Hindawi
Security and Communication Networks
Volume 2022, Article ID 1321198, 17 pages
https://doi.org/10.1155/2022/1321198

Research Article

Privacy-Preserving Outsourced Logistic Regression on Encrypted Data from Homomorphic Encryption

Xiaopeng Yu,[1] **Wei Zhao**,[1] **Yunfan Huang**,[1] **Juan Ren**,[1] **and Dianhua Tang**[1,2]

[1]*Science and Technology on Communication Security Laboratory, Chengdu 610041, China*
[2]*School of Computer Science and Engineering, University of Electronic Science and Technology of China, Chengdu 611731, China*

Correspondence should be addressed to Dianhua Tang; tangdianhua86@163.com

Received 23 March 2022; Revised 11 May 2022; Accepted 24 May 2022; Published 21 July 2022

Academic Editor: Debiao He

Logistic regression is a data statistical technique, which is used to predict the probability that an event occurs. For some scenarios where the storage capabilities and computing resources of the data owner are limited, the data owner wants to train the logistic regression model on the cloud service provider, while the high sensitivity of training data requires effective privacy protection methods that enable efficient model training without exposing information about the training data to untrusted cloud service providers. Recently, several works have used cryptographic techniques to implement privacy-preserving logistic regression in such application scenarios. However, on large-scale training datasets, the existing works still have the problems of long model training time and poor model performance. To solve these problems, based on the homomorphic encryption (HE), we propose an efficient privacy-preserving outsourced logistic regression (P^2OLR) on encrypted training data, which enables data owners to utilize the powerful storage and computing resources of cloud service providers for logistic regression analysis without exposing data privacy. Furthermore, the proposed scheme can pack multiple messages into one ciphertext and perform the same arithmetic evaluations on multiple plaintext slots by using the batching technique and single instruction multiple data (SIMD) mechanism in HE. On three public training datasets, the experimental results show that, compared with the existing schemes, the proposed scheme has better performance in terms of the encryption and decryption time of the data owner, the storage of encrypted training data, and the training time and accuracy of the model.

1. Introduction

Logistic regression (LR) [1] is a popular classification method, which has been used in numerous practical applications including cancer diagnosis [2], credit scoring [3], genome-wide association study [4], and more. LR can not only be applied to the problem of predicting the probability of occurrence of various events, but also is competitive with other classification algorithms in terms of prediction accuracy. In some practical application setting, the data owners have the limited computing and storage resources, and thus wants to outsource some of the heavy computation in logistic regression model training, the outsourced data analysis [5] has received considerable attention recently, which enables data owners to train a LR model using the powerful storage capacity and computing resources of cloud service providers [6].

However, the high sensitivity of training data requires to perform an effective privacy protection [7–10] that enable efficient and secure logistic regression analysis without leaking information about the training data to untrusted cloud service provider. Recently, to meet such application requirements, based on the cryptographic techniques like secure multiparty computation (MPC) [11] and homomorphic encryption (HE) [12], there have been several researches on the privacy-preserving logistic regression (PPLR) [13–22], which enables data owners to employ the service providers' powerful data storage and computing resources for logistic regression model training without exposing its own data privacy. Specifically, the data owner encrypts its training data, and sends encrypted training data to the service provider. The service provider can train a logistic regression model on encrypted training data, and returns the encrypted training result to the data owner. The data owner can decrypt the encrypted training result to obtain final training result.

Unfortunately, on large-scale training dataset, the existing PPLR schemes [13–22] still have the bottlenecks of high model training time and low model precision. To solve these problems, based on the HE cryptographic technique [23] that has the property that the operation results on ciphertexts are consistent with those on plaintexts, we design an efficient privacy-preserving outsourced logistic regression (P^2OLR). The main contributions are as follows:

(1) Firstly, we propose a method for achieving P^2OLR on encrypted data from HE. To speed up the model training, the proposed P^2OLR scheme employs the batching technique to pack multiple elements into multiple plaintext slots, encrypts them into one ciphertext, and performs the same arithmetic operations to multiple plaintext slots in the SIMD mechanism.

(2) Secondly, we evaluate the proposed P^2OLR on three public datasets [18]. Under the same experimental environment, compared with the related P^2OLR [17, 18, 22], the model training time of the proposed P^2OLR is reduced by more than 71.7%, and the proposed P^2OLR has a better model performance.

The rest of this paper is arranged as follows. We present the related works in Section 2. We review the preliminaries related to our P^2OLR in Section 3. In Section 4, ourP^2OLR is described. The performance evaluation for our P^2OLR is presented in Section 5. The security analysis of our P^2OLR is shown in Section 6. Finally, we conclude in Section 7.

2. Related Works

There have been a lot of works on achieving PPLR using cryptographic techniques. In this paper, we mainly focus on the PPLR based on HE. To outsource the LR model training to a cloud service provider in a privacy-preserving manner, based on the HE scheme (FV) [24], Charlotte et al. [13] proposed an algorithm to train a LR model on an homomorphically encrypted dataset, which is implemented based on the FV-NFLlib library [25]. However, the accuracy of model is poor due to the use of a quadratic polynomial to approximate the sigmoid function. Furthermore, the training time grows linearly in the number of training samples. Using the HE scheme (FV) [24] and 1 bit gradient descent (GD) method, Chen et al. [14] presented a method to train LR over encrypted data, which is implemented through the SEAL library [26], and allows an arbitrary number of iterations by using bootstrapping [27] in FV, but bootstrapping introduces a significant decrease in performance. Focusing on the prediction process of LR, based on the HE scheme (BGV) [28], Li and Sun [15] proposed a secure protocol to solve the data leakage problem during the LR prediction process, and implement their scheme by the HElib library [29]. Based on the Chimera framework [30] that allows switching between HE schemes TFHE [31] and CKKS [23], Carpov et al. [16] proposed a solution to achieve semi-parallel LR on encrypted genomic data, which performs the bootstrapping [27] without re-encrypting the genomic data for an arbitrary number of iterations, and is implemented by using TFHE library [32] and HEAAN library [33].

Adapting the packing and parallelization techniques of approximate HE scheme (CKKS) [23], Kim et al. [17] proposed a PPLR, which is implemented through using the HEAAN library [33], and uses least squares approximation to improve the accuracy and efficiency of LR model training. However, as the number of iterations increases, the parameters of the CKKS scheme also need to become larger, which makes the training time increase dramatically. Kim et al. [18] applied the HE scheme (CKKS) [23] to achieve PPLR. Their scheme is implemented via using the HEAAN library [33]. Moreover, they devised an encoding method to decrease the storage of encrypted training data and adapted Nesterov's accelerated GD method to reduce the number of iterations as well as the computational cost. However, their scheme requires the assumption that both the number of training samples and features are power-of-two, which makes the scheme unsuitable for practical applications. To reduce the number of iterations, Cheon et al. [19] proposed an ensemble GD method based on the HE scheme (CKKS) [23], and applied it to the PPLR, in which they approximate the sigmoid function using a polynomial of 5-degree obtained by least squares approximation. Their scheme is implemented based on the HEAAN library [34]. To run a genome-wide association study on encrypted data, using the SIMD capabilities of HE scheme (CKKS) and Nesterov's accelerated GD, Bergamaschi et al. [20] introduced a method for homomorphic training of LR model, which is implemented based on the HElib library [29]. To protect the private information of both parties, based on the HE scheme (CKKS) [23] and gradient sharing technology, Wei et al. [21] proposed a protocol to train an LR model on vertically distributed data between two parties, which does not require trusted third-party nodes and is implemented by the HElib library [29]. Based on the HE scheme (CKKS) [23], Fan et al. [22] offered a PPLR algorithm, where they approximate the sigmoid function in LR by Taylor's theorem, and use row encoding to encrypt training samples, but as the number of samples increased, this will lead to longer model training time.

3. Preliminaries

3.1. System Model. As can be seen in Figure 1, the system model of the proposed P^2OLR considers two entities, namely a data owner (DO) and a service provider (SP). For readability, the definitions of the notations in this paper are shown in Table 1. DO: It has limited computational resources, and wants to use SP's data analysis service on encrypted data to train a LR model without revealing its own training data privacy. SP: It is a semi-trusted entity with powerful data storage and computing capabilities, and can provide data analysis and statistical services on encrypted data for DO. Specifically, DO chooses poly_modulus_degree N, coeff_modulus Q, and runs key_generation algorithm to generate the secret_key sk, public_key pk, relinearization_key rk, galois_key gk. Next, DO encrypts the training data $\mathrm{D} \in \mathbb{R}^{m\times n}$ into ciphertexts D, encrypts the initial weight $\{w_0^{(0)}, w_1^{(0)}, \cdots, w_{n-1}^{(0)}\}$ into ciphertexts $W^{(0)}$, encrypts the learning rate α into one ciphertext $\alpha/\mathbf{m}$, and sends N, Q, Δ, pk, rk, gk, t, D, $W^{(0)}$, α/m to SP. SP performs the P^{2}OLR algorithm and returns the ciphertext result $W^{(t)}$ of the t -th iteration to DO. DO decrypts the ciphertext result $W^{(t)}$ to obtain final result $\{w_0^{(t)}, w_1^{(t)}, \cdots, w_{n-1}^{(t)}\}$.

3.2. Homomorphic Encryption. Homomorphic encryption (HE) is a cryptographic technique, which allows operations on ciphertexts without decryption, and guarantees that the computation results on ciphertexts are consistent with the computation results on plaintexts. We adopt the HE scheme (CKKS) [23] based on the Ring Learning with Errors (RLWE) problem, which can encrypt multiple elements in one ciphertext and supports the single instruction multiple data (SIMD) operations. Suppose $\Phi_M(X) = X^N + 1$ denotes the M -th cyclotomic polynomial, where N is power of 2. $\mathscr{R} = \mathbb{Z}[X]/(X^N + 1)$ denotes the cyclotomic ring of polynomials. $\mathscr{R}_q = \mathscr{R}/q\mathscr{R} = \mathbb{Z}_q[X]/(X^N + 1)$ denotes the residue ring of $\mathscr{R}$ modulo q. $\mathbb{H}$ denotes a subring of complex vector $\mathbb{C}^N$ that is isomorphic to $\mathbb{C}^{N/2}$. σ: $\mathscr{R} \longrightarrow \sigma(\mathscr{R}) \subseteq \mathbb{H}$ denotes a canonical embedding that transforms a plaintext polynomial $\mathscr{R}$ into a complex vector $\mathbb{H}$. π: $\mathbb{H} \longrightarrow \mathbb{C}^{N/2}$ denotes a natural projection that transforms a complex vector $\mathbb{C}^N$ to $\mathbb{C}^{N/2}$. HE scheme (CKKS) [23] supports the operations as follows, which can be found in the Appendix. For ease of description, we define the Algorithms 1–9.

3.3. Sigmoid Approximation. Since the existing HE scheme can only effectively support polynomial arithmetic computations, the computation of sigmoid function $\sigma(x) = 1/(1 + e^{-x})$ using HE is a barrier to the realization of P^{2}OLR. To find a approximate polynomial of $\sigma(x)$, adapting the least squares method, we consider the 7° polynomial $g(x) = a_0 + a_1 x + a_3 x^3 + a_5 x^5 + a_7 x^7$ over the domain $[-8, 8]$, where $a_0 = 1/2$, $a_1 = 1.73496/8$, $a_3 = 4.19407/8^3$, $a_5 = 5.43402/8^5$, $a_7 = 2.50739/8^7$. $\sigma(x)$ and $g(x)$ can be seen in Figure 2, the maximum errors between $\sigma(x)$ and $g(x)$ are about 0.032. $g(x)$ over encrypted data $\mathbf{x}$ from HE can be achieved by the Algorithm 10.

