McMahan, Eider Moore, Daniel Ramage, and Blaise Agüera y Arcas. 2016. Federated learning of deep networks using model averaging. arXiv:abs/1602.05629. Retrieved from https://arxiv.org/abs/1602.05629

[15] Payman Mohassel and Peter Rindal. 2018. ABY3: A mixed protocol framework for machine learning. In Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security. CCS 2018. David Lie, Mohammad Mannan, Michael Backes, and Xiaofeng Wang (Eds.). ACM, 35–52. DOI: https://doi.org/10.1145/3243734.3243760

[16] Payman Mohassel and Yupeng Zhang. 2017. SecureML: A system for scalable privacy-preserving machine learning. In Proceedings of the 2017 IEEE Symposium on Security and Privacy, SP 2017. IEEE Computer Society, 19–38. DOI: https://doi.org/10.1109/SP.2017.12

[17] Swaroop Ramaswamy, Rajiv Mathews, Kanishka Rao, and Françoise Beaufays. 2019. Federated learning for emoji prediction in a mobile keyboard. arXiv:1906.04329. Retrieved from https://arxiv.org/abs/1906.04329

[18] Adi Shamir. 1979. How to share a secret. Communications of the ACM 22, 11 (1979), 612–613. DOI: https://doi.org/10.1145/359168.359176

[19] Lichao Sun, Jianwei Qian, and Xun Chen. 2021. LDP-FL: Practical private aggregation in federated learning with local differential privacy. In Proceedings of the 30th International Joint Conference on Artificial Intelligence. Zhi-Hua Zhou (Ed.), ijcai.org, 1571–1578. DOI: https://doi.org/10.24963/ijcai.2021/217

[20] Sameer Wagh, Divya Gupta, and Nishanth Chandran. 2019. SecureNN: 3-party secure computation for neural network training. Proceedings of Privacy Enhancing Technologies 2019, 3 (2019), 26–49. DOI: https://doi.org/10.2478/popets-2019-0035

[21] Qianjun Wei, Qiuyi Yang, Peng Zhai, Zhiqiang Ge, and Yonggang Zhang. 2021. Privacy-preserving two-parties logistic regression on vertically partitioned data using asynchronous gradient sharing. Peer-to-Peer Networking and Applications 14, 3 (2021), 1379–1387. DOI: https://doi.org/10.1007/s12083-020-01017-x

[22] Yuncheng Wu, Shao Feng Cai, Xiaokui Xiao, Gang Chen, and Beng Chin Ooi. 2020. Privacy preserving vertical federated learning for tree-based models. Proceedings of the VLDB Endowment 13, 11 (2020), 2090–2103. Retrieved from http://www.vldb.org/pvldb/vol13/p2090-wu.pdf

[23] Qiuyi Yang, Yang Liu, Tianjian Chen, and Yongxin Tong. 2019. Federated machine learning: Concept and applications. ACM Transactions on Intelligent Systems and Technology 10, 2 (2019), 12:1–12:19. DOI: https://doi.org/10.1145/3298981

[24] Shengwen Yang, Bing Ren, Xuhui Zhou, and Liping Liu. 2019. Parallel distributed logistic regression for vertical federated learning without third-party coordinator. arXiv:1911.09824. Retrieved from https://arxiv.org/abs/1911.09824

[25] Andrew Chi-Chih Yao. 1982. Protocols for secure computations (extended abstract). In Proceedings of the 23rd Annual Symposium on Foundations of Computer Science. IEEE Computer Society, 160–164. DOI: https://doi.org/10.1109/SFCS.1982.38

ACM Transactions on Knowledge Discovery from Data, Vol. 18, No. 3, Article 51. Publication date: December 2023.
[26] Qiao Zhang, Cong Wang, Hongyi Wu, Chunsheng Xin, and Tran V. Phuong. 2018. GELU-Net: A globally encrypted, locally unencrypted deep neural network for privacy-preserved learning. In Proceedings of the 27th International Joint Conference on Artificial Intelligence, IJCAI 2018. Jérôme Lang (Ed.), ijcai.org, 3933–3939. DOI: https://doi.org/10.24963/ijcai.2018/547

[27] Yifei Zhang and Hao Zhu. 2020. Additively homomorphical encryption based deep neural network for asymmetrically collaborative machine learning. arXiv:2007.06849. Retrieved from https://arxiv.org/abs/2007.06849

[28] Di Zhao, Ming Yao, Wanwan Wang, Hao He, and Xin Jin. 2022. NTP-VFL - A new scheme for non-3rd party vertical federated learning. In Proceedings of the ICMLC 2022: 14th International Conference on Machine Learning and Computing. ACM, 134–139. DOI: https://doi.org/10.1145/3529836.3529841

[29] Huafei Zhu, Rick Siow Mong Goh, and Wee Keong Ng. 2020. Privacy-preserving weighted federated learning within the secret sharing framework. IEEE Access 8 (2020), 198275–198284. DOI: https://doi.org/10.1109/ACCESS.2020.3034602

[30] Ligeng Zhu and Song Han. 2020. Deep leakage from gradients. In Proceedings of the Federated Learning—Privacy and Incentive. Qiang Yang, Lixin Fan, and Han Yu (Eds.), Lecture Notes in Computer Science, Vol. 12500, Springer, 17–31. DOI: https://doi.org/10.1007/978-3-030-63076-8_2

Received 12 September 2022; revised 14 August 2023; accepted 12 October 2023