Hindawi
Security and Communication Networks
Volume 2022, Article ID 9983463, 19 pages
https://doi.org/10.1155/2022/9983463

Research Article

Privacy-Preserving Outsourcing Scheme for SVM on Vertically Partitioned Data

Guowei Qiu,[1] **Hua Huo**,[2] **Xiaolin Gui**,[1] **and Huijun Dai**[1]

[1]*School of Computer Science and Technology, Xi'an Jiaotong University, Xi'an 710049, China*
[2]*College of Information Engineering, Henan University of Science and Technology, Luoyang 471023, China*

Correspondence should be addressed to Hua Huo; pacific_huo@126.com

Received 8 October 2021; Revised 18 February 2022; Accepted 17 March 2022; Published 10 June 2022

Academic Editor: Zhili Zhou

Support vector machine (SVM) is an important technique for data classification. Traditional SVM assumes free access to data. If the data are split and held by different users, for privacy reasons, users are likely unwilling to submit their data to a third party for classification. In this paper, by using additive homomorphic encryption and random transformations (matrix transformation and vector decomposition), we design a privacy-preserving outsourcing scheme for conducting Least Squares SVM (LS-SVM) classification on vertically partitioned data. In our system, multiple data owners (users) submit their encrypted data to two non-colluding service providers, which conduct SVM algorithm on it. During the execution of our algorithm, neither service provider learns anything about the input data, the intermediate results, or the predicted result. In other words, our algorithm is encrypted in the whole process. Extensive theoretical analysis and experimental evaluation demonstrate the correctness, security, and efficiency of the method.

1. Introduction

1.1. Background. SVM [1] is a powerful machine learning algorithm. It uses some given data points to learn a model which separates the feature space into two parts. All data in each part are considered to belong to one category. Then, the model could be used to predict the category of a new data point. Thus far, SVM classification has been successfully used in many fields, such as pattern recognition [2], diagnosis and classification of diseases [3, 4], and financial forecast [5]. Traditional SVM supposes that data are centralized and can be freely accessed. However, there is a widespread practice in today's cloud computing environment; that is, the data for learning are split and held by multiple users, and the learning process is outsourced to some cloud service providers. For example, multiple financial institutions (banks, insurance companies, etc.) want to construct a model used for assessing personal credit. Clearly, the more the data used for learning are, the more accurate the model will be. To obtain a more accurate model, these institutions should contribute their data to a service provider for learning a SVM model. However, in this procedure, the traditional SVM will expose their data to the service providers. This is not permitted because of legal and moral constraints. The key to solving this problem is to develop a privacy-preserving SVM (PP-SVM) algorithm which obtains valid results without disclosing the users' data.

PP-SVM is a privacy-preserving machine learning (PPML) algorithm [6]. Generally speaking, a PPML algorithm modifies the original machine learning algorithm from two aspects: changing the system architecture and combining different security techniques with the original algorithm. For the first aspect, some architectures used by people include one client and one server [7, 8], multiclient and one server [9, 10], and multiclient and multiserver [11] architecture. Thus far, it is still difficult to design a highly efficient algorithm for one server architecture. In contrast, multiserver provides more possibilities for designing a highly efficient algorithm, but it also increases communication costs. For the second aspect, the security techniques often used in PPML include fully homomorphic encryption [7, 12], additive homomorphic encryption [11, 13–15], secret

sharing [15–17], garbled circuits [17, 18], and differential privacy [19]. In this paper, we mainly use additive homomorphic encryption to construct our PP-SVM on a multiclient/two-server architecture.

1.2. Related Work. In the last two decades, some attention has been given to PP-SVM algorithm.

Laur et al. [8] designed a cryptographically private SVM with additive homomorphic encryption [20], linear secret sharing [21], and conditional oblivious transfer [22]. Their algorithm only works on a system consisting of one server and one client, where the server owns the input data points and the client owns the corresponding class labels. Their system is not suitable for multiclient situation.

Yu et al. [23] proposed an algorithm for PP-SVM based on secure set intersection cardinality [9] and commutative public-key encryption. Vaidya et al. [14] constructed three PP-SVM methods using secure addition, secure scalar product, and homomorphic encryption for vertically, horizontally, and arbitrarily partitioned data. The algorithms proposed by Yu et al. and Vaidya et al. are secure multiparty computing system. Omer et al. [13] proposed a PP-SVM algorithm based on additive homomorphic encryption, which is suitable for multiclient and one server systems. In above three algorithms, the input data points are concealed, but the corresponding class labels and kernel matrix are exposed to other participants. Exposing these messages to others can sometimes reveal some privacy about the input data (see the appendix in [24]). Therefore, we want to hide all this information in our algorithm.

Liu et al. [12] proposed their PP-SVM algorithm based on fully homomorphic encryption [25, 26] and secure addition method. In their algorithm, each user communicates with the server many times, and each user performs a lot of computation by itself. These operations increase the workload of users and make their system look more like a collaborative rather than an outsourcing computing system.

Park et al. [7] used CKKS [27] scheme, a fully homomorphic encryption, to construct a training algorithm based on Least Squares SVM [28]. Their method is suitable for the case of one client and one server, not directly for the case of multiple users.

Thus far, all known methods of fully homomorphic encryption are still not efficient enough for practical use. We hope to design a sufficiently effective algorithm, which should reduce the work of users as much as possible.

Wang et al. [11] first built a secure computation system and then designed a PP-SVM algorithm on it. Their computation system consists of eight secure operations, including secure integer multiplication, secure inner product, and secure floating-point addition/subtraction. They use a distributed two-trapdoor public-key cryptosystem [29] to set up this computation system, which ensures that their algorithm has better performance. This method can work on a multiclient/multiserver architecture. Compared with their methods, our method in this paper is simple and easy to understand.

In addition, because our algorithm is based on LS-SVM, the service providers need to solve a linear system of equations. To ensure the data security, our algorithm employs random transformations to hide the coefficients of the equations. Similar data masking techniques have been used by some researchers. Lei et al. [30] proposed a secure outsourcing method to compute the inverse matrix. They use a sparse matrix to mask the original matrix, and then send it to the cloud for computing the inverse matrix. Chen et al. [31] proposed a secure outsourcing scheme for solving large-scale systems of linear equations based on matrix transformation in the fully malicious model. Chen et al. [32] designed two protocols for secure outsourcing of linear regression. They use random orthogonal matrix to mask the original equations and then outsource the equation solving process to the cloud. However, all their methods only work in one client and one server architecture. In this paper, we combine homomorphic encryption and random transformation technique to tackle the problem under multiclient/two-server framework.

1.3. The Target Problem, Architecture, and Security Model

1.3.1. Target Problem. In this paper, we focus on the data privacy issues when multiple users submit their data to some cloud servers for SVM classification.

Specifically, suppose there are m data pairs $(\mathbf{x}_1, y_1)$, $\ldots$, $(\mathbf{x}_m, y_m)$ for training and a new data point $\mathbf{x}$ for predicting, where each y_i is the class label corresponding to the data point $\mathbf{x}_i$ and is shared by all users. Each vector $\mathbf{x}_i$ (including $\mathbf{x}$) is vertically partitioned among r users as shown in Figure 1, where

$$\mathbf{x}_i = \left(x_1^{(i)}, x_2^{(i)}, \ldots, x_n^{(i)}\right) = \left(\mathbf{x}_{i_1}, \mathbf{x}_{i_2}, \ldots, \mathbf{x}_{i_r}\right). \quad (1)$$

The $\mathbf{x}_{i_k} = (x_{n_{k-1}+1}^{(i)}, x_{n_{k-1}+2}^{(i)}, \ldots, x_{n_k}^{(i)}) \in \mathbb{R}^{n_k - n_{k-1}}$ is the private data of user$_k$ who is unwilling to share it with others. Meanwhile, all users do not want to reveal y_i to any server.

We want to design a PP-SVM algorithm that enables the cloud server to perform SVM algorithm using all user data without learning any input data or prediction results.

1.3.2. Our System Architecture. Our system architecture consists of user user$_A$, multiple users, and two service providers (SP1 and SP2), as shown in Figure 2. They each undertake the following tasks.

(1) user$_A$: It sends a service request to all data owners (users). Then, all users and service providers perform PP-SVM algorithm. user$_A$ may or may not be one of the data owners.

(2) Users: Each user provides different attribute values of the same set of objects to a service provider in an encrypted form. The data can be used for training or prediction.

(3) SP1, SP2: They collaboratively learn a SVM model and predict the category of a new data point. In the training process, SP1 plays the role of Evaluator and

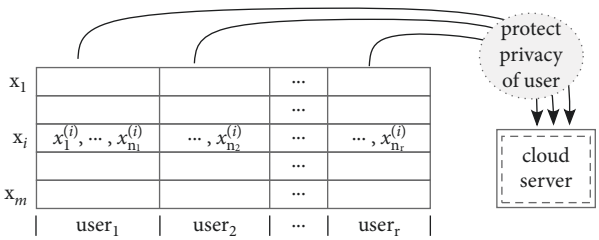

Figure 1: The problem model.

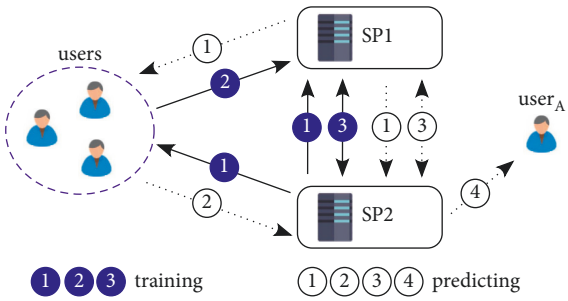

Figure 2: The system architecture.

SP2 plays the role of Crypto Service Provider (CSP). In the predicting process, SP1 and SP2 switch their roles. These two roles have the following responsibilities.

(1) CSP generates a key pair (pk, sk) of the Paillier cryptosystem [20] and then sends the public key pk to all other participators. It also performs some computations to help Evaluator execute our algorithm.
(2) Evaluator executes our training and predicting algorithms by collaborating with CSP. It performs all homomorphic computations in our protocols.

Figure 2 shows that the training and predicting procedure can be divided into the following steps as a whole.

(1) CSP sends the public key to other participators.
(2) All users send encrypted data to Evaluator.
(3) Evaluator and CSP collaboratively perform the privacy-preserving training or predicting protocols.
(4) Evaluator sends the predicted result to user_A.

1.3.3. Security Model. Each party in our system is semi-honest. This means that each service provider or user performs our protocols faithfully but hopes to learn some privacy of others by observing the execution of the protocols.

Meanwhile, we suppose the two service providers are non-colluding. Service providers are generally large companies that are heavily regulated; we believe this condition is not difficult to achieve.

In addition, we found that our protocol for computing RBF kernel (Protocol 3) may be inefficient. Therefore, we designed a new protocol (Protocol 4) as an alternative. However, the new protocol requires that there is no collusion between any two participants.

1.4. Our Contributions. In this paper, we propose a privacy-preserving outsourcing scheme for SVM classification on vertically partitioned data. Our algorithm is based on LS-SVM. Paillier encryption and random transformation are the main techniques in this work. Our algorithm has the following features.

1.4.1. Secure Outsourced SVM Training. Our algorithm enables two service providers to solve LS-SVM equation in an encrypted state. In this procedure, the two service providers cannot learn the input data (all $\mathbf{x}_i$ and y_i), the solution of LS-SVM equation, and the training parameters.

1.4.2. Secure Outsourced SVM Prediction. Our algorithm enables two service providers to make prediction for a new data point in an encrypted state. In this procedure, the two service providers learn nothing about the input data or the value of the decision function. The encrypted outcome will be sent to user user_A, and then user_A can use it to compute the value of the decision function by himself.

1.4.3. Concealing Important Intermediate Results. In our algorithm, the kernel matrix and decision function are hidden, which makes it more difficult for the service provider to infer some information about the input data.

In addition, we conduct a comprehensive security analysis of the protocol and prove its security under various collusion scenarios. Experiments show that our algorithm has good accuracy and performance.

2. Preliminaries

In general, we use lowercase letters to indicate numbers (e.g., x_i, y_i, a, α_i), lowercase boldface letters (e.g., $\mathbf{x}_i$, $\mathbf{e}$, $\boldsymbol{\alpha}$) for vectors, and uppercase boldface letters (e.g., $\mathbf{Q}$, $\mathbf{A}$) for matrices.