3.4. Logistic Regression. Logistic regression (LR) is a statistical analysis method for predicting the probability of an event. We consider the case where the predicted value is a binary dependent variable. Assuming that a dataset consists of m samples of the form $\{y_i, \mathbf{x}_i\}$ with $y_i \in \{0, 1\}$ and $\mathbf{x}_i = \{x_{i,1}, x_{i,2}, \cdots, x_{i,N/2-1}\} \in \mathbb{R}^{n-1}$, the goal of LR is to find the optimal parameters $\mathbf{w} = \{w_0, w_1, \cdots, w_{n-1}\}$ that minimizes the negative log-likelihood function (loss function) $J(w) = 1/m \cdot \sum_{i=0}^{m-1} (y_i \cdot \log(\sigma(d_i \cdot w^T)) + (1 - y_i) \cdot (1 - \log(\sigma(d_i \cdot w^T))))$, where $\mathbf{d}_i = \{1, \mathbf{x}_i\}$. A common method for minimizing loss function $J(\mathbf{w})$ is a gradient descent (GD) algorithm, which finds the local extremum of a loss function by following the direction of the gradient. The gradient of $J(\mathbf{w})$ with respect to $\mathbf{w}$ is calculated by $\nabla J(w) = 1/m \cdot \sum_{i=0}^{m-1} ((\sigma(-d_i \cdot w^T) - y_i) \cdot d_i)$. Let $\mathbf{w}^k$ be the regression parameters and α^k is a learning rate in the k-th iteration of the GD algorithm, the GD algorithm can update $\mathbf{w}^{k+1}$ by $\mathbf{w}^{k+1} \leftarrow \mathbf{w}^k - \alpha^k/m \cdot \sum_{i=0}^{m-1} ((\sigma(\mathbf{d}_i \cdot \mathbf{w}^\mathrm{T}) - y_i) \cdot \mathbf{d}_i)$.

4. Privacy-Preserving Outsourced Logistic Regression

Based on the HE scheme, we propose a P^{2}OLR, where we employ the batching method to pack multiple elements into multiple plaintext slots, and encrypt them into one ciphertext, and then perform the same arithmetic evaluations to multiple plaintext slots through the SIMD mechanism. To reduce the parameters of HE scheme (CKKS) as well as improve the performance of P^{2}OLR, the proposed P^{2}OLR allows the interaction between DO and SP during iterative training. Specifically, SP returns the ciphertext training result to DO after certain number of iterations t. DO decrypts the ciphertext training result, and determines whether the performance of the model has met the requirements. If so, stops training. Otherwise, sends encrypted weights to SP to continue training. Let

$$D = \begin{bmatrix} y_0 \\ y_1 \\ \vdots \\ y_{m-1} \\ x_{0,1} \\ x_{1,1} \\ \vdots \\ x_{m-1,1} \\ x_{0,2} \\ x_{1,2} \\ \vdots \\ x_{m-1,2} \\ \cdots \\ \cdots \\ \ddots \\ \cdots \\ x_{0,n-1} \\ x_{1,n-1} \\ \vdots \\ x_{m-1,n-1} \end{bmatrix}. \tag{1}$$

denote the training data sets held by DO, where D consists of m samples of the form $\{y_i, x_{i,1}, x_{i,2}, \cdots, x_{i,N/2-1}\}$ with $y_i \in \{0, 1\}$ and $\{x_{i,1}, x_{i,2}, \cdots, x_{i,N/2-1}\} \in \mathbb{R}^{n-1}$. The first column of D denotes the label, other columns D denote the features. Since DO has limited computational resources, DO

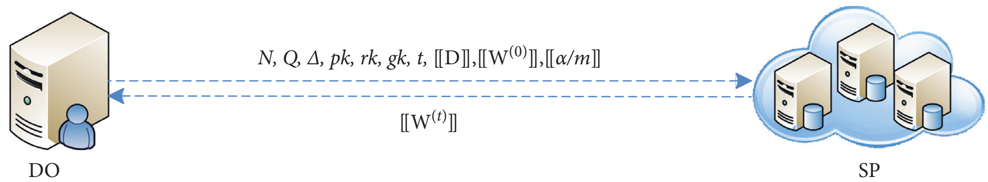


FIGURE 1: System model.

TABLE 1: The definitions of the notations.

Notations	Definitions
$\mathbf{x}$	A message vector $[x_0, x_1, \cdots, x_{N/2-1}]$
$\langle \mathbf{x} \rangle$	The plaintext of message vector $\mathbf{x}$
$\mathbf{x}$	The ciphertext of message vector $\mathbf{x}$
X	A list $\{\mathbf{x}_0, \mathbf{x}_1, \ldots, \mathbf{x}_{n-1}\}$
X_i	The i ciphertext of ciphertext list X_i
$\mathbf{x} * \mathbf{y}$	The multiplication of x and y, namely $[x_0 \cdot y_0, x_1 \cdot y_1, \cdots, x_{N/2-1} \cdot y_{N/2-1}]$
$\mathbf{x} \cdot \mathbf{y}$	The product of x and y, namely $[x_0 \cdot y_0, x_1 \cdot y_1, \cdots, x_{N/2-1} \cdot y_{N/2-1}]$
$\mathbf{x} + \mathbf{y}$	The addition of x and y, namely $[x_0 + y_0, x_1 + y_1, \cdots, x_{N/2-1} + y_{N/2-1}]$
$\mathbf{x} - \mathbf{y}$	The subtraction of x and y, namely $[x_0 + y_0, x_1 + y_1, \cdots, x_{N/2-1} + y_{N/2-1}]$

Input: x
Output: x
(1) encode_double (x, Δ, x)
(2) encrypt ($\langle x \rangle$, x)
(3) **return**: x

ALGORITHM 1: $x = \mathrm{Enc}(x)$.

Input: X
Output: $\{x_0, x_1, \cdots, x_{n-1}\}$
(1) **for** ($i = 0$ to $n - 1$) do
(2) decrypt (X_i, $\langle x_i \rangle$)
(3) decode_double ($\langle x_i \rangle$, x_i)
(4) $x_i = \mathbf{x}_\mathbf{i}$.get(0)
(5) **end for**
(6) **return**: $\{x_0, x_1, \ldots, x_{n-1}\}$

ALGORITHM 2: $\{x_0, x_1, \cdots, x_{n-1}\} = \mathrm{Dec}(X)$.

Input: x, y
Output: $\mathbf{x} * \mathbf{y}$
(1) mod_switch_to_inplace (y, x.parms_id())
(2) multiply (x, y, $x * y$)
(3) relinearize_inplace ($x * y$, rk)
(4) rescale_to_next_inplace ($x * y$)
(5) $x * y$.set_scale (Δ)
(6) **return**: $x * y$

ALGORITHM 3: $x * y = \mathrm{Mul}(x, y)$.

Input: x, y
Output: $x * y$
(1) encode_double (y, Δ, $\langle y \rangle$)
(2) mod_switch_to_inplace ($\langle y \rangle$, x.parms_id())
(3) multiply_plain (x, $\langle y \rangle$, $x * y$)
(4) rescale_to_next_inplace ($x * y$)
(5) $x * y$.set_scale (Δ)
(6) **return**: $x * y$

ALGORITHM 4: $x * y = \mathrm{Mul\_Plain}(x, y)$.

Input: x, y
Output: $x + y$
(1) mod_switch_to_inplace (y, x.parms_id())
(2) add (x, y, $x + y$)
(3) **return**: $x + y$

ALGORITHM 5: $x + y = \mathrm{Add}(x, y)$.

Input: x, y
Output: $x + y$
(1) encode_double (y, Δ, $\langle y \rangle$)
(2) mod_switch_to_inplace ($\langle y \rangle$, x.parms_id())
(3) add_plain (x, $\langle y \rangle$, $x + y$)
(4) **return**: $x + y$

ALGORITHM 6: $x + y = \mathrm{Add\_Plain}(x, y)$.

Input: x, y
Output: $x - y$
(1) mod_switch_to_inplace (y, x.parms_id())
(2) sub (x, y, $x - y$)
(3) **return**: $x - y$

Algorithm 7: $x - y = \text{Sub}(x, y)$.

Input: x, y
Output: x
(1) mod_switch_to_inplace (x, y.parms_id())
(2) add _inplace (x, y)
(3) **return**: x

Algorithm 8: $x = \text{Add\_Inplace}(x, y)$.

Input: $x = [x_0, x_1, \cdots, x_{N/2-1}]$
Output: $y = [\sum_{i=0}^{N/2-1} x_i, \sum_{i=0}^{N/2-1} x_i, \cdots, \sum_{i=0}^{N/2-1} x_i]$
(1) $y = x$
(2) **for** ($k = N/2$; $k \geq 1$; $k = k/2$) **do**
(3) rotate_vector (y, k, gk, z)
(4) add_inplace (y, z)
(5) **end for**
(6) **return**: y

Algorithm 9: $y = \text{Rotate\_Sum}(x)$.

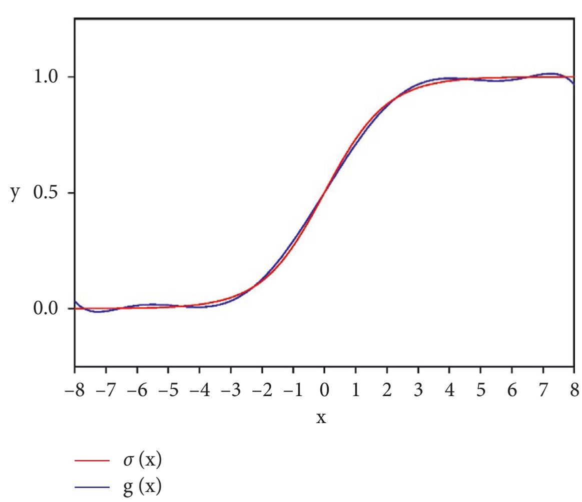


Figure 2: Sigmoid approximation.

wants to outsource to SP to train a LR model without disclosing its own training data privacy. The specific description of the proposed P^{2}OLR is as follows.

