Journal of Information Hiding and Privacy Protection
DOI:10.32604/jihpp.2021.026944
Article

Verifiable Privacy-Preserving Neural Network on Encrypted Data

Yichuan Liu[1], Chungen Xu[1,*], Lei Xu[1], Lin Mei[1], Xing Zhang[2] and Cong Zuo[3]

[1]Nanjing University of Science and Technology, Nanjing, 210014, China
[2]Jiangsu University, Zhenjiang, 212013, China
[3]Nanyang Technological University, 639798, Singapore
[*]Corresponding Author: Chungen Xu. Email: xuchung@njust.edu.cn
Received: 01 January 2022; Accepted: 02 March 2022

Abstract: The widespread acceptance of machine learning, particularly of neural networks leads to great success in many areas, such as recommender systems, medical predictions, and recognition. It is becoming possible for any individual with a personal electronic device and Internet access to complete complex machine learning tasks using cloud servers. However, it must be taken into consideration that the data from clients may be exposed to cloud servers. Recent work to preserve data confidentiality has allowed for the outsourcing of services using homomorphic encryption schemes. But these architectures are based on honest but curious cloud servers, which are unable to tell whether cloud servers have completed the computation delegated to the cloud server. This paper proposes a verifiable neural network framework which focuses on solving the problem of data confidentiality and training integrity in machine learning. Specifically, we first leverage homomorphic encryption and extended diagonal packing method to realize a privacy-preserving neural network model efficiently, it enables the user training over encrypted data, thereby protecting the user's private data. Then, considering the problem that malicious cloud servers are likely to return a wrong result for saving cost, we also integrate a training validation modular Proof-of-Learning, a strategy for verifying the correctness of computations performed during training. Moreover, we introduce practical byzantine fault tolerance to complete the verification progress without a verifiable center. Finally, we conduct a series of experiments to evaluate the performance of the proposed framework, the results show that our construction supports the verifiable training of PPNN based on HE without introducing much computational cost.

Keywords: Homomorphic encryption; verifiable neural network; privacy-preserving; secure computation

1 Introduction

As artificial intelligence enters the public field of vision, machine learning is playing an increasingly important role in the lives of people around the world. However, deep learning models typically require additional parameters for low tolerance levels and large models are trained on large datasets. For instance, the deep learning model ResNet-50 has 15 million parameters, and the dataset ImageNet has 14 million images. Training ResNet-50 on ImageNet using a single NVIDIA M40 GPU takes about two weeks. For a personal computer or a general server, this would be too computationally heavy to afford.

The cloud environment provides a way to perform complex tasks without requiring local machines with powerful computing capabilities. Because of its flexibility and economy, the cloud computing paradigm is becoming more and more popular among enterprises and organizations. While driving these costs down, cloud technology is also giving rise to security threats. Since machine learning is outsourced to a cloud server, the cloud server has access to the training data. It is difficult to prevent the cloud server from accessing the data and sharing it with others. In the case that the data owner (or the user) could not keep control of the data. Searchable encryption [1,2] is used to preserve the privacy data (such as medical data) on the cloud server. The technology is also applied to Internet of Things [3,4]. But it is insufficient to support the evaluation on encrypted data. Privacy-preserving machine learning (PPML) is proposed to solve the problem by completing the training on encrypted data or perturbed data.

PPML relies on cryptographic techniques, such as Homomorphic Encryption (HE), Garbled Circuits (GC), and Differential Privacy (DP), which ensure that data is not exposed to cloud servers. But GC-based solutions usually limit the number of participants and require frequent communication. Because data owner is not expected to carry out too much computation or be online for a long time, GC is not an ideal choice in real situations. DP reduces the availability of the model due to the introduction of perturbed data. Additive homomorphic encryption is used to realize PPML [5] but it is hard to accomplish the complex training progress. Therefore, we select HE as the technique to protect the privacy of sensitive data.

We first use HE to encrypt data and send it to the cloud server. The cloud server can train the model without accessing the raw data. Since the parameters generated in the training phase are also encrypted, the cloud server learns nothing about the raw data and the model parameters. The diagonal method is not only used to improve efficiency in the prediction phase, but also is extended to the more general matrix to perform the training progress.

However, this method is not able to ensure the integrity of the training. An undesired result is likely to be returned to the user as a result since a cloud server attempts to reduce computational cost. A verification algorithm that can be applied to machine learning is expected to arise. Jia et al. [6] designed a strategy Proof-of-Learning (PoL) to prove that cloud servers have performed the computational tasks assigned during the training phase. In their strategy, a prover is required to generate a proof that a verifier could verify the correctness of the computation performed during the training phase by the prover. This strategy can be applied to various machine learning models that use the gradient descent method to update their parameters.