2.1. SVM and LS-SVM Overview. SVM algorithm consists of two phases: training and predicting. In SVM training phase, it uses all training data to learn a linear or nonlinear model represented by a decision function $f(\mathbf{x})$. Then, in SVM prediction phase, it computes $f(\mathbf{x})$ at a given $\mathbf{x}$. The function value is used to predict the category of the given data.The SVM model is actually a hyperplane called decision boundary, which separates the training data into two categories. The training process finds the hyperplane by maximizing the margin between two categories of data; Figure 3 shows a two-dimensional case. It can be described as follows. Suppose there are m data pairs $(\mathbf{x}_1, y_1)$, $\ldots$, $(\mathbf{x}_m, y_m)$ for SVM training, where $\mathbf{x}_i \in \mathbb{R}^n$ is a vector in the feature space and $y_i \in \{-1, +1\}$ is the class label of $\mathbf{x}_i$. The SVM learns a decision boundary $f(\mathbf{x}) = \langle \mathbf{w}, \mathbf{x} \rangle + b = 0$ by solving the following optimization problem:

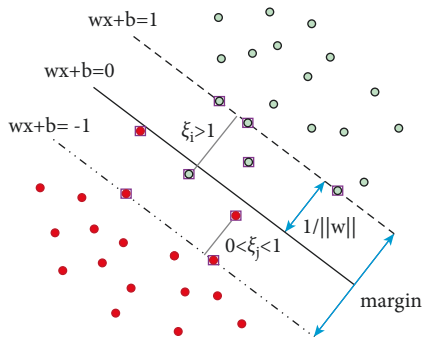

Figure 3: The SVM maximizes the margin between two classes of data (points in squares are support vectors).

$$\min_{\mathbf{w},b,\xi} \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{m}\xi_i$$
$$\text{s.t. } y_i\left(\langle \mathbf{w}, \mathbf{x}_i\rangle + b\right) \geq 1 - \xi_i, \xi_i > 0, \quad \text{for } i = 1, \ldots, m, \tag{2}$$

where $\xi = (\xi_1, \ldots, \xi_m)$ is the slack variables that allow some data points to lie within the margin. Figure 2 is a two-dimensional example. The classification rule induced by $f(\mathbf{x})$ is $\operatorname{sgn}(f(\mathbf{x}))$. The problem equation (2) is usually solved in its dual form, which often combines with the kernel trick to deal with the nonlinear decision boundary. Finally, the form of its dual problem is

$$\min_{\alpha} \frac{1}{2}\sum_{i,j=1}^{m}\alpha_i\alpha_j y_i y_j K\left(\mathbf{x}_i, \mathbf{x}_j\right) - \sum_{i=1}^{m}\alpha_i$$
$$\text{s.t. } \sum_{i=1}^{m}\alpha_i y_i = 0, \quad 0 \leq \alpha_i \leq C\,(i = 1, \ldots, m), \tag{3}$$

where $K(\mathbf{x}_i, \mathbf{x}_j) = \langle \phi(\mathbf{x}_i), \phi(\mathbf{x}_j)\rangle$ is a kernel function, and $\phi(\mathbf{x})$: $\mathbb{R}^n \longrightarrow \mathbb{R}^l$ is a basis function. The decision function is

$$f(\mathbf{x}) = \sum_{i=1}^{m}\alpha_i y_i K(\mathbf{x}_i, \mathbf{x}) + b, \tag{4}$$

where $\alpha_i\,(0 \leq i \leq m)$ are the solutions to the dual problem equation (3). b can be obtained as follows: select α_j that satisfies $1 \leq \alpha_j \leq C$; compute $b = y_j - \sum_{i=1}^{m}\alpha_i y_i K(\mathbf{x}_i, \mathbf{x}_j)$.

Three popular choices for kernel function $K(\mathbf{x}_i, \mathbf{x}_j)$ are

$$\begin{aligned} &\text{Linear kernel} && : \langle \mathbf{x}_i, \mathbf{x}_j\rangle \\ &\text{Polynomial kernel} && : \left(a\langle \mathbf{x}_i, \mathbf{x}_j\rangle + c\right)^p \\ &\text{RBF kernel} && : \exp\left(-\sigma\left\|\mathbf{x}_i - \mathbf{x}_j\right\|^2\right) \end{aligned} \quad . \tag{5}$$

The matrix $\mathbf{K} = (K(\mathbf{x}_i, \mathbf{x}_j))_{m\times m}$ is the kernel matrix.

Although problem equation (3) is only quadratic programming, it is still difficult to solve in the encrypted state, because the inequality constraints are difficult to deal with. The algorithm in this paper is based on the following LS-SVM.

$$\min_{\mathbf{w},b,e} \frac{1}{2}\|\mathbf{w}\|^2 + \frac{\gamma}{2}\sum_{i=1}^{m}e_i^2$$
$$\text{s.t. } y_i\left(\langle \mathbf{w}, \phi(\mathbf{x}_i)\rangle + b\right) = 1 - e_i, \quad \text{for } i = 1, \ldots, m. \tag{6}$$

We can also transform this problem to its dual form with Lagrange multiplier method. Likewise, we can apply the kernel trick to its dual problem. The kernel functions (5) can also be used in LS-SVM. Most important of all, we can obtain the coefficients of its decision function by solving LS-SVM equation as follows:

$$\mathbf{Q}\boldsymbol{\beta} = \mathbf{e}, \tag{7}$$

where

$$\mathbf{Q} = \begin{bmatrix} 0 & y_1 & \cdots & y_m \\ y_1 & & & \\ \vdots & & \boldsymbol{\Omega} + \gamma^{-1}\mathbf{I} & \\ y_m & & & \end{bmatrix} \boldsymbol{\beta} = \begin{bmatrix} b \\ \alpha_1 \\ \vdots \\ \alpha_m \end{bmatrix} \mathbf{e} = \begin{bmatrix} 0 \\ 1 \\ \vdots \\ 1 \end{bmatrix}, \tag{8}$$

and $\omega_{ij} = y_i y_j K(\mathbf{x}_i, \mathbf{x}_j)$ is the (i, j)th entry of $\boldsymbol{\Omega}$. In this paper, we will solve this linear equation in an encrypted form and then secretly compute $f(\mathbf{x})$ for a new data point $\mathbf{x}$. For more details about SVM, see [1, 28].

2.2. The Paillier Cryptosystem. The Paillier cryptosystem is an additive homomorphic encryption scheme which satisfies semantic security [33]. Semantic security makes it impossible for any polynomial algorithm to gain extra information about a plaintext when given only its ciphertext and public key. As an asymmetric encryption, Paillier cryptosystem has a key pair (pk, sk), where pk is the public key and sk is the private key.

Let $E(\cdot)$ be the encryption algorithm and $D(\cdot)$ be the decryption algorithm. Given $a, b \in \mathbb{Z}_n$, the Paillier encryption satisfies the following properties:

$$E(a + b) = E(a) \cdot E(b) \bmod n^2,$$
$$E(\mathrm{ka}) = E(a)^k \bmod n^2 \quad \forall k \in \mathbb{Z}_n. \tag{9}$$

For more details, see chapter 13 of [34].

In our system, CSP generates the key pair (pk, sk) and sends pk to other parties. To simplify the description, for any $\mathbf{A} = (a_{\mathrm{ij}})_{m\times n}$ or $\mathbf{b} = (b_i)_m$, we use $E(\mathbf{A})$ or $E(\mathbf{b})$ to denote a matrix or vector whose each entry is $E(a_{\mathrm{ij}})$ or $E(b_i)$.

2.3. Data Representation. We use fixed-point representation in our system. More precisely, we represent a real number with a fixed-point integer. For a fixed-point real number a which dedicates q bits to the fractional part, we use the integer $a \cdot 2^q$ to represent it. Supposing a and b are fixed-point numbers in our system, we perform the arithmetic operations as follows:

$$(a \pm b)_{\text{fixed}} = a \pm b,$$
$$(\text{ab})_{\text{fixed}} = \left\lfloor \frac{(\text{ab})}{2^q} \right\rfloor, \tag{10}$$
$$\left(\frac{a}{b}\right)_{\text{fixed}} = \left\lfloor \left(\frac{a}{b}\right) \cdot 2^q \right\rfloor \quad \text{for } a, b > = 0,$$

where the operation $\lfloor \cdot \rfloor$ rounds a real number down to the nearest integer.

Because the cryptosystem works only for nonnegative integers, we use 2's complement in our system to ensure all numbers are positive. However, this causes another problem, the operation $(a/b)_{\text{fixed}}$ listed above is not correct for the complement of a negative number. Therefore, when performing a division operation, we first transform the 2's complement to its true form and then perform division on it, and the result of division will be transformed back to the complement for subsequent computation.

2.4. The Protocol $EncMul(E(a), E(b)) \longrightarrow E(ab)$. The protocol EncMul $(\cdot, \cdot)$ is a two-party protocol that accomplishes the following task: Evaluator provides $E(a), E(b)$; CSP keeps the private key; finally, Evaluator obtains $E(ab)$, and the two servers cannot learn anything about a or b.

Elmehdwi et al. [35] proposed this protocol and Liu et al. [29] gave a similar one which used multiple keys. This protocol is used in our methods to compute the polynomial or RBF kernel functions. The description of the EncMul protocol is as follows:

(1) Evaluator chooses two random positive integers r_1 and r_2; computes $E(a + r_1) = E(a) \cdot E(r_1)$, $E(b + r_2) = E(b) \cdot E(r_2)$; and then sends $E(a + r_1)$, $E(b + r_2)$ to CSP.

(2) CSP obtains $a + r_1 = D(E(a + r_1))$, $b + r_2 = D(E(b + r_2))$; then computes

$$T = E(\text{ab} + \text{ar}_2 + \text{br}_1 + r_1 r_2) = E((a + r_1)(b + r_2)); \tag{11}$$

and returns T to Evaluator.

(3) Evaluator computes

$$E(\text{ab}) = T \cdot E(\text{ar}_2)^{-1} \cdot E(\text{br}_1)^{-1} \cdot E(r_1 r_2)^{-1}. \tag{12}$$

In this protocol, $E(x)^{-1}$ is the modular inverse of $E(x)$. This protocol is based on the equation $\text{ab} = (a + r_1)(b + r_2) - \text{ar}_2 - \text{br}_1 - r_1 r_2$.

3. Our PP-SVM Training Algorithm

In the training phase, SP1 is Evaluator and SP2 is CSP.

3.1. Main Steps of Our PP-SVM Training Algorithm. The key of our training algorithm is to solve the LS-SVM equation $\mathbf{Q}\beta = \mathbf{e}$ securely. Figure 4 shows the procedure of PP-SVM training, which includes several steps as follows:

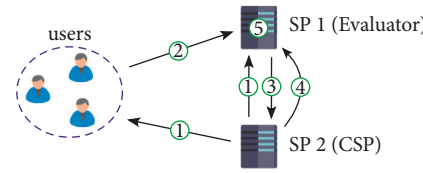

Figure 4: The procedure of our PP-SVM training.

(1) SP2 sends pk_2 to other participators.

(2) All users send encrypted data to SP1.

(3) SP1 computes $E(\mathbf{Q})$ and $E(\mathbf{C}) = E(\mathbf{QR})$ and then sends $E(\mathbf{C})$ to SP2 ($\mathbf{R}$ is used for masking $\mathbf{Q}$).

(4) SP2 decrypts $E(\mathbf{C})$, solves $\mathbf{C}\delta = \mathbf{e}$, decomposes δ into $\delta = t_1\delta_1 + t_2\delta_2$, and then sends δ_1, δ_2 to SP1 ($t_1, t_2 \in \mathbb{R}$ and $\delta_1, \delta_2 \in \mathbb{R}^{m+1}$).

(5) SP1 computes $\zeta = \mathbf{R}\delta_1$, $\eta = \mathbf{R}\delta_2$ (removing perturbation).

Finally, the SP1 has random vectors ζ, η; SP2 has random numbers t_1, t_2. These data satisfy $\beta = t_1\zeta + t_2\eta$, where β is the solution of LS-SVM equation. Because the two service providers do not collude, neither of them knows β.

Furthermore, during the training procedure, SP1 and SP2 should not learn anything about $\mathbf{Q}$. The reason for this is that someone can learn the kernel matrix from $\mathbf{Q}$, because the difference between them is small, and thus can learn some information about $\mathbf{x}_i$ from the kernel matrix, especially when the kernel function is simple (see the appendix in [24]).

We use the notations $\text{SP}1_{(\text{E})}$ and $\text{SP}2_{(\text{C})}$ instead of SP1 and SP2 in this section to make their roles easy to identify.

3.2. Securely Computing Matrices $E(\Omega)$ and $E(\mathbf{Q})$

3.2.1. Case 1: PP-SVM with Linear Kernel. For a linear-kernel matrix, its (i, j)th entry $K(\mathbf{x}_i, \mathbf{x}_j) = \langle \mathbf{x}_i, \mathbf{x}_j \rangle = \sum_{k=1}^{r} \langle \mathbf{x}_{i_k}, \mathbf{x}_{j_k} \rangle$. For $\Omega = (\omega_{\text{ij}})_{m \times n}$, we have

$$\begin{aligned} E(\omega_{\text{ij}}) &= E(y_i y_j K(\mathbf{x}_i, \mathbf{x}_j)) \\ &= E\left(\sum_{k=1}^{r} \langle y_i \mathbf{x}_{i_k}, y_j \mathbf{x}_{j_k} \rangle\right) \\ &= \prod_{k=1}^{r} E(\langle y_i \mathbf{x}_{i_k}, y_j \mathbf{x}_{j_k} \rangle). \end{aligned} \tag{13}$$

Based on (13), we design Protocol 1.

Protocol 1. Compute $E(\mathbf{Q})$ for linear-kernel PP-SVM.

Input: user_k provides $\mathbf{x}_{i_k}$; each y_i is shared by all users, where $1 \le i \le m$ and $1 \le k \le r$.

Output: SP1 obtains $E(\mathbf{Q})$.

(1) Each user_k computes $E(\langle y_{i_k} \mathbf{x}_{i_k}, y_j \mathbf{x}_{j_k} \rangle)$ locally and sends them to SP1, where $1 \le i, j \le m$, $1 \le k \le r$.

(2) For any $1 \le i, j \le m$, SP1 computes $E(\Omega)$ as follows:

$E(\omega_{ij}) = \prod_{k=1}^{r} E(\langle y_i \mathbf{x}_{i_k}, y_j \mathbf{x}_{j_k} \rangle)$.

(3) User sends $E(\gamma^{-1})$ and all $E(y_i)$ to SP1, who computes each $E(\omega_{ii} + \gamma^{-1}) = E(\omega_{ii}) \cdot E(\gamma^{-1})$. Then, SP1 obtains $E(\mathbf{Q})$ (γ is the parameter in (7)).

3.2.2. Case 2: PP-SVM with Polynomial Kernel. For a polynomial kernel matrix, its (i, j)th entry $K(\mathbf{x}_i, \mathbf{x}_j) = (a\langle \mathbf{x}_i, \mathbf{x}_j \rangle + c)^p = (\sum_{k=1}^{r} a\langle \mathbf{x}_{i_k}, \mathbf{x}_{j_k} \rangle + c)^p$. Hence,

$$E(\omega_{ij}) = E(y_i y_j K(\mathbf{x}_i, \mathbf{x}_j)) = E\left(y_i y_j \left(\sum_{k=1}^{r} a\langle \mathbf{x}_{i_k}, \mathbf{x}_{j_k} \rangle + c \right)^p \right). \quad (14)$$

Based on (14), we design Protocol 2.

Protocol 2. Compute $E(\mathbf{Q})$ for polynomial kernel PP-SVM.

Input: user$_k$ provides $\mathbf{x}_{i_k}$ all y_i, and a, c, p in the kernel function are shared by all users $(1 \le i \le m, 1 \le k \le r)$.

