Hindawi
Security and Communication Networks
Volume 2021, Article ID 8876893, 16 pages
https://doi.org/10.1155/2021/8876893

Research Article

Distributed Outsourced Privacy-Preserving Gradient Descent Methods among Multiple Parties

Zuowen Tan,[1] Haohan Zhang ,[2] Peiyi Hu,[1] and Rui Gao[2]

[1]*Department of Computing Science and Technology, Jiangxi University of Finance & Economics, Nanchang 330032, China*
[2]*Research Center for Information Technology Security, CEPREI, Guangzhou 510610, China*

Correspondence should be addressed to Haohan Zhang; zhang.haohan@aliyun.com

Received 13 April 2020; Revised 31 January 2021; Accepted 25 March 2021; Published 22 April 2021

Academic Editor: Amir Anees

The Internet of Things (IoT) is one of the latest internet evolutions. Cloud computing is an important technique which realizes the computational demand of largely distributed IoT devices/sensors by employing various machine learning models. Gradient descent methods are widely employed to find the optimal coefficients of a machine learning model in the cloud computing. Commonly, the data are distributed among multiple data owners, whereas the target function is held by the model owner. The model owner can train its model over data owner's data and provide predictions. However, the dataset or the target function's confidentiality may not be kept in secret during computations. Thus, security threats and privacy risks arise. To address the data and model's privacy mentioned above, we present two new outsourced privacy-preserving gradient descent (OPPGD) method schemes over horizontally or vertically partitioned data among multiple parties, respectively. Compared to previously proposed solutions, our methods improve in comprehensiveness in a more general scene. The data privacy and the model privacy are preserved during the whole learning and prediction procedures. In addition, the execution performance evaluation demonstrates that our schemes can help the model owner to optimize its target function and provide exact prediction with high efficiency and accuracy.

1. Introduction

The Internet of Things (IoT) is the latest internet evolution which provides multifarious novel digital, smart services and products by integrating abundant devices into networks [1]. It enables the communication between the physical world and the cyberspace [2]. IoT system contains radio-frequency identifications, wireless sensor networks, and the cloud computing [3]. Cloud computing realizes the computational demand of large-scale distributed IoT devices or sensors through various machine learning methods. Since IoT devices have tiny memory, the collected data are required to be stored and managed by the cloud servers [3–5]. Data can be downloaded from the cloud for different purposes such as machine learning. However, since there may exist sensitive data such as physiological data, location data, and some other data which are closely related to our personal information [6], this exposes the data to security breaches. Therefore, IoT not only provides convenience but also brings about security and privacy issues [7]. How to deal with security, privacy, and trust has been one of the main barriers in developing IoT in the real world [8, 9]. Most of the existing work on the protection of sensitive data is based on the secure communication channels and authorization [10]. In our paper, we focus on the protection of sensitive data in machine learning or deep learning. The data can be protected during the transmission phase, the computation phase, and the prediction phase. Furthermore, the computation and prediction results' privacy can also be preserved.

In machine learning or deep learning, the prediction function is usually called the decision model. The model coefficients' quality determines the accuracy of the model. In order to minimize the error of the model, the optimal coefficients are indispensable. This process is called model learning. Gradient descent methods are effective methods to

find the optimal coefficients of the decision model, such as linear regression, hyperplane decision classification, and neural networks. Gradient descent methods conclude four types: classical gradient descent method (GD), stochastic gradient descent method (SGD), minibatch stochastic gradient descent method (minibatch SGD), and momentum. Through these methods, the optimal prediction function can be obtained after several iterations.

In the cloud computing, the cloud server offers huge storage and computing capacity. The model owner initializes the prediction function, and the training data are distributed among different data owners who hope to get desired results with these data by cloud servers without exposing their privacy. These data form an enormous training dataset which is divided into different disjoint subsets held by different data owners. The dataset partition can be horizontal or vertical. The number of data owners can be two, even more than two. As is known to all, the channel transmission is not secure in our real life. In addition, data owners, the model owner, and the cloud server do not trust each other. When they train a decision model together, they worry about that any other participant may get information from their own data. So, they encrypt their training data or the decision model with their own public keys or blind their data to preserve confidentiality before delivering them to the cloud server. The training data and the decision model can be kept confidential during the whole cloud computing. After finishing training the decision model, the model owner learns the model securely based on the training dataset with the help of the cloud server. At this time, the clients can get the prediction about their request data from the cloud server according to this decision model.

At present, although a lot of researchers focus on the data privacy protection or the model privacy protection when gradient descent methods are utilized to optimize machine learning models, few schemes can provide both data privacy and model privacy at the same time. Beyond that, some privacy-preserving gradient descent schemes can protect data owners' privacy, but they are not applied to an outsourcing computation. In addition, the dataset's partition is usually horizontal or vertical in the distributed system. In many previous literature studies, few schemes can be applied to two different partitioned datasets at the same time. Besides, both training data and the decision model are held only by data owners rather than the model owner. In fact, it is more practical that the models are held by the model owner rather than the data owner. Motivated by the above, we construct two novel outsourcing gradient descent methods to solve these problems.

Generally speaking, it is necessary to preserve the privacy of the training data, the decision model, and the request data during the model training. Assume that there exists a training dataset X, and the corresponding label vector is $\mathbf{y}$. Each row of the dataset represents one sample $\mathbf{x}$ with a set of attributes. By $f(\mathbf{x})$, we denote the prediction function which maps the sample $\mathbf{x}_i$ into its corresponding category label y_i. According to the partition of the dataset, each data owner has part of data samples or part of the attributes. The model owner holds the coefficients of the prediction function $f(\mathbf{x})$. The target of data owners and the model owner is to minimize the error of the prediction function and obtain the optimal coefficients ultimately through the gradient descent methods. Thus, the model owner holds the optimal decision model. Then, it can provide the client accurate prediction. In this paper, we focus on outsourced gradient descent methods over distributed data among multiple parties which conclude data owners, the model owner, the cloud server, and the client. Both horizontal and vertical partition of the dataset are discussed. For the horizontally partitioned dataset, two or multiple data owners hold different samples with the same attributes, whereas two or more data owners hold all same samples but with different sets of attributes when the dataset is vertically partitioned.

1.1. Contributions. To address the privacy when performing gradient descent methods by multiple parties via the cloud computing, we propose two OPPGD schemes over horizontally or vertically distributed data. Our main contributions of this paper are summarized as follows:

(1) We design an outsourced privacy-preserving scalar product (OPPSP) algorithm. The cloud server computes the inner product of two vectors encrypted under different keys securely. For example, one data owner and the model owner hold one vector, respectively. Both parties first encrypt their own vector with their own key and send the encrypted vector to the cloud server. Then, the cloud server computes the scalar product of these two encrypted vectors.

(2) We propose two secure and comprehensive schemes to perform OPPGD over horizontally or vertically distributed dataset, respectively. The number of data owners can be two or more than two. The prediction functions are linear regression or neural networks. The OPPGD schemes are applied to classical GD, SGD, minibatch SGD, and momentum. It is worth noting that our schemes are with higher applicability and practicability contrasted with other schemes.

(3) We demonstrate that our OPPGD schemes are privacy-preserving. The computational cost and communication complexities are discussed. The analyses show that our OPPGD schemes are with high efficiency and accuracy.

1.2. Organization. The remainder of this paper is as follows. In Section 2, we discuss the related works on privacy-preserving gradient descent methods. In Section 3, we briefly introduce some preliminaries, Elgamal homomorphic cryptosystem [11], and gradient descent methods. In Section 4, we describe the system model, problem statements, the threat model, and the system requirements. We present two OPPGD schemes and prove their correctness, security, and complexity in Section 5. The performance evaluation of the schemes is analyzed in Section 6. Section 7 makes a conclusion on our OPPGD schemes.

2. Related Works

In this section, we review works on privacy-preserving gradient descent methods among parties. According to the existence or absence of cloud servers, the existing works can be classified into two categories.

2.1. The Absence of Cloud Servers. Wan et al. [12] presented the first privacy-preserving scheme for gradient descent methods. They proposed a generic formulation of gradient descent methods by defining the prediction function $f(x)$ as a composition $g^{\circ}h(x)$. The formulation is used to perform the specific iteration-based algorithm in linear regression or neural networks. In our paper, we also use this formulation. However, the partition of the dataset discussed in their scheme [12] is only vertical. Han et al. [13] extended the scheme [12] to the horizontally distributed dataset and proposed the least square approach to perform gradient descent methods. Both schemes [12, 13] utilize a secure scalar product to gain their privacy preservation, but they cannot be applied to the outsourced model. Gabor Danner and Jelasity [14] designed a novel fully distributed privacy-preserving minibatch SGD that can avoid collecting any personal data centrally. Their scheme does not require the precise sum of gradients. A tree topology and homomorphic encryption are employed to produce a "quick and dirty" partial sum. The protocol can resist collusion attacks. Hegedus and Jelasity [15] adopted differential privacy technology to solve privacy-preserving stochastic distributed gradient descent methods. Mehnaz et al. [16] designed two secure gradient descent schemes over horizontally partitioned data and vertically partitioned data via a secure sum protocol. Later, they designed a secure gradient descent method scheme [17] without Yao's circuits over the arbitrarily partitioned dataset. Based on output perturbation, Wu et al. [18] devised a novel "bolt-on" differentially private algorithm for stochastic gradient descent.

2.2. The Existence of Cloud Servers. Liu et al. [19] designed an encrypted gradient descent method. Both data owners and the cloud server perform operations collaboratively to learn the target function without leaking any data privacy. They extended their scheme to the outsourced model by utilizing the BGN cryptosystem. However, their protocol is only suitable for a two-party scenario. Shokri and Shmatikov [20] learnt an accurate neural network model without sharing input datasets by using the stochastic gradient descent method. After the parameter server initializes the parameter vector, it updates the parameters with the help of the cloud server without leaking any privacy. Kim et al. [21] provided a practical frame for mainstream learning models such as logistic regression. They calculated the gradient descent method securely by using homomorphic encryption, but this is inefficient. Since the required bit length of ciphertext modulus per iteration is too long, it also takes up too much space. Francisco-Javier et al. [22] realized training supervised machine learning over ciphertext. Through the gradient descent method, the server optimizes the predicted training model without exposing the data or the training model. Mohassel and Zhang [23] used the stochastic gradient descent method to construct new and efficient privacy-preserving machine learning protocols for linear regression, logistic regression, and neural network. Their protocol is involved with a two-server model. Data providers distribute their private data among two noncolluding servers, while the servers train models on the joint data through secure two-party computation techniques. Li et al. [24] also presented a multikey privacy-preserving deep learning scheme in the cloud computing environment. Their protocols realize outsourced multilayer backpropagation network learning via the gradient descent methods. Ma et al. [25] took advantage of two noncolluding servers' framework to build a new outsourced model of the privacy-preserving neural network. However, the model owner can only make prediction rather than learning the model itself.