Table 1: Comparison of representative privacy-preserving approaches and verification methods in deep neural networks

Proposed Work	Privacy	Integrity	Model	Approach
Abadi et al. [7]	training/inference	no	DNN	DP
Yu et al. [8]	training/inference	no	DNN	DP
Deepsecure [9]	training/inference	no	DNN	GC
Nandakumar et al. [10]	training/inference	no	MLP	HE
NN-EMD [11]	training/inference	no	MLP	FE
SecureML [12]	training/inference	no	MLP	MPC
CryptoNN [13]	training/inference	no	CNN	FE
Glyph [14]	training/inference	no	MLP	HE
Madi et al. [15]	inference	inference	MLP	HE
vCNN [16]	inference	inference	CNN	zk-SNARKs
VeriML [17]	inference	inference	CNN	zk-SNARKs
VPPNN (our work)	inference	training	MLP	HE

We combine the PPNN based on HE with PoL to complete the security and validation tasks. PoL makes us verify the correctness of computation performed during training. We summarize existing privacy-preserving neural network in Table 1. To the best of our knowledge, this is the first work to not only protect the privacy of the data but provide guarantees of the training integrity.

The remainder of this paper is organized as follows: In Section 2, we discuss related work. In Section 3, we give mathematical definitions and a brief description of the CKKS scheme. In Section 4, we give two system models to perform the verification tasks. In Section 5, we make a detailed description of our VPPNN algorithm. In Section 6, real datasets are used to train VPPNN and the results of the experiments are illustrated. We conclude the paper in Section 7.

2 Related Work

Many investigations on security in the context of machine learning have concentrated on the confidentiality of data and the integrity of model predictions. Because of the inefficiency of homomorphic encryption, there are not so many approaches based on homomorphic encryption that ensure the privacy of the training data. We focus the work which realize the training utilizing homomorphic encryption.

Hesamifard et al. [18] presented the first approach to training neural network models using fully homomorphic encryption. As a way to avoid the heavy computation cost of bootstrapping, the server requires the client to decrypt the ciphertext whose noise level is about to the budget and re-encrypt it to get refresh ciphertext used to continue to evaluate. The fact that multiple rounds of communication are required during computation does not meet our expectations for noninteractive computation. Nandakumar et al. [10] proposed a fully homomorphic deep learning method for training in a noninteractive way firstly. They used a solution by Crawford et al. [19] to get a low-precision approximation of required functions. Lou et al. [14] used the switching method between TFHE and CKKS proposed by [20] and transfer learning on DNN training to improve test accuracy and reduce the number of MACs between ciphertext and ciphertext in convolutional layers.

For verifiable computation in machine learning, most work focus on prove the correctness of the result returned by the prediction model. Ghodsi et al. [21] proposed an interactive proof protocol requiring multiple interactions. The protocol reduces the cost of computation at the expense of lower accuracy. Madi et al. [15] used the homomorphic hash functions to verify the first few layers of a PPNN based on Lee et al. [16] proposed their efficient methods to give guarantees to the convolutional neural network (CNN) by means of zero-knowledge Succinct Non-Interactive Arguments of Knowledge (zk-SNARKs). Zhao et al. [17] achieved outcome verifiability in support vector machine prediction services. These verifiable approaches are either unable to protect the data or provide a complete way to verify the correctness of the whole training computations. And in these papers, it is necessary to modify the algorithm for meeting the verifying requirement.

3 Preliminaries

3.1 Basic Notations

We denote vectors in bold lowercase, e.g., $\mathbf{x}$, denote matrixes in bold capital, e.g., $\mathbf{A}$. For two vectors $\mathbf{a}, \mathbf{b}, \langle \mathbf{a}, \mathbf{b} \rangle$ denotes the dot product of $\mathbf{a}$ and $\mathbf{b}$. For a real number r, $\lfloor r \rceil$ denotes the nearest integer to r, $\lfloor r \rfloor$ denotes the largest integer less than or equal to r, $\lceil r \rceil$ denotes the smallest integer greater than r. For a power-of-two integer M, let $\Phi_M(X)$ be the M-th cyclotomic polynomial of degree $N = \phi(M)$. Let $\mathcal{R} = \mathbb{Z}[X]/(\Phi_M)$ be the ring of integer of $\mathbb{Q}[X]/\big(\Phi_M(X)\big)$. We denote $\mathcal{R}_q = \mathcal{R}/q\mathcal{R}$. We use $x \leftarrow D$ to represent the sampling x from a distribution D.

3.2 Homomorphic Encryption

Homomorphic encryption is a cryptography technology that allows a user to evaluate on encrypted data without decrypting it and generate a result that matches the value evaluated on the raw data. To put it another way, the encryption algorithm is a homomorphism between the plaintext space and the ciphertext space. The slight difference is that the homomorphic encryption scheme adds noise to the plaintext for security. This technology allows the evaluator to perform calculations without knowing the decryption key or the original plaintext data, and the data owner can delegate the calculations to untrusted cloud servers.

Cheon et al. [22] presented a method to construct an FHE scheme supporting the arithmetic computation of approximate numbers in an encrypted state. Most HE schemes need to apply some noise to hide messages for security. Cheon et al. [22] assumed encrypted numbers have initial precision, and the noise generated during the homomorphic evaluation keeps smaller than the precision. The CKKS scheme considers the noise added to the plaintext to be a part of the error arising during homomorphic computations. Then the error would be reduced by rescaling, an evaluation that manages the magnitude of plaintext. By the way, the decryptor could gain an approximate value with definite precision. We could express the homomorphic encryption as follows:

$$\mathrm{Eval}(\mathrm{Enc}(m_0), \cdots, \mathrm{Enc}(m_n)) = \mathrm{Enc}(\mathrm{Eval}(m_0, \cdots, m_n))$$

In the CKKS scheme, several complex numbers are allowed to encode a single element in the message space $\mathcal{R}$. This lets us view plaintext as a vector of complex numbers. All ciphertexts are viewed as an encrypted array of fixed-point numbers unless otherwise indicated. The ability to encode multiple numbers into a single ciphertext not only reduces the time take to encrypt messages, but also makes us have the ability to perform the computation more efficiently by Single Instruction Multiple Data (SIMD) structure.