Output: SP $1_{(E)}$ obtains $E(\mathbf{Q})$.

(1) user$_1$ computes $E(y_i y_j)$, $E(c)$, and $E(a\langle \mathbf{x}_{i_1}, \mathbf{x}_{j_1} \rangle)$; other users compute $E(a\langle \mathbf{x}_{i_k}, \mathbf{x}_{j_k} \rangle)(1 \le i, j \le m, 2 \le k \le r)$. Then, all users send these data to SP $1_{(E)}$.

(2) For $1 \le i, j \le m$, SP $1_{(E)}$ computes

$$T_{ij} = E\left(\sum_{k=1}^{r} a\langle \mathbf{x}_{i_k}, \mathbf{x}_{j_k} \rangle + c \right) = \prod_{k=1}^{r} E(a\langle \mathbf{x}_{i_k}, \mathbf{x}_{j_k} \rangle) \cdot E(c). \quad (15)$$

(3) user$_1$ sends p to SP $1_{(E)}$. For $1 \le i, j \le m$, let $S_{ij} = T_{ij}$. SP $1_{(E)}$ and SP $2_{(C)}$ repeatedly execute

$$S_{ij} = \text{EncMul}(S_{ij}, T_{ij}) \quad (16)$$

$p - 1$ times. Then, SP $1_{(E)}$ obtains

$$E(K(\mathbf{x}_i, \mathbf{x}_j)) = S_{ij} = E\left(\left(\sum_{k=1}^{r} a\langle \mathbf{x}_{i_k}, \mathbf{x}_{j_k} \rangle + c \right)^p \right). \quad (17)$$

(4) SP $1_{(E)}$ and SP $2_{(C)}$ compute

$$E(\omega_{ij}) = \text{EncMul}(S_{ij}, E(y_i y_j))(1 \le i, j \le m). \quad (18)$$

Then, SP $1_{(E)}$ obtains $E(\Omega)$.

(5) user$_1$ sends $E(\gamma^{-1})$ and all $E(y_i)$ to SP $1_{(E)}$, who computes $E(\omega_{ii} + \gamma^{-1}) = E(\omega_{ii}) \cdot E(\gamma^{-1})$. Then, SP $1_{(E)}$ obtains $E(\mathbf{Q})$.

Remark 1. Because y_i cannot be disclosed to service provider, two service providers must execute EncMul to generate $E(\omega_{ij})$, as shown in step 4. Another method is that SP1 sends all $E(K(\mathbf{x}_i, \mathbf{x}_j))$ to one user, who computes $E(\omega_{ij}) = E(K(\mathbf{x}_i, \mathbf{x}_j))^{y_i y_j}$ and returns it to SP1. In this way, two service providers can reduce the execution of EncMul once.

In addition, the degree p is the only parameter sent directly to SP1.

3.2.3. Case 3: PP-SVM with RBF Kernel. For a RBF kernel matrix, its (i, j)th entry

$$\begin{aligned} K(\mathbf{x}_i, \mathbf{x}_j) &= \exp\left(-\sigma \left\| \mathbf{x}_i - \mathbf{x}_j \right\|^2\right) \\ &= \exp\left(-\sigma\left(\langle \mathbf{x}_i, \mathbf{x}_i \rangle - 2\langle \mathbf{x}_i, \mathbf{x}_j \rangle + \langle \mathbf{x}_j, \mathbf{x}_j \rangle\right)\right) \\ &= \exp\left(-\sigma \sum_{k=1}^{r} \left(\langle \mathbf{x}_{i_k}, \mathbf{x}_{i_k} \rangle - 2\langle \mathbf{x}_{i_k}, \mathbf{x}_{j_k} \rangle + \langle \mathbf{x}_{j_k}, \mathbf{x}_{j_k} \rangle \right) \right) \\ &= \prod_{k=1}^{r} \exp\left(-\sigma \langle \mathbf{x}_{i_k}, \mathbf{x}_{i_k} \rangle - 2\langle \mathbf{x}_{i_k}, \mathbf{x}_{j_k} \rangle + \langle \mathbf{x}_{j_k}, \mathbf{x}_{j_k} \rangle\right). \end{aligned} \quad (19)$$

Let $\Delta_k^{ij} = \exp(-\sigma(\langle \mathbf{x}_{i_k}, \mathbf{x}_{i_k} \rangle - 2\langle \mathbf{x}_{i_k}, \mathbf{x}_{j_k} \rangle + \langle \mathbf{x}_{j_k}, \mathbf{x}_{j_k} \rangle))$; we have

$$E(\omega_{ij}) = E(y_i y_j K(\mathbf{x}_i, \mathbf{x}_j)) = E\left(y_i y_j \prod_{k=1}^{r} \Delta_k^{ij} \right). \quad (20)$$

According to (9), we can design the following two methods for computing $E(\Omega)$:

(A) SP $1_{(E)}$ collects $E(\Delta_k^{ij})$ and $E(y_i y_j \Delta_1^{ij})$ from users. Then, two servers use EncMul to compute $E(\omega_{ij})$.

(B) user$_1$ chooses some random h^{ij}; all users try to compute $T_{ij} = h^{ij} \prod_{k=1}^{r} \Delta_k^{ij}$. SP $1_{(E)}$ collects $E(y_i y_j / h^{ij})$ and T_{ij}; then, it can compute $E(\omega_{ij})$ with them.

Protocol 3 is designed based on method A.

Protocol 3. (method A)Compute $E(\mathbf{Q})$ for PP-SVM with RBF kernel.

Input: user$_k$ provides $\mathbf{x}_{i_k}$, and each y_i is shared by all users, where $1 \le i \le m$ and $1 \le k \le r$.

Output: SP $1_{(E)}$ obtains $E(\mathbf{Q})$.

(1) For any $1 \le i, j \le m$, user$_k$ $(2 \le k \le r)$ computes $E(\Delta_k^{ij})$ and user$_1$ computes $E(y_i y_j \Delta_1^{ij})$. Then, all users send these data to SP $1_{(E)}$.

(2) Let $T_{ij} = E(y_i y_j \Delta_1^{ij})$. SP $1_{(E)}$ and SP $2_{(C)}$ repeatedly execute

$$T_{ij} = \text{EncMul}\left(E\left(\Delta_k^{ij}\right), T_{ij}\right), \tag{21}$$

for $k = 2, 3, \ldots, r$. Then, SP $1_{(E)}$ obtains

$$E\left(\omega_{ij}\right) = T_{ij} = E\left(y_i y_j \prod_{k=1}^{r} \Delta_k^{ij}\right) \quad (1 \le i, j \le m). \tag{22}$$

(3) user_1 sends $E(\gamma^{-1})$ and all $E(y_i)$ to SP $1_{(E)}$, who computes $E(\omega_{ii} + \gamma^{-1}) = E(\omega_{ii}) \cdot E(\gamma^{-1})$. Then, SP $1_{(E)}$ obtains $E(\mathbf{Q})$.

Remark 2. Protocol 3 will call EncMul $r - 1$ times. This leads to inefficiency because EncMul is a time-consuming protocol.

Protocol 4 is designed based on the method B.

Protocol 4. (method B) Compute $E(\mathbf{Q})$ for PP-SVM with RBF kernel.

Input: user_k provides $\mathbf{x}_{i_k}$, and each y_i is shared by all users, where $1 \le i \le m$ and $1 \le k \le r$.

Output: SP $1_{(E)}$ obtains $E(\mathbf{Q})$.

(1) user_1 chooses random real numbers $h^{ij} > 1$ $(1 \le i, j \le m)$ and a random integer $f > 0$, where $f/h^{ij} > 2^g$, $g > 10$. Then, user_1 computes $T_{ij} = h^{ij}\Delta_1^{ij}$ for $1 \le i, j \le m$.

(2) user_1 sends each T_{ij} to user_2, and user_2 computes $T_{ij} = T_{ij}\Delta_2^{ij}$ and then sends T_{ij} to user_3. One by one, other users do the same thing until user_r computes $T_{ij} = T_{ij}\Delta_r^{ij}$ and sends these data to SP $1_{(E)}$.

(3) user_1 computes $E(\lfloor f y_i y_j / h^{ij} \rfloor)$ for $1 \le i, j \le m$; sends $E(\lfloor f y_i y_j / h^{ij} \rfloor)$ to SP $1_{(E)}$; and sends f to SP $1_{(E)}$ and SP $2_{(C)}$.

(4) SP $1_{(E)}$ computes $E(\Omega)$ as follows:

$$E\left(f^2\omega_{ij}\right) = E\left(\frac{f^2 y_i y_j}{h^{ij}} \prod_{k=1}^{r} \Delta_k^{ij}\right) \approx E\left(\left\lfloor \frac{\text{fy}_i y_j}{h^{ij}} \right\rfloor\right)^{\lfloor f T_{ij} \rfloor}, \tag{23}$$

where $1 \le i, j \le m$.

(5) user_1 sends $E(f^2\gamma^{-1})$ and all $E(f^2 y_i)$ to SP $1_{(E)}$, who computes $E(f^2\omega_{ii} + f^2\gamma^{-1}) = E(f^2\omega_{ii}) \cdot E(f^2\gamma^{-1})$. Then, SP $1_{(E)}$ obtains $E(f^2\mathbf{Q})$.

Remark 3. The data $|y_i y_j / h^{ij}| < 1$. To improve the accuracy, we multiply $y_i y_j / h^{ij}$ by f and use the integer part of $\text{fy}_i y_j / h^{ij}$ in subsequent computations. Proposition 1 will illustrate why it require $f/h^{ij} > 2^g$ and $g > 10$.

In this paper, we use Linear-SVM, Poly-SVM, RBF-SVM(A), and RBF-SVM(B) to denote the PP-SVM methods which use Protocols 1–4, respectively.

The RBF-SVM(B) is far more efficient than RBF-SVM(A), but it requires that any two participants do not collude with each other (see Section 3.5.2).

3.3. Securely Computing β and Splitting It between Two Service Providers. The decision function, which we call SVM model, can be written as follows:

$$f(\mathbf{x}) = \sum_{i=1}^{m} \alpha_i y_i K(\mathbf{x}_i, \mathbf{x}) + b = \boldsymbol{\beta} \cdot \mathbf{k}(\mathbf{x}), \tag{24}$$

where

$$\begin{aligned} \boldsymbol{\beta} &= (b, \alpha_1, \alpha_2, \ldots, \alpha_m), \\ \mathbf{k}(\mathbf{x}) &= (1, y_1 K(\mathbf{x}_1, \mathbf{x}), \ldots, y_m K(\mathbf{x}_m, \mathbf{x})). \end{aligned} \tag{25}$$

In this section, the two service providers will securely compute β. Because β discloses the relationship between the input points and the hyperplane $f(\mathbf{x})$ to some extent, we do not reveal β to service provider. To this end, we will randomly decompose β into a linear combination, then split this combination into two parts, and provide different parts to Evaluator and CSP. This ensures that both of the service providers learn nothing about β.

In a word, we require that none of the service providers can learn anything about $\mathbf{x}_i$, y_i, β, or $K(\mathbf{x}_i, \mathbf{x})$ when executing our algorithm.

Protocol 5. Compute β and split it between two servers.

Input: SP $1_{(E)}$ inputs $E(\mathbf{Q})$; CSP has the private key.

Output: SP $1_{(E)}$ obtains vectors ζ, η; SP $2_{(C)}$ obtains real numbers t_1, t_2. These data satisfy $\beta = t_1\zeta + t_2\eta$.

(1) $SP1_{(E)}$ chooses an invertible random $\mathbf{R} \in \mathbb{Z}_+^{(m+1)\times(m+1)}$. For Linear-SVM, Poly-SVM, and RBF-SVM(A), let $\mathbf{C} = \mathbf{QR}$; SP $1_{(E)}$ computes each entry of $E(\mathbf{C})$ as follows:

$$E\left(C_{ij}\right) = E\left(\sum_{k=1}^{m+1} Q_{ik} R_{kj}\right) = \prod_{k=1}^{m+1} E\left(Q_{ik}\right)^{R_{kj}}. \tag{26}$$

For RBF-SVM(B), let $\mathbf{C} = (f^2\mathbf{Q})\mathbf{R}$; SP $1_{(E)}$ computes

$$E\left(C_{ij}\right) = \prod_{k=1}^{m+1} E\left(f^2 Q_{ik}\right)^{R_{kj}}, \tag{27}$$

where C_{ij}, Q_{ik}, R_{kj} are the elements of $\mathbf{C}$, $\mathbf{Q}$, $\mathbf{R}$. Then, SP $1_{(E)}$ sends $E(\mathbf{C})$ to SP $2_{(C)}$.

(2) SP $2_{(C)}$ obtains $\mathbf{C} = D(E(\mathbf{C}))$. Especially for method RBF-SVM(B), SP2 computes $\mathbf{C} = \mathbf{C}/f^2$. Then, SP2 solves $\mathbf{C}\delta = \mathbf{e}$ and obtains δ.

(3) Then, $SP2_{(C)}$ randomly divides δ into a linear combination as follows:

$$\boldsymbol{\delta} = t_1\boldsymbol{\delta}_1 + t_2\boldsymbol{\delta}_2\left(t_1, t_2 \in \mathbb{R}, \boldsymbol{\delta}_1, \boldsymbol{\delta}_2 \in \mathbb{R}^{m+1}\right). \quad (28)$$

Then, $SP2_{(C)}$ sends vectors δ_1, δ_2 to $SP1_{(E)}$.

(4) $SP1_{(E)}$ computes $\zeta = \mathbf{R}\delta_1$ and $\eta = \mathbf{R}\delta_2$.

Remark 4. Step 1 uses an invertible matrix $\mathbf{R}$ to mask $\mathbf{Q}$ in an encrypted state. Step 3 decomposes δ into a linear combination. These random transformations prevent SP1 and SP2 from learning anything about $\mathbf{Q}$ or δ.

It is worth noting that keeping $\mathbf{Q}$ confidential is a key step to ensure the security of β. If someone obtains $\mathbf{Q}$, they can compute $\beta = \mathbf{Q}^{-1}\mathbf{e}$, where $\mathbf{e} = (0, 1, \ldots, 1)$ is the right-hand side of (6).