2.3. The Other Works on Privacy Preservation for Machine Learning. Aside from the above privacy-preserving gradient descent methods, there are also plenty of other works on privacy-preserving computation over distributed data among multiple parties under the cloud environment. Liu et al. [26] constructed an efficient privacy-preserving method to compute outsourced data. They [27] also proposed a privacy-preserving outsourced calculation toolkit, which allows data owners to securely outsource their data to the cloud for storage and calculation. Rady et al. [28] designed a new architecture that achieves the confidentiality and integrity of query results of the outsourced database. Yu et al. [29] devised a verifiable outsourced computation scheme over encrypted data by employing fully homomorphic encryption and polynomial factorization algorithm. Chamikara et al. [30] presented an efficient and scalable nonreversible perturbation algorithm of data mining without leaking privacy of big data via optimal geometric transformations. Li et al. [31] proposed a novel outsourced privacy-preserving classification scheme based on homomorphic encryption. In their scheme, multiple parties outsource securely their sensitive data to an untrusted evaluator for storing and processing. Li et al. [32] devised a novel scheme for a classifier owner to provide users with the privacy-preserving classification service by delegating a cloud server. However, they focus on two concrete secure classification protocols: naive Bayes classifier and hyperplane decision classifier. Park et al. [33] described a privacy-preserving naive Bayes protocol. No intermediate interactions are required between the server and the clients. Hence, their protocols can alleviate the heavy computational cost of fully homomorphic encryption. Li et al. [34] proposed an outsourced privacy-preserving C4.5 decision tree algorithm over both horizontally and vertically partitioned datasets. They used the BCP cryptosystem to present an outsourced privacy-preserving weighted average protocol. Rong et al. [35] presented a series of privacy-preserving building blocks for verifiable and privacy-preserving association rule mining under the hybrid cloud environment. Li et al. [36] used an efficient homomorphic encryption with multiple keys to design an outsourced privacy-preserving ID3 data mining

solution. Xue et al. [37] built a differential privacy-based privacy-preserving classification system for secure edge computing. Yang et al. [38] realized privacy-preserving medical record sharing in the cloud computing environment. Kaur et al. [39] devised an efficient privacy-preserving collaborative filtering for the healthcare recommender system over arbitrary distributed data. In our work, we aim at designing outsourced privacy-preserving gradient descent methods among multiple parties. To the best of our knowledge, there has not been any work which addresses the issue comprehensively.

3. Preliminaries

In this section, we introduce some preliminaries for our outsourced privacy-preserving gradient descent schemes.

3.1. The Elgamal Homomorphic Cryptosystem. The Elgamal cryptosystem [11] comprises the following algorithms: preparation, key generation, encryption, and decryption:

Preparation (λ): given a security parameter λ. The system generates the public parameter PP as follows. The system first chooses a large prime number N and a random number g less than N. And it publishes the multiplicative cyclic group G of prime order N with the generator g. The public parameter PP = (Ng)

KeyGen (PP): taking PP as the input, each party P_i randomly selects a number sk_i less than N as its private key and computes $\mathrm{pk}_i = g^{\mathrm{sk}_i} \bmod N$ as its public key.

$\mathrm{Enc}_{\mathrm{pk}_i}(M_i)$: P_i selects a random integer r_i which is coprime to $(N-1)$ and encrypts its plaintext M_i with the public key pk_i to generate the ciphertext $C_i = (C_{i1}C_{i2})$

$$\begin{aligned} C_{i1} &= g^{r_i} \bmod N \\ C_{i2} &= \mathrm{pk}_i^{r_i} m \bmod N \end{aligned} \tag{1}$$

$\mathrm{Dec}_{\mathrm{pk}_i}(C_i)$: each party P_i decrypts C_i with its secret key sk_i and obtains the plaintext M_i. The decryption process is

$$M_i = C_{i2}C_{i1}^{-\mathrm{sk}_i} \bmod N \tag{2}$$

Its correctness is early confirmed.

$$\begin{aligned} C_{i2}C_{i1}^{-\mathrm{sk}_i} \bmod N &= \left(\mathrm{pk}_i^{r_i} M_i\right)\left(g^{r_i}\right)^{-\mathrm{sk}_i} \bmod N \\ &= \left(g^{\mathrm{sk}_i r_i} M_i\right)\left(g^{r_i}\right)^{-\mathrm{sk}_i} \bmod N \\ &= M_i \end{aligned} \tag{3}$$

The semantic security of the Elgamal cryptosystem is based on the hardness assumption of discrete logarithm problems over finite fields.

3.2. The Key Conversion System. As for the secure outsourced computation over the dataset among multiple parties, the essential difficulty is how to deal with different ciphertexts encrypted under different keys which are sent from multiple parties. Based on Gentry's fully homomorphic encryption [40], we transform the ciphertext under different keys into the ciphertext under the same key. Take two parties, Alice and Bob, as an example. Assume that their respective key pairs are $(\mathrm{pk}_a\mathrm{sk}_a)$ and $(\mathrm{pk}_b\mathrm{sk}_b)$. For a plaintext m, its ciphertext encrypted under key pk_a is $[m]_{\mathrm{pk}_a}$. The goal is to switch encrypted $[m]_{\mathrm{pk}_a}$ into a new ciphertext $[m]_{\mathrm{pk}_b}$ which is encrypted under the public key pk_b. The conversion can be divided into the following steps:

Rekey generation $(\mathrm{pk}_b\mathrm{sk}_a)$: taking pk_b and sk_a as the input, it outputs the rekey $\widetilde{\mathrm{sk}_a} = \{\widetilde{\mathrm{sk}_{ai}}\}_{i=1}^{I}$, where $\widetilde{\mathrm{sk}_{ai}} \longleftarrow \mathrm{Encrypt}(\mathrm{pk}_b\mathrm{sk}_{ai})$ is the i-th binary representation of sk_a

Reencryption $(\mathrm{pk}_b, [m]_{\mathrm{pk}_a})$: taking public key pk_b and ciphertext $[m]_{\mathrm{pk}_a}$ as inputs, it outputs $\widetilde{m} = \{\widetilde{m}_i\}_{i=1}^{I}$, where $\widetilde{p}_i \longleftarrow \mathrm{Encrypt}(\mathrm{pk}_b[p_i]_{\mathrm{pk}_a})$

Evaluation algorithm $(\mathrm{pk}_b D_\Pi, \widetilde{p}_i\widetilde{\mathrm{sk}_{ai}})$: taking the public key pk_b, rekey $\widetilde{\mathrm{sk}_a}$, ciphertext $\widetilde{m}$, and a decryption circuit D_Π, it outputs $[m]_{\mathrm{pk}_b} \longleftarrow \mathrm{Evaluate}(\mathrm{pk}_b D_\Pi \widetilde{m}\widetilde{\mathrm{sk}_a})$

3.3. Gradient Descent Methods. Assume that D is the dataset of data samples, $\{(\mathbf{x}_i y_i) | i = 1, 2 \ldots N\}$, where the vector $\mathbf{x}_i = [x_{i1}x_{i2} \ldots x_{im}]$ presents the i-th sample's m attributes and y_i denotes the target attribute. The goal is to determine a prediction function $f(x)$ such that $f(\mathbf{x}_i)$ is as close to y_i as possible. Thus, when one makes prediction about the test data, the basic strategy is to make the prediction function to produce the smaller error. Gradient descent methods are always applied to search $f(x)$'s optimal coefficients. The technique can minimize the prediction error. The whole process can be described as follows. At the beginning, one determines the loss function $L(x)$, randomly initializes a coefficient vector of $f(x)$, and calculates the current error about the learning dataset. If the current error is not ideal, one can take the derivative of $L(x)$ with respect to the vector, modify the coefficient vector, and update $f(x)$ based on the derivative. Then, one recalculates the loss and repeats optimizing its model until the minimum error appears. To this end, one can generate the optimal value through several iterations.

There are four main gradient descent methods, such as classical GD, SGD, minibatch SGD, and momentum. In classical GD, the loss function is determined by all samples in each iteration which leads to high computational complexity. For SGD, the loss function is determined by a random sample every iteration which reduces computing overhead. However, this method has one weakness that, sometimes, the final coefficient vector may be the local optimal value rather than the global optimal value. When the loss function is determined by a batch of random samples every iteration, the gradient descent method is called minibatch SGD. The minibatch SGD has classical GD's and SGD's advantages and overcomes their weaknesses. So far, SGD is the most widely applied in machine learning. Momentum is the latest gradient descent method which greatly improves the accuracy and speed of the prediction. Beside

the learning rate η, the coefficient vector $\omega = \gamma\omega - \eta\nabla\omega$ in momentum contains a new parameter γ, the attenuation rate. However, our schemes can be applied to the above four main gradient descent methods.

The error function of every sample $\mathbf{x}_i$ is $E(f(\mathbf{x}_i)y_i) = (y_i - f(\mathbf{x}_i))^2$. Given l arbitrary samples, the loss function is (12)

$$L = \frac{1}{l}\sum_{i=1}^{l} E(f(\mathbf{x}_i)y_i) = \frac{1}{2l}\sum_{i=1}^{l}(y_i - f(\mathbf{x}_i))^2 \quad (4)$$

The prediction function $f(x)$ is a composition function of two functions $g(z)$ and $z = h(x)$, where $g(z)$ is any differentiable function and $h(x)$ is a linearly separable function: $h(\mathbf{x}_i) = \sum_{j=1}^{m}\omega_j x_{ij}$, where $\omega = (\omega_1\omega_2 \ldots \omega_m)$ is the coefficient vector of the prediction function. When $l = 1$, the method is SGD, when $1 < l < N$, the method is minibatch SGD, whereas when $l = N$, the method is GD. Subsequently, we update the coefficient vector $\omega = \omega - \eta\nabla\omega$, where $\nabla\omega = \partial L\partial\omega$ and η is a constant parameter called learning rate. When the coefficient vector is $\omega = \gamma\omega - \eta\nabla\omega$, where γ is a constant parameter called attenuation rate, this method is momentum.