(1) DO generates $\{sk, pk, rk, gk\}$, computes $l = 2m/N$, calls the Algorithm 1 to encrypt the training data D into $l \times n$ ciphertexts

Input: x
Output: $g(x)$
(1) $x * x = \mathrm{Mul}(x, x)$
(2) $x * x * x * x = \mathrm{Mul}(x * x, x * x)$
(3) $x * x * x * x * x * x = \mathrm{Mul}(x * x * x * x, x * x)$
(4) $a_7 * x = \mathrm{Mul\_Plain}(x, a_7)$
(5) $a_7 * x * x * x * x * x * x * x = \mathrm{Mul}(x * x * x * x * x * x, a_7 * x)$
(6) $a_5 * x = \mathrm{Mul\_Plain}(x, a_5)$
(7) $a_5 * x * x * x * x = \mathrm{Mul}(x * x * x * x, a_5 * x)$
(8) $a_3 * x = \mathrm{Mul\_Plain}(x, a_3)$
(9) $a_3 * x * x * x = \mathrm{Mul}(x * x, a_3 * x)$
(10) $a_1 * x = \mathrm{Mul\_Plain}(x, a_1)$
(11) $a_0 + a_1 * x = \mathrm{Add\_Plain}(a_1 * x, a_0)$
(12) $a_0 + a_1 * x - a_3 * x * x * x = \mathrm{Sub}(a_3 * x * x * x, a_0 + a_1 * x)$
(13) $a_0 + a_1 * x - a_3 * x * x * x + a_5 * x * x * x * x * x = \mathrm{Add}(a_5 * x * x * x * x * x, a_0 + a_1 * x - a_3 * x * x * x)$
(14) $a_0 + a_1 * x - a_3 * x * x * x + a_5 * x * x * x * x * x - a_7 * x * x * x * x * x * x * x = \mathrm{Sub}(a_7 * x * x * x * x * x * x * x, a_0 + a_1 * x - a_3 * x * x * x + a_5 * x * x * x * x * x)$
(15) **return**: $g(x) = a_0 + a_1 * x - a_3 * x * x * x + a_5 * x * x * x * x * x - a_7 * x * x * x * x * x * x * x.$

ALGORITHM 10: $g(x) = \mathrm{Sigmoid\_Approximation}(x)$.

$$
\begin{aligned}
\llbracket \mathbf{y}_0^T \rrbracket &= \mathrm{Enc}\left(\left[y_0, y_1, \cdots, y_{N/2-1}\right]\right),\\
\llbracket \mathbf{y}_1^T \rrbracket &= \mathrm{Enc}\left[\left(y_{N/2}, y_{N/2+1}, \cdots, y_{N-1}\right)\right],\\
\llbracket \mathbf{y}_{l-1}^T \rrbracket &= \mathrm{Enc}\left(\left[y_{m-(l-1)\cdot N/2}, y_{m-(l-1)\cdot N/2+1}, \cdots, y_{m-1}\right]\right),\\
\llbracket \mathbf{x}_{0,1}^T \rrbracket &= \mathrm{Enc}\left(\left[x_{0,1}, x_{1,1}, \cdots, y_{N/2-1,1}\right]\right),\\
\llbracket \mathbf{x}_{0,n-1}^T \rrbracket &= \mathrm{Enc}\left(\left[x_{0,n-1}, x_{1,n-1}, \cdots, y_{N/2-1,n-1}\right]\right),\\
\llbracket \mathbf{x}_{0,2}^T \rrbracket &= \mathrm{Enc}\left(\left[x_{0,2}, x_{1,2}, \cdots, y_{N/2-1,2}\right]\right),\\
\llbracket \mathbf{x}_{1,1}^T \rrbracket &= \mathrm{Enc}\left(\left[x_{N/2,1}, x_{N/2+1,1}, \cdots, y_{N-1,1}\right]\right),\\
\llbracket \mathbf{x}_{1,2}^T \rrbracket &= \mathrm{Enc}\left(\left[x_{N/2,2}, x_{N/2+1,2}, \cdots, y_{N-1,2}\right]\right),\\
\llbracket \mathbf{x}_{1,n-1}^T \rrbracket &= \mathrm{Enc}\left(\left[x_{N/2,n-1}, x_{N/2+1+1,n-1}, \cdots, y_{N-1,n-1}\right]\right),\\
\llbracket \mathbf{x}_{l-1,1}^T \rrbracket &= \mathrm{Enc}\left(\left[x_{m-(l-1)\cdot N/2,1}, x_{m-(l-1)\cdot N/2+1,1}, \cdots, y_{m-1,1}\right]\right),\\
\llbracket \mathbf{x}_{l-1,2}^T \rrbracket &= \mathrm{Enc}\left(\left[x_{m-(l-1)\cdot N/2,2}, x_{m-(l-1)\cdot N/2+1,2}, \cdots, y_{m-1,2}\right]\right)\\
\llbracket \mathbf{x}_{l-1,n-1}^T \rrbracket &= \mathrm{Enc}\left(\left[x_{m-(l-1)\cdot N/2,n-1}, x_{m-(l-1)\cdot N/2+1,n-1}, \cdots, y_{m-1,n-1}\right]\right).
\end{aligned}
\tag{2}
$$

calls the Algorithm 1 to encrypt the initial weight $\left\{w_0^{(0)}, w_1^{(0)}, \cdots, w_{n-1}^{(0)}\right\}$ into n ciphertexts

$$
\begin{aligned}
\llbracket \mathbf{w}_0^{(0)} \rrbracket &= \mathrm{Enc}\left(\left[\underbrace{w_0^{(0)}, w_0^{(0)}, \cdots, w_0^{(0)}}_{N/2}\right]\right),\\
\llbracket \mathbf{w}_1^{(0)} \rrbracket &= \mathrm{Enc}\left(\left[\underbrace{w_1^{(0)}, w_1^{(0)}, \cdots, w_1^{(0)}}_{N/2}\right]\right),\\
\llbracket \mathbf{w}_{N-1}^{(0)} \rrbracket &= \mathrm{Enc}\left(\left[\underbrace{w_1^{(0)}, w_1^{(0)}, \cdots, w_1^{(0)}}_{N/2}\right]\right),
\end{aligned}
\tag{3}
$$

calls the Algorithm 1 to encrypt the learning rate α into one ciphertext

$$
\llbracket \frac{\boldsymbol{\alpha}}{\mathbf{m}} \rrbracket = \mathrm{Enc}\left(\left[\underbrace{\alpha/m, 0, 0, \cdots, 0}_{N/2}\right]\right). \tag{4}
$$

and sends $\mathbf{y}_0^T$, $\mathbf{y}_1^T$, $\cdots$, $\mathbf{y}_{l-1}^T$, $\mathbf{x}_{0,1}^T$, $\mathbf{x}_{0,2}^T$, $\cdots$, $\mathbf{x}_{0,n-1}^T$, $\mathbf{x}_{1,1}^T$, $\mathbf{x}_{1,2}^T$, $\cdots$, $\mathbf{x}_{1,n-1}^T$, $\mathbf{x}_{l-1,1}^T$, $\mathbf{x}_{l-1,2}^T$, $\cdots$, $\mathbf{x}_{l-1,n-1}^T$, $w_0^{(0)}$, $\mathbf{w}_1^{(0)}$, $\cdots$, $\mathbf{w}_{n-1}^{(0)}$, $\alpha/\mathbf{m}$, N, Q, sk, pk, rk, gk, t to SP.

(2) SP computes ciphertexts

$$
\begin{aligned}
\llbracket \mathbf{x}_{i,0}^T \rrbracket &= Enc\left(\left[\underbrace{1, 1, \cdots, 1}_{N/2}\right]\right), \ (i = 1, 2, \cdots, l-2),\\
\llbracket \mathbf{x}_{l-1,0}^T \rrbracket &= \mathrm{Enc}\left(\left[\underbrace{1, 1, \cdots, 1}_{m-(l-1)\cdot N/2}\right]\right),
\end{aligned}
\tag{5}
$$

and sets the lists

$$
\begin{aligned}
\mathrm{Y} &= \left\{[\![\mathbf{y}_0^T]\!], [\![\mathbf{y}_1^T]\!], \cdots, [\![\mathbf{y}_{l-1}^T]\!]\right\},\\
[\![\mathrm{X}_0]\!] &= \left\{[\![\mathbf{x}_{0,0}^T]\!], [\![\mathbf{x}_{0,1}^T]\!], [\![\mathbf{x}_{0,2}^T]\!], \cdots, [\![\mathbf{x}_{0,n-1}^T]\!]\right\},\\
[\![\mathrm{X}_1]\!] &= \left\{[\![\mathbf{x}_{1,0}^T]\!], [\![\mathbf{x}_{1,1}^T]\!], [\![\mathbf{x}_{1,2}^T]\!], \cdots, [\![\mathbf{x}_{1,n-1}^T]\!]\right\},\\
[\![\mathrm{X}_{l-1}]\!] &= \left\{[\![\mathbf{x}_{l-1,0}^T]\!], [\![\mathbf{x}_{l-1,1}^T]\!], [\![\mathbf{x}_{l-1,2}^T]\!], \cdots, [\![\mathbf{x}_{l-1,n-1}^T]\!]\right\},\\
[\![\mathrm{W}^{(0)}]\!] &= \left\{[\![\mathbf{w}_0^{(0)}]\!], [\![\mathbf{w}_1^{(0)}]\!], \cdots, [\![\mathbf{w}_{n-1}^{(0)}]\!]\right\}.
\end{aligned}
\tag{6}
$$

Next, SP calls the Algorithm 11, and returns the ciphertext result $W^{(t)}$ to DO.

(3) DO calls the Algorithm 2 to decrypt the ciphertext result $W^{(t)}$ into the result $\left\{w_0^{(t)}, w_1^{(t)}, \cdots, w_{n-1}^{(t)}\right\} = \mathrm{Dec}(\mathrm{W}^{(t)})$. Next, DO judges whether $\left\{w_0^{(t)}, w_1^{(t)}, \cdots, w_{n-1}^{(t)}\right\}$ has met the requirements. If so, terminates the training. Otherwise, DO calls the Algorithm 1 to encrypt $\left\{w_0^{(t)}, w_1^{(t)}, \cdots, w_{n-1}^{(t)}\right\}$ into n ciphertexts

$$
\begin{aligned}
[\![\mathbf{w}_0^{(0)}]\!] &= \mathrm{Enc}\left(\left[\underbrace{w_0^{(t)}, w_0^{(t)}, \cdots, w_0^{(t)}}_{N/2}\right]\right),\\
[\![\mathbf{w}_1^{(0)}]\!] &= \mathrm{Enc}\left(\left[\underbrace{w_1^{(t)}, w_1^{(t)}, \cdots, w_1^{(t)}}_{N/2}\right]\right),\\
[\![\mathbf{w}_{n-1}^{(0)}]\!] &= \mathrm{Enc}\left(\left[\underbrace{w_{n-1}^{(t)}, w_{n-1}^{(t)}, \cdots, w_{n-1}^{(t)}}_{N/2}\right]\right).
\end{aligned}
\tag{7}
$$

and sends $w_0^{(0)}$, $\mathbf{w}_1^{(0)}$, $\cdots$, $\mathbf{w}_{n-1}^{(0)}$ to SP to continue training.

5. Performance Evaluation

We implement all experiments on a 32-core Intel Xeon CPU with 256 GB RAM. We compare the performance of the proposed $\mathrm{P^2OLR}$ with the related $\mathrm{P^2OLR}$ [17, 18, 22]. We employ 5-fold cross-validation method to obtain the validity of the experimental results. For [17, 18], the implementations are publicly available at [35, 36], respectively, which use the HEAAN library [33] to provide HE cryptographic operations. For [22] and the proposed $\mathrm{P^2OLR}$, we employ the Microsoft SEAL library [26] for the HE cryptographic operations. For all experiments, we set the learning rate $\alpha = 0.01$, random initial weight vector $\left\{w_0^{(0)}, w_1^{(0)}, \cdots, w_{n-1}^{(0)}\right\}$ maximum number of iterations $\lambda = 20$, and scaling factor $\Delta = 2^{40}$. To guarantee $\kappa = 128$ bit security, the scheme [17] takes the polynomial-modulus-degree $N = 2^{17}$, coefficient-modulus Q around 2204 to 2406 bits; the scheme [18] sets the $N = 2^{16}, Q = 1176$ bits; the scheme [22] chooses the $N = 2^{15}$, $Q = 320$ bits; For the proposed $\mathrm{P^2OLR}$, we select $N = 2^{15}$, $Q = 512$ bits. Using the three datasets [18]: D_1—Umaru Impact Study, D_2—Myocardial Infarction Study from Edinburgh, D_3—Nhanes III, we compare the proposed $\mathrm{P^2OLR}$ with the related $\mathrm{P^2OLR}$ [17, 18, 22] in terms of the encryption time (E. time) and decryption time (D. time) of DO, storage of encrypted training data, and training time (T. time), accuracy, precision, recall, F1-score and AUC of model. All comparison results are shown as an average of 10 experiments. The performance comparisons of the proposed $\mathrm{P^2OLR}$ and the related $\mathrm{P^2OLR}$ [17, 18, 22] are shown in Table 2.