SP1 and SP2 will use ζ, η, t_1, and t_2 to compute $f(\mathbf{x})$ in prediction phase.

3.4. Correctness of Our Training Algorithm. It is easy to verify the correctness of Protocols 1–3 by using the properties of Paillier encryption.

In Protocol 4, we use secure multiplication which uses h^{ij} to conceal each user's Δ_k^{ij}. The integers $\lfloor fy_i y_j/h^{ij} \rfloor$ and $\lfloor fT_{ij} \rfloor$ are used instead of $fy_i y_j/h^{ij}$ and fT_{ij} in this protocol, which introduces error into system. Fortunately, the following proposition proves that the error is small.

Proposition 1. *(precision)In Protocol 4, the use of $\lfloor fy_i y_j/h^{ij} \rfloor$ and $\lfloor fT_{ij} \rfloor$ introduces errors into the system, but the actual effect on each element in $\mathbf{Q}$ is less than $1/2^{g-1}$.*

Proof: In Protocol 4, SP1 computes

$$E\left(f^2\omega_{ij}\right) = E\left(\frac{fy_i y_j}{h^{ij}} fT_{ij}\right) \approx E\left(\left\lfloor \frac{fy_i y_j}{h^{ij}} \right\rfloor\right)^{\lfloor fT_{ij} \rfloor}. \quad (29)$$

We use $\lfloor fy_i y_j/h^{ij} \rfloor \lfloor fT_{ij} \rfloor$ instead of $(fy_i y_j/h^{ij})fT_{ij}$. Let $\mathbf{Q}'$ be a matrix and its (i, j)th entry be $\lfloor fy_i y_j/h^{ij} \rfloor \lfloor fT_{ij} \rfloor$.

$$\begin{aligned}
\mathrm{Err} &= \left| \frac{fy_i y_j}{h^{ij}} fT_{ij} - \left\lfloor \frac{fy_i y_j}{h^{ij}} \right\rfloor \lfloor fT_{ij} \rfloor \right| \\
&= \left| \frac{f}{h^{ij}} fT_{ij} - \left\lfloor \frac{f}{h^{ij}} \right\rfloor \lfloor fT_{ij} \rfloor \right| \left(y_i y_j = 1 \text{ or } -1\right) \\
&\le \left| \frac{f}{h^{ij}} fT_{ij} - \frac{f}{h^{ij}} \lfloor fT_{ij} \rfloor \right| + \left| \frac{f}{h^{ij}} \lfloor fT_{ij} \rfloor - \left\lfloor \frac{f}{h^{ij}} \right\rfloor \lfloor fT_{ij} \rfloor \right| \quad (30) \\
&< \frac{f}{h^{ij}} + \left\lfloor fh^{ij} \prod_{k=1}^{r} \Delta_k^{ij} \right\rfloor \\
&< \frac{f}{h^{ij}} + fh^{ij}\left(0 < \Delta_k^{ij} < 1\right).
\end{aligned}$$

In step 2, SP2 computes $\mathbf{C} = \mathbf{C}/f^2 = (\mathbf{Q}'/f^2)\mathbf{R}$. Clearly, it is equivalent to replacing $\mathbf{Q}$ with $\mathbf{Q}'/f^2$. This means the error for each entry in the matrix $\mathbf{Q}$ is Err/f^2. We have

$$\frac{Err}{f^2} < \frac{1}{fh^{ij}} + \frac{h^{ij}}{f} < \frac{1}{2^g\left(h^{ij}\right)^2} + \frac{1}{2^g} < \frac{1}{2^{g-1}} \left(f > 2^g\right). \quad (31)$$

For example, if the $g = 11$, the error is less than 10^{-3}. The following proposition proves that the output of Protocol 5 is a linear combination of β.

Proposition 2. *(correctness)In Protocol 5, SP1 has ζ and η; SP2 has t_1 and t_2. These data satisfy $\beta = t_1\zeta + t_2\eta$, where β is the solution of LS-SVM equation.*

Proof:

$$t_1\boldsymbol{\zeta} + t_2\boldsymbol{\eta} = \mathbf{R}\left(t_1\boldsymbol{\delta}_1 + t_2\boldsymbol{\delta}_2\right) = \mathbf{R}\boldsymbol{\delta} = \mathbf{R}\left(\mathbf{R}^{-1}\mathbf{Q}^{-1}\right)\mathbf{e} = \boldsymbol{\beta}. \quad (32)$$

□

3.5. Security of Our Training Algorithm

3.5.1. Security of the Protocol Itself. Protocols 1–4 only use Paillier encryption and protocol EncMul. Paillier encryption is semantically secure, and the security of EncMul has been confirmed by previous studies. Therefore, Protocols 1–4 do not disclose input data. The following proposition proves the security of Protocol 5.

Proposition 3. *(security) In Protocol 5, two semi-honest and non-colluding service providers learn nothing about $\mathbf{Q}$ and β of the LS-SVM equation.*

Proof: $SP1_{(E)}$ only has the plaintext ζ and η. These data have nothing to do with $\mathbf{Q}$; hence, SP1 cannot learn anything about $\mathbf{Q}$. In addition, SP1 probably knows $t_1\zeta + t_2\eta = \beta$. This gives it $m + 1$ equations but $m + 3$ unknowns (t_1, t_2, β). Therefore, SP1 cannot also learn anything about β.

$SP2_{(C)}$ only knows the matrix $\mathbf{C}$. Even if it knows $\mathbf{C} = \mathbf{QR}$, it still cannot obtain anything about $\mathbf{Q}$ or $\mathbf{R}$, because $\mathbf{C} = \mathbf{QR}$ gives it $(m + 1)^2$ equations but $2(m + 1)^2$ unknowns in $\mathbf{Q}$ and $\mathbf{R}$. Meanwhile, CSP learns nothing about β without $\mathbf{R}$, because $\beta = \mathbf{R}\delta$ and it only has δ.

3.5.2. Analysis of Collusion. Suppose there exists collusion between some participants. In Protocols 1–3, all users only provide data, and there is no communication between users. Therefore, we only need to discuss the collusion between users and one service provider.

Proposition 4. *(collusion) For Linear-SVM, Poly-SVM, and RBF-SVM(A), collusion between some users and SP1 or SP2 cannot help them to learn anything they do not know.*

Proof. There are two cases of collusion.

Case 1: Some users collude with SP1. These users can provide their data to SP1. However, SP1 still cannot obtain any part of $\mathbf{Q}$ because each entry of $\mathbf{Q}$ is divided between all users. SP1 and these users cannot learn anything they do not know.

Case 2: Some users collude with SP2. SP2 has **C** and it knows **C** = **QR**. Without **R**, matrix **C** cannot help them learn anything else. Hence, they also cannot obtain **Q** because only some users collude with SP2. Without **Q**, SP2 cannot compute **R** and β. SP2 and these users cannot learn anything they do not know.

For Protocol 4, simply requiring that there is no collusion between two service providers cannot prevent privacy leakage. Protocol 4 uses random h^{ij} to keep any user from learning other user's Δ_k^{ij}. This is similar to the secure multiparty summation protocol proposed by Clifton et al. [36], except that our arithmetic operation is multiplication. If any two users do not collude, this multiparty multiplication is secure. However, if there are two users colluding with each other, they may obtain the other user's data. For example, user_{k-1} and user_{k+1} can easily compute user_k's $\Delta_k^{ij} = (\text{user}_{k'} \text{sT}_{ij})/(\text{user}_{k-1'} \text{sT}_{ij})$. For more discussion on this problem, please see [37]. □

4. Our PP-SVM Prediction Algorithm

In this paper, each training data point $\mathbf{x}_i$ $(1 \le i \le m)$ is vertically partitioned between users. These users could be banks, insurance companies, e-commerce firms, etc., and they are unwilling to share their data with others. Meanwhile, the new data point **x** for prediction also includes the same attributes as $\mathbf{x}_i$. Indeed, it is difficult for someone else to collect the data for prediction from these users. Accordingly, we assume the data **x** is also vertically partitioned between these users.

4.1. Main Steps of Our PP-SVM Prediction Algorithm. In the prediction phase, user_A wants to know about the category of object *G*. Therefore, it sends a request to all data owners (users); then, all users provide data **x** about *G*, and service providers will perform PP-SVM prediction algorithm. To determine the category of **x**, user_A needs to know $f(\mathbf{x})$ as follows:

$$f(\mathbf{x}) = \langle \boldsymbol{\beta}, \mathbf{k} \rangle = t_1 \langle \boldsymbol{\zeta}, \mathbf{k} \rangle + t_2 \langle \boldsymbol{\eta}, \mathbf{k} \rangle, \tag{33}$$

where $\mathbf{k} = (1, k_1, k_2, \ldots, k_m)$ and $k_i = y_i K(\mathbf{x}_i, \mathbf{x})$, and ζ and η are held by SP1 and SP2. Figure 5 shows the procedure of prediction, which includes several steps as follows:

Let $\overline{\mathbf{k}} = (k_1, \ldots, k_m)$, $\overline{\zeta} = (\zeta_1, \ldots, \zeta_m)$, and $\overline{\eta} = (\eta_1, \ldots, \eta_m)$ in the following steps.

(1) SP1 sends pk_1 to other participators.

(2) All users send encrypted data to SP2.

(3) SP2 computes $E(\overline{\mathbf{k}}) = (E(k_1), \ldots, E(k_m))$ (maybe with the help of SP1).

(4) user_A chooses $u_1, u_2 \in \mathbb{R}$ and sends them to SP1.

(5) SP1 chooses $\varepsilon \in \mathbb{Z}_+^m$; computes $\zeta' = u_1(\overline{\zeta} + \varepsilon)$, $\eta' = u_2(\overline{\eta} + \varepsilon)$, $E(\varepsilon)$; and then sends them to SP2.

(6) SP2 chooses $\mathbf{s} \in \mathbb{Z}_+^m$ and computes $e_1 = \langle \mathbf{s}, \zeta' \rangle$, $e_2 = \langle \mathbf{s}, \eta' \rangle$, $E(d) = E(\langle \mathbf{s}, \varepsilon \rangle)$, $E(\mathbf{p}) = E(\overline{\mathbf{k}} + \mathbf{s})$. Then, it keeps e_1, e_2 by itself and sends $E(d), E(\mathbf{p})$ to SP1.

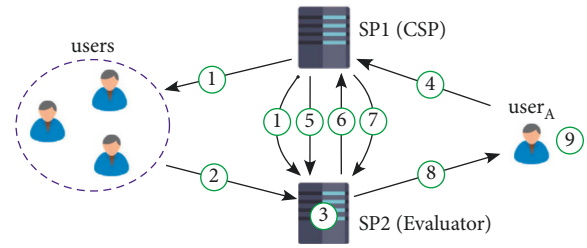

Figure 5: The procedure of our PP-SVM prediction.

(7) SP1 decrypts the data received; computes $v_1 = u_1(\langle \overline{\zeta}, \mathbf{p} \rangle + \zeta_0 + d)$ and $v_2 = u_2(\langle \overline{\eta}, \mathbf{p} \rangle + \eta_0 + d)$; and then sends v_1, v_2 to SP2.

(8) SP2 computes $w_1 = t_1(v_1 - e_1)$ and $w_2 = t_2(v_2 - e_2)$ and then sends w_1, w_2 to user_A.

(9) user_A computes the $\text{out}_y = w_1/u_1 + w_2/u_2$.

Finally, user_A obtains the out_y, which is the value of $f(\mathbf{x})$. Because using the random vectors $\boldsymbol{\varepsilon}$ and **s** in this procedure, each service provider cannot learn the data held by the other party.

4.2. Our Protocol for PP-SVM Prediction. In this phase, SP1 and SP2 switch their roles in the training phase. SP1 becomes CSP and SP2 becomes Evaluator. We use the notations $\text{SP1}_{(C)}$ and $\text{SP2}_{(E)}$ instead of SP1 and SP2 in Protocol 6 to show their roles. Let the new data point $\mathbf{x} = (\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_r)$, where $\mathbf{x}_k$ is owned by user_k.

Protocol 6. Privacy-preserving SVM prediction.

Input: user_k provides $\mathbf{x}_k (1 \le k \le r)$; $\text{SP1}_{(C)}$ and $\text{SP2}_{(E)}$ provide their data obtained in the training phase

Output: only user_A obtains the value of $f(\mathbf{x})$.

(1) Compute $E(y_i K(\mathbf{x}_i, \mathbf{x}))$, where $1 \le i \le m$.

Case 1: Linear-SVM.

(a) Each user_k computes $E(\langle y_i \mathbf{x}_{i_k}, \mathbf{x}_k \rangle)$ locally and sends them to $\text{SP2}_{(E)}$, where $1 \le i \le m$.

(b) $\text{SP2}_{(E)}$ computes the following data for $1 \le i \le m$.

$$E(y_i K(\mathbf{x}_i, \mathbf{x})) = \prod_{k=1}^{r} E\left(\langle y_i \mathbf{x}_{i_k}, \mathbf{x}_k \rangle\right). \tag{34}$$

Case 2: Poly-SVM.

(a) Each user_k computes $E(a\langle \mathbf{x}_{i_k}, \mathbf{x}_k \rangle)$ and user_1 computes all $E(y_i)$ and $E(c)$. They send all of them to $\text{SP2}_{(E)}$.

(b) For $1 \le i \le m$, $\text{SP2}_{(E)}$ computes

$$T_i = E\left(\sum_{k=1}^{r} a\langle \mathbf{x}_{i_k}, \mathbf{x}_k \rangle + c\right) = \prod_{k=1}^{r} E\left(a\langle \mathbf{x}_{i_k}, \mathbf{x}_k \rangle\right) \cdot E(c). \tag{35}$$

(c) Let $S_i = T_i$. $\mathrm{SP}\,2_{(E)}$ and $\mathrm{SP}\,1_{(C)}$ repeatedly execute $S_i = \mathrm{EncMul}(S_i, T_i)$ $p-1$ times. Then, $\mathrm{SP}\,2_{(E)}$ obtains

$$S_i = E(K(\mathbf{x}_i, \mathbf{x})) = E\left(\left(\sum_{k=1}^{r} a\langle \mathbf{x}_{i_k}, \mathbf{x}_k\rangle + c\right)^p\right). \quad (36)$$

(d) Finally, $\mathrm{SP}\,2_{(E)}$ and $\mathrm{SP}\,1_{(C)}$ compute

$$E(y_i K(\mathbf{x}_i, \mathbf{x})) = \mathrm{EncMul}(E(y_i), S_i). \quad (37)$$

Then, $\mathrm{SP}\,2_{(E)}$ obtains $E(y_i K(\mathbf{x}_i, \mathbf{x}))$ for $1 \le i \le m$.