For each sample $\mathbf{x}_i$, there is a derivative $\partial E(f(\mathbf{x}_i)y_i)\partial\omega$. Thus, we calculate $\nabla\omega = (\partial L\partial\omega) = (1l)\sum_{i=1}^{l}\partial E(f(\mathbf{x}_i)y_i)\partial\omega$

$$\begin{aligned}\frac{\partial E(f(\mathbf{x}_i)y_i)}{\partial\boldsymbol{\omega}} &= \frac{\partial E(f(\mathbf{x}_i)y_i)}{\partial f(\mathbf{x}_i)}\frac{\partial f(\mathbf{x}_i)}{\partial\boldsymbol{\omega}} \\ &= \frac{\partial E(f(\mathbf{x}_i)y_i)}{\partial f(\mathbf{x}_i)}\frac{\partial g(h(\mathbf{x}_i))}{\partial h(\mathbf{x}_i)}\frac{\partial h(\mathbf{x}_i)}{\partial\boldsymbol{\omega}}\end{aligned} \quad (5)$$

As the function $f(x)$ changes, $\nabla\omega$ is also different. Here, we discuss two specific functions used in linear regression and neural network.

In linear regression, the prediction function for an arbitrary sample $\mathbf{x}_i$ is $f(\mathbf{x}_i) = \sum_{j=1}^{m}\omega_j x_{ij}$. Then,

$$\begin{aligned}\frac{\partial E(f(\mathbf{x}_i)y_i)}{\partial\boldsymbol{\omega}} &= \frac{\partial}{\partial\boldsymbol{\omega}}\left(\frac{1}{2}(y_i - f(\mathbf{x}_i))^2\right) \\ &= \frac{1}{2}\mathbf{x}_i(y_i - f(\mathbf{x}_i)) \\ &= \frac{1}{2}(\mathbf{x}_i f(\mathbf{x}_i) - \mathbf{x}_i y_i)\end{aligned} \quad (6)$$

In neural networks, $f(x)$ is also called as activation function that is a sigmoid function, $(z) = 1(1 + e^{-\alpha z})$, or tanh function, $f(z) = (e^{\alpha z} - e^{-\alpha z})(e^{\alpha z} + e^{-\alpha z})$. If the function is a sigmoid function, the prediction function for an arbitrary sample $\mathbf{x}_i$ is $f(\mathbf{x}_i) = 1(1 + e^{-\alpha\sum_{j=1}^{m}\omega_j x_{ij}})$. Then,

$$\begin{aligned}\frac{\partial E(f(\mathbf{x}_i)y_i)}{\partial\boldsymbol{\omega}} &= \frac{\partial}{\partial\boldsymbol{\omega}}\left(\frac{1}{2}(y_i - f(\mathbf{x}_i))^2\right) \\ &= -\frac{1}{2}\alpha(y_i - f(\mathbf{x}_i))f(\mathbf{x}_i)(1 - f(\mathbf{x}_i)) \\ &= -\frac{1}{2}\alpha(y_i - f(\mathbf{x}_i))f(\mathbf{x}_i)(1 - f(\mathbf{x}_i)) \\ &= \frac{1}{2}\alpha\left(\mathbf{x}_i y_i f^2(\mathbf{x}_i) + \mathbf{x}_i f^2(\mathbf{x}_i) - \mathbf{x}_i f^3(\mathbf{x}_i) - \mathbf{x}_i y_i f(\mathbf{x}_i)\right)\end{aligned} \quad (7)$$

Through the Taylor expansion formula, the function $f(x)$ can be expanded into a polynomial $T(a)$. Then, we have

$$\begin{aligned}\frac{\partial E(f(\mathbf{x}_i)y_i)}{\partial\boldsymbol{\omega}} \approx \frac{1}{2}\alpha\Bigg(&\mathbf{x}_i y_i T^2\left(\sum_{j=1}^{m}\omega_j x_{ij}\right) + \mathbf{x}_i T^2\left(\sum_{j=1}^{m}\omega_j x_{ij}\right) \\ &- \mathbf{x}_i T^3\left(\sum_{j=1}^{m}\omega_j x_{ij}\right) - \mathbf{x}_i y_i T\left(\sum_{j=1}^{m}\omega_j x_{ij}\right)\Bigg)\end{aligned} \quad (8)$$

4. Models and Requirements

4.1. System Model. As shown in Figure 1, the system comprises five entities: data owners, a model owner, a cloud server, a key conversion server, and a trusty decryption server. Each entity is described as follows:

Data owner (DO): after receiving the public parameter PP, each DO generates their own key pair and encrypts their data. Then, DOs send their respective ciphertext to the cloud server, depicted as Step 1 in Figure 1. After MO has finished training the model, one DO can request the CS and MO to make prediction.

Cloud server (CS): assume that CS can provide DOs and MO with unlimited computation and storage service. After receiving vectors encrypted by every DO and MO, the CS executes the OPPSP algorithm and finally sends the encrypted results back to DOs as Step 2.

Model owner (MO): MO holds the target function which contains the coefficient vector, learning rate, or the attenuation rate. MO encrypts the target function's coefficients with its own key and then sends the ciphertext to the CS and executes the OPPSP algorithm. After receiving $\nabla\omega$, MO updates its model until it gets the optimal coefficients. Moreover, it can provide DO with prediction services.

Key conversion server (KCS): KCS runs the conversion algorithm and switches different ciphertexts encrypted under DOs' respective keys into a new intermediate ciphertext under the same key, which is depicted as Step 3.

Trusty decryption server (TDS): assume that TDS is trusty. It only provides decryption service. TDS will not conspire with other parties. After receiving new encrypted results from the KCS as Step 4, the TDS decrypts these results and performs few computations to acquire the final results. In the end, TDS sends the intermediate results back to the MO, as depicted in Step 5.

In our system model, each entity is semihonest except TDS. All the entities have some background knowledge of the attribute names, class names, and the number of their attributes. Each data owner has a part of the complete dataset, which can be partitioned horizontally or vertically.

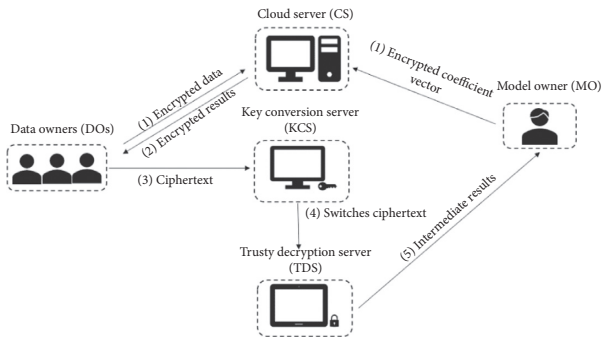

Figure 1: The system model.

When the dataset is distributed vertically, all data owners have the class value vector. The complete attribute dataset X is of size $n \times m$, and the target vector $\mathbf{y}$ is represented as follows:

$$X = \begin{bmatrix} x_{1,1} & \cdots & x_{1m} \\ \vdots & \ddots & \vdots \\ x_{n1} & \cdots & x_{nm} \end{bmatrix}$$
$$y = [y_1 y_2 \cdots y_n]^T \tag{9}$$

where y_i is $\mathbf{x}_i$'s corresponding class value.

For the horizontally partitioned dataset, each data owner has n samples with all the attributes and the corresponding class value, as described in Figure 2. For the vertically partitioned dataset, each data owner has m_i attributes with all the samples and the corresponding class value. The data owner P_i's data can be depicted as in Figure 3

The scheme consists of the preparation phase, the training phase, and the prediction phase. An overview of the scheme can be described as follows:

Preparation phase: according to the public parameter PP, DOs and MO generate their respective key pairs. They also share a secret value k in advance. Then, DOs encrypt their dataset with their respective keys, while MO encrypts the coefficient vector of its model with his public key. Then, DOs and MO send their ciphertext to CS, respectively.

Training phase: CS performs the OPPSP algorithm and sends the results back to DOs. Next, DOs perform decryption and send the results to the KCS. KCS switches these encrypted results and sends the final results to the TDS. TDS decrypts the results and sends the results to the MO. MO can update the coefficients to optimize the model.

Prediction phase: with the help of the CS, the MO makes prediction for the DO's query.

4.2. Problem Statement. Let D_i be the dataset of data owner DO_i. All datasets are disjoint and composed of the complete dataset. Each dataset $D_i = \{\mathbf{x}_j^i\} \subset X$ is of size p_i, where the integers $j \in [1 p_i]$ and $i \in [1t]$. If the dataset is partitioned horizontally, $\mathbf{x}_j^i \in R^m$. If the dataset is partitioned vertically, $\mathbf{x}_j^i \in R^n$. MO holds the coefficient vector $\omega \in R^m$ of the target function $f(x)$ and the target vector $\mathbf{y} \in R^m$

$x_{1,1}$	$x_{1,2}$	$x_{1,3}$	$\cdots$	$x_{1,m}$
$x_{2,1}$	$x_{2,2}$	$x_{2,3}$	$\cdots$	$x_{2,m}$
$x_{3,1}$	$x_{3,2}$	$x_{3,3}$	$\cdots$	$x_{3,m}$
$x_{4,1}$	$x_{4,2}$	$x_{4,3}$	$\cdots$	$x_{4,m}$
$\vdots$	$\vdots$	$\vdots$	$\cdots$	$\vdots$
$x_{n-1,1}$	$x_{n-1,2}$	$x_{n-1,3}$	$\cdots$	$x_{n-1,m}$
$x_{n,1}$	$x_{n,2}$	$x_{n,3}$	$\cdots$	$x_{n,m}$

Figure 2: Horizontally partitioned dataset.

$x_{1,1}$	$x_{1,2}$	$x_{1,3}$	$\cdots$	$x_{1,m}$
$x_{2,1}$	$x_{2,2}$	$x_{2,3}$	$\cdots$	$x_{2,m}$
$x_{3,1}$	$x_{3,2}$	$x_{3,3}$	$\cdots$	$x_{3,m}$
$x_{4,1}$	$x_{4,2}$	$x_{4,3}$	$\cdots$	$x_{4,m}$
$\vdots$	$\vdots$	$\vdots$	$\cdots$	$\vdots$
$x_{n-1,1}$	$x_{n-1,2}$	$x_{n-1,3}$	$\cdots$	$x_{n-1,m}$
$x_{n,1}$	$x_{n,2}$	$x_{n,3}$	$\cdots$	$x_{n,m}$

Figure 3: Vertically partitioned dataset.