From Table 2, we can see that, compared with the related $\mathrm{P^2OLR}$ [17, 18, 22], the proposed $\mathrm{P^2OLR}$ has a better performance. Specifically, as shown in Figure 3, under the training dataset D_1, the encryption time of DO in the proposed $\mathrm{P^2OLR}$ is 2.01 s, which is reduced by nearly 71.4%, 7.8%, and 93.3% respectively compared with the encryption time of DO in [17, 18, 22]; under the training dataset D_2, the encryption time of DO in the proposed $\mathrm{P^2OLR}$ is 2.16 s, which is reduced by nearly 73.6%, 2.3%, and 96.8% respectively compared with the encryption time of DO in [17, 18, 22]; under the training dataset D_3, the encryption time of DO in the proposed $\mathrm{P^2OLR}$ is 3.49 s, which is reduced by nearly 75.9%, 81.6%, and 75.0% respectively compared with the encryption time of DO in [17, 18, 22].

As can be seen in Figure 4, under the training dataset D_1, the decryption time of DO in the proposed $\mathrm{P^2OLR}$ is 0.23 s, which is reduced by almost 95.3% and 41.0% respectively in comparison to the decryption time of DO in [17, 18]; under the training dataset D_2, the decryption time of DO in the proposed $\mathrm{P^2OLR}$ is 0.26 s, which is reduced by almost 95.0% and 36.6% respectively in comparison to the decryption time of DO in [17, 18]; under the training dataset D_3, the decryption time of DO in the proposed $\mathrm{P^2OLR}$ is 0.45 s, which is reduced by almost 96.1% and 6.1% respectively in comparison to the decryption time of DO in [17, 18]. The decryption time of DO in [22] is smaller in comparison to that of the proposed $\mathrm{P^2OLR}$.

As described in Figure 5, under the training dataset D_1, the storage of encrypted training data in the proposed $\mathrm{P^2OLR}$ is 72.00 MB, compared with the storage of encrypted training data in [17, 22], which is reduced by nearly 88.9% and 95.0%; under the training dataset D_2, the storage of encrypted training data in the proposed $\mathrm{P^2OLR}$ is 80.00 MB, compared with the storage of encrypted training data in [17, 22], which is reduced by nearly 89.0% and 97.4%; under the training dataset D_3, the storage of encrypted training data in the proposed $\mathrm{P^2OLR}$ is 128.00 MB, compared with the storage of encrypted training data in [17, 18, 22], which is reduced by nearly 89.4%, 13.0% and 99.7% respectively. Although the storage of encrypted training data for dataset D_1 and D_2 in [18] is smaller than that of the proposed $\mathrm{P^2OLR}$, as the number of samples m and features n increases, for dataset D_3, the storage of encrypted training data in the proposed $\mathrm{P^2OLR}$ is smaller than that of [22].

As displayed in Figure 6, under the training dataset D_1, the training time of model in the proposed $\mathrm{P^2OLR}$ is 2.64 min, which is reduced by almost 96.6%, 73.8%, and 90.1% respectively than the training time of model in [17, 18, 22]; under the training dataset D_2, the training time of model in the proposed $\mathrm{P^2OLR}$ is 2.91 min, which is reduced by almost 96.5%, 71.7%, and 95.0% respectively than the training time of model in [17, 18, 22]; under the training dataset D_3, the training time of model in the proposed $\mathrm{P^2OLR}$ is 4.21 min, which is reduced by almost 96.5%, 79.8%, and 99.4% respectively than the training time of model in [17, 18, 22].

Input: Y, X_0, X_1, $\cdots$, X_{l-1}, $w^{(0)}$, $\alpha/\mathbf{m}$, N, Q,., sk, pk, rk, gk, t
Output: $W^{(t)}$
for $(k = 0$ to $t-1)$ **do**
  for $(i = 0$ to $l-1)$ **do**
    for $(j = 0$ to $n-1)$ **do**
      $C_{ij} = \mathrm{Mul}(W^{(k)}{}_j, X_{ij})$
    end for
    $O_i = 0$
    for $(j = 0$ to $n-1)$ **do**
      $O_i = \mathrm{Add\_Inplace}(O_i, C_{ij})$
    end for
    $G_i = \mathrm{Sigmoid\_Approximation}(O_i)$
    $G'_i = \mathrm{Sub}(G_i, Y_i)$
    for $(j = 0$ to $n-1)$ **do**
      $C'_{ij} = \mathrm{Mul}(G'_i, X_{ij})$
    end for
  end for
  for $(j = 0$ to $n-1)$ **do**
    $Z_i = 0$
    for $(i = 0$ to $l-1)$ **do**
      $Z_j = \mathrm{Add\_Inplace}(Z_j, C'_{ij})$
    end for
    $Z'_j = \mathrm{Rotate\_Sum}(Z_j)$
    $Z'_j = \mathrm{Mul}(Z'_j, \alpha/m)$
    $W^{(k+1)}{}_j = \mathrm{Sub}(W^{(k)}{}_j, Z''_j)$
    $W'^{(k+1)}{}_j = \mathrm{Mul\_Plain}(W^{(k+1)}{}_j, 1)$
    $W''^{(k+1)}{}_j = \mathrm{Rotate\_Sum}(W'^{(k+1)}{}_j)$
  end for
end for
return: $W^{(t)}$

ALGORITHM 11: P^2OLR.

TABLE 2: Performance comparisons.

Dataset	m	n	λ	Scheme	E. time (s)	D. time (s)	Storage (MB)	T. time (min)	Accuracy (%)	Precision (%)	Recall (%)	F1-score (%)	AUC
D_1	575	8	20	[17]	7.04	4.93	648.56	77.35	74.8	92.3	71.4	80.5	0.68
				[18]	2.18	0.39	36.75	10.09	74.4	90.9	71.4	80.0	0.65
				[22]	30.28	0.06	1438.75	26.67	74.4	90.9	71.4	80.0	0.66
				P^2OLR	2.01	0.23	72.00	2.64	80.6	95.6	77.4	85.5	0.73
D_2	1253	9	20	[17]	8.17	5.19	726.86	83.57	81.6	89.7	82.4	85.9	0.82
				[18]	2.21	0.41	36.75	10.28	83.0	90.4	83.5	86.8	0.86
				[22]	68.35	0.06	3133.75	57.35	82.7	90.4	82.9	86.5	0.86
				P^2OLR	2.16	0.26	80.00	2.91	90.6	95.1	90.6	92.8	0.88
D_3	15649	15	20	[17]	14.48	11.65	1203.00	121.95	79.1	50.0	61.2	55.0	0.83
				[18]	13.96	0.48	147.00	20.86	79.2	50.2	61.3	55.2	0.71
				[22]	823.56	0.06	39123.75	718.25	77.9	52.4	62.2	56.9	0.71
				P^2OLR	3.49	0.45	128.00	4.21	83.7	60.3	64.2	62.2	0.85

As illustrated in Figure 7, under the training dataset D_1, the average accuracy of model in the proposed P^2OLR is 80.6%, which has nearly 5.8%, 6.2%, and 6.2% improvement respectively compared with the average accuracy of model in [17, 18, 22]; under the training dataset D_2, the average accuracy of model in the proposed P^2OLR is 90.6%, which has nearly 9.0%, 7.6%, and 7.9% improvement respectively compared with the average accuracy of model in [17, 18, 22]; under the training dataset D_3, the average accuracy of model in the proposed P^2OLR is 83.7%, which has nearly 4.6%, 4.5%, and 5.8% improvement respectively compared with the average accuracy of model in [17, 18, 22].

As illustrated in Figure 8, under the training dataset D_1, the average precision of model in the proposed P^2OLR is 95.6%, which has nearly 3.3%, 4.7%, and 4.7% improvement respectively compared with the average precision of model in [17, 18, 22]; under the training dataset D_2, the average precision of model in the proposed P^2OLR is 95.1%, which has nearly 5.4%, 4.7%, and 4.7% improvement respectively compared with the average precision of model in [17, 18, 22];

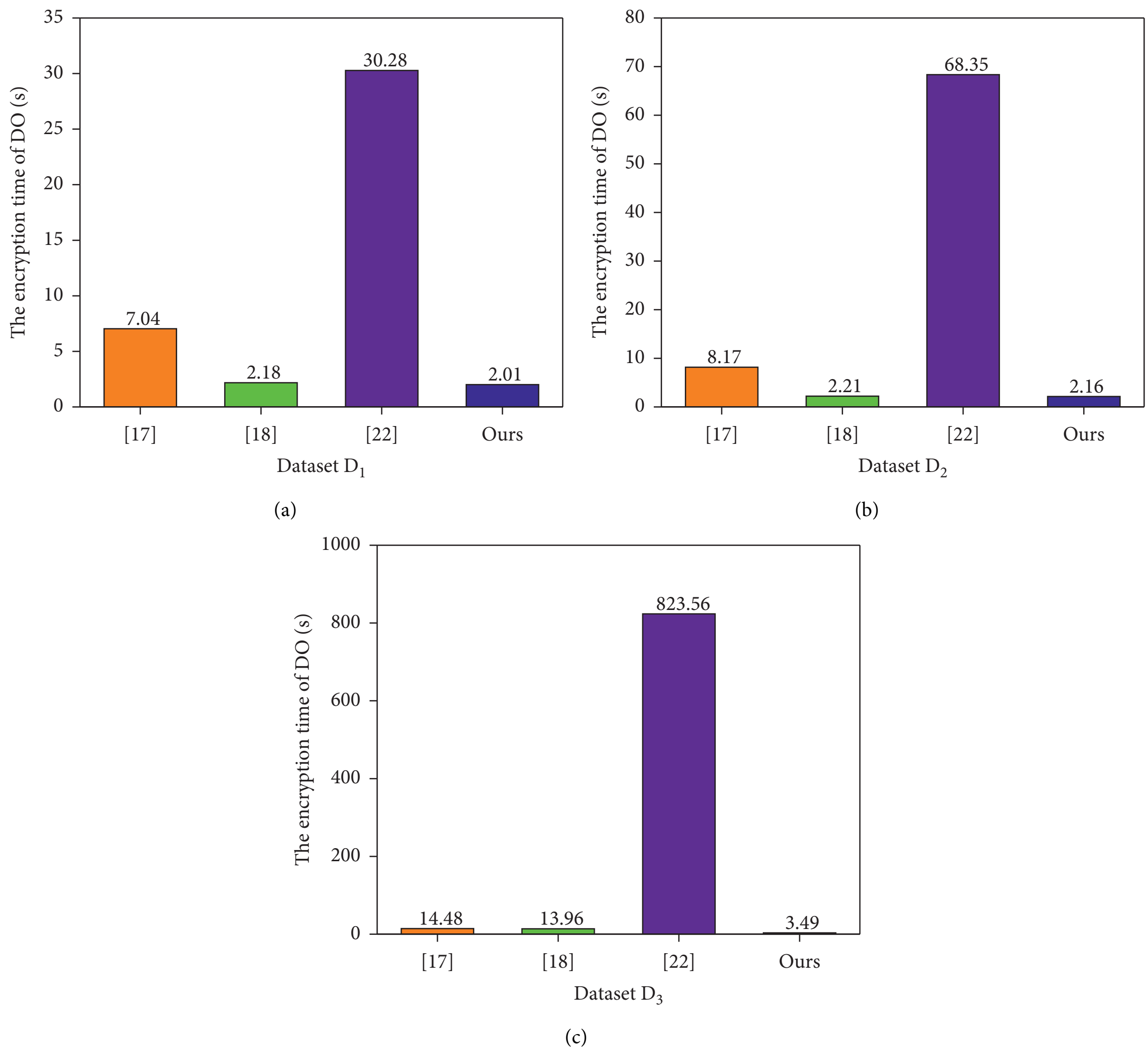


Figure 3: The encryption time of DO.