Case 3 (for PP-SVM with RBF kernel):

Choose one from the following two methods.

Method RBF-SVM(A):

(a) Let $\Delta_k^i = \exp(-\sigma(\langle \mathbf{x}_{i_k}, \mathbf{x}_{i_k}\rangle - 2\langle \mathbf{x}_{i_k}, \mathbf{x}_k\rangle + \langle \mathbf{x}_k, \mathbf{x}_k\rangle))$. $\mathrm{user}_k\,(k \ge 2)$ computes $E(\Delta_k^i)$; user_1 computes $E(y_i\Delta_1^i)$, where $1 \le i \le m$. Then, all these data are sent to $\mathrm{SP}\,2_{(E)}$.

(b) For any $1 \le i \le m$, let $T_i = E(y_i\Delta_1^i)$. Evaluator and CSP repeatedly execute $T_i = \mathrm{EncMul}(E(\Delta_k^i), T_i)$ for $k = 2, \ldots, r$. Finally, $\mathrm{SP}\,2_{(E)}$ obtains

$$E(y_i K(\mathbf{x}_i, \mathbf{x})) = T_i = E\left(y_i \prod_{k=1}^{r} \Delta_k^i\right). \quad (38)$$

Method RBF-SVM(B):

(a) Let $\Delta_k^i = \exp(-\sigma(\langle \mathbf{x}_{i_k}, \mathbf{x}_{i_k}\rangle - 2\langle \mathbf{x}_{i_k}, \mathbf{x}_k\rangle + \langle \mathbf{x}_k, \mathbf{x}_k\rangle))$. user_1 chooses random real numbers $h^i > 1$ and computes $T_i = h^i\Delta_1^i$ for any $1 \le i \le m$.

(b) user_1 sends each T_i to user_2, and user_2 computes $T_i = T_i\Delta_2^i$ and then sends T_i to user_3. One by one, other users do the same thing until user_r computes $T_i = T_i\Delta_r^i$ and sends these data to $\mathrm{SP}\,2_{(E)}$.

(c) user_1 chooses a random integer $f > 2^{10}h^i$, computes $E(\lfloor \mathrm{fy}_i/h^i \rfloor)$, sends $E(\lfloor \mathrm{fy}_i/h^i \rfloor)$ to $\mathrm{SP}\,2_{(E)}$, and sends f to $\mathrm{SP}\,1_{(C)}$ and $\mathrm{SP}\,2_{(E)}$.

(d) For $1 \le i \le m$, $\mathrm{SP}\,2_{(E)}$ computes

$$E\left(f^2 y_i K(\mathbf{x}_i, \mathbf{x})\right) = E\left(f^2 y_i \prod_{k=1}^{r} \Delta_k^i\right) = E\left(\left\lfloor \frac{\mathrm{fy}_i}{h^i} \right\rfloor\right)^{fT_i}. \quad (39)$$

(2) user_A chooses two random real numbers u_1, u_2 and sends them to Evaluator, where $|u_1| > 1$, $|u_2| > 1$.

(3) $\mathrm{SP}\,1_{(C)}$ chooses positive random integers $\varepsilon_1, \ldots, \varepsilon_m$ and computes $E(\varepsilon_1), E(\varepsilon_2), \ldots, E(\varepsilon_m)$ and

$$\begin{aligned} \zeta_1' &= u_1(\zeta_1 + \varepsilon_1), \ldots, \zeta_m' = u_1(\zeta_m + \varepsilon_m), \\ \eta_1' &= u_2(\eta_1 + \varepsilon_1), \ldots, \eta_m' = u_2(\eta_m + \varepsilon_m), \end{aligned} \quad (40)$$

where $\mathrm{SP}\,1_{(C)}$'s data $\zeta = (\zeta_0, \zeta_1, \ldots, \zeta_m)$ and $\eta = (\eta_0, \eta_1, \ldots, \eta_m)$ are obtained in Protocol 5. Then, $\mathrm{SP}\,1_{(C)}$ sends all $E(\varepsilon_i)$, η_i', and ζ_i' to $\mathrm{SP}\,2_{(E)}$.

(4) $\mathrm{SP}\,2_{(E)}$ chooses positive random integers $s_1, \ldots, s_m$. Let $p_i = y_i K(\mathbf{x}_i, \mathbf{x}) + s_i$ $(1 \le i \le m)$, $d = \sum_{i=1}^{m} s_i\varepsilon_i$. $\mathrm{SP}\,2_{(E)}$ computes

$$\begin{aligned} E(p_i) &= E(y_i K(\mathbf{x}_i, \mathbf{x})) \cdot E(s_i), \\ E(d) &= E\left(\sum_{i=1}^{m} s_i\varepsilon_i\right) = \prod_{i=1}^{m} E(\varepsilon_i)^{s_i}, e_1 = \sum_{i=1}^{m} s_i\zeta_i', e_2 = \sum_{i=1}^{m} s_i\eta_i', \end{aligned} \quad (41)$$

and sends all $E(p_i), E(d)$ to $\mathrm{SP}\,1_{(C)}$.

(5) $\mathrm{SP}\,1_{(C)}$ obtains $p_i = D(E(p_i))$, $d = D(E(d))$. For Linear-SVM, Poly-SVM, and RBF-SVM(A), $\mathrm{SP}\,1_{(C)}$ computes

$$\begin{aligned} v_1 &= u_1\left(\sum_{i=1}^{m} \zeta_i p_i + \zeta + d\right), \\ v_2 &= u_2\left(\sum_{i=1}^{m} \eta p_i + \eta_0 + d\right). \end{aligned} \quad (42)$$

For RBF-SVM(B), $\mathrm{SP}\,1_{(C)}$ computes

$$\begin{aligned} v_1 &= u_1\left(\sum_{i=1}^{m} \zeta_i p_i + f^2\zeta_0 + d\right), \\ v_2 &= u_2\left(\sum_{i=1}^{m} \eta_i p_i + f^2\eta_0 + d\right). \end{aligned} \quad (43)$$

Then, $\mathrm{SP}\,1_{(C)}$ sends v_1, v_2 to $\mathrm{SP}\,2_{(E)}$.

(6) For Linear-SVM, Poly-SVM, and RBF-SVM(A), $\mathrm{SP}\,2_{(E)}$ computes

$$\begin{aligned} w_1 &= t_1(v_1 - e_1), \\ w_2 &= t_2(v_2 - e_2). \end{aligned} \quad (44)$$

For RBF-SVM(B), $\mathrm{SP}\,2_{(E)}$ computes

$$\begin{aligned} w_1 &= \frac{t_1(v_1 - e_1)}{f^2}, \\ w_2 &= \frac{t_2(v_2 - e_2)}{f^2}. \end{aligned} \quad (45)$$

Then, $\mathrm{SP}\,2_{(E)}$ sends w_1, w_2 to user_A.

(7) user_A computes $f(\mathbf{x})$ as follows:

$$\frac{w_1}{u_1} + \frac{w_2}{u_2}. \tag{46}$$

We choose $|u_1| > 1$, $|u_2| > 1$ to make the computation more accurate.

4.3. Correctness and Security of Prediction Algorithm. In Protocol 6, two service providers compute $E(y_i K(\mathbf{x}_i, \mathbf{x}))$ in step 1. This step is similar to computing $E(\Omega)$ in Protocols 1–4. Its correctness is obvious. The following proposition proves that the output of Protocol 6 is $f(\mathbf{x})$.

Proposition 5. *(correctness) In Protocol 6, user_A finally obtains the value*

$$f(\mathbf{x}) = \sum_{i=1}^{m} \alpha_i y_i K(\mathbf{x}_i, \mathbf{x}) + b. \tag{47}$$

Proof. For Linear-SVM, Poly-SVM, and RBF-SVM(A), user_A finally obtains the following value:

$$\begin{aligned}
&\frac{w_1}{u_1} + \frac{w_2}{u_2} \\
&= \frac{t_1(v_1 - e_1)}{u_1} + \frac{t_2(v_2 - e_2)}{u_2} \\
&= \frac{t_1\left(u_1\left(\sum_{i=1}^{m} \zeta p_{ii} + \zeta_0 + d\right) - u_1\left(\sum_{i=1}^{m} s_i(\zeta_i + \varepsilon_i)\right)\right)}{u_1} \\
&\quad + \frac{t_2\left(u_2\left(\sum_{i=1}^{m} \eta_i p_i + \eta_0 + d\right) - u_2\left(\sum_{i=1}^{m} s_i(\eta_i + \varepsilon_i)\right)\right)}{u_2} \\
&= t_1\left(\sum_{i=1}^{m} \zeta_i y_i K(\mathbf{x}_i, \mathbf{x}) + \zeta_0 + d - \sum_{i=1}^{m} s_i \varepsilon_i\right) \\
&\quad + t_2\left(\sum_{i=1}^{m} \eta_i y_i K(\mathbf{x}_i, \mathbf{x}) + \eta_0 + d - \sum_{i=1}^{m} s_i \varepsilon_i\right) \\
&= \sum_{i=1}^{m} (t_1 \zeta_i + t_2 \eta_i) y_i K(\mathbf{x}_i, \mathbf{x}) + t_1 \zeta_0 + t_2 \eta_0 \\
&= \sum_{i=1}^{m} \alpha_i y_i K(\mathbf{x}_i, \mathbf{x}) + b \quad (t_1 \boldsymbol{\zeta} + t_2 \boldsymbol{\eta} = \boldsymbol{\beta}).
\end{aligned} \tag{48}$$

There is a similar proof process for RBF-SVM(B).

The next proposition guarantees the security of Protocol 6. □

Proposition 6. *(security) In Protocol 6, two non-colluding service providers cannot learn anything about the data $\mathbf{x}_i$, y_i, $\mathbf{x}$, $f(\mathbf{x})$, β, and $K(\mathbf{x}_i, \mathbf{x})$.*

Proof. We do not discuss step 1 because it is essentially the same as computing $E(\Omega)$ in Protocols 1–4. The following discussion is about the other steps.

$\text{SP 1}_{(C)}$ knows plaintext d, p_i, ζ, and η. The ζ and η have nothing to do with p_i and d. In fact, even if it knows

$$d = \sum_{i=1}^{m} s_i \varepsilon_i,\ p_i = y_i K(\mathbf{x}_i, \mathbf{x}) + s_i \quad (1 \le i \le m) \tag{49}$$

and regards each $y_i K(\mathbf{x}_i, \mathbf{x})$ as an unknown variable u_i, the number of unknowns (including u_i and s_i) still far exceeds the number of equations given by p_i and d. This ensures that SP1 learns nothing from p_i and d. Meanwhile, SP1 still does not know t_1, t_2, which means it still cannot learn β. Thus, SP1 knows nothing about $\mathbf{x}_i$, y_i, $\mathbf{x}$, $f(\mathbf{x})$, β, or $K(\mathbf{x}_i, \mathbf{x})$.

$\text{SP 2}_{(E)}$ only receives plaintext ζ_i', η_i', v_1, v_2, where $1 \le i \le m$. Other encrypted data it has are not helpful in understanding private information. Under this circumstance, even if it knows the following equations:

$$\begin{aligned}
\zeta_i' &= u_1(\zeta_i + \varepsilon_i),\ v_1 = u_1\left(\sum_{i=1}^{m} \zeta_i p_i + \zeta_0 + d\right), \\
\eta_i' &= u_2(\eta_i + \varepsilon_i),\ v_2 = u_2\left(\sum_{i=1}^{m} \eta_i p_i + \eta_0 + d\right),
\end{aligned} \tag{50}$$

the number of unknowns (including ζ_i, η_i, ε_i, p_i, u_1, u_2, d) is still far more than the number of equations. It cannot obtain any new information from these equations. Hence, SP2 learns nothing about $\mathbf{x}_i$, y_i, $\mathbf{x}$, $f(\mathbf{x})$, β, or $K(\mathbf{x}_i, \mathbf{x})$. The following proposition discusses the collusion between user_A and one service provider. □

Proposition 7. *(collusion) In Protocol 6, collusion between user_A and SP1 or SP2 cannot help them to learn anything they do not know.*

Proof. There are two cases of collusion.

Case 1user_A colludes with SP1. user_A can provide w_1, w_2 to SP1. Even if they know the following equations:

$$\begin{aligned}
&\zeta_i + \varepsilon_i, \eta_i + \varepsilon_i \quad (1 \le i \le m), \\
&\sum_{i=1}^{m} \zeta_i p_i + \zeta_0 + d, \sum_{i=1}^{m} \eta_i p_i + \eta_0 + d,
\end{aligned} \tag{51}$$

there are still unknowns $t_1, t_2, e_1, e_2, s_1, \ldots, s_m$ in these equations. The number of unknowns still exceeds the number of equations. SP1 and user_A cannot learn anything with these equations.

Case 2: user_A colludes with SP2. user_A can give SP2 u_1 and u_2, which can be used by SP2 to simplify (50). Then, SP2 obtains the values of the following expressions:

$$\zeta_i + \varepsilon_i, \eta_i + \varepsilon_i, \quad (1 \le i \le m),$$
$$\sum_{i=1}^{m} \zeta_i p_i + \zeta_0 + s, \tag{52}$$
$$\sum_{i=1}^{m} \eta_i p_i + \eta_0 + d.$$

Indeed, the number of unknowns $(\zeta_i, \eta_i, \varepsilon_i, p_i, d)$ is still far more than the number of these expressions. SP2 and user_A cannot learn anything with these equations. □

5. Experiments

In this section, we evaluate our algorithm on accuracy and execution time. The datasets used in our experiments, including Liver Disorders, Sonar, and Breast Cancer Wisconsin, are obtained from the UCI Machine Learning Repository [38].