Our goal is to train the MO's target function with DOs' datasets. MO needs to get $\nabla\omega$ to optimize the coefficients of the target function $f(x)$ after renewing coefficients over MO's coefficients and DOs' datasets. We discuss two kinds of machine learning methods: linear regression and neural network. For linear regression, each MO's task is to obtain encrypted $\mathbf{x}_i(y_i - f(\mathbf{x}_i))$ of every sample $\mathbf{x}_i$ with the help of

the CS. For the neural network, each MO's task is to obtain $\mathbf{x}_i f(\mathbf{x}_i)(y_i - f(\mathbf{x}_i))(1 - f(\mathbf{x}_i))$ for every sample $\mathbf{x}_i$. After getting the results $\nabla\omega$, MO chooses one gradient descent method to refresh its coefficients. In the end, MO can provide accurate prediction services about the query through its optimal target function.

Since each P_i encrypts D_i with its public key pk_i and MO encrypts its coefficients with its public key pk, CS performs computations only over encrypted data. TDS performs the decryption algorithm of the final results, while DO and MO share a secret value k. This can prevent the TDS from getting the information about the coefficients.

4.3. Threat Model. Assume that all the entities except TDS are semihonest, honest-but-curious. In other words, these entities follow the protocol, but they may try to obtain as much as secret information from the message which they receive.

Consider two kinds of adversaries in this model: an external adversary and an internal adversary. An external adversary may obtain some information, i.e., encrypted data or encrypted results, during every iteration via public channels. An internal adversary could refer to a malicious data owner DO, the model owner MO, the cloud server CS, or the key conversion server KCS. The goal of a malicious DO is to extract the coefficients of target function $f(x)$. An internal adversary KCS tries to extract the intermediate results and the MO's coefficient vector, while the goal of an adversary MO is to reveal the information of each DO's partitioned dataset. In addition, if the CS is an internal adversary, it tries to acquire MO's coefficients or DO's datasets.

4.4. Privacy Requirements. In the outsourced gradient descent schemes, privacy preservation is essential. In our model, we assume that the cloud server is semihonest. In order to measure the extent of privacy preservation, now, we define two privacy preservation levels.

Definition 1. Explicit privacy leakage means that privacy may be exposed during the computation of the cloud server or among the message transmission over public channels. If an outsourced computation scheme can prevent the explicit privacy leakage, we call it achieving the level-1 privacy.

Definition 2. Implicit privacy leakage means that one's privacy may be leaked by deducing from results of the cloud server. If an outsourced computing scheme can prevent the implicit privacy leakage, we call it achieving the level-2 privacy.

In our OPPGD scheme, DOs' data and MO's coefficient vector are uploaded to the cloud server in the ciphertext. Explicit privacy leakage means that DOs' data or MO's coefficient vector and final desired results are leaked during the scheme. Implicit privacy leakage means that it is impossible to deduce DOs' data or MO's coefficient vector from intermediate results. Our OPPGD schemes can realize level-1 privacy or level-2 privacy.

5. Two OPPGD Schemes

In this section, we present two outsourced privacy-preserving gradient descent schemes over horizontally partitioned data or vertically partitioned data. For simplicity, we make the following assumptions. When data are horizontally partitioned, each DO has only one record with all the attributes and the class value. When data are vertically partitioned, each DO has one attribute of all the samples and the corresponding class vector. An outsourced privacy-preserving gradient descent scheme is composed of the preparation phase, the training phase, and the prediction phase. Now, we first describe the OPPGD scheme over horizontally partitioned data.

5.1. OPPGD Scheme over Horizontally Partitioned Data

5.1.1. Preparation Phase. The phase is involved with several essential algorithms, parameter generation, key pair generation, and encryption.

Step 1: the system runs Algorithm 1 to generate PP = (g, N) and SP = k

Step 2: after receiving the PP, DOs, MO, and TDS operate Algorithm 2 to obtain their own key pair $(\mathrm{pk}_i \mathrm{sk}_i)$, $(\mathrm{pk}_M\, \mathrm{sk}_M)$, and $(\mathrm{pk}_*\, \mathrm{sk}_*)$

Step 3: DO encrypts its $\mathbf{x}_i$ and y_i to be $[\mathbf{x}_i]_{\mathrm{pk}_i} = \mathrm{pk}_i^{r_i}\mathbf{x}_i \bmod N$ and $[y_i]_{\mathrm{pk}_i} = \mathrm{pk}_i^{r_i} y_i \bmod N$. Then, MO encrypts its coefficient vector ω to be $[[\omega_1]_{\mathrm{pk}_M}[\omega_2]_{\mathrm{pk}_M} \ldots [\omega_m]_{\mathrm{pk}_M}]$ by Algorithm 3, where $\mathbf{x}_i = (x_{i1}x_{i2} \ldots x_{im})$ and $[\omega_i]_{\mathrm{pk}_M} = \mathrm{pk}_M^{r_M}\omega_i \bmod N$

5.1.2. Training Phase

Step 4: each DO sends their encrypted $[\mathbf{x}_i]_{\mathrm{pk}_i}$ and $[y_i]_{\mathrm{pk}_i}$ to the CS, and MO sends $[\omega]_{\mathrm{pk}_M}$ to the CS.

Step 5: CS operates Algorithm 4 and obtains the encrypted scalar product vector $\mathbf{S}$ after receiving $[\mathbf{x}_i]_{\mathrm{pk}_i}$, $[y_i]_{\mathrm{pk}_i}$, and $[\omega]_{\mathrm{pk}_M}$ from DOs and MO, where $s_i = [\mathbf{x}_i]_{\mathrm{pk}_i} \cdot [\omega]_{\mathrm{pk}_M}$. In addition, CS also makes some other computations over some components of $\nabla\omega$. To be specific, CS computes $\mathbf{I}_{i1}$ and $\mathbf{I}_{i2}$ in the linear regression model or computes $\mathbf{I}_{i3}$, I_{i4}, $\mathbf{I}_{i5}$, and $\mathbf{I}_{i6}$ in the neural network model,

Input: the security parameter λ
Output: the public parameter PP = $\{g, N\}$, a secret value k
(1) generate a prime N, choose a primitive element g in Z_N^*
(2) generate a secret value k
(3) **end**

Algorithm 1: Parameter generation.

Input: the public parameter PP$\{gN\}$ and a secret value k
Output: the key pair (pk sk)
(1) choose sk < N
(2) compute pk = $g^{\text{sk}} \bmod N$
(3) **end**
(4) return (pk sk)

Algorithm 2: Key pair generation.

Input: the key pair (pk sk), a message m, and a random integer r_i which is a coprime to $N - 1$
Output: the encrypted message $[m]_{\text{pk}}$
(1) choose a random integer r which is a coprime to $N - 1$
(2) compute $[m]_{\text{pk}} = \text{pk}^r m \bmod N$, g^r **mod N**
(3) **end**
(4) **return** $[\mathbf{m}]_{\mathbf{pk}}$ and $g^r \bmod N$

Algorithm 3: Encryption.

Input: two encrypted vectors $[a]_A$ and $[b]_B$, where $a = (a_1 a_2 \ldots a_m)$ and $b = (b_1 b_2 \ldots b_m)$
Output: encrypted scalar product s
(1) CS computes $s = \sum_{j=1}^{m} [a_i]_A \cdot [b_i]_B$
(2) **end**
(3) return **s**

Algorithm 4: Outsourced privacy-preserving scalar product.

Input: an encrypted message $[m]_{\text{pk}}$, its corresponding key pair (pk sk), and ciphertext $g^r \bmod N$
Output: m
(1) compute: $m = [m]_{\text{pk}} \cdot g^{-r\text{sk}} \bmod N$
(2) **end**
(3) **return m**

Algorithm 5: Decryption.

$$\begin{aligned}
\mathbf{I}_1' &= g^{2\mathrm{sk}_i r_i}\mathbf{x}_i y_i \bmod N\\
\mathbf{I}_2' &= g^{\mathrm{sk}_M r_M}\mathbf{x}_i \sum_{j=1}^{m}\omega_j x_{ij} \bmod N\\
\mathbf{I}_3' &= g^{2\mathrm{sk}_M r_M}\mathbf{x}_i y_i T^2\left(\sum_{j=1}^{m}\omega_j x_{ij}\right) \bmod N\\
\mathbf{I}_4' &= g^{2\mathrm{sk}_M r_M}\mathbf{x}_i T^2\left(\sum_{j=1}^{m}\omega_j x_{ij}\right) \bmod N\\
\mathbf{I}_5' &= [\mathbf{x}_i]_{\mathrm{pk}_i} s_i^3 = g^{3\mathrm{sk}_M r_M}\mathbf{x}_i T^3\left(\sum_{j=1}^{m}\omega_j x_{ij}\right) \bmod N\\
\mathbf{I}_6' &= [\mathbf{x}_i]_{\mathrm{pk}_i}[y_i]_{\mathrm{pk}_i} s_i = g^{\mathrm{sk}_M r}\mathbf{x}_i y_i T\left(\sum_{j=1}^{m}\omega_j x_{ij}\right) \bmod N
\end{aligned} \tag{10}$$

Step 6: CS sends the above encrypted results to the DO.

After receiving encrypted scalar product **S**, DO performs decryption operation. The TDS and the MO perform decryption as shown in Algorithm 5

Step 7: once DOs receive encrypted results $\mathbf{I}_{i1}\mathbf{I}_{i2}$ or $\mathbf{I}_{i3}$, $\mathbf{I}_{i4}$, $\mathbf{I}_{i5}$, $\mathbf{I}_{i6}$ from CS, DO runs Algorithm 5 to get the new ciphertext:

$$\begin{aligned}
k\mathbf{I}_1^{'} &= [k\mathbf{x}_i y_i]_{\mathrm{pk}^*}\\
k\mathbf{I}_2^{'} &= \left[k\mathbf{x}_i \cdot \sum_{j=1}^{m}\omega_j x_{ij}\right]_{\mathrm{pk}^*}\\
k\mathbf{I}_3^{'} &= \left[k\mathbf{x}_i y_i T^2\left(\sum_{j=1}^{m}\omega_j x_{ij}\right)\right]_{\mathrm{pk}^*}\\
k\mathbf{I}_4^{'} &= \left[k\mathbf{x}_i T^2\left(\sum_{j=1}^{m}\omega_j x_{ij}\right)\right]_{\mathrm{pk}^*}\\
k\mathbf{I}_5^{'} &= \left[k\mathbf{x}_i T^3\left(\sum_{j=1}^{m}\omega_j x_{ij}\right) \bmod N\right]_{\mathrm{pk}^*}\\
k\mathbf{I}_6^{'} &= \left[k\mathbf{x}_i y_i T\left(\sum_{j=1}^{m}\omega_j x_{ij}\right)\right]_{\mathrm{pk}^*}
\end{aligned} \tag{11}$$

Step 8: DO blinds above ciphered data with the security parameter k to be $k\mathbf{I}_1' k\mathbf{I}_2'$ in the linear regression model or $k\mathbf{I}_3'$, $k\mathbf{I}_3'$, $k\mathbf{I}_5'$, $k\mathbf{I}_6'$ in the neural network model.