In this paper, we mainly consider four homomorphic operations: additions, multiplications, rotations and scale-multiplications. For packed ciphertext $\boldsymbol{x_0}, \boldsymbol{x_1}$, we let $\mathrm{Add}(\boldsymbol{x_0}, \boldsymbol{x_1})$ or $+$ denote the homomorphic addition, let $\mathrm{Mult}(\mathbf{x_0}, \mathbf{x_1})$ or $\times$ denote the homomorphic multiplication. We denote $\mathrm{Rot}(\boldsymbol{x}; i)$ packed vector x rotated to left direction by i slots. We treat a ciphertext as a vector for convenience. That is, $\boldsymbol{x}$ indicates a ciphertext encoded from a vector. The relinearisation and rescale technologies are performed after homomorphic multiplication. Both technologies are ignored in next section for convenience.

3.3 Diagonal Packing Method

In the CKKS scheme, packing is used to encrypt multiple elements to a single ciphertext to speed up the matrix-vector multiplication. A naïve method to complete the linear transformation of x is to pack each row of the matrix into one ciphertext and to compute the inner product between the ciphertext of the row and the vector x. Then, it is necessary to compute the multiplication of the inner product and a unit vector (only one element 1 and the others 0) to obtain the intermediate results. Finally, all intermediate results are rotated and summed to get the result. Fig. 1 gives an intuitive understanding. The left side describes the above row packing. The entries in a single color live in the same ciphertext. The entries in the vector x are filled with different colors to show how the vector is rotated. The blank entries mean the corresponding element is zero.

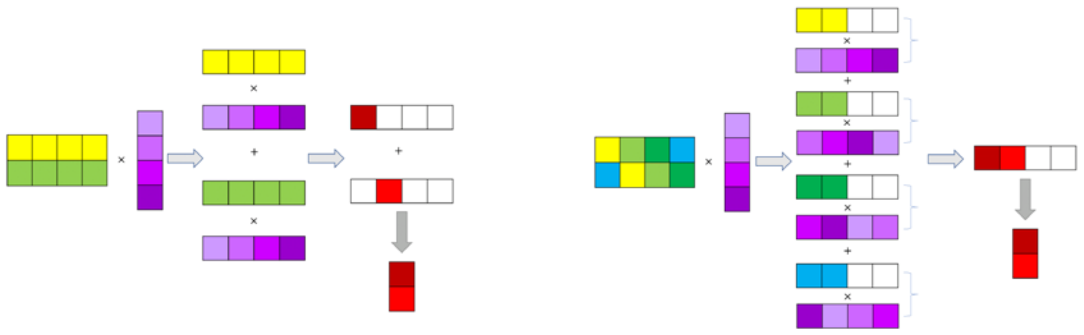

Figure 1: Linear transformation with row packing and diagonal packing

GAZELLE [23] proposed the diagonal packing method. We give a brief description of the diagonal packing method. As shown in Fig. 1 on the right, we multiply the packed diagonal component of the matrix with the rotated vector x and sum intermediate results to get the result. For a $M \times N$ matrix, the diagonal packing method reduces the number of rotations from $\mathrm{M} \times \log_2 N$ times to M times and avoid the multiplication between ciphertext and the unit vector. The multiplications and the rotations are heavy computations we need to consider. So the diagonal packing method is more efficient than the row packing method.

Algorithm 1 Matrix-vector multiplication with diagonal packing

```
Input: Encrypted Matrix $\mathbf{A}_{M\times N} = \{\mathbf{a}_j, 0 \le j \le N-1\}, \mathbf{a_j} = \{a_{0j}, a_{1(j+1)}, \cdots, a_{(M-1)((j+M-1)\%N)}\}$, Input Vector $\mathbf{x}$,,
Output: $\mathbf{y}$
1: for $i = 0$ to $N-1$do
2:     $\mathbf{y}_i \leftarrow$ Mult $(\mathbf{a_i}, \mathrm{Rot}(\mathbf{x}; i))$
3:     if $i = 0$ then
4:         $\mathbf{y} \leftarrow \mathbf{y}_i$
5:     else
6:         $\mathbf{y} \leftarrow \mathbf{y} + \mathbf{y}_i$
7:     end if
8: end for
9: return $\mathbf{y}$
```

4 System Model

In this section, we assume a distributed system where nodes are connected by a network. Each node corresponds to a cloud server. Fig. 2 illustrates our system architecture. The data owner chooses to outsource the training task to several cloud servers. The data owner is not willing to expose the training data and the model parameters to the cloud server due to privacy and interest considerations. It is also not expected that cloud server infers sensitive information through intermediate values.