5.1. Experimental Settings. All our experiments are conducted on a Dell laptop with an Intel i7-4800MQ CPU (8 cores at 2.7 GHz) and 32 GB RAM, running Ubuntu Linux 20.04. We implement our protocols in Python and its extensions: gmpy2 [39] and bigfloat [40]. These two modules support high-precision arithmetic.

Before starting our experiment, we normalize the input data $\mathbf{x}_i, \mathbf{x}$ to the range $[0, 1]$ and set the class label y_i to 1 or −1. We adopt a Paillier cryptosystem with 1024-bit modulus in our system. This means all input data and intermediate results only could be integers in $[-2^{512}, 2^{512})$, where negative integers are represented by 2's complement.

For the polynomial kernel, we only consider the quadratic polynomial $(a\langle \mathbf{x}_i, \mathbf{x}_j \rangle + c)^2$.

5.2. Accuracy

5.2.1. The Effect of Different Fraction Length on the Error. We adopt fixed-point numbers in our system. Fixed-point arithmetic operation will bring truncation error.

Experiment 1 investigates the effect of the fraction length (q), which is used to convert a real number a to an integer $\lfloor a \cdot 2^q \rfloor$. The dataset Liver Disorders is used in this experiment. We randomly choose 50 records for training and 20 records for prediction. Moreover, we split input data between 2 users.

For different fraction lengths, we compare our methods with the ordinary SVM and compute the error of $f(\mathbf{x})$ between them at data points for prediction. For different kernel SVM, we choose parameter values as follows:

(1) Linear: $\gamma = 2$.

(2) Polynomial: $\gamma = 3, a = 1.5, c = 1$.

(3) RBF(A): $\gamma = 2, \sigma = 11.1$.

(4) RBF(B): $\gamma = 2, \sigma = 11.1, h \in [9e3, 10^4], f = 10^7, 10^{12}$.

In method RBF-SVM(B), h is used for secure multiplication. Giving h a wider range (e.g., $h \in [10, 10^4]$) can make our protocol more secure. In our experiment, the small range of h actually limits f/h to a small range. Then, we can study the influence of fraction length on the error of RBF-SVM(B).

Table 1: Output errors of all methods with different q (experiment 1).

Fraction length q	8	16	24	32
Linear	$1.2e-02$	$6.8e-05$	$2.4e-07$	$6.9e-10$
Polynomial	$9.9e-02$	$4.8e-04$	$1.1e-06$	$3.9e-09$
RBF(A)	$4.1e-03$	$2.4e-05$	$1.2e-07$	$2.9e-10$
RBF(B) $(f = 10^7)$	$1.0e-03$	$7.2-04$	$9.1e-04$	$9.0e-04$
RBF(B) $(f = 10^{12})$	$8.8e-09$	$8.7e-09$	$7.9e-09$	$9.2e-09$

Table 2: Output errors of RBF-SVM(B) with different f/h (experiment 2).

$f/h \approx 1$	$f/h \approx 10^2$	$f/h \approx 10^3$	$f/h \approx 10^5$	$f/h \approx 10^8$
Error result	6.1e − 03	8.3e − 04	5.3e − 06	4.8e − 09

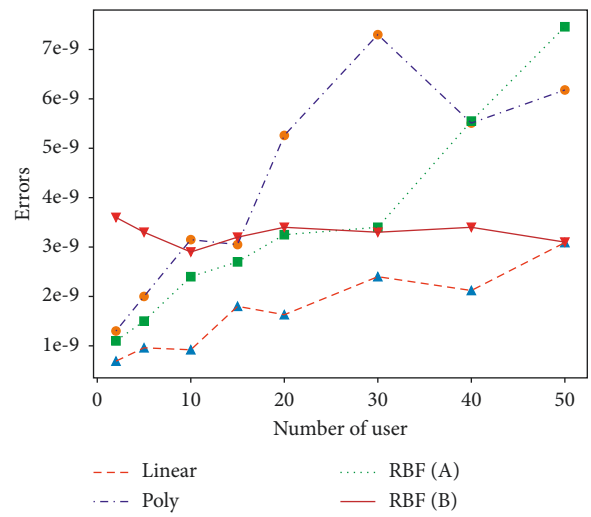

Figure 6: The error of the predicted values (experiment 3).

The results of experiment 1 are shown in Table 1. For our Linear-SVM, Poly-SVM, and RBF-SVM(A), it shows that the error decreases with the increase of fraction length q. However, for RBF-SVM(B), the variation of error is very small with the increase of q. The reason for this is that the RBF-SVM(B) performs fewer homomorphic computations than the other methods. We suggest $q \ge 32$ in most practical problems, which provides enough accuracy to guarantee the correctness of the classification.

From the last two lines of Table 1, we know that the value of f/h has a significant impact on the accuracy of RBF-SVM(B). Experiment 2 is conducted to evaluate the effect of different f/h. In experiment 2, we choose $q = 32, \gamma = 2, \sigma = 11.1, h \in [9000, 10^4]$. The results are shown in Table 2. When $f/h \approx 1$, the predicted value is wrong. With the increase of f/h, the error of the predicted values decreases linearly.

5.2.2. The Effect of Different Numbers of Users on the Error. Our methods merge all users' data to generate $E(\mathbf{Q})$. More users mean more fixed-point computations, which may

Table 3: The complexity of our algorithm at different stages.

Methods	Linear-SVM	Poly-SVM	RBF-SVM(A)	RBF-SVM(B)
Stage A (each user)	mul: $\mathcal{O}(\mathrm{nm}^2)$ add: $\mathcal{O}(\mathrm{nm}^2)$ Enc: m^2	mul: $\mathcal{O}(\mathrm{nm}^2)$ add: $\mathcal{O}(\mathrm{nm}^2)$ Enc: m^2	mul: $\mathcal{O}(\mathrm{nm}^2)$ add: $\mathcal{O}(\mathrm{nm}^2)$ exp: m^2 Enc: m^2	mul: $\mathcal{O}(\mathrm{nm}^2)$ add: $\mathcal{O}(\mathrm{nm}^2)$ exp: m^2 rnd: m^2 mul: rm^2 Enc: m^2
Stage B (SP1, SP2)	mul: $(m+1)^2 m + \mathrm{rm}^2$ add: $(m+1)^2 m$ rnd: $(m+1)^2$ pow: $(m+1)^3$	mul: $(m+1)^2 m + \mathrm{rm}^2$ add: $(m+1)^2 m$ rnd: $(m+1)^2$ pow: $(m+1)^3$ EncMul: pm^2	mul: $(m+1)^2 m$ add: $(m+1)^2 m$ rnd: $(m+1)^2$ pow: $(m+1)^3$ EncMul: $(r-1)m^2$	mul: $(m+1)^2 m$ add: $(m+1)^2 m$ rnd: $(m+1)^2$ pow: $(m+1)^3 + m^2$
Stage C (SP1, SP2)	mul: $\mathcal{O}(m^3)$, div: $\mathcal{O}(m^3)$, add: $\mathcal{O}(m^3)$ Dec: m^2			
Stage D (each user)	mul: $\mathcal{O}(\mathrm{nm})$ add: $\mathcal{O}(\mathrm{nm})$ Enc: m	mul: $\mathcal{O}(\mathrm{nm})$ add: $\mathcal{O}(\mathrm{nm})$ Enc: m	mul: $\mathcal{O}(\mathrm{nm})$ add: $\mathcal{O}(\mathrm{nm})$ exp: m Enc: m	mul: $\mathcal{O}(\mathrm{nm})$ add: $\mathcal{O}(\mathrm{nm})$ exp: m rnd: m mul: rm Enc: m
Stage D (SP1, SP2 user_A)	mul: $\mathcal{O}(rm)$ add: $\mathcal{O}(m)$ Enc: m Dec: m rnd: $2m$ pow: m	mul: $\mathcal{O}(rm)$ add: $\mathcal{O}(m)$ Enc: m Dec: m rnd: $2m$ pow: m EncMul: pm	mul: $\mathcal{O}(m)$ add: $\mathcal{O}(m)$ Enc: m Dec: m rnd: $2m$ pow: m EncMul: $(r-1)m$	mul: $\mathcal{O}(m)$ add: $\mathcal{O}(m)$ Enc: m Dec: m rnd: $2m$ pow: m

Note. n is the number of features, m is the number of data points (records), r is the number of users, exp is the e^x, rnd is the random integer generation, pow is the modular exponentiation, Enc is the encryption, Dec is the decryption. Tasks performed by one user or tasks performed by all users in sequence.

bring more errors to our system. Experiment 3 studies the influence of the number of users on the accuracy.

The dataset Sonar is used in experiment 3. We choose 40 records from it for training and 20 records for prediction. Furthermore, we set the fraction length $q = 32$. In addition, for RBF-SVM(B), we choose $h \in [2^8, 2^{12}]$ and $f = 2^{40}$. Then, we perform each method for the number of users $r = 2, 5, 10, 15, 30, 40, 50$. Figure 6 shows the errors of $f(\mathbf{x})$ for our methods and the ordinary SVM.

For our Linear-SVM, Poly-SVM, and RBF-SVM(A), Figure 6 shows that the errors usually increase with the increasing number of users, but the errors increase very slowly. When the number of users r goes from 2 to 50, the errors increase by less than 7 times.

We know that the RBF-SVM(A) needs to execute EncMul $r - 1$ times when computing $E(\omega_{\mathrm{ij}})$. Meanwhile, the Poly-SVM executes EncMul only twice for any number of users when computing $E(\omega_{ij})$. However, in Figure 6, we find little difference in accuracy between the methods RBF-SVM(A) and Poly-SVM. This means multiple calls to EncMul have little impact on accuracy.

In addition, the errors of RBF-SVM(B) are almost invariant with the increasing number of users. We believe the reason is that all users use high-precision floating-point operations to merge data (secure multiplication), and then convert the results to fixed-point numbers with only one operation. Therefore, the number of users has little impact.

5.3. Execution Time. To investigate the performance of our algorithm, we divide each method into 4 stages:

Stage A: each user computes part of matrix $E(\Omega)$ with local data.

Stage B: Evaluator computes $E(\Omega)$, $E(\mathbf{Q})$, and $E(\mathbf{C})$ in an encrypted form.

Stage C: CSP solves the encrypted equation; finally, CSP has t_1, t_2; and Evaluator obtains ζ, η.

Stage D: all users submit a new data point together, and two service providers make prediction for it.

5.3.1. The Complexity of Our Algorithm at Different Stages. Except arithmetic operations (add, mul, div), we also take random number generation (rnd), e^x (exp), modular exponentiation (pow), encryption (Enc), decryption (Dec), and EncMul as basic operations.

Table 3 shows the complexity of each method at different stages. The time spent on different operations varies greatly. When we set the fraction length $q = 32$ and limit the range of random numbers to $[1, 2^{32}]$, the execution time of these operations has the following relationship:

$$\begin{aligned} 1\ \mathit{EncMul} &\approx 4.5\,\mathrm{Dec} \approx 9\,\mathrm{Enc} \approx 180\,\mathrm{pow} \approx 1800\,\mathrm{mul} \\ &\approx 18000\ \mathrm{exp}\ \approx 18000\,\mathrm{add} \approx 10^5\,\mathrm{rnd}. \end{aligned} \tag{53}$$

This relationship is not fixed. It varies with fraction length and random numbers and operands. For example, the exp in our system executes operations much faster than pow. The reason is that users use small numbers (plaintexts) for exp computation, but SP uses very large integers (ciphertexts, random numbers) for pow operation. In addition,

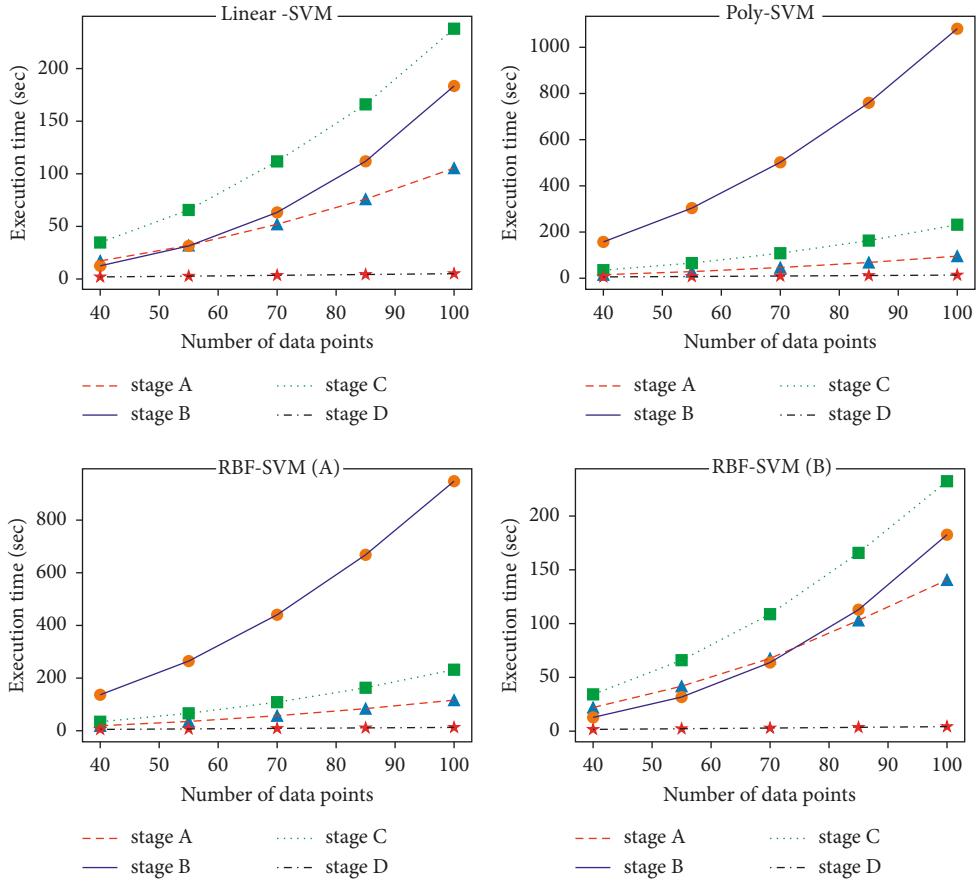

Figure 7: The execution time of each stage for different numbers of data points (experiment 4).

because the pow operations in stage B use a lot of random numbers, only changing the range of random numbers will have a great impact on the running time.