Step 9: DO sends these blinded encrypted results to the KCS.

Step 10: KCS operates Algorithm 6 to convert the blinded encrypted results $k\mathbf{I}_1'$, $k\mathbf{I}_2'$ or $k\mathbf{I}_3'$, $k\mathbf{I}_3'$, $k\mathbf{I}_5'$, $k\mathbf{I}_6'$ to be new results $k\mathbf{I}_1^{'}$ and $k\mathbf{I}_2^{'}$ in the linear regression model or $k\mathbf{I}_3^{'}$, $k\mathbf{I}_4^{'}$ $k\mathbf{I}_5^{'}$, and $k\mathbf{I}_6^{'}$ in the neural network model,

$$\begin{aligned}
k\mathbf{I}_1^* &= k\mathbf{x}_i y_i\\
k\mathbf{I}_2^* &= k\mathbf{x}_i \sum_{j=1}^{m}\omega_j x_{ij}\\
k\mathbf{I}_3^* &= k\mathbf{x}_i y_i T^2\left(\sum_{j=1}^{m}\omega_j x_{ij}\right)\\
k\mathbf{I}_4^* &= k\mathbf{x}_i T^2\left(\sum_{j=1}^{m}\omega_j x_{ij}\right) \quad \omega_j x_{ij}\\
k\mathbf{I}_5^* &= k\mathbf{x}_i T^3\left(\sum_{j=1}^{m}\omega_j x_{ij}\right)\\
k\mathbf{I}_6^* &= k\mathbf{x}_i y_i T\sum_{j=1}^{m}
\end{aligned} \tag{12}$$

which are all encrypted under the TDS's key pk^*

Step 11: subsequently, the KCS sends the above intermediate results $k\mathbf{I}_1^{'}$, $k\mathbf{I}_2^{'}$ or $k\mathbf{I}_3^{'}$, $k\mathbf{I}_4^{'}$, $k\mathbf{I}_5^{'}$, $k\mathbf{I}_6^{'}$ to the TDS.

Step 12: TDS runs Algorithm 5 and gets where

$$\begin{aligned}
\mathbf{I}_{i1} &= [\mathbf{x}_i]_{\mathrm{pk}_i}[y_i]_{\mathrm{pk}_i} = g^{2\mathrm{sk}_i r_i}\mathbf{x}_i y_i \bmod N\\
\mathbf{I}_{i2} &= [\mathbf{x}_i]_{\mathbf{pk}_i} s_i = g^{2\mathrm{sk}_i r_i + \mathrm{sk}_M r_M}\mathbf{x}_i \sum_{j=1}^{d}\omega_j x_{ij} \bmod N\\
\mathbf{I}_{i3} &= [\mathbf{x}_i]_{\mathrm{pk}_i}[y_i]_{\mathrm{pk}_i} s_i^2\\
&= g^{4\mathrm{sk}_i r_i + 2\mathrm{sk}_M r}\mathbf{x}_i y_i\left(\sum_{j=1}^{m}\omega_j x_{ij}\right)^2 \bmod N\\
I_{i4} &= [x_i]_{\mathrm{pk}_i} s_i^2 = g^{3\mathrm{sk}_i r_i + 2\mathrm{sk}_M r_M} x_i T^2\left(\sum_{j=1}^{m}\omega_j x_{ij}\right) \bmod N\\
\mathbf{I}_{i5} &= [\mathbf{x}_i]_{\mathrm{pk}_i} s_i^3\\
&= g^{4\mathrm{sk}_i r_i + 3\mathrm{sk}_M r_M}\mathbf{x}_i T^3\left(\sum_{j=1}^{m}\omega_j x_{ij}\right) \bmod N\\
\mathbf{I}_{i6} &= [\mathbf{x}_i]_{\mathrm{pk}_i}[y_i]_{\mathrm{pk}_i} s_i\\
&= g^{3\mathrm{sk}_i r_i + \mathrm{sk}_M r}\mathbf{x}_i y_i T\left(\sum_{j=1}^{m}\omega_j x_{ij}\right) \bmod N
\end{aligned} \tag{13}$$

Input: the ciphertext $[\mathrm{ms}]_{\mathrm{pk}_o}$ of the message ms with the original public key pk_o, the decryption circuit D_{Π_o} of the original encryption, and the target key pk_*
Output: the ciphertext $[\mathrm{ms}]_{\mathrm{pk}_*}$ of the message ms.
(1) compute: $[\mathrm{ms}]_{\mathrm{pk}_*} \leftarrow \mathrm{Evaluation}\ (\mathrm{pk}_b D_{\Pi_o}\ [\mathrm{ms}]_{\mathrm{pk}_o} [\mathrm{sk}_o]_{\mathrm{pk}_*})$
(2) **end**
(3) **return** $[\mathbf{ms}]_{\mathrm{pk}_*}$

Algorithm 6: Ciphertext conversion.

Input: the update information $\nabla\omega$, the coefficient vector ω, the learning rate η, and the attenuation rate γ
Output: the renew coefficient vector ω'
(1) compute $\omega' = \omega - \eta\nabla\omega$ or $\omega' = \gamma\omega - \eta\nabla\omega$
(2) **end**
(3) return ω'

Algorithm 7: Renewing the coefficient vector.

Initialization: DO's encrypted query feature vector $[q_i]_{\mathrm{pk}_i}$, the corresponding key pair $(\mathrm{pk}_i, \mathrm{sk}_i)$, MO's encrypted coefficient vector $[\omega]_{\mathrm{pk}_M}$, and the corresponding key pair $(\mathrm{pk}_M, \mathrm{sk}_M)$, where $q_i = (q_{i1}q_{i2} \ldots q_{im})$ and $\omega = (\omega_1\omega_2 \ldots \omega_m)$

Target: prediction result pr'
Step 1: DO and MO send $[q_i]_{\mathrm{pk}_i}$ and $[\omega]_{\mathrm{pk}_M}$ to the CS, respectively.
Step 2: CS computes pr, whereas $\mathrm{pr} = \sum_{j=1}^{m} [q_{ij}]_{\mathrm{pk}_i} [\omega_{ij}]_{\mathrm{pk}_M} = \sum_{j=1}^{m} g^{\mathrm{sk}_i r_i + \mathrm{sk}_M r_M} q_{ij} \cdot \omega_{ij} \bmod N$
Step 3: CS sends pr to the MO.
Step 4: MO runs Algorithm 5 and decrypts pr with its key pair $(\mathrm{pk}_M, \mathrm{sk}_M)$ and obtains $\overline{\mathrm{pr}} = \sum_{j=1}^{m} g^{\mathrm{sk}_i r_i} q_{ij}\omega_{ij} \bmod N$
Step 5: MO sends $\overline{\mathrm{pr}}$ to each DO.
Step 6: MO runs Algorithm 5 to decrypt $\overline{\mathrm{pr}}$ with its key pair $(\mathrm{pk}_M, \mathrm{sk}_M)$ and gets access to the desired prediction result: $\mathrm{pr}' = \sum_{j=1}^{m} q_{ij} \cdot \omega_{ij} \bmod N$

Algorithm 8: Subprotocol prediction.

and then TDS makes some simple computations: $k\mathbf{l}_1 = k\mathbf{I}_2^* - k\mathbf{I}_1^* = k(\mathbf{x}_i \sum_{j=1}^{m} \omega_j x_{ij} - \mathbf{x}_i y_i)$ in the linear regression model or $k\mathbf{l}_2 = k\mathbf{I}_3^* + k\mathbf{I}_4^* - k\mathbf{I}_5^* - k\mathbf{I}_6^* = k(\mathbf{x}_i \sum_{j=1}^{m} \omega_j x_{ij} - \mathbf{x}_i y_i)$ in the neural network model to get the final results $k\mathbf{l}_1$ or $k\mathbf{l}_2$

Step 13: TDS sends $k\mathbf{l}_1$ or $k\mathbf{l}_2$ to the MO.

5.1.3. Prediction Phase

In this phase, DO requests prediction with the help of the CS and MO.

Step 14: MO receives $k\mathbf{l}_1$ or $k\mathbf{l}_2$ and removes the security parameter k to obtain different $\nabla\omega$ of each sample $\mathbf{x}_i$

Step 15: then, the MO chooses one gradient descent method and then optimizes its coefficient vector through Algorithm 7

Step 16: each of the DO encrypts a query feature vector $[\mathbf{q}_i]_{\mathrm{pk}_i}$, and the MO encrypts its optimal coefficient vector $[\omega]_{\mathrm{pk}_M}$

Step 17: each of the DO and MO sends $[\mathbf{q}_i]_{\mathrm{pk}_i}$ and $[\omega]_{\mathrm{pk}_M}$ to the CS, respectively.

Step 18: finally, MO, CS, and DO operate together to help the DO to extract the prediction results by operating subprotocol prediction (Algorithm 8).

5.2. OPPGD Scheme over Vertically Partitioned Data. The OPPGD scheme over vertically partitioned data is a little different from the OPPGD scheme over horizontally partitioned data. After receiving $[\mathbf{x}_i]_{\mathrm{pk}_i}$, $[y_i]_{\mathrm{pk}_i}$, and $[\omega]_{\mathrm{pk}}$, CS executes Algorithm 4 n times in the first scheme, whereas CS operates Algorithm 4 nm times in the second scheme. This is because one record's m attributes are sent to the CS by its DO,

respectively. In addition, when the KCS receives the blinded encrypted results, it needs to add blinded encrypted results together m times to get the inner product of a record and the coefficient vector. For simplicity, we omit the same steps of the OPPGD scheme over vertically partitioned data as the steps of the OPPGD scheme over horizontally partitioned data.

5.3. Scheme Correctness. Now, we prove the correctness of our proposed OPPGD scheme over horizontally partitioned data. The correctness of the other scheme can be verified in a similar manner.