The distributed system may undergo Byzantine failures [24], i.e., completely arbitrary behaviors of some cloud servers involved. We assume most of computer services are honest but curious. That is, they faithfully follow our protocol, but they are interested in infer some secret information. For instance, the unencrypted parameters, the feature vector and target value of training samples. On the other hand, part of computer servers is malicious, these malicious servers can return corrupted result to the data owner for saving cost. We could call them Byzantine devices. The assumption is under practical consideration that most cloud servers' providers are well-known company. They should not be motivated to return corrupted results because this hurts reputation. But a verify algorithm is still necessary for us. (Only if we can prove the server is malicious will the reputation of cloud servers' providers is affected.) The retrain is the heaviest computation in the verification progress, which can be considered as part of the neural network training algorithm. So the main computational tasks in the verification procedure are assigned to other cloud servers. We need to delegate the verification algorithm to multiple servers due to the possibility of collusion between malicious servers. We adopted a method is like Byzantine Fault Tolerant (BFT) [25] to perform the whole verification progress. We assume the number of malicious servers will not exceed $\left\lfloor \frac{n}{3} \right\rfloor$, n is the number of cloud servers.

Next, we give a description of the whole progress:

- CS_i performs the training task using stochastic gradient descent (SGD), generates a PoL and send it to DO.
- DO receives the PoL and verifies the validity of the signature. After confirming the identity of the sender, DO multicasts a request to others node for verifying the PoL.

- cs_j executes the request and send the result to DO.
- DO checks the validity of the received result, the PoL validity is done if more than $\lfloor\frac{n}{3}\rfloor$ different replicas are proved validity.

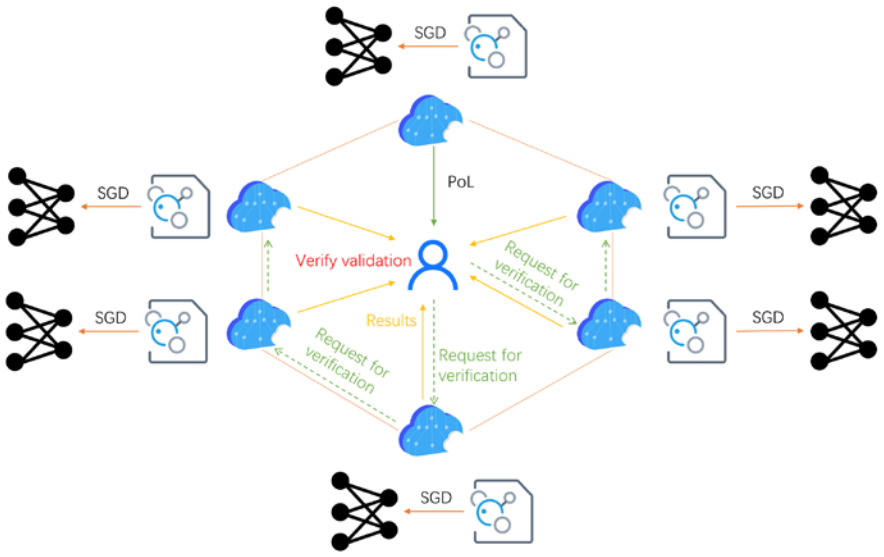

Figure 2: Byzantine-tolerant Verifiable Privacy-Preserving Neural Network

5 The Algorithms in VPPNN

5.1 PPNN

Artificial Neural Network or Neural Network is a kind of data-driven model that is hierarchical and non-linear architecture consisting of several layers, where each layer is composed of several neural units that receive the processed data from the previous layer and send data to the next layer. Each neural unit includes a linear transformation and a non-linear activation function. The basic structure of a neural unit is shown in Fig. 3. Such a structure makes it possible to extract abstract features from raw data through the non-linear function. The output of each node in the network applying a non-linear activation function to the weighted average of its inputs, which includes a bias term that always emits value 1. Similarly, each layer in the neural network consists of several units regards the results from the previous layer and send the performed data to the next layer.

As mentioned above, the operations which make up of the forward propagation include linear transformations and active functions. It is a necessary condition that two kinds of operations on encrypted data are completed.

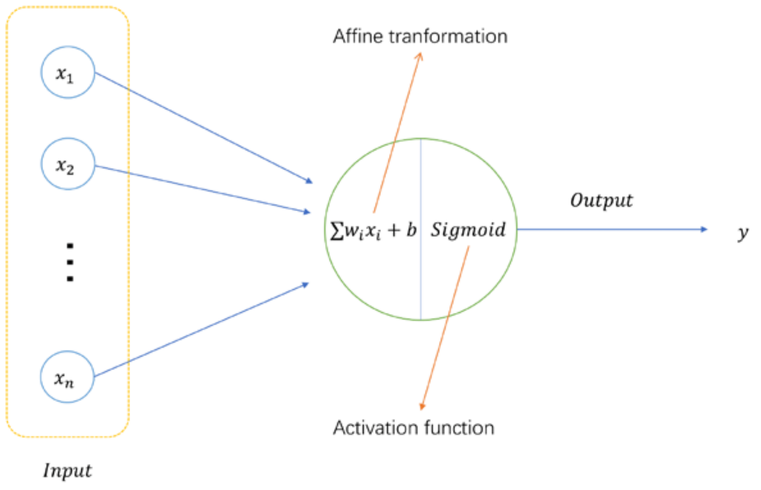

Figure 3: The neural unit model

Feedforward neural network (FNN) is the quintessential learning model, which is the model we used in this paper. In this FNN, each layer of the neural network carries out a linear transformation of the input vector, and executes the activation function for each element of the resulting vector, which is then output as the input to the next layer. This is not only what the neural network needs to perform in the prediction phase, but also the crucial component of the training. This progress usually refers to as forward propagation.