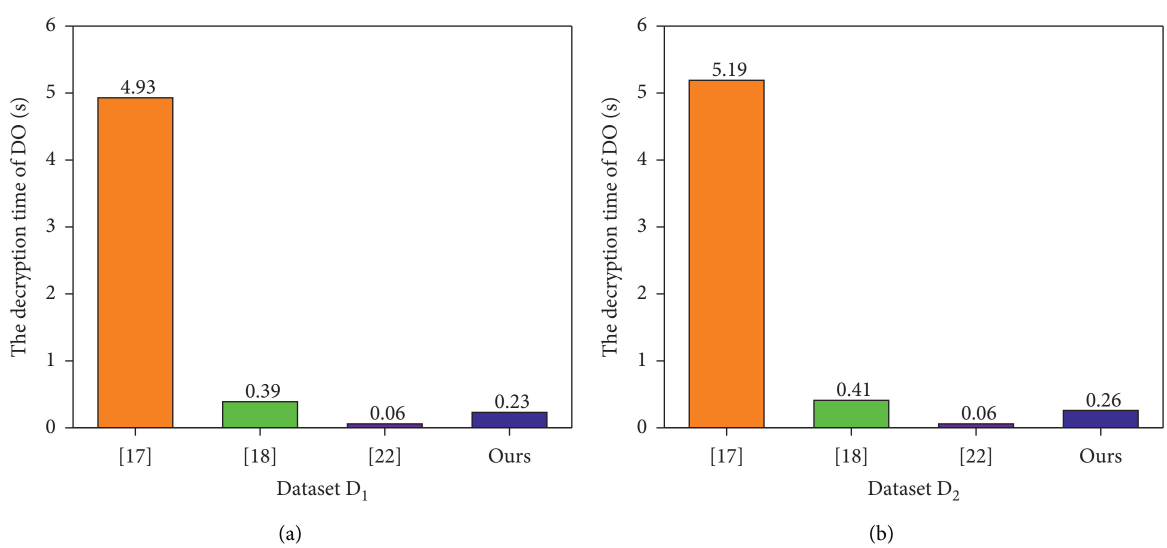
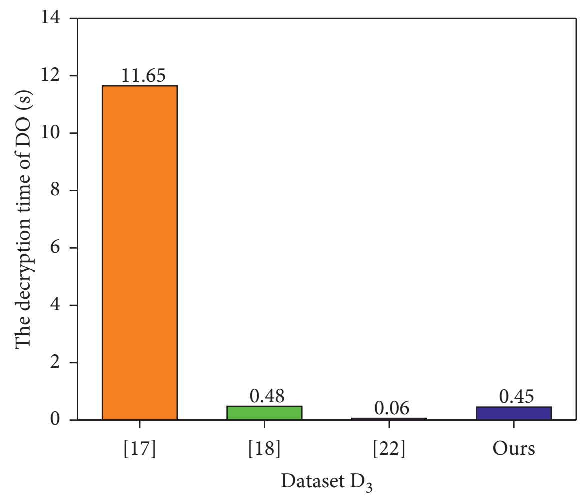


Figure 4: Continued.



(c)

Figure 4: The decryption time of DO.

under the training dataset D_3, the average precision of model in the proposed P^2OLR is 60.3%, which has nearly 10.3%, 10.1%, and 7.9% improvement respectively compared with the average precision of model in [17, 18, 22].

As illustrated in Figure 9, under the training dataset D_1, the average recall of model in the proposed P^2OLR is 77.4%, which has nearly 6.0%, 6.0%, and 6.0% improvement respectively compared with the average recall of model in [17, 18, 22]; under the training dataset D_2, the average recall of model in the proposed P^2OLR is 90.6%, which has nearly 8.2%, 7.1%, and 7.7% improvement respectively compared with the average recall of model in [17, 18, 22]; under the training dataset D_3, the average recall of model in the proposed P^2OLR is 64.2%, which has nearly 3.0%, 2.9%, and 2.0% improvement respectively compared with the average recall of model in [17, 18, 22].

As illustrated in Figure 10, under the training dataset D_1, the average F1-score of model in the proposed P^2OLR is 85.5%, which has nearly 5.0%, 5.5%, and 5.5% improvement respectively compared with the average F1-score of model in [17, 18, 22]; under the training dataset D_2, the average F1-score of model in the proposed P^2OLR is 92.8%, which has nearly 6.9%, 4.0%, and 4.3% improvement respectively compared with the average F1-score of model in [17, 18, 22]; under the training dataset D_3, the average F1-score of model in the proposed P^2OLR is 62.2%, which has nearly 7.2%, 7.0%, and 5.3% improvement respectively compared with the average F1-score of model in [17, 18, 22].

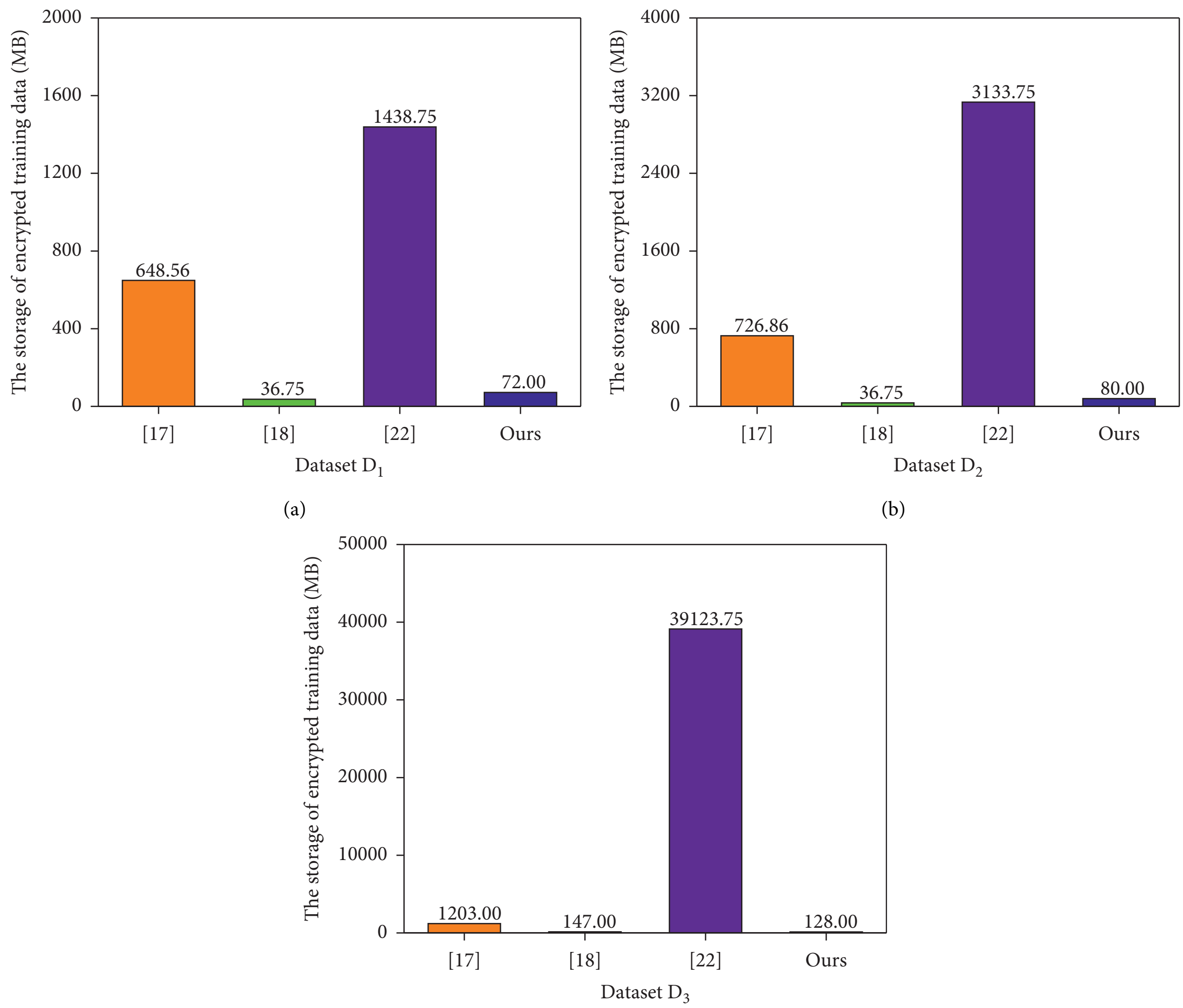
As demonstrated in Figure 11, under the training dataset D_1, the AUC of model in the proposed P^2OLR is 0.73, compared with the AUC of model in [17, 18, 22], which has nearly 0.05, 0.08, and 0.07 improvement respectively; under the training dataset D_2, the AUC of model in the proposed P^2OLR is 0.88, compared with the AUC of model in [17, 18, 22], which has nearly 0.06, 0.02, and 0.02 improvement respectively; under the training dataset D_3, the AUC of model in the proposed P^2OLR is 0.85, compared with the AUC of model in [17, 18, 22], which has nearly 0.02, 0.14, and 0.14 improvement respectively.

6. Security Analysis

In a semi-honest adversary model, we assume that DO and SP hold the public key *pk*, relinearization key *rk*, galois key *gk*, and only DO holds the secret key *sk*. For our P^2OLR that evaluates deterministic function f, following the simulation-based paradigm [37], we consider the security model for security analysis, namely, DO encrypts its private data $\mathbf{x}$ and sends $\mathbf{x}$ to SP. SP performs the homomorphic operations on $\mathbf{x}$ to obtain $\mathbf{y}$, homomorphically evaluates $f(\mathbf{x})$ on $\mathbf{x}$ to obtain $f(\mathbf{x})$, and sends $f(\mathbf{x})$ to DO. DO decrypts $f(\mathbf{x})$ and obtains $f(\mathbf{x})$.

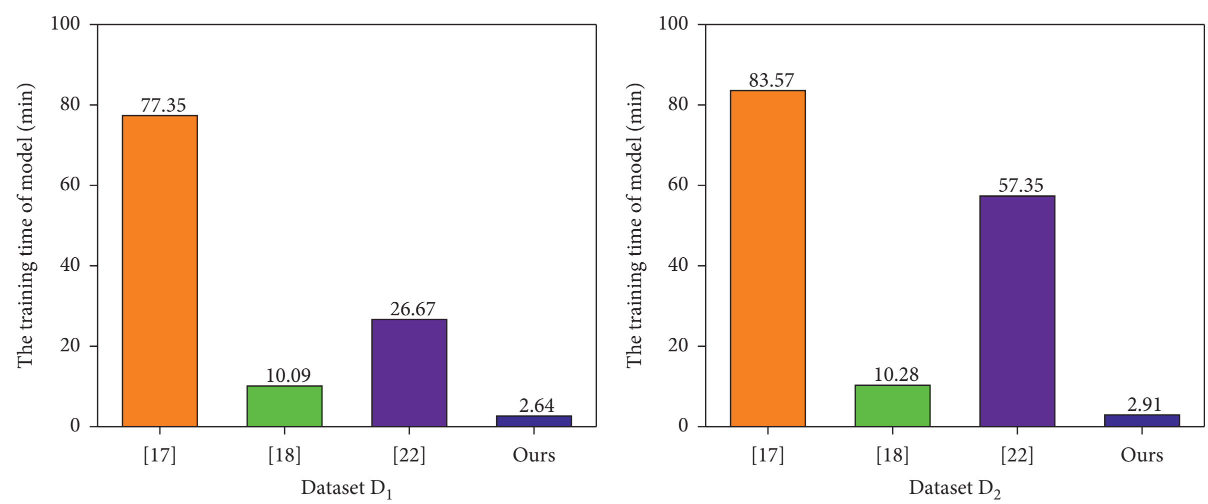
Theorem 1. *We assume that SP is a semi-honest entity and assume that DO and SP do not collude with each other. Let* $\mathbf{x}$ *be a private data of DO. If the HE scheme [23] provides semantic security, after performing the homomorphic operations on* $\mathbf{x}$ *and the evaluation of* $f(\mathbf{x})$ *on* $\mathbf{x}$*, DO learns* $f(\mathbf{x})$ *but nothing else, SP learns nothing.*