Theorem 1. *MO can correctly obtain $\nabla\omega$ to update its coefficient vector.*

Proof. After receiving $[\mathbf{x}_i]_{\text{pk}_i}$, $[y_i]_{\text{pk}_i}$, and $[\omega]_{\text{pk}}$, CS computes an encrypted scalar product $\mathbf{S}$, where $s_i = \sum_{j=1}^{m} g^{\text{sk}_i r_i + \text{sk}_M r} \omega_j x_{ij} \bmod N$. For linear regression, CS calculates $\mathbf{I}_1$ and $\mathbf{I}_2$, whereas for the neural network, CS calculates $\mathbf{I}_3$, $\mathbf{I}_4$, $\mathbf{I}_5$, and $\mathbf{I}_6$. After receiving the encrypted results from the CS, each DO decrypts the message sent from the CS and obtains $\mathbf{I}_1'$ and $\mathbf{I}_2'$ or $\mathbf{I}_3'$, $\mathbf{I}_4'$, $\mathbf{I}_5'$, and $\mathbf{I}_6'$ in linear regression or the neural network, respectively. Then, it blinds these encrypted results with k to be $k\mathbf{I}_1'$ and $k\mathbf{I}_2'$ or $k\mathbf{I}_3'$, $k\mathbf{I}_4'$, $k\mathbf{I}_5'$, and $k\mathbf{I}_6'$ and sends them to the KCS. Consequently, KCS converts the ciphertext into $k\mathbf{I}_1'$ and $k\mathbf{I}_2'$ or $k\mathbf{I}_3'$, $k\mathbf{I}_4'$, $k\mathbf{I}_5'$, and $k\mathbf{I}_6'$ under the key pk^* of the TDS. TDS decrypts the above intermediate results through Algorithm 5 to produce $k\mathbf{I}_1^*$ and $k\mathbf{I}_2^*$ or $k\mathbf{I}_3^*$, $k\mathbf{I}_4^*$, $k\mathbf{I}_5^*$, and $k\mathbf{I}_6^*$. Then, it computes $k\mathbf{I}_2^* - k\mathbf{I}_1^*$ or $k\mathbf{l}_2 = k\mathbf{I}_3^* + k\mathbf{I}_4^* - k\mathbf{I}_5^* - k\mathbf{I}_6^*$ and generates the final results $k\mathbf{l}_1$ or $k\mathbf{l}_2$ for linear regression or the neural network. Ultimately, after the MO receives them, he removes the security parameter k and obtains $\nabla\omega = \mathbf{x}_i \sum_{j=1}^{m} \omega_j x_{ij} - \mathbf{x}_i y_i$ in linear regression or $\nabla\omega = (12)\alpha(\mathbf{x}_i\ y_i T^2(\sum_{j=1}^{m} \omega_j x_{ij}) + \mathbf{x}_i T^2(\sum_{j=1}^{m} \omega_j x_{ij}) - \mathbf{x}_i T^3(\sum_{j=1}^{m} \omega_j x_{ij}) - \mathbf{x}_i y_i T(\sum_{j=1}^{m} \omega_j x_{ij}))$ in the neural network which are equal to equation (3) or equation (5), respectively. Then, MO can achieve accurate $\nabla\omega$ □

6. Privacy and Complexity Analysis

We will analyze the privacy, computational cost, and communication overhead of the OPPGD scheme over horizontally partitioned data. We can perform analysis of the OPPGD scheme over vertically partitioned data in terms of the privacy, computational cost, and communication overhead in almost the same way. For simplicity, we omit the latter.

6.1. Privacy Analysis. According to the definitions of two different privacy levels in Section 4.4, we conduct the privacy analysis of our proposed OPPGD scheme over horizontally partitioned data.

Proof. Upon the hardness assumption of the Diffie–Hellman problem, our proposed OPPGD schemes achieve level-1 privacy against any probabilistic polynomial-time adversary. □

Proof. Now, we show that our scheme can preserve MO's model privacy and DO's data privacy.

In Step 3 of Algorithm 3, MO and DO hide their input via Elgamal encryption. After receiving $[\mathbf{x}_i]_{\text{pk}_i}$, $[y_i]_{\text{pk}_i}$, and $[\omega]_{\text{pk}}$, the CS runs Algorithm 4 and obtains the encrypted scalar product $\mathbf{S}$. Especially, MO's and every DO's encrypted input are $g^{\text{sk}_M r_M}\omega \bmod N$ and $\{g^{\text{sk}_i r_i}\mathbf{x}_i \bmod N\, g^{\text{sk}_i r_i} y_i \bmod N\}_{i=1}^{n}$. Upon the hardness assumption of the Diffie–Hellman problem, although CS knows MO and DO's public keys $g^{\text{sk}_i} \bmod N$ and $g^{\text{sk}_r} \bmod N$, it is still impossible for them to acquire their secret keys sk_i and sk. Since the randomness r_i and r are chosen by DO and MO, respectively, any adversary who attempts to solve $\{g^{\text{sk}_i r_i} y_i \bmod N,\ g^{\text{sk}_M r_M}\omega \bmod N\}$ from the public keys $\{g^{r_i} \bmod N,\ g^{r_M} \bmod N\}$ will have to be faced with two instances of Diffie–Hellman problems. Thus, DO's $\mathbf{x}_i$ and y_i and MO's ω will not be exposed to other parties. When the KCS performs Algorithm 6 to convert the encrypted results $\{\mathbf{I}_1'\mathbf{I}_2', \mathbf{I}_3'\mathbf{I}_4', \mathbf{I}_5', \mathbf{I}_6'\}$, it receives MO and DO's secret keys encrypted under the TDS's public key. However, TDS is a trusty decryption server, so KCS cannot obtain TDS's secret key, which means KCS knows nothing about MO and DO's secret keys and their private value. So, the encrypted results $\{k\mathbf{I}_1'k\mathbf{I}_2'k\mathbf{I}_3'k\mathbf{I}_4'k\mathbf{I}_5'k\mathbf{I}_6'\}$ cannot leak any secret information. Next, TDS runs Algorithm 5 and obtains encrypted $\nabla\omega$. However, without the secret value k, TDS cannot obtain $\nabla\omega$. Hence, MO's model parameters will not be exposed.

Since MO's coefficient vector, gradient $\nabla\omega$, and DO's data will not face the privacy problem, our OPPGD schemes can provide level-1 privacy. □

Theorem 3. *Upon the hardness assumption of knapsack problems, our OPPGD schemes can provide level-2 privacy against any probabilistic polynomial-time adversary.*

Proof. After receiving the encrypted results from the CS, DOs run Algorithm 5 to generate new encrypted results under MO's key. For linear regression, DO knows $\{\mathbf{I}_2', g^{\text{sk}_r} \bmod N, \mathbf{x}_i y_i\}$. For neural networks, DO knows $\{\mathbf{I}_3'\mathbf{I}_4', \mathbf{I}_5', \mathbf{I}_6', g^{\text{sk}_r} \bmod N, \mathbf{x}_i y_i\}$. However, with the knowledge of the information, it is still impossible to acquire ω. This is because that the knapsack problem is assumed to be difficult: given a scalar product z and a vector $\mathbf{a}$, it is hard to find vector $\mathbf{b}$ that satisfies $z = \mathbf{ab}$

Consequently, MO's coefficient vector and gradient results $\nabla\omega$ cannot be deduced from the intermediate results all over the scheme.

Therefore, we conclude that our schemes can achieve level-2 privacy. □

6.2. Theoretical Efficiency Analysis. Now, we carry out the theoretical efficiency analysis of the schemes. We consider the situation for linear regression. Assume that the MO chooses the SGD method to update its coefficient vector within one epoch. In essence, MO optimizes its coefficients within several epochs. In the following, we analyze the feasibility of our proposed schemes in detail in terms of

Table 1: Complexity cost of the proposed OPPGD scheme.

			Phase			
			Preparation	Training	Prediction	Total
Computation cost	DO	Multiplications	$\mathcal{O}(\mathrm{mn}+n)$	$\mathcal{O}(3\mathrm{mn})$	$\mathcal{O}(2m)$	$\mathcal{O}(4\mathrm{mn}+n+2m)$
	MO	Multiplications	$\mathcal{O}(m)$	$\mathcal{O}(\mathrm{mn})$	$\mathcal{O}(m+1)$	$\mathcal{O}(2m+\mathrm{mn}+1)$
		Additions		$\mathcal{O}(m)$		$\mathcal{O}(m)$
	CS	Multiplications		$\mathcal{O}(3\mathrm{mn})$	$\mathcal{O}(m+1)$	$\mathcal{O}(3\mathrm{mn}+m+1))$
		Additions		$\mathcal{O}(\mathrm{mn}-n)$	$\mathcal{O}(m-1)$	$\mathcal{O}(\mathrm{mn}+m+n-1)$
	KCS	Multiplications		$\mathcal{O}(2\mathrm{mn})$		$\mathcal{O}(2\mathrm{mn})$
	TDS	Multiplications		$\mathcal{O}(2\mathrm{mn})$		$\mathcal{O}(2\mathrm{mn})$
		Additions		$\mathcal{O}(\mathrm{mn})$		$\mathcal{O}(\mathrm{mn})$
Communication overhead	Total round		$\mathcal{O}(n+1)$	$\mathcal{O}(2n+2)$	$\mathcal{O}(4)$	$\mathcal{O}(3n+7)$
	Total bits to transmit		$\mathcal{O}((\mathrm{mn}+2m)\lvert N\rvert)$	$\mathcal{O}(6\mathrm{nm}\lvert N\rvert)$	$\mathcal{O}(4\mathrm{nm}\lvert N\rvert)$	$\mathcal{O}((11\mathrm{nm}+2m)\lvert N\rvert)$

Table 2: Running time of the OPPGD scheme with the dataset of n tuples.

Tuples	KeyGen (ms)	Encryption (ms)	Training (ms)	Prediction (ms)
1000	279.47	293.07	31146.57	0.06
2000	559.03	585.97	31338.61	0.06
3000	838.42	878.63	31507.91	0.06
4000	1117.97	1172.11	31677.20	0.06
5000	1397.61	1464.20	31846.51	0.06
6000	1677.09	1756.98	32015.80	0.06

computational cost and communication overhead. Both computational cost and communication overhead are shown in Table 1

6.2.1. Computational Cost. Assume that the dataset contains n records, each of which has m attributes, and one class value in the OPPGD scheme over horizontally partitioned data. In Step 3, DOs and MO operate Algorithm 3 $\mathcal{O}(\mathrm{mn}+n)$ and $\mathcal{O}(m)$ times, respectively. Thus, $\mathcal{O}(\mathrm{mn}+n+m)$ multiplications are required. In Step 5, CS performs OPPSP to calculate encrypted scalar product $\mathbf{S}$. It requires 3mn multiplications and $\mathcal{O}(n(m-1))$ additions. In Step 7, DO performs nm decryptions to generate the encrypted results which are under the MO's key. In Step 8, DO needs $\mathcal{O}(2\mathrm{mn})$ multiplications to blind encrypted results with the security parameter K. In Step 10, KCS performs $\mathcal{O}(2\mathrm{mn})$ multiplications to convert the encrypted results into new results. In Step 12, TDS performs $\mathcal{O}(2\mathrm{mn})$ decryptions and makes mn subtractions to obtain the final results. In Step 14, MO performs $\mathcal{O}(\mathrm{mn})$ multiplications and obtains $\nabla\omega$. MO operates SGD to update its coefficient vector by executing $\mathcal{O}(m)$ times of multiplications and additions. In Step 16, both DO and MO perform $\mathcal{O}(m)$ encryption operations to encrypt their query and the optimal coefficient vector. In Step 18, in order to generate prediction results, CS performs $\mathcal{O}(\mathrm{mn})$ multiplication and $\mathcal{O}(m-1)$ additions, while both DO and MO perform one encryption operation, respectively.