5.3.2. The Effect of Different Numbers of Records on the Execution Time. The number of input data points m has a great impact on the execution time, because all participators need to deal with $m \times m$ matrix. The problem is which stage is the bottleneck of performance.

Experiment 4 investigates the effect of different m on the execution time. The data used in this experiment are taken from the dataset Breast Cancer Wisconsin. We choose $m = 40$, 55, 70, 85, and 100 records for training and 1 record for prediction. Moreover, we split the data between $r = 2$ users to reduce the impact of the number of users. We set the fraction length to $q = 32$ and the range of random numbers to $[1, 2^{32}]$. The results are shown in Figure 7. Note that the execution time of stage A is the average execution time of all users.

It is easy to notice that stages C of all methods are the same. Therefore, the graphs of stages C of all methods in Figure 7 are also the same. They can be regarded as a benchmark for comparison.

Figure 7 shows that the Poly-SVM and RBF-SVM(A) spend most of the time on stage B, and the execution time of stage B grows much faster than that of stage C with the increase of m. On the other hand, for the Linear-SVM or RBF-SVM(B), there is no huge difference between the execution time of stage B and that of stage C. This is because Poly-SVM and RBF-SVM(A) call protocol EncMul many times in stage B, while the other methods do not call it.

Among all methods, we find that when the number of data points is the same, the maximum execution time of stage A or D does not exceed 200% of the minimum execution time of the same stage. This is not a significant difference.

Comparing the efficiency of all methods under the same conditions, we have RBF-SVM(B) $\approx$ Linear-SVM $\gg$ Poly-SVM $\approx$ RBF-SVM(A). Calling EncMul takes up most of the execution time.

5.3.3. The Effect of Different Numbers of Users on the Execution Time. Experiment 5 studies the effect of different numbers of users (r) on the execution time. We use the same dataset and parameter settings as experiment 3 in this experiment. The results are shown in Figure 8. For each stage, we analyze the change of execution time as follows.

(1) Stage A. Figure 8 shows that the execution time of stage A changes very little for all methods. The reasons are as follows.

For any of Linear-SVM, Poly-SVM, or RBF-SVM(A), all users execute the protocol independently of each other. It is reasonable to assume that all users execute the protocol at

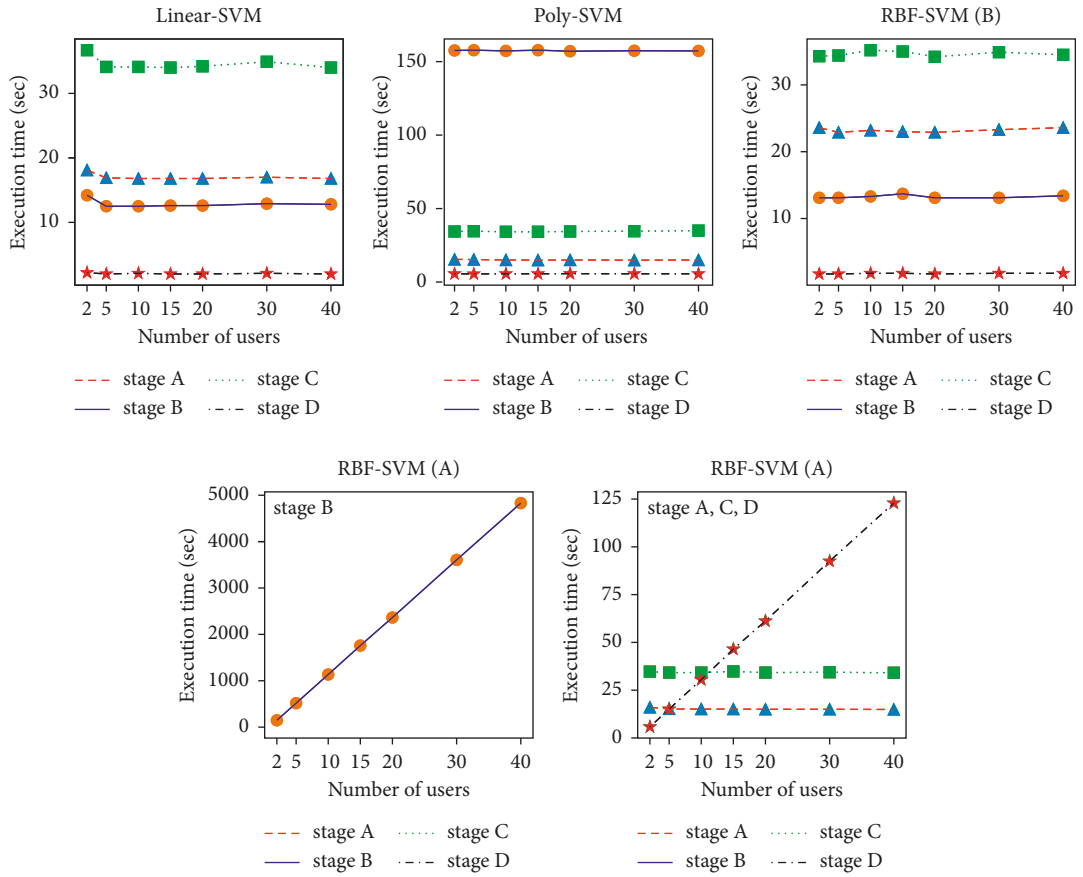

Figure 8: The execution time of each stage for different numbers of users (experiment 5).

this stage in parallel. For these methods, we take the average execution time of all users as the execution time of stage A, which is almost fixed with changing r.

In the method RBF-SVM(B), all users in turn merge local data into a matrix (computing $T_{ij} = T_{ij}\Delta_k^{ij}$ by each user, user$_k$). Although this step is related to r, it takes up only a small fraction of the execution time. The encryption operations performed by the last user take up most of the execution time of stage A.

In sum, the execution time of stage A for all methods is hardly affected by the number of users.

(2) Stage B. In stage B, Linear-SVM and Poly-SVM will perform multiplication r times for generating each entry of $E(\Omega)$. For both of these two methods, this is the only step in stage B related to r. However, the time consumed by this step is far less than that consumed by other steps. For Linear-SVM, the pow operations are the most time-consuming steps and are independent of r. For Poly-SVM, the execution of EncMul takes up most of the execution time and has nothing to do with r. In addition, the computation of RBF-SVM(B) in stage B is completely independent of r. Hence, for Linear-SVM, Poly-SVM, and RBF-SVM(B), the execution time of stage B remains almost unchanged, as shown in Figure 8.

Although both Poly-SVM and RBF-SVM(A) call EncMul in stage B, Poly-SVM with a kernel $(a\langle \mathbf{x}_i, \mathbf{x}_j\rangle + c)^p$ performs EncMul p times for each entry of $E(\Omega)$, while RBF-SVM(A) performs EncMul $r-1$ times to do the same thing. Fortunately, the degree p is often a small integer. This reduces the negative impact of EncMul on Poly-SVM. However, for RBF-SVM(A), the situation is quite different. As shown in Figure 8, its execution time of stage B increases sharply with the increase of r. This disadvantage will affect the application of this method to a certain extent. For this reason, we design RBF-SVM(B) as an alternative.

(3) Stage C. The tasks of all our methods at this stage are the same and independent of r. Figure 8 shows that the execution time of stage C is almost equal and fixed for all methods.

(4) Stage D. In stage D, all methods need to compute the kernel function, which is the only step that may be related to the number of users. This step is similar to some computation procedures in stages A and B. The difference is that each method only needs to compute m values in stage D, but m^2 values in stages A and B.

Only the execution time of RBF-SVM(A) will be significantly affected because it will call EncMul EncMul $r-1$ times when computing each kernel function. This explains why, in Figure 8, the execution time of stage D of Linear-SVM, Poly-SVM, and RBF-SVM(B) changes very little with the increase of r, while the execution time of stage D of RBF-SVM(A) increases steadily.

5.4. Comparison with Another PP-SVM Algorithm. Park et al. [7] propose a PP-SVM training algorithm based on LS-SVM and they use CKKS [27] encryption to encrypt the input data. In this paper, we use the notation $\cdot$ to denote CKKS encryption algorithm. Their algorithm can be used in one client and one server environment. The client provides the encrypted matrix $\mathbf{Q}$ to the server, which uses gradient descent method to solve the equation $\mathbf{Q}^T\mathbf{Q}\beta = \mathbf{Q}^T\mathbf{e}$.

5.4.1. The Modified Version of Park's Training Method. We modify Park et al. method as Protocol 7 (MP's method) to deal with the same problem as ours, and then we compare this modified method with our algorithm. We do not put the counterpart of RBF-SVM(B) in Protocol 7, because its performance is equivalent to that of Linear-SVM.

Protocol 7. (MP's method): The modified version of Park's training method.

Input: each user_k inputs $\mathbf{x}_{i_k}$, and y_i is shared by all users, where $1 \le i \le m$, $1 \le k \le r$. The public key is shared by all participators, and user_c has the private key.

Output: Server obtains β.

(1) Users perform computation with local data.

(a) Linear kernel:
Each user_k computes $\langle y_i\mathbf{x}_{i_k}, y_j\mathbf{x}_{j_k}\rangle$, and user_1 also computes all y_i. All these data are sent to server.

(b) Poly kernel:
Each user_k computes $a\langle \mathbf{x}_{i_k}, \mathbf{x}_{j_k}\rangle$, and user_1 also computes all y_i and c. All these data are sent to server.

(c) RBF kernel (A):
Each user_k computes $\Delta_k^{(ij)}$, and user_1 also computes y_i. Then, all these data are sent to server.

(2) Server computes $\mathbf{Q}$.

(a) Linear kernel:
Server computes $\Omega_{ij} = \sum_{k=1}^{r} \langle y_i\mathbf{x}_{i_k}, y_j\mathbf{x}_{j_k}\rangle$.

(b) Poly kernel:
Server computes $T_1 = \sum_{k=1}^{r} a\langle \mathbf{x}_{i_k}, \mathbf{x}_{j_k}\rangle + c = \sum_{k=1}^{r} a\langle \mathbf{x}_{i_k}, \mathbf{x}_{j_k}\rangle + c$. Then, let $T_2 = T_1$; server repeatedly computes $T_2 = T_2T_1$ $p-1$ times and then computes $\Omega_{ij} = y_iy_jT_2$.

(c) RBF kernel (A):
Server computes $\Omega_{ij} = \prod_{k=1}^{r} \Delta_k^{(ij)} y_iy_j$.
Then, server computes $\Omega_{ii} + \gamma^{-1} = \Omega_{ii} + \gamma^{-1}$. So far, sever has obtained all entries of $\mathbf{Q}$.

(3) Server computes β by gradient descent method.

(a) Server first computes $\mathbf{Q}^T\mathbf{Q}$ and $\mathbf{Q}^T\mathbf{e}$ with $\mathbf{Q}$.

(b) Server chooses an initial vector β_1 and then repeatedly computes β_k as follows:

Table 4: (Experiment 6) comparison of server-side training time between the two methods with different numbers of users (sec.).

Number of users (r)	Linear kernel		Poly kernel		RBF kernel (A)	
	MP's	Ours	MP's	Ours	MP's	Ours
5	426.1	46.6	425.5	192.4	463.5	550.6
10	435.8	47.8	445.2	191.5	496.3	1169.7
20	445.5	46.8	430.9	192.5	562.1	2398.1
40	429.1	47.9	445.1	192.3	693.6	4865.2

Table 5: (Experiment 7) comparison of server-side training time between the two methods with different numbers of data points (sec.).

Num. of data points (m)	Linear kernel		Poly kernel		RBF kernel (A)	
	MP's	Ours	MP's	Ours	MP's	Ours
40	430	47	447	192	439	171
55	1063	97	1080	369	1071	331
70	1920	175	1951	610	1932	549
85	3300	278	3330	923	3332	832

Table 6: Two methods' complexity of training phase.

MP's method	Ours		
	Linear kernel	Poly kernel	RBF kernel
$O(m^3)$ mul	$O(m^3)$ pow $O(m^2)$ Dec	$O(m^3)$ pow $O(m^2)$ EncMul $O(m^2)$ Dec	$O(m^3)$ pow $O(m^2)$ EncMul $O(m^2)$ Dec

CKKS multiplication.

$$\begin{cases} [\![\mathbf{p}_k]\!] = [\![\mathbf{Q}^T\mathbf{Q}]\!][\![\boldsymbol{\beta}_k]\!] - [\![\mathbf{Q}^T\mathbf{e}]\!] \\ [\![\boldsymbol{\beta}_{k+1}]\!] = [\![\boldsymbol{\beta}_k]\!] - [\![s_k\mathbf{p}_k]\!] \end{cases}, \tag{54}$$

where $k = 1, 2, \ldots, N$ and s_k is learning rate.

The server sent all β_k to user_c who decrypts them and determine if the iteration should be stopped.

Clearly, this modified Park's method solves a problem just like ours. In this protocol, step 3 is the same as Park et al. method. Step 1 and step 2 adopt the ideas of this article. Of course, MP's method uses CKKS encryption to compute both addition and multiplication between ciphers.

In this section, we compare MP's method with ours. For a fair comparison, we do not use any packing technique in all the following experiments. Moreover, because the time spent on the client and prediction phase is far less than the training time spent on the server, we only focus on the execution time of training.

The server-side computation in the training phase includes computing $E(\mathbf{Q})$, computing $E(\mathbf{Q}^T\mathbf{Q})$ with $E(\mathbf{Q})$, and performing the gradient descent method. Obviously, the latter two computation processes are only related to the number of data points.

Table 7: (Experiment 7) the training time of MP's method and the time consumed by two main steps of MP's method (sec.).