Security Proof. The security proof of the proposed P^2OLR follows the simulation-based paradigm [37]. Let the view of DO and SP during the evaluation be $\mathscr{V}_{DO}$ *and* $\mathscr{V}_{SP}$*, respectively. The view* $\mathscr{V}_{SP}$ *of SP consists of* $\{pk, rk, gk, \mathbf{x}, \mathbf{y}, f(\mathbf{x})\}$. *We construct a simulator* $\mathscr{S}_{SP}$ *as*



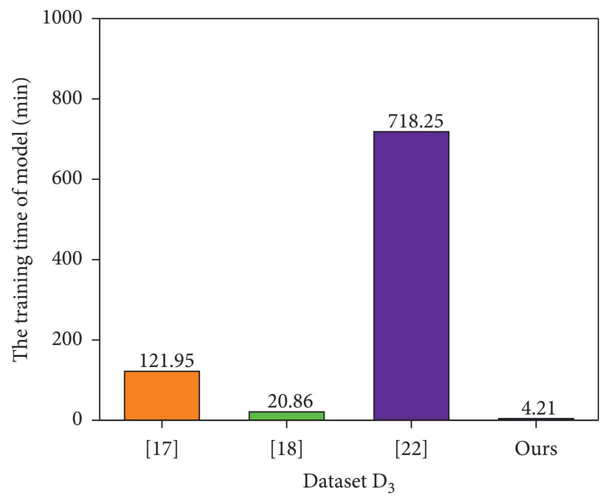
(a) (b) (c)

Figure 5: The storage of encrypted training data.



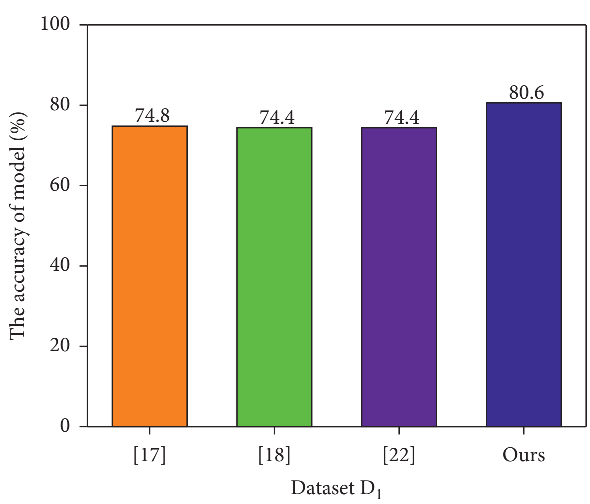
(a) (b)

Figure 6: Continued.

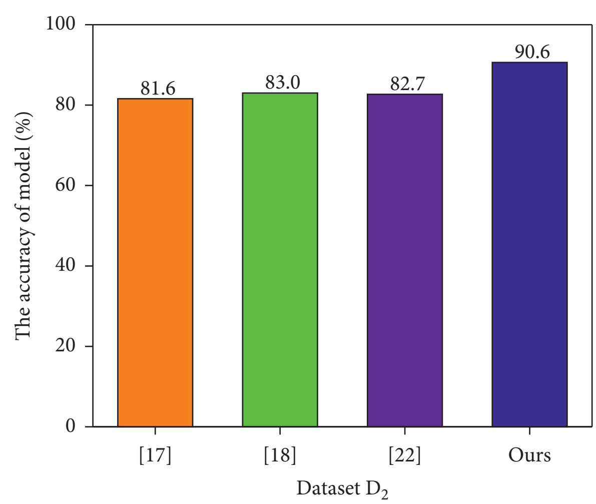


(c)

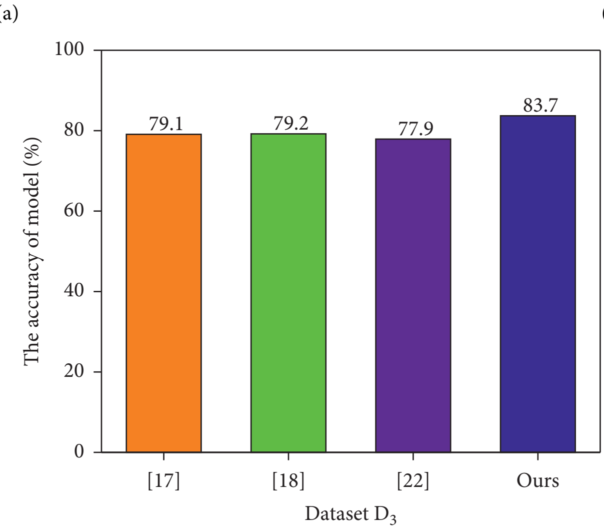
Figure 6: The training time of model.



(a)



(b)



(c)

Figure 7: The accuracy of model.

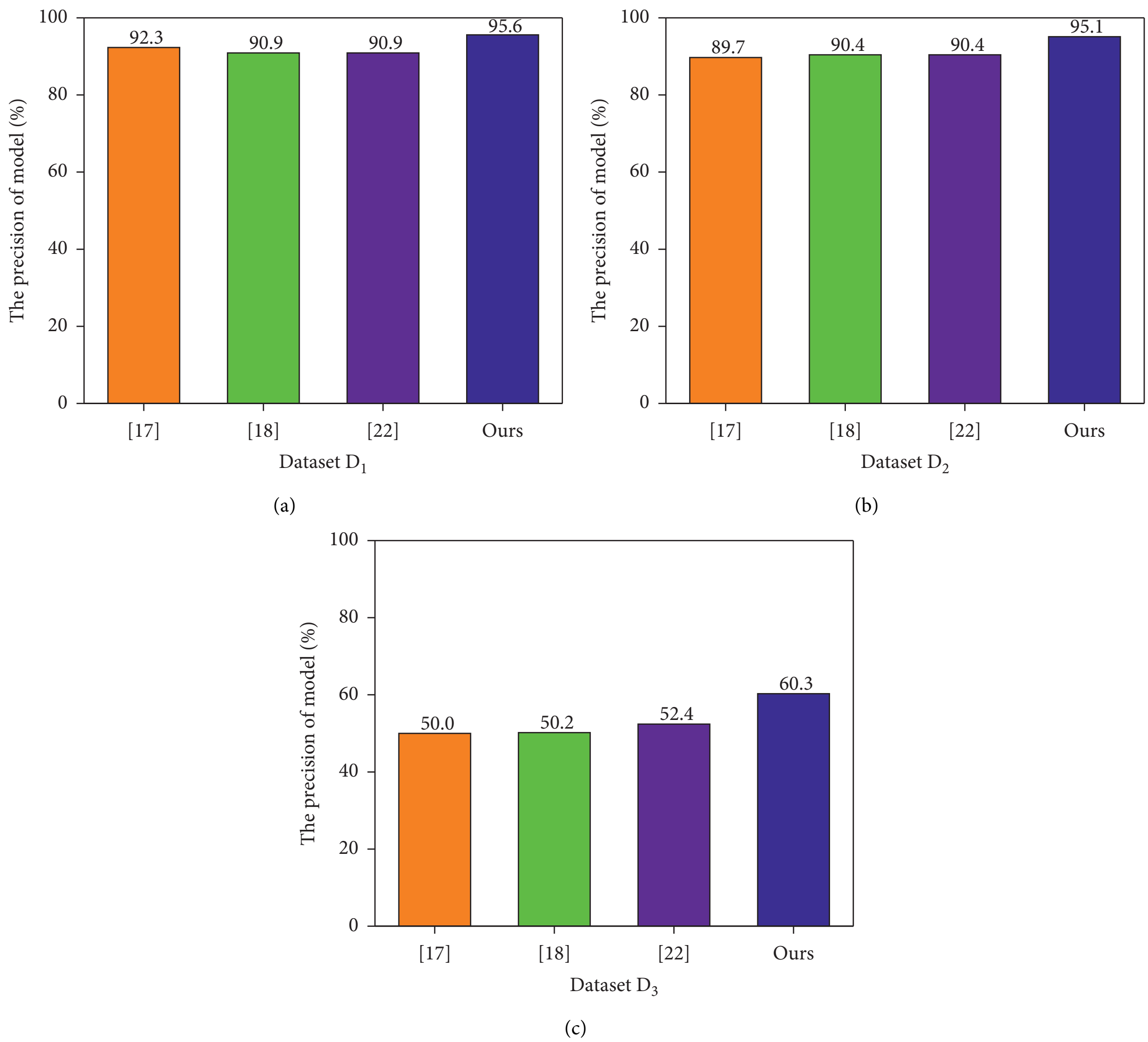


Figure 8: The precision of model.