Since the CKKS scheme only supports the evaluation of polynomial functions, the evaluation of usual activation functions such as the Sigmoid function, the Tanh function, the Rectified Linear Unit, and so on, remains to be the biggest obstacle for evaluation. A polynomial is required to approximate the activation function. The Taylor polynomials that are used to approximate the activation function easily come to our minds. But it is necessary for obtaining the desired accuracy to arise the degree of Taylor polynomials. With certainty, too many multiplications make bootstrapping necessary which greatly hinders the implementation of the HE scheme. So the Taylor polynomial is not a good candidate for approximation because it is a local approximation near a certain point.

So, we use the least squares approximation method to substitute the Sigmoid function. That is, to solve the following problem:

$$Argmin \int_I \left(f(x) - \sigma(x)\right)^2$$

The function $\sigma(x)$ is Sigmoid function, I is the required interval. We can obtain polynomials $f_3(x)$ and $f_7(x)$:

$$f_3(x) = 0.5 + 0.15012x - 0.00159301x^3, \text{ where } x \in [-8,8]$$

$$f_7(x) = 0.5 + 0.21687x - 8.19154 \times 10^{-3}x^3 + 1.65833 \times 10^{-4}x^5 \\ -1.19562 \times 10^{-6}x^7, \text{ where } x \in [-8,8]$$

Besides the forward propagation, the backpropagation is another component in the neural network training. Backpropagation could be viewed as the reverse process of feedforward. It is necessary to compute linear transformations and partial derivatives of the loss with respect to parameters. The loss function and its derivative are easy to approximate by the above method. So we concentrate on how the linear transformations are accomplished in the backpropagation.

The GAZELLE [23] did not provide a method to complete the matrix-vector multiplication when the number of rows in the matrix are more than the number columns. So we propose a similar approach to solve the problem with diagonal packing. The main difference is that it is necessary to padding the vector $\boldsymbol{x}$ before the linear transformation. For example, $M = 8,\ N = 2,\ \boldsymbol{x} = [1, 2]$, what we do is obtain a new M-

dimensional vector $\boldsymbol{x}' = [1,2,1,2,1,2,1,2]$. The number of operations performed is the same as the number of ciphertexts corresponding to the matrix. The procedure of this evaluation is shown in Algorithm 3 and Fig. 4. Our research targets secure neural network training. We set the number of hidden layer neurons to a power of 2 for simplicity. Similarly, we fill the number of neurons in the input layer to a power of 2. When performing operations related to matrix $A_{M\times N}$, we think that both M and N are powers of 2.

In the backpropagation the matrix transpose is necessary. The goal is to transpose matrix $A_{M\times N}$ into matrix $B_{N\times M}$. Fortunately, the two kinds of matrix have similar structure because of the same packing method. What we need is merging several original "short" vectors into a "long" vector. As shown in Fig. 4 on the right, A consists of M ciphertexts, while B consists of N ciphertexts. The only thing we need to do is combine the $\lceil \mathrm{M/N} \rceil$ ciphertexts in A to one ciphertext in B and repeat N times because of the similar structures in two matrixes. There is no multiplication consumed in this progress. It makes us reduce the times of bootstrapping or use smaller parameters. The procedures of matrix transpose are shown in Algorithm 2.