6.2.2. Communication Overhead. Next, we analyze the communication complexity of each entity in our proposed schemes. In Step 4, DO and MO communicate with CS n rounds and one round, respectively. It takes

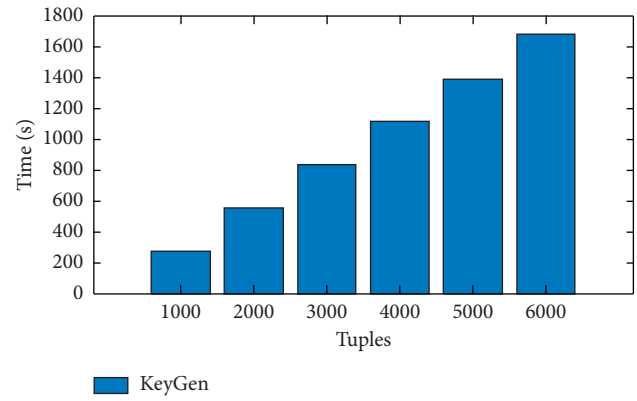

Figure 4: The running time in the KeyGen algorithm.

$\mathcal{O}((2\mathrm{nm}+2m)\lvert N\rvert)$ bits to transmit. In Step 6, since CS sends the encrypted results to DOs, its communication overhead required is $\mathcal{O}(n)$. So, it requires $\mathcal{O}(2\mathrm{nm}\lvert N\rvert)$ bits. Moreover, in Step 9, when each DO sends blinded encryption results to the KCS, the communication overhead is $\mathcal{O}(n)$. Thus, $\mathcal{O}(2\mathrm{nm}\lvert N\rvert)$ bits are required to be transmitted. In Step 11, KCS sends new intermediate results to TDS via $\mathcal{O}(1)$ round with $\mathcal{O}(\mathrm{nm}\lvert N\rvert)$ bits. In Step 13, the communication overhead between TDS and MO is $\mathcal{O}(1)$. It costs $\mathcal{O}(\mathrm{nm}\lvert N\rvert)$ bits to transmit. In Step 17, DO and MO send the encrypted feature vectors to CS, respectively, with the communication overhead of $\mathcal{O}(2)$ which costs $\mathcal{O}(2\mathrm{nm}\lvert N\rvert)$ to transmit. In Step 18, $\mathcal{O}(2)$ communication cost is required for the DO to obtain its desired prediction results, while $\mathcal{O}(2\mathrm{nm}\lvert N\rvert)$ bits are transmitted. Hence, the communication cost of the scheme is $\mathcal{O}(3n+7)$ in total.

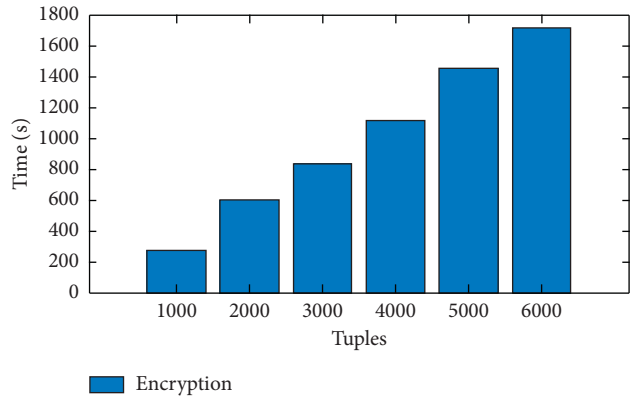

Figure 5: The running time in the encryption algorithm.

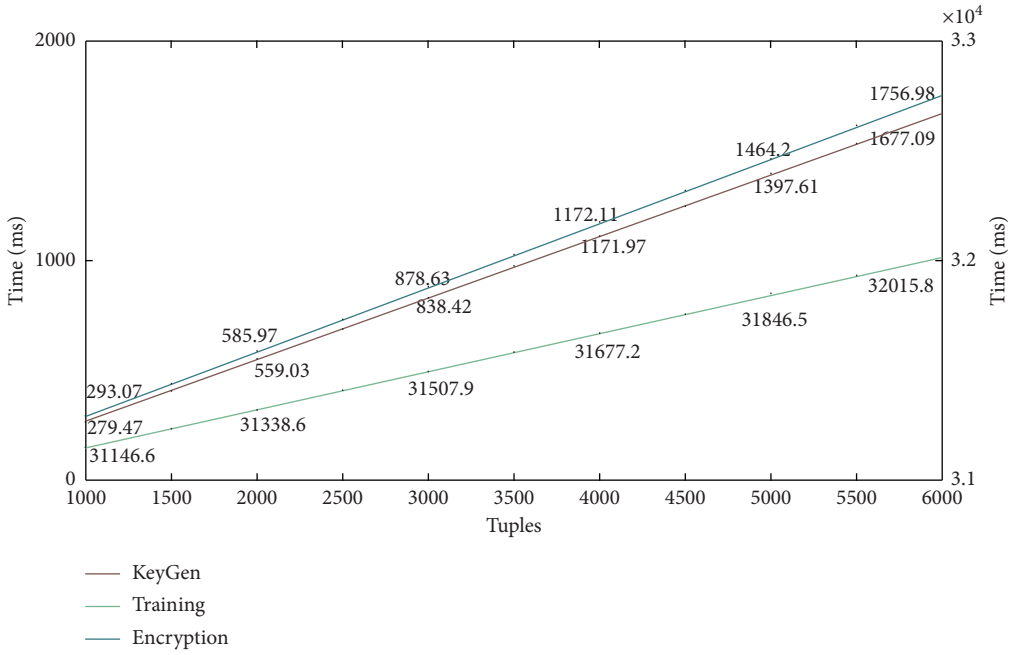

Figure 6: The running time in the KeyGen algorithm, the encryption algorithm, and the training phase.

Table 3: The running time of the key generation algorithm and the evaluation algorithm based on the key dimension.

Key bit-length	KeyGen (s)	Evaluation (s)	Training (s)
2048 bits	40	31	31.8
8192 bits	480	180	180.8

7. Performance Evaluation

In this section, we evaluate the efficiency of the OPPGD scheme over horizontally partitioned data by using a custom simulator built in JAVA. The running time of the OPPGD scheme over vertically partitioned data can be evaluated in a similar way. The scenario we focus on in our paper is the data are partitioned among multiple data owners, and the target function is owned by the model owner. The model owner can not only train its model over data owner's data but also provide users with predictions. To the best of our knowledge, no other prior work in the literature discusses this scenario. So, we present detailed performance evaluation of our schemes rather than comparing them to previous works.

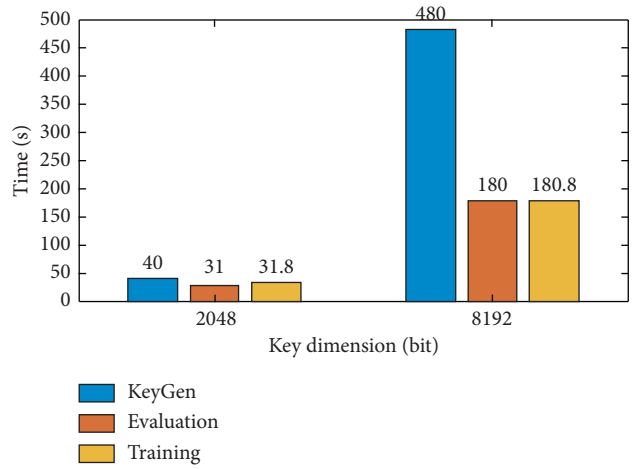

Figure 7: The running time of the key generation algorithm and the evaluation algorithm based on the key dimension.

Table 4: The total time cost of each entity in the OPPGD scheme.

Tuples	DO (ms)	MO (ms)	CS (ms)	KCS (ms)	TDS (ms)
1000	52.48	13.09	41.95	26.09	28.43
2000	104.93	26.13	83.89	52.19	57.67
3000	157.38	39.18	125.66	78.28	86.5
4000	209.83	52.23	167.76	104.38	115.34
5000	262.28	65.27	209.43	130.47	144.17
6000	314.73	78.33	251.31	156.57	173.01

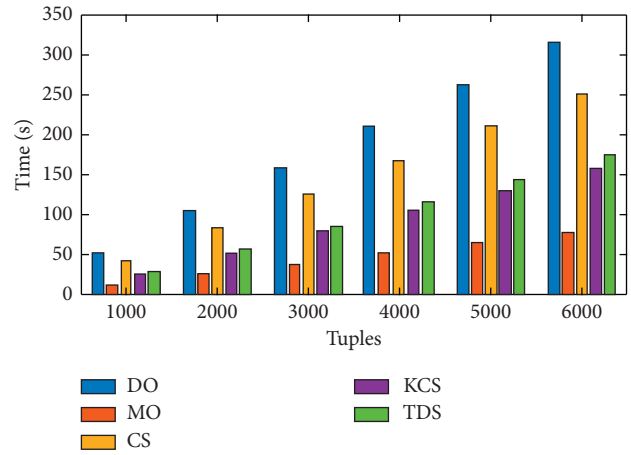

Figure 8: The running time of each entity in the OPPGD scheme with different key bit-lengths.

There are five entities in the scheme: the model owner MO, the data owner DO, the cloud server CS, the key conversion server KCS, and the trusty decryption server TDS.

We run the data owners DOs and the model owner MO on a laptop with Intel Xeon(R) E5-1620 3.50 GHZ CPU processor and 16 GB RAM memory. The cloud server CS, the key conversion server KCS, and the trusty decryption server TDS sides are operated on a computer with Intel(R) Core (TM) i7-4770 3.40 GHz CPU processor and16 GB RAM memory.