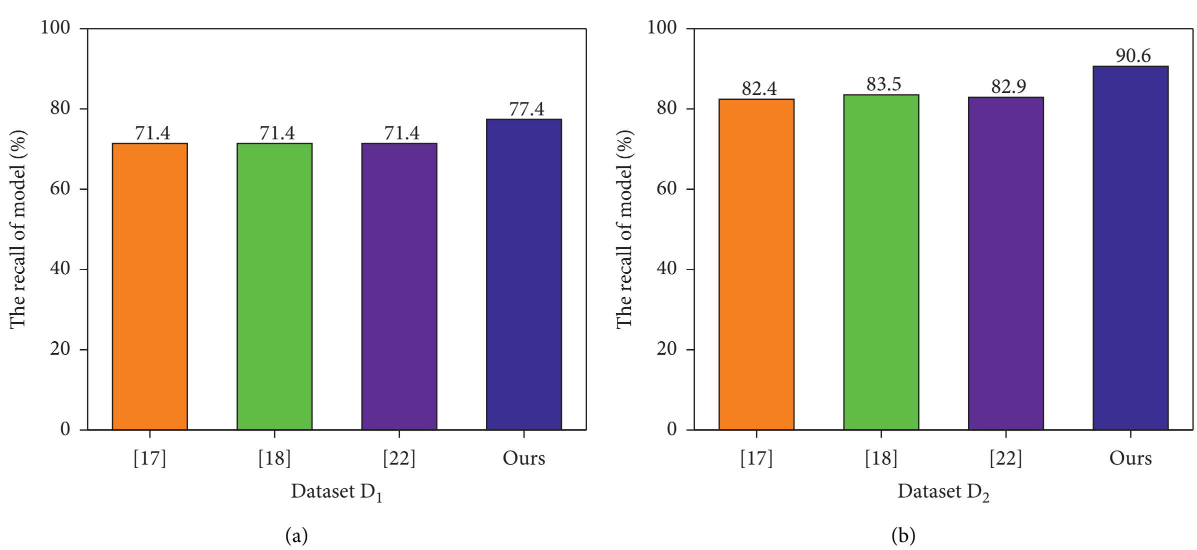
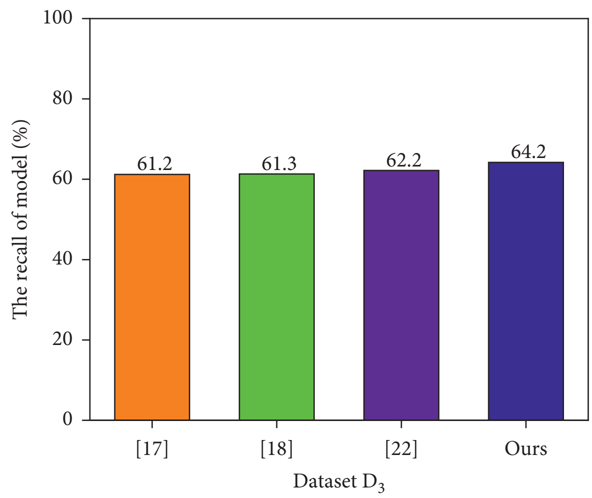
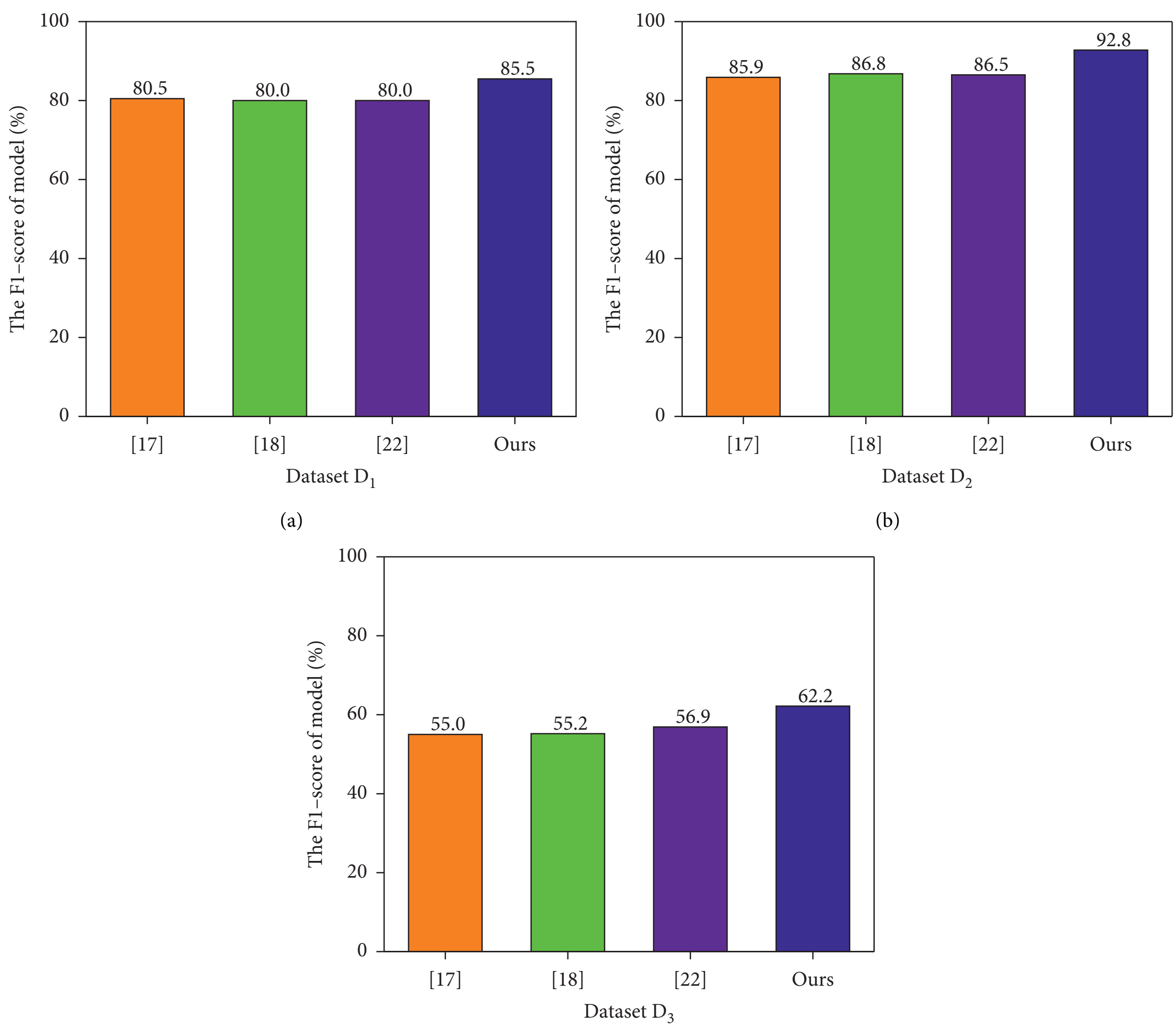


Figure 9: Continued.



(c)

FIGURE 9: The recall of model.



(c)

FIGURE 10: The F1-score of model.

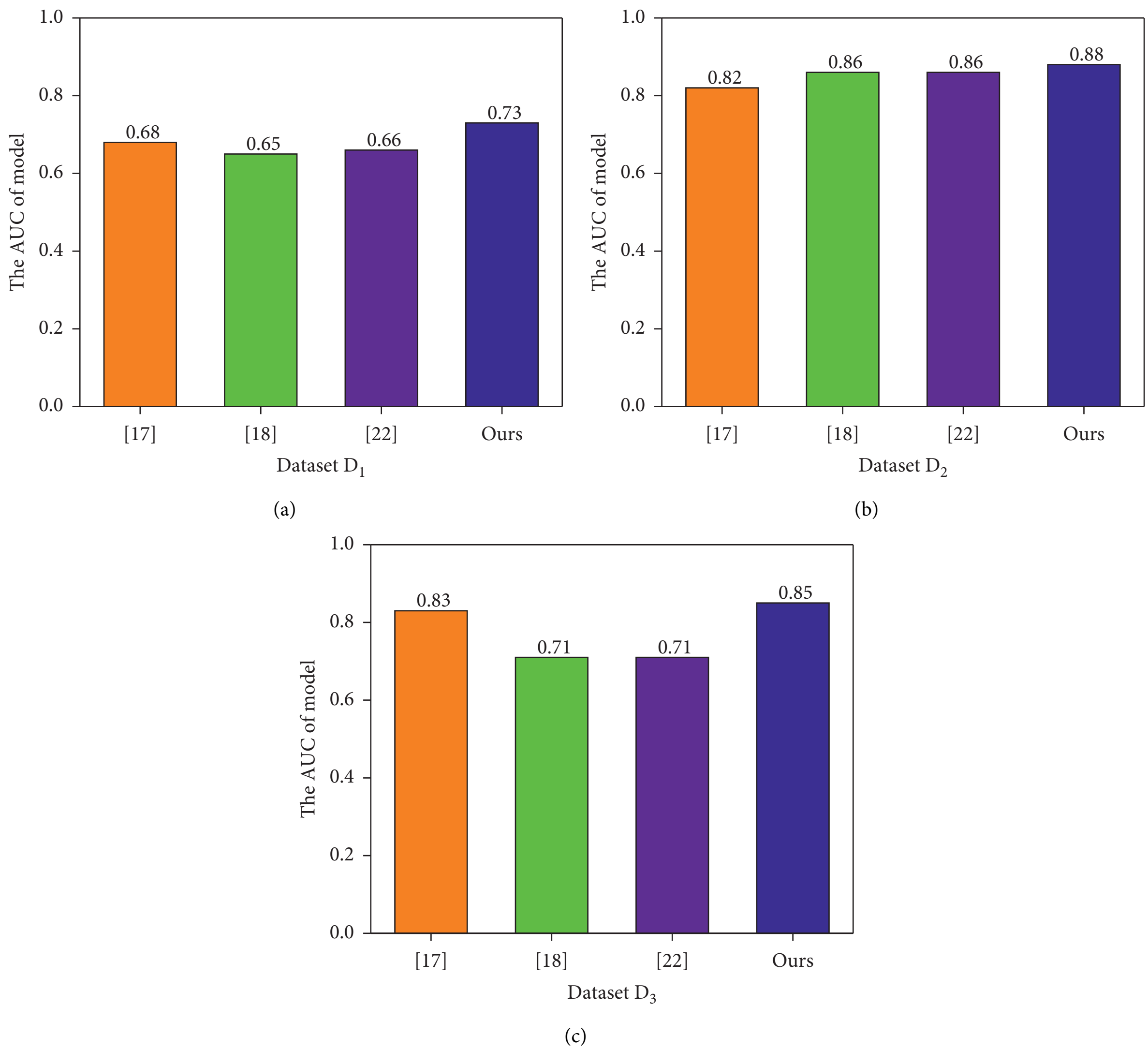


Figure 11: The AUC of model.

follows. $\mathcal{S}_{SP}$ *randomly chooses input data* $\mathbf{x}', \mathbf{y}', f(\mathbf{x}')$. *Then,* $\mathcal{S}_{SP}$ *simulates* $\mathcal{V}_{SP}$ *by* $\mathcal{V}_{SP}' = \{pk, rk, gk, \mathbf{x}', \mathbf{y}', f(\mathbf{x}')\}$. *Since the HE scheme [23] provides semantic security by assumption,* $\mathcal{V}_{SP}$ *and* $\mathcal{V}_{SP}'$ *are indistinguishable. Therefore, the proposed* P^2*OLR is secure against a semi-honest SP.*

7. Conclusion

In this paper, we present a method for achieving a P^2OLR on encrypted training data, which enables data owners to utilize the powerful storage and computing resources of cloud service providers for logistic regression analysis without exposing the privacy of training data. We take advantage of the batching technique and SIMD mechanism in HE to speed up the training progress. On the three public datasets, compared with the related P^2OLR schemes [17, 18, 22], the model training time of the proposed P^2OLR is reduced by more than 71.7%, and the proposed P^2OLR has over 4.5%, 3.3%, 2.0%, 4.0%, and 0.02 performance in terms of the accuracy, precision, recall, F1-score, and AUC of model. There are still some limitations in applying our scheme to arbitrary datasets and performing arbitrary number of iterations on encrypted training data. In the future, we will extend our scheme to efficiently support P^2OLR with arbitrary number of iterations.

Appendix

(1) key_generation(*params*) $\longrightarrow$ $\{sk, pk, rk, gk\}$: Given the poly_modulus_degree N and coeff_modulus Q, it returns the secret_key sk, public_key pk, relinearization_key rk, galois_key gk.

(2) encode_double($\mathbf{x}$, Δ, $\mathbf{x}$): Given the message vector $\mathbf{x} \in \mathbb{C}^{N/2}$ and scaling factor Δ, it expands $\mathbf{x}$ to $\mathbb{H}$ by $\pi^{-1}(\mathbf{x})$, scales $\pi^{-1}(x)$ by $\Delta \cdot \pi^{-1}(\mathbf{x})$, and outputs the plaintext $\mathbf{x} = \sigma^{-1}(\Delta \cdot \pi^{-1}(\mathbf{x})) \in \mathcal{R}$.

(3) decode_double($\mathbf{x}$, $\mathbf{x}$): Given the plaintext $\mathbf{x}$, it computes
$\sigma \cdot \mathbf{x} = \sigma \cdot \sigma^{-1}(\Delta \cdot \pi^{-1}(\mathbf{x})) = \Delta \cdot \pi^{-1}(\mathbf{x}) \in \mathbb{H}$,
$\Delta^{-1} \cdot [\Delta\pi^{-1}(\mathbf{x}) \approx \pi^{-1}(\mathbf{x})]$, and outputs the message vector $\mathbf{x} = \pi \cdot \pi^{-1}(\mathbf{x}) \in \mathbb{C}^{N/2}$.

(4) encrypt ($\mathbf{x}$, $\mathbf{x}$): Given the plaintext $\mathbf{x}$, it encrypts $\mathbf{x}$ into a ciphertext $\mathbf{x}$, and outputs the ciphertext $\mathbf{x}$.

(5) decrypt ($\mathbf{x}$, $\mathbf{x}$): Given a ciphertext $\mathbf{x}$, it decrypts $\mathbf{x}$ into a plaintext $\mathbf{x}$, and outputs the plaintext $\mathbf{x}$.