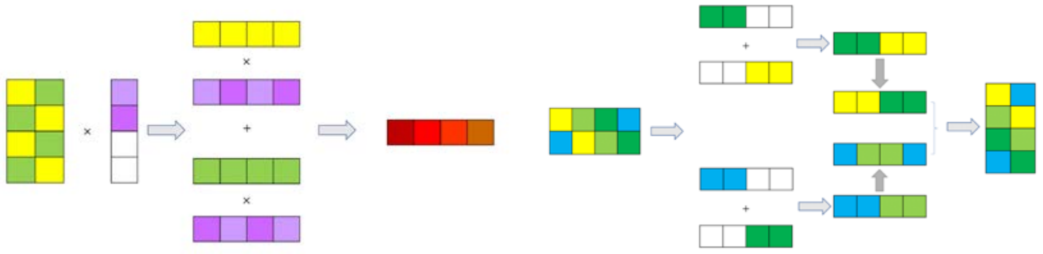

Figure 4: Linear Transformation and Transpose in the backpropagation

Algorithm 2 Transpose with diagonal packing

```
Input: Encrypted Matrix $\mathbf{A}_{M\times N} = \{\mathbf{a}_j, 0 \le j \le N-1\}, \mathbf{a}_j = \{a_{0j}, a_{1(j+1)}, \cdots, a_{(M-1)((j+M-1)\%N}, 0, \cdots, 0)\}$,,
Output: $\mathbf{B}_{N\times M} = \{\mathbf{b}_i, 0 \le i \le M-1\}, \mathbf{b}_i = \{a_{i0}, a_{i1}, \cdots, a_{iM}, \cdots, a((i-1)\%M)N\}$
1: for $i = 0$ to $M-1$ do
2:     for $j = 0$ to $\frac{N}{M} - 1$ do
3:         if $j = 0$ then
4:             $\mathbf{b}_i \leftarrow \mathbf{a}_{\mathbf{N-M+i}}$
5:         else
6:             $\mathbf{b}_i \leftarrow \mathbf{b}_\mathbf{i} + \mathrm{Rot}(\mathbf{a}_{i+(j-1)M}; -jM)$
7:         end if
8:     end for
9:     $\mathbf{b}_i = \mathrm{Rot}\ (\mathbf{b}_i; M-i)$
10: end for
11: return $\mathbf{B} = \{\mathbf{b}_i\}$
```

Algorithm 3 Matrix-vector multiplication with diagonal packing in backpropagation

```
Input: Encrypted Matrix $\mathbf{B}_{N\times M} = \{\mathbf{b}_i, 0 \le i \le i \le M-1\}$, $\mathbf{b}_i = \{b_{0i}, a_{1(i+1)}, \cdots, a_{(N-1)((j+N-1)\%M)}\}$, the N-dimensional Input Vector $\mathbf{x} =$
```

```
$\{x_0, x_1, \cdots, x_{M-1}, 0, \cdots, 0\}$,,
Output: $\mathbf{y}$
1: for $i = 0$ to $\left\lceil \log_2 \frac{N}{M} \right\rceil$ do
2:     $\mathbf{x} \leftarrow \mathbf{x} + \mathrm{Rot}(\mathbf{x}; -2^i)$
3: end for
4: for $i = 0$ to $M - 1$ do
5:     $\mathbf{y}_i \leftarrow$ Mult ($\mathbf{b}_i$, Rot ($\mathbf{x}; i$)
6:     if $i = 0$ then
7:         $y \leftarrow \mathbf{y}_i$
8:     else
9:         $\mathbf{y} \leftarrow \mathbf{y} + \mathbf{y}_i$
10:    end if
11: end for
12: return $\mathbf{y}$
```

Table 2 shows the number of homomorphic evaluations for linear transformation and matrix transpose. The matrix $A_{M \times N}$ corresponds to the parameter matrix of one layer of the neural network. It is oblivious that the number of rotations and scalar-multiplications is reduced by a large margin in the linear transformation. Diagonal method improves the efficiency of the matrix transposition more significantly. The improvement makes a positive effect on efficiency when applying a more complex model.

Packing method	Evaluation	Add	Rot	Mult	Scalar-Mult
Diagonal packing	LT in FF	$N-1$	$N-1$	N	1
	LT in BP	$M-1$	$M-1$	M	1
	transpose	$N-M$	$N-M$	0	0
Row packing	LT in FF	$M-1$	$M\lceil log\, N\rceil$	M	M
	LT in BP	$N-1$	$N\lceil log\, M\rceil$	N	N
	transpose	$MN-M$	$MN-M$	0	MN

Table 2: Computational complexity of linear transformation and matrix transpose

5.2 A PoL on the PPNN

Just like in the previous introduction, the data owner needs to delegate an untrusted cloud server to compute a function $f(\boldsymbol{x})$ and sends some data $\boldsymbol{x}$ that needs to be dealt with. But the data owner has no efficient way to prove the integrity of computations on authenticated data. Verifiable computation allows the client to delegate a cloud server to verify the correctness of the computation. The PoL proposed by Jia et al. [1] provides a great tool to complete the above task. The prover needs to generate a PoL in the training phase, as shown in Algorithm 4. The verifier executes the Algorithm 5, and the verification is passed if success is returned.

In the verification mechanism, $\mathcal{T}$ reveals to $\mathcal{V}$ some information during training as its PoL. The PoL includes:

- $\mathbb{W}$: the values of the weights encrypted by HE at periodic intervals during training.
- $\mathbb{I}$: the corresponding indices of the data points from the training set which are used to compute

said weights.

- $\mathbb{H}$: the signature of $\mathbb{I}$ with the prover's private key.
- $\mathbb{Y}$: the hyper-parameters.

The main idea of PoL is that the verifier can verify the correctness of the entire computational process by only recomputing the largest update due to the difficulty of inverting gradient descent. As illustrated by Jia et al. [6], any estimation error introduced by an adversary wishing to recreate a proof at a smaller computational cost would be easier to detect for these large model updates. They give the correctness analysis [6] of the gradient descent mechanism from the perspective of entropy growth.