Num. of data points (m)	MP's method			Computing$E(\mathbf{Q}^T\mathbf{Q})$	Gradient descent
	Linear	Poly	RBF		
40	430	447	439	282	140
55	1063	1080	1071	751	258
70	1920	1951	1932	1458	430
85	3300	3330	3332	2632	625

Note. $T_{\text{MP's training}} = T_{E(\mathbf{Q})} + T_{E(\mathbf{Q}^T\mathbf{Q})} + T_{\text{GD method}}$.

5.4.2. Comparison of Server-Side Training Time between the Two Methods with Different Numbers of Users. In experiment 6, we choose 40 records from Sonar to be the training dataset. The data are vertically partitioned between $r = 5, 10, 20, 40$ users. The experimental results are shown in Table 4.

For the cases of linear and polynomial kernel, Table 4 shows that our methods are more efficient. Although it takes longer to run the MP's method, the training time of it with linear or polynomial kernel keeps almost unchanged with the increase of r. We believe the reason is as follows. When the number of records m is fixed, the MP's method takes a fixed time to compute $E(\mathbf{Q}^T\mathbf{Q})$ and perform the gradient descent method. Meanwhile, although the time required to compute $E(\mathbf{Q})$ is related to r, the MP's Linear-kernel and Poly-kernel methods mainly perform efficient CKKS addition when computing $E(\mathbf{Q})$. The time they spent in computing $E(\mathbf{Q})$ is far less than that spent in computing $E(\mathbf{Q}^T\mathbf{Q})$ and performing the gradient descent method. Therefore, we see in Table 4 that the server-side training time of MP's method with Linear-kernel and Poly-kernel changes little with the increase of r. For RBF kernel, Table 4 shows that the training time of the two methods increases with the increase of r. The MP's RBF kernel method mainly executes CKKS multiplication for computing $E(\mathbf{Q})$. The CKKS multiplication is a time-consuming operation, but it is still much more efficient than protocol *EncMul*. We know that our RBF-SVM(A) method spends most of its time executing EncMul, this makes MP's method have better performance than RBF-SVM(A), as shown in Table 4.

5.4.3. Comparison of Server-Side Training Time between the Two Methods with Different Numbers of Data Points. In experiment 7, we choose $m = 40, 60, 80, 100$ records from Sonar to be training dataset and vertically partition the training data between $r = 2$ users.

Table 5 shows the impact of m on server-side training time. For any type of kernel, the training time of both methods increases with the increase of m. However, the training time of MP's method not only far exceeds ours, but also grows faster. Table 6 shows the two methods' complexity of training phase. Meanwhile, in our experiment, the time consumed by the operations in Table 6 has the following relationship:

$$1\,\text{EncMul} \approx 4.5\,\text{Dec} \approx 10\,\text{CKKS mul} \approx 180\,\text{pow}. \quad (55)$$

The data in Table 5 illustrates that $O(m^3)$ CKKS multiplication grows faster than $O(m^3)$ pow + $O(m^2)$ EncMul + $O(m^2)$ Dec with the increase of m.

For MP's method, computing $E(\mathbf{Q}^T\mathbf{Q})$ and performing the gradient descent method take up most of the training time, as shown in Table 7. Actually, the computation of $E(\mathbf{Q}^T\mathbf{Q})$ is the preparation for the gradient descent method. Tables 5 and 7 illustrate that the gradient descent method is not a very efficient method in solving the equation.

Another uncertainty of the gradient descent method is how many times the iteration should be conducted before obtaining enough accurate result. The number of iterations has a significant impact on the training time. Park et al. performed the iteration 10 times in their experiments. It seems all right for their experiments, but there is no theoretical basis for doing this.

The above comparisons show that each of the two methods has advantages and disadvantages. This leads to a simple idea of using CKKS encryption in our algorithm instead of Paillier encryption. This will combine the advantages of the two methods and seems more efficient. This idea is attractive, but it is not the focus of this paper. We left it as future work.

6. Conclusion

In this paper, we presented a privacy-preserving SVM algorithm on vertically partitioned data. Non-colluding two-server architecture is the starting point of this work. This architecture makes it possible for us to design a PP-SVM algorithm only using additive homomorphic encryption and random transformation method. Unlike previous research, our algorithm conceals all users' training data, the decision function, the data for prediction, and the predicted result. In short, our algorithm is encrypted in the whole process. The experiments show that our Linear-SVM, Poly-SVM, and RBF-SVM(B) are highly efficient, and our RBF-SVM(A) is still attractive for a small number of data owners. For future work, we will consider improving the efficiency of our algorithm with other encryption techniques and extending our idea to the case of horizontally partitioned data.

Data Availability

The data that support the findings of this study are openly available in UCI Machine Learning Repository at https://archive.ics.uci.edu/ml, [38].

Conflicts of Interest

The authors declare that there are no conflicts of interest regarding the publication of this paper.

Acknowledgments

This work was supported by the National Natural Science Foundation of China (No. 61672210), Fundamental Research Funds for the Central Universities (No. xjj2018023), and Shaanxi Industrial Research Project (No. 2019GY-005).

References

[1] V. N. Vapnik, *The Nature of Statistical Learning Theory*, Springer, Heidelberg, Germany, 2000.

[2] H. Qian, Y. Mao, W. Xiang, and Z. Wang, "Recognition of human activities using svm multi-class classifier," *Pattern Recognition Letters*, vol. 31, no. 2, pp. 100–111, 2010.

[3] H. Zhu, X. Liu, R. Lu, and H. Li, "Efficient and privacy-preserving online medical prediagnosis framework using nonlinear svm," *IEEE Journal of Biomedical and Health Informatics*, vol. 21, no. 3, pp. 838–850, 2017.

[4] D. Jain and V. Singh, "A two-phase hybrid approach using feature selection and adaptive svm for chronic disease classification," *International Journal of Computers and Applications*, pp. 1–13, 2019.

[5] S.-C. Huang, Y.-C. Tang, C.-W. Lee, and M.-J. Chang, "Kernel local Fisher discriminant analysis based manifold-regularized svm model for financial distress predictions," *Expert Systems with Applications*, vol. 39, no. 3, pp. 3855–3861, 2012.

[6] M. Al-Rubaie and J. M. Chang, "Privacy-preserving machine learning: threats and solutions," *IEEE Security & Privacy*, vol. 17, no. 2, pp. 49–58, 2019.

[7] S. Park, J. Byun, J. Lee, J. H. Cheon, and J. Lee, "He-friendly algorithm for privacy-preserving svm training," *IEEE Access*, vol. 8, no. 99, pp. 57414–57425, 2020.

[8] S. Laur, H. Lipmaa, and T. Mielikäinen, "Cryptographically private support vector machines," in *Proceedings of the KDD'06: 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 618–624, ACM, Philadelphia, Pennsylvania, 20 August 2006.

[9] J. Vaidya and C. Clifton, "Secure set intersection cardinality with application to association rule mining," *Journal of Computer Security*, vol. 13, no. 4, pp. 593–622, 2005.

[10] P. Li, T. Li, H. Ye, J. Li, X. Chen, and Y. Xiang, "Privacy-preserving machine learning with multiple data providers," *Future Generation Computer Systems*, vol. 87, pp. 341–350, 2018.

[11] J. Wang, L. Wu, H. Wang, K.-K. R. Choo, and D. He, "An efficient and privacy-preserving outsourced support vector machine training for internet of medical things," *IEEE Internet of Things Journal*, vol. 8, no. 1, pp. 458–473, 2021.

[12] F. Liu, W. K. Ng, and W. Zhang, "Encrypted Svm for Outsourced Data Mining," in *Proceedings of the 2015 IEEE 8th International Conference on Cloud Computing*, pp. 1085–1092, IEEE, New York, NY, USA, 27 June 2015.

[13] M. Z. Omer, H. Gao, and F. Sayed, "Privacy Preserving in Distributed Svm Data Mining on Vertical Partitioned Data," in *Proceedings of the 2016 3rd International Conference on Soft Computing Machine Intelligence (ISCMI)*, pp. 84–89, IEEE, Dubai, UAE, 23 November 2016.

[14] J. Vaidya, H. Yu, and X. Jiang, "Privacy-preserving svm classification," *Knowledge and Information Systems*, vol. 14, pp. 161–178, 2008.

[15] P. Mohassel and Y. Zhang, "Secureml: A System for Scalable Privacy-Preserving Machine learning," in *Proceedings of the 2017 IEEE Symposium on Security and Privacy (SP)*, pp. 19–38, IEEE, San Jose, CA, USA, 22 May 2017.

[16] K. Bonawitz, V. Ivanov, B. Kreuter et al., "Practical secure aggregation for privacy-preserving machine learning," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pp. 1175–1191, IEEE, Dallas, TX, USA, 30 October 2017.

[17] M. S. Riazi, C. Weinert, O. Tkachenko, E. M. Songhori, T. Schneider, and F. Koushanfar, "Chameleon: A Hybrid Secure Computation Framework for Machine Learning Applications," in *Proceedings of the ACM AsciaCCS'18*, pp. 707–721, IEEE, Incheon, Korea, June 2018.

[18] B. D. Rouhani, M. S. Riazi, and F. Koushanfar, "Deepsecure: scalable provably-secure deep learning," in *Proceedings of the 55th Annual Design Automation Conference*, 24 June 2018.

[19] K. Chaudhuri, C. Monteleoni, and A. D. Sarwate, "Differentially private empirical risk minimization," *Journal of Machine Learning Research*, vol. 12, pp. 1069–1109, 2011.

[20] P. Paillier, "Public-key cryptosystems based on composite degree residuosity classes," *Advances in Cryptology — EUROCRYPT*, Springer-Verlag, vol. '99, , pp. 223–238, 1999.

[21] M. Ito, A. Saito, and T. Nishizeki, "Secret sharing scheme realizing general access structure," *Electronics and Communications in Japan*, vol. 72, no. 9, pp. 56–64, 1989.

[22] G. D. Crescenzo, R. Ostrovsky, and S. Rajagopalan, "Conditional oblivious transfer and timed-release encryption," *Advances in Cryptology Eurocrypt*, 1999.

[23] H. Yu, X. Jiang, and J. Vaidya, "Privacy-preserving Svm Using Nonlinear Kernels on Horizontally Partitioned Data," in *Proceedings of the Acm Symposium on Applied Computing*, pp. 603–610, IEEE, Dijon, France, 23 April 2006.

[24] R. Hall, S. E. Fienberg, and Y. Nardi, "Secure multiple linear regression based on homomorphic encryption," *Journal of Official Statistics*, vol. 27, no. 4, pp. 669–691, 2011.

[25] C. Gentry, "A Fully Homomorphic Encryption Scheme," Ph.D. thesis, Stanford University, Stanford, California, 2009.

[26] V. Vaikuntanathan, "Computing blindfolded: new developments in fully homomorphic encryption," 2011 IEEE 52nd annual symposium on foundations of computer science," vol. 5–16, 2011.

[27] J. H. Cheon, A. Kim, M. Kim, and Y. Song, "Homomorphic encryption for arithmetic of approximate numbers, Advances in Cryptology - ASIACRYPT 2017," in *Advances in Cryptology – ASIACRYPT 2017*, T. Takagi and T. Peyrin, Eds., Springer International Publishing, Heidelberg, Germany, pp. 409–437, 2017.

[28] J. Ak. Suykens and J. Vandewalle, "Least squares support vector machine classifiers," *Neural Processing Letters*, vol. 9, pp. 293–300, 1999.

[29] X. Liu, R. H. Deng, K.-K. R. Choo, and J. Weng, "An efficient privacy-preserving outsourced calculation toolkit with multiple keys," *IEEE Transactions on Information Forensics and Security*, vol. 11, no. 11, pp. 2401–2414, 2016.

[30] X. Lei, X. Xiaofeng Liao, T. Tingwen Huang, H. Huaqing Li, and C. Chunqiang Hu, "Outsourcing large matrix inversion computation to a public cloud," *IEEE Transactions on Cloud Computing*, vol. 1, no. 1, p. 1, 2013.

[31] X. Chen, X. Xinyi Huang, J. Jin Li, J. Jianfeng Ma, W. Wenjing Lou, and D. S. Wong, "New algorithms for secure outsourcing of large-scale systems of linear equations," *IEEE Transactions on Information Forensics and Security*, vol. 10, no. 1, pp. 69–78, 2015.

[32] F. Chen, T. Xiang, X. Lei, and J. Chen, "Highly efficient linear regression outsourcing to a cloud," *IEEE Transactions on Cloud Computing*, vol. 2, no. 4, pp. 499–508, 2014.

[33] S. Goldwasser, S. Micali, and C. Rackoff, "The knowledge complexity of interactive proof systems," *SIAM Journal on Computing*, vol. 18, no. 1, pp. 186–208, 1989.

[34] D. R. Stinson and M. Paterson, *Cryptography: Theory and Practice*, Chapman & Hall/CRC, Boca Raton, Florida, Fourth Edition, 2018.

[35] Y. Elmehdwi, B. K. Samanthula, and W. Jiang, "Secure K-Nearest Neighbor Query over Encrypted Data in Outsourced Environments," in *Proceedings of the 2014 IEEE 30th International Conference on Data Engineering*, pp. 664–675, IEEE, Chicago, IL, USA, 31 March 2014.

[36] C. Clifton, M. Kantarcioglu, J. Vaidya, X. Lin, and M. Y. Zhu, "Tools for privacy preserving distributed data mining," *ACM SIGKDD Explorations Newsletter*, vol. 4, no. 2, pp. 28–34, 2002.

[37] T. Ranbaduge, D. Vatsalan, and P. Christen, "Secure multi-party summation protocols: are they secure enough under collusion?" *Transactions on Data Privacy*, vol. 13, no. 1, pp. 25–60, 2020.

[38] D. Dua and C. Graff, "UCI Machine Learning Repository," 2019, http://archive.ics.uci.edu/ml.

[39] C. V. Horsen, "gmpy2 – python extension module that supports multiple-precision arithmetic," 2022, https://pypi.org/project/gmpy2/.

[40] M. Dickinson, "Bigfloat – python package providing arbitrary-precision correctly-rounded binary floating-point arithmetic," 2019, https://pypi.org/project/bigfloat/.