(6) add ($\mathbf{x}$, $\mathbf{y}$, $\mathbf{x}+\mathbf{y}$): Given two ciphertexts $\mathbf{x}$ and $\mathbf{y}$, it computes $\mathbf{x}+\mathbf{y}$ and saves the result as a new ciphertext $\mathbf{x}+\mathbf{y}$.

(7) add_inplace($\mathbf{x}$, $\mathbf{y}$): Given two ciphertexts $\mathbf{x}$ and $\mathbf{y}$, it computes $\mathbf{x}+\mathbf{y}$ and saves the result in ciphertext $\mathbf{x}$.

(8) add_plain($\mathbf{x}$, $\mathbf{y}$, $\mathbf{x}+\mathbf{y}$): Given a ciphertext $\mathbf{x}$ and a plaintext $\mathbf{y}$, it computes $\mathbf{x}+\mathbf{y}$ and saves the result as a new ciphertext $\mathbf{x}+\mathbf{y}$.

(9) sub($\mathbf{x}$, $\mathbf{y}$, $\mathbf{x}-\mathbf{y}$): Given two ciphertexts $\mathbf{x}$ and $\mathbf{y}$, it computes $\mathbf{x}-\mathbf{y}$ and saves the result as a new ciphertext $\mathbf{x}-\mathbf{y}$.

(10) multiply($\mathbf{x}$, $\mathbf{y}$, $\mathbf{x}*\mathbf{y}$): Given two ciphertexts $\mathbf{x}$ and $\mathbf{y}$, it computes $\mathbf{x}*\mathbf{y}$ and saves the result as a new ciphertext $\mathbf{x}*\mathbf{y}$.

(11) multiply_plain($\mathbf{x}$, $\mathbf{y}$, $\mathbf{x}*\mathbf{y}$): Given a ciphertext $\mathbf{x}$ and a plaintext $\mathbf{y}$, it computes $\mathbf{x}*\mathbf{y}$ and saves the result as a new ciphertext $\mathbf{x}*\mathbf{y}$.

(12) mod_switch_to_inplace($\mathbf{x}\backslash$ $\mathbf{x}$, $\mathbf{y}$.parms_id()): Given a ciphertext/plaintext $\mathbf{x}\backslash$ $\mathbf{x}$ and a levels $\mathbf{y}$.parms_id() of ciphertext $\mathbf{y}$, it switches the levels of $\mathbf{x}\backslash$ $\mathbf{x}$ to $\mathbf{y}$.parms_id().

(13) relinearize_inplace($\mathbf{x}$, rk): Given a ciphertext $\mathbf{x}$ and a relinearization_key rk, it relinearizes $\mathbf{x}$ and saves the result in ciphertext $\mathbf{x}$.

(14) rescale_to_next_inplace($\mathbf{x}$): Given a ciphertext $\mathbf{x}$, it switches the modulo of $\mathbf{x}$ to the next levels, reduces the length of the plaintext accordingly, and saves the result in ciphertext $\mathbf{x}$.

(15) set_scale(Δ): Given a scaling factor Δ, it scales the ciphertext $\mathbf{x}$ by computing $\mathbf{x}$.set_scale(Δ), and outputs the ciphertext $\mathbf{x}$.

(16) rotate_vector($\mathbf{x}$, k, gk, $\mathbf{y}$): Given a ciphertext $\mathbf{x} = [x_0, x_1, \cdots, x_{N/2-1}]$, a rotation value k, and galois_key gk, it rotates $\mathbf{x}$ left by k, and saves the result as a new ciphertext $\mathbf{y} = [x_k, x_{k+1}, \cdots, x_{N/2-1}, x_0, x_1, \cdots, x_{k-1}]$.

Data Availability

Previously reported Umaru Impact Study, Myocardial Infarction dataset from Edinburgh and Nhanes III datasets were used to support this study and are available at https://doi.org/10.1186/s12920-018-0401-7. These prior studies (and datasets) are cited at relevant places within the text as references [18].

Conflicts of Interest

The authors declare no conflicts of interest.

Acknowledgments

This work was supported by the National Natural Science Foundation of China under Grant no. U19B2021.

References

[1] Y. Jiang, J. Hamer, C. Wang et al., "SecureLR: secure logistic regression model via a hybrid cryptographic protocol," *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, vol. 16, no. 1, pp. 113–123, 2019.

[2] V. V. P. Wibowo, Z. Rustam, A. R. Laeli, and A. A. Said, "Logistic regression and logistic regression-genetic algorithm for classification of liver cancer data," in *Proceedings of theInternational Conference on Decision Aid Sciences and Application*, pp. 244–248, Sakheer, Bahrain, December 2021.

[3] B. Liu, L. Lu, Q. Zeng, and Y. Li, "Implementation of credit scoring card model based on logistic regression and lightgbm," in *Proceedings of the International Conference on Control Science and Electric Power Systems*, pp. 175–178, Shanghai, China, May 2021.

[4] Z. Han, L. Lu, and H. Liu, "A differential privacy preserving approach for logistic regression in genome-wide association studies," in *Proceedings of the International Conference on Networking and Network Applications*, pp. 181–185, Daegu, Korea (South), October, 2019.

[5] X. Jiang, M. Kim, K. Lauter, and Y. Song, "Secure outsourced matrix computation and application to neural networks," in *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security*, pp. 1–23, Toronto, Canada, October 2018.

[6] J. M. Cortas-Mendoza, A. Tchernykh, M. Babenko, L. B. Pulido-Gaytan, and A. Avetisyan, "Privacy-preserving logistic regression as a cloud service based on residue number system," in *Proceedings of the 6th Russian Supercomputing Days*, pp. 598–610, Moscow, Russia, September 2020.

[7] P. Mohassel and Y. Zhang, "SecureML: "A system for scalable privacy-preserving machine learning," in *Proceedings of the IEEE Symposium on Security and Privacy*, pp. 19–38, San Jose, CA, USA, May 2017.

[8] J. Feng, L. Liu, Q. Pei, and K. Li, "Min-max cost optimization for efficient hierarchical federated learning in wireless edge networks," *IEEE Transactions on Parallel and Distributed Systems*, p. 1, 2022.

[9] J. Feng, W. Zhang, Q. Pei, J. Wu, and X. Lin, "Heterogeneous computation and resource allocation for wireless powered federated edge learning systems," *IEEE Transactions on Communications*, vol. 70, no. 5, pp. 3220–3233, 2022.

[10] S. Mao, L. Liu, N. Zhang et al., "Reconfigurable intelligent surface-assisted secure mobile edge computing networks," *IEEE Transactions on Vehicular Technology*, p. 1, 2022.

[11] A. C. Yao, "Protocols for secure computations," in *Proceedings of the 23rd Annual IEEE Symposium on Foundations of Computer Science*, pp. 1–5, Chicago, Illinois, USA, November 1982.

[12] C. Gentry, "Fully homomorphic encryption using ideal lattices," in *Proceedings of the 41st Symposium on Theory of Computing*, pp. 169–178, Bethesda, Maryland, USA, June 2009.

[13] B. Charlotte and V. Frederik, "Privacy-preserving logistic regression training," *BMC Medical Genomics*, vol. 11, no. 4, pp. 13–21, 2018.

[14] H. Chen, R. Gilad-Bachrach, K. Han et al., "Logistic regression over encrypted data from fully homomorphic encryption," *BMC Medical Genomics*, vol. 11, no. S4, p. 81, 2018.

[15] Z. Li and M. Sun, "Privacy-preserving classification of personal data with fully homomorphic encryption: an application to high-quality ionospheric data prediction," *Machine Learning for Cyber Security*, pp. 437–446, 2020.

[16] S. Carpov, N. Gama, M. Georgieva, and J. R. Troncoso-Pastoriza, "Privacy-preserving semi-parallel logistic regression training with fully homomorphic encryption," *BMC Medical Genomics*, vol. 13, no. S7, p. 88, 2020.

[17] 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. e19, 2018.

[18] A. Kim, Y. Song, M. Kim, K. Lee, and J. H. Cheon, "Logistic regression model training based on the approximate homomorphic encryption," *BMC Medical Genomics*, vol. 11, no. S4, p. 83, 2018.

[19] J. H. Cheon, D. Kim, Y. Kim, and Y. Song, "Ensemble method for privacy-preserving logistic regression based on homomorphic encryption," *IEEE Access*, vol. 6, pp. 46938–46948, 2018.

[20] F. Bergamaschi, S. Halevi, T. T. Halevi, and H. Hunt, "Homomorphic training of 30,000 logistic regression models," *Applied Cryptography and Network Security*, vol. 11464, pp. 592–611, 2019.

[21] Q. Wei, Q. Li, Z. Zhou, Z. Ge, and Y. Zhang, "Privacy-preserving two-parties logistic regression on vertically partitioned data using asynchronous gradient sharing," *Peer-to-Peer Networking and Applications*, vol. 14, no. 3, pp. 1379–1387, 2020.

[22] Y. Fan, J. Bai, X. Lei et al., "Privacy preserving based logistic regression on big data," *Journal of Network and Computer Applications*, vol. 171, p. 102769, 2020.

[23] J. H. Cheon, A. Kim, M. Kim, and Y. Song, "Homomorphic encryption for arithmetic of approximate numbers," in *Proceedings of the Advances in Cryptology - ASIACRYPT 2017: 23rd International Conference on the Theory and Application of Cryptology and Information Security*, vol. 10624, pp. 409–437, Hong Kong, China, December 2017.

[24] J. Fan and F. Vercauteren, "Somewhat practical fully homomorphic encryption", IACR Cryptology Eprint Archive," 2012, https://eprint.iacr.org/2012/144.

[25] "FV-NFLlib," 2016, https://github.com/CryptoExperts/FV-NFLlib.

[26] "Seal," 2021, https://github.com/microsoft/SEAL.

[27] C. Ilaria, N. Gama, M. Georgieva, and M. Izabachne, "Faster fully homomorphic encryption: bootstrapping in less than 0.1 seconds," in *Proceedings of the Advances in Cryptology ASIACRYPT 2016: 22nd International Conference on the Theory and Application of Cryptology and Information Security*, vol. 10031, pp. 3–33, Hanoi, Vietnam, December 2016.

[28] Z. Brakerski, C. Gentry, and V. Vaikuntanathan, "(Leveled) fully homomorphic encryption without bootstrapping," in *Proceedings of the 3rd Innovations in Theoretical Computer Science*, pp. 309–325, New Yark, USA, June 2012.

[29] "HElib," 2021, https://github.com/homenc/HElib.

[30] C. Boura, N. Gama, M. Georgieva, and D. Jetchev, "Chimera: combining ring-lwe-based fully homomorphic encryption schemes," *Journal of Mathematical Cryptology*, vol. 14, no. 1, pp. 316–338, 2020.

[31] I. Chillotti, N. Gama, M. Georgieva, and M. Izabachène, "TFHE: fast fully homomorphic encryption over the torus," *Journal of Cryptology*, vol. 33, no. 1, pp. 34–91, 2020.

[32] "Tfhe," 2021, https://github.com/tfhe/tfhe.

[33] "Heaan," 2022, https://github.com/snucrypto/HEAAN.

[34] "Heaan," 2019, https://github.com/kimandrik/HEAAN.

[35] "Helr," 2019, https://github.com/K-miran/HELR.

[36] "Heml," 2018, https://github.com/kimandrik/IDASH2017.

[37] R. Küsters, A. Datta, J. C. Mitchell, and A. Ramanathan, "On the relationships between notions of simulation-based security," *Journal of Cryptology*, vol. 21, no. 4, pp. 492–546, 2008.