Algorithm 4 Generating a PoL

```
Require: Encrypted dataset $E(D)$, Hyper-parameters $\mathbb{Y}$
Require: $E, S, l$ { the number of epochs, the number of steps per epoch, the length of
checkpointing interval }
Require: $\lambda, H${the given distribution, a hash function}
1: Init: $\mathbb{W} \leftarrow \{\}, \mathbb{I} \leftarrow \{\}, \mathbb{H} \leftarrow \{\}$
2: if $W_0$ then
3:     $W_0 \leftarrow \chi$
4: end if
5: for $i \leftarrow 0, \cdots, E-1$ do
6:     $I \leftarrow \text{getpatches}(E(D), S)$
7:     for $j \leftarrow 0, \cdots, S-1$ do
8:         $t \leftarrow e \cdot S + s$
9:         $W_{t+1} \leftarrow SGD(W_t, E(D)[I_s], Y_t)$
10:        $\mathbb{I} \leftarrow \mathbb{I} \cdot \text{add}(I_t)$
11:        $\mathbb{H} \leftarrow \mathbb{H} \cdot \text{add}\big(H(D[I_t])\big)$
12:        if $\text{t mod k} = 0$ then
13:            $\mathbb{W} \leftarrow \mathbb{W} \cdot \text{add}(W_t)$
14:        else
15:            $\mathbb{W} \leftarrow \mathbb{W} \cdot \text{add}(nil)$
16:        end if
17:    end for
18: end for
19: $\mathcal{P} \leftarrow (\mathbb{W}, \mathbb{I}, \mathbb{H}, \mathbb{Y})$
20: return $\mathcal{P}, H(\mathcal{P}, sk_{\mathcal{T}})$
```

A point to note is that the cloud computer is unable to sort the set of updates and comparison the ciphertext and the plaintext due to having no access to the private key of HE. So it is necessary to perform several rounds of interactions with the date owner. In the verification process, there are two parts that need to be compared:

1. Before the retraining, we firstly need to compare the difference of parameters.

2. After the retraining is completed, the training result needs to be compared with slack parameter to determine whether the verifying result is passed.

In these two steps, the user only needs to decrypt the ciphertext to obtain the plaintext, and send the compared result to the server. Normal electronic devices are also fully capable of performing the computation. Although data owner needs to perform part of verification work, it has a minimal impact on the personal computer in the entire verification procedure. We think the cost of verification algorithm is acceptable. Because this process does not involve overly complex interactions, for convenience, we treat it as an algorithm.

Algorithm 5 Verifying a PoL

```
Require: $\mathcal{P}, E(D), Q, \delta$ {the PoLs, the encrypted dataset, query budget, the limitation of tolerance}
1: $\mathbb{W}, \mathbb{I}, \mathbb{H}, \mathbb{Y} \leftarrow \mathcal{P}$
2: if InitializationVerification ($W_0$) = FAIL then
3:     return FAIL
4: end if
5: $i \leftarrow 0$ {the number of current epoch}
6: incre ← {}
7: for $j \leftarrow 0, \cdots, \left\lfloor \frac{T-1}{l} \right\rfloor$ do
8:     if $j \neq 0 \wedge j \bmod l \neq 0$ then
9:         $\text{incre}_i \leftarrow \text{incre}_i\text{.add}\left(d_1(W_t - W_{t-k})\right)$
10:    end if
11:    $i' = \left\lfloor \frac{j}{s} \right\rfloor$
12:    if $i' = i + 1$ then
13:        $DO$ compute incre $_i$ ← sorting ($\text{incre}_i$)
14:        incre ← incre.add (increi)
15:    end if
16: end for
17: for $k \leftarrow 1, \cdots, E$ do
18:     $\mathcal{V}$ executes Retrain ($\text{incre}_i$)
19:     if Retrain(idx) = FAIL then
20:         return FAIL
21:     end if
22: end for
23: return Success
```

In Algorithm 5, the InitializationVerification could prevent adversarial prover forging a PoL by the PoL generated by other honest trainers. For specific analysis, please refer to [6]. Generally, machine learning models are initialized by sampling randomly from a particular distribution. We can use the Kolmogorov-Smirnov (KS) test to check if the parameters are sampled from the particular distribution. The KS test [6] is a statistical test to check whether samples come from a specific distribution. In our scheme, since our distributed learning does not come from multiple data sources, the parameters can be given by the data owner together with the dataset.

Algorithm 6 Retrain

```
Require: $\mathcal{P}, f, E(D), Q, \delta$, idx
```

```
1: for $i \leftarrow 1, \cdots, Q$ do
2:     $t =$ incre [[$i - 1$]]
3:     VerifySignature $\left(H_t, E(D[I_t])\right)$
4:     $W'_t \leftarrow W_t$
5:     for $i \leftarrow 0, \cdots, k - 1$ do
6:         $SGD(W'_{t+i}, E(D[I_{t+i}]), M_{t+i})$
7:         $DO$ compute $d_2(W'_{t+k}, W_{t+k})$
8:         $CS$ send $d_2(W'_{t+k}, W_{t+k})$ to $DO$.
9:     end for
10:    if $d_2(W'_{t+k}, W_{t+k}) \leq \delta$ then
11:        return FAIL
12:    end if
13: end for
14: return Success
```

6 Security Analysis

Cloud servers only have access to encrypted data and parameters. It avoids data leakage caused by reconstruction attack or membership inference attack. The confidentiality of data and models completely depends on the security of the CKKS encryption scheme. But as mentioned in [26], the decryption results must not be published, or shared with anybody not completely trusted.

Next, we discuss how to determine the validity of the verification process. In our assumption the number of malicious servers will not exceed $\lfloor\frac{n}{3}\rfloor$, it validates successfully when more than $\lfloor\frac{n}{3}\rfloor$ verification results performed are valid. But it is so expensive that verifying cost is more than training. For saving cost, we could use the PoF as a seed and generate a random number to select the cloud servers that performing the verifying algorithm. For instance, we select 11 cloud servers as verifying servers when $n = 100$. When the results of most validation servers are consistent, the validation result is correct with 0.896 probability.