In our experiments, DO's data X are represented as one $n * m$ matrix, where n ranges from 1000 to 6000 and $m = 50$. We evaluate the computational efficiency of our OPPGD schemes without considering communication latency. We simulate four stages: the KeyGen algorithm, the encryption algorithm, the training phase, and the prediction phase. As the data size n changes, the corresponding time cost is also different. When the key bit-length is 2048 bits, the running time of each stage of the schemes with the number of data tuples can be seen from Table 2. The calculation of the OPPGD scheme is mainly in the training stage, while the calculation cost of the rest stages is very low. We use the histogram to explicitly present the running time in the KeyGen algorithm and the encryption algorithm in Figures 4

and 5. The running time in the KeyGen algorithm, the encryption algorithm, and the training phase is shown in Figure 6. In addition, when the data dimension is 6000, the running time mainly verified in the KeyGen algorithm, the evaluation algorithm, and the training phase based on various key bit-lengths is different. So, we simulate these stages and the running time, as shown in Table 3 and Figure 7. When the key bit-length is 2048 bits, the total running time of each entity in our OPPGD scheme is shown in our Table 4. According to the variation of the tuples or key bit-lengths, the running time of each party is shown in Figure 8

8. Conclusion

Massive work on the protection of sensitive data of IoT devices is based on the secure communication channels and authorization. In our paper, we focus on the protection of data which are collected by the IoT devices, stored, and calculated on the cloud end and the privacy of the machine learning model which is held by the MO. Gradient descent methods are employed comprehensively to train a machine learning model in the cloud computing environment. In order to preserve data privacy and model privacy during the cloud computing, we propose two secure schemes to perform outsourced privacy-preserving gradient descent methods over a horizontally or vertically distributed dataset. The proposed schemes enable the model owner (MO) to train its learning model and obtain the optimal coefficient vector based on the dataset owned by the DO with the help of CS, TDS, and KCS. After the MO improves its model, it can offer prediction service to the DO. Both the privacy of the MO's model and DO's dataset can be protected. Complexity and performance evaluation are also given in detail. In the future work, we will try to optimize our system to reduce the number of entities.

Data Availability

The data used to support the findings of this study are available from the corresponding author upon request.

Conflicts of Interest

The authors declare that they have no conflicts of interest.

Acknowledgments

This work was partially supported by the Ministry of Science and Technology of the People's Republic of China (Grant no. 2018YFB0803505), the National Natural Science Foundation of China (Grant nos. 61862028 and 61702238), the Natural Science Foundation of Jiangxi Province (Grant no. 20181BAB202016), and the Science and Technology Project of Provincial Education Department of Jiangxi (GJJ160430 and GJJ180288).

References

[1] L. Xiao, X. Wan, X. Lu, Y. Zhang, and D. Wu, "IoT security techniques based on machine learning: How do IoT devices use AI to enhance security?" *IEEE Signal Processing Magazine*, vol. 35, no. 5, pp. 41–49, 2018.

[2] Z. Yan, P. Zhang, and A. V. Vasilakos, "A survey on trust management for internet of things," *Journal of Network and Computer Applications*, vol. 42, pp. 120–134, 2014.

[3] I. Andrea, C. Chrysostomou, and G. C. Hadjichristofi, "Internet of things: security vulnerabilities and challenges," 2015.

[4] N. Kshetri, "Can blockchain strengthen the internet of things?" *IT Professional*, vol. 19, no. 4, pp. 68–72, 2017.

[5] M. T. Thai, "A disruptive integration," *Computer*, vol. 51, pp. 48–53, 2018.

[6] L. Chen, S. Thombre, K. Järvinen et al., "Robustness, Security and privacy in location-based services for future IoT: a Survey," *IEEE Access*, vol. 5, pp. 8956–8977, 2017.

[7] I. Yaqoob, I. A. T. Hashem, A. Ahmed, S. M. A. Kazmi, and C. S. Hong, "Internet of things forensics: Recent advances, taxonomy, requirements, and open challenges," *Future Generation Computer Systems*, vol. 92, pp. 265–275, 2019.

[8] J. Daubert, A. Wiesmaier, and P. Kikiras, "A view on privacy & trust in IoT," 2015.

[9] Z. Zhang, M. C. Y. Cho, C. Wang, C. Hsu, C. Chen, and S. Shieh, "IoT security: ongoing challenges and research opportunities," 2014.

[10] P. P. Jayaraman, X. Yang, A. Yavari, D. Georgakopoulos, and X. Yi, "Privacy preserving Internet of Things: From privacy techniques to a blueprint architecture and efficient implementation," *Future Generation Computer Systems*, vol. 76, pp. 540–549, 2017.

[11] T. Elgamal, "A public key cryptosystem and a signature scheme based on discrete logarithms," *IEEE Transactions on Information Theory*, vol. 31, no. 4, pp. 469–472, 1985.

[12] L. Wan, W. K. Ng, S. Han, and V. C. S. Lee, "Privacy-preservation for gradient descent methods," 2007.

[13] S. Han, W. K. Ng, L. Wan, and V. C. S. Lee, "Privacy-preserving gradient-descent methods," *IEEE Transactions on Knowledge and Data Engineering*, vol. 22, no. 6, pp. 884–899, 2010.

[14] G. Danner and M. Jelasity, "Fully Distributed Privacy preserving mini-batch gradient descent learning," *IFIP International Conference on Distributed Applications and Interoperable Systems*, vol. 9038, pp. 30–44, 2015.

[15] I. Hegedűs and M. Jelasity, "Distributed differentially private stochastic gradient descent: an empirical study," 2016.

[16] S. Mehnaz, G. Bellala, and E. Bertino, "A secure sum protocol and its application to privacy-preserving multi-party analytics," 2017.

[17] S. Mehnaz and E. Bertino, "Privacy-preserving multi-party analytics over arbitrarily partitioned Data," 2017.

[18] X. Wu, F. Li, A. Kumar, K. Chaudhuri, S. Jha, and J. Naughton, "Bolt-on differential privacy for scalable stochastic gradient descent-based analytics," 2017.

[19] F. Liu, W. K. Ng, and W. Zhang, "Encrypted gradient descent protocol for outsourced data mining," 2015.

[20] R. Shokri and V. Shmatikov, "Privacy-preserving deep learning," 2015.

[21] M. Kim, Y. Song, S. Wang, Y. Xia, and X. Jiang, "Secure logistic regression based on homomorphic encryption: Design and Evaluation," *JMIR Medical Informatics*, vol. 6, no. 2, p. 19, 2018.

[22] F. Gonzalez-Serrano, A. Amor-Martin, and J. Casamayón-Antón, "Supervised machine learning using encrypted training data," *International Journal of Information Security*, vol. 7, no. 2, pp. 365–377, 2017.

[23] P. Mohassel and Y. Zhang, "SecureML: a system for scalable privacy-preserving machine learning," 2017.

[24] P. Li, J. Li, Z. Huang et al., "Multi-key privacy-preserving deep learning in cloud computing," *Future Generation Computer Systems*, vol. 74, pp. 76–85, 2017.

[25] X. Ma, X. Chen, and X. Zhang, "Non-interactive privacy-preserving neural network prediction," *Information Sciences*, vol. 481, pp. 507–519, 2019.

[26] X. Liu, B. Qin, R. H. Deng, and Y. Li, "An efficient privacy-preserving outsourced computation over public data," *IEEE Transactions on Services Computing*, vol. 10, no. 5, pp. 756–770, 2017.

[27] X. Liu, R. Deng, K. R. Choo, Y. Yang, and H. Pang, "Privacy-Preserving outsourced calculation toolkit in the cloud," *IEEE Transactions on Dependable and Secure Computing*, vol. 2018, p. 1, 2018.

[28] M. Rady, T. Abdelkader, and R. Ismail, "Integrity and confidentiality in cloud outsourced data," *Ain Shams Engineering Journal*, vol. 10, no. 2, pp. 275–285, 2019.

[29] X. Yu, R. Zhang, and Z. Rui, "Verifiable outsourced computation over encrypted data," *Information Sciences*, vol. 479, pp. 372–385, 2019.

[30] M. A. P. Chamikara, P. Bertok, D. Liu, S. Camtepe, and I. Khalil, "Efficient privacy preservation of big data for accurate data mining," *Information Sciences*, vol. 2019, 2019.

[31] P. Li, J. Li, Z. Huang, C. Gao, W. Chen, and K. Chen, "Privacy-preserving outsourced classification in cloud computing," *Cluster Computing*, vol. 21, no. 1, pp. 277–286, 2018.

[32] T. Li, Z. Huang, P. Li, Z. Liu, and C. Jia, "Outsourced privacy-preserving classification service over encrypted data," *Journal of Network and Computer Applications*, vol. 106, pp. 100–110, 2018.

[33] H. Park, P. Kim, H. Kim, K.-W. Park, and Y. Lee, "Efficient machine learning over encrypted data with non-interactive communication," *Computer Standards & Interfaces*, vol. 58, pp. 87–108, 2018.

[34] Y. Li, Z. L. Jiang, L. Yao, X. Wang, S. M. Yiu, and Z. Huang, "Outsourced privacy-preserving C4.5 decision tree algorithm over horizontally and vertically partitioned dataset among multiple parties," *Cluster Computing*, vol. 22, pp. 1581–1593, 2019.

[35] H. Rong, H. Wang, J. Liu, F. Tang, and M. Xian, "Verifiable and privacy-preserving association rule mining in hybrid cloud environment," 2019.

[36] Y. Li, Z. L. Jiang, X. Wang, and S. M. Yiu, "Privacy-preserving ID3 data mining over encrypted data in outsourced environments with multiple keys," *IEEE International Conference on Computational Science and Engineering (CSE) and IEEE International Conference on Embedded and Ubiquitous Computing (EUC)*, vol. 1, pp. 548–555, 2017.

[37] W. Xue, Y. Shen, C. Luo, W. Hu, and A. Seneviratne, "A privacy-preserving system for edge-based classification," 2018.

[38] J.-J. Yang, J.-Q. Li, and Y. Niu, "A hybrid solution for privacy preserving medical data sharing in the cloud environment," *Future Generation Computer Systems*, vol. 43, pp. 74–86, 2015.

[39] H. Kaur, N. Kumar, and S. Batra, "An efficient multi-party scheme for privacy preserving collaborative filtering for healthcare recommender systemficient Multi-party Scheme for Privacy Preserving Collaborative Filtering for Healthcare Recommender System," *Future Generation Computer Systems*, vol. 86, pp. 297–307, 2018.

[40] R. Gennaro, C. Gentry, B. Parno et al., "Non-interactive verifiable computing: outsourcing computation to untrusted workers," in *Proceedings of the 30th Annual Cryptology Conference Advances in Cryptology-CRYPTO 2010*, Lecture Notes in Computer Science, vol. 6223, pp. 465–482, Springer, Berlin, Germany, 2010, https://www.iacr.org/archive/crypto2010/62230459/62230459.pdf.