7 Experiments

Tests were done on Intel Core i5-10400F CPU, running at 2.90 GHz. The machine has 16 GB of main memory.

We worked with the cyclotomic ring $\mathbb{Z}[X]/X^M + 1$ with $M = 2^{15}$. It also allows to set the M as a smaller integer such as $M = 2^{10}$, which makes operations speed up more than twenty times. A smaller M makes homomorphic operations run faster but has a lower level which gives rise to more bootstrapping operations.

We constructed a neural network that consist of three hiding layers. Student Performance Data Set is used to train and test the VPPNN model. The evolution of test accuracy over multiple epochs is shown in Fig. 5 on the left.

In the verification procedure, we set the query budget Q as 1. It means that we only need to check the largest update only once. But the time spent on verifying mainly depends on initialization verification. The checkpointing interval is a hyperparameter of the proposed PoL method and is related to the storage cost. Its reciprocal can be viewed as the frequency of checking in the proof algorithm. As shown in Fig. 5 on the right, ratio of verifying cost to training cost is no more than 0.4. It is an efficient verification algorithm.

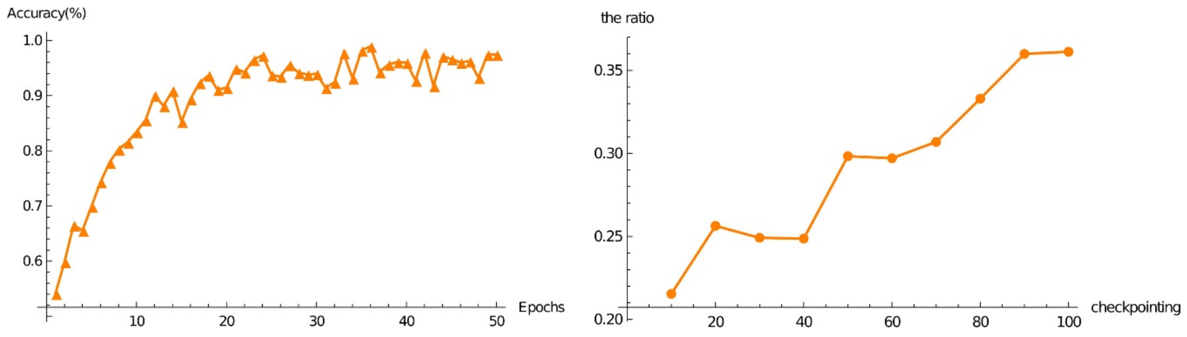

Figure 5: Classification accuracy and the ratio of verifying cost to training cost

8 Conclusion

In this paper, we present a privacy-preserving neural network framework using homomorphic encryption, which preserves the confidentiality of the training data and the parameters under several malicious cloud servers. We extend the diagonal packing method so that it can be used in the neural network training process. It reduces the number of multiplications and rotations so as to improves the efficiency of neural network. By combining PPML with PoL, we provide a guarantee that the computations assigned to each computer server are performed completely. This is our first attempt at protecting the confidentiality of the data and ensuring training integrity.

Funding Statement: The work is supported by the National Natural Science Foundation of China (No. 62072240), the National Natural Science Foundation of China (No. 61902156), and the Natural Science Foundation of Jiangsu Province under Grant BK20210330.

Conflicts of Interest: The authors declare that they have no conflicts of interest to report regarding the present study.

References

[1] L. Li, C. Xu, X. Yu, B. Dou and C. Zuo, "Searchable encryption with access control on keywords in multi-user setting," *Journal of Cyber Security*, vol. 2, no. 1, pp. 9–23, 2020.

[2] L. Xu, C. G. Xu, J. K. Liu, C. Zuo and P. Zhang, "Building a dynamic searchable encrypted medical database for multi-client," *Information Sciences*, vol. 527, pp. 394–405, 2020.

[3] L. Mei, C. Xu, L. Xu, X. Yu and C. Zuo, "Verifiable identity-based encryption with keyword search for IoT from lattice," *Computers, Materials & Continua*, vol. 68, no. 2, pp. 2299–2314, 2021.

[4] C. Xu, L. Mei, J. Cheng, Y. Zhao and C. Zuo, "IoT services: Realizing private real-time detection via authenticated conjunctive searchable encryption," *Journal of Cyber Security*, vol. 3, no. 1, pp. 55–67, 2021.

[5] D. P. Dong, Y. Wu, L. Z. Xiong and Z. H. Xia, "A privacy preserving deep linear regression scheme based on homomorphic encryption," *Journal on Big Data*, vol. 1, no. 3, pp. 145–150, 2019.

[6] H. Jia, M. Yaghini, C. A. Choquette-Choo, N. Dullerud, A. Thudi *et al.*, "Proof-of-learning: Definitions and practice," in *2021 IEEE Sym. on Security and Privacy (SP)*, pp. 1039–1056, 2019.

[7] M. Abadi, A. Chu, I. Goodfellow, H. B. McMahan, I. Mironov *et al.*, "Deep learning with differential privacy," in *Proc. of the 2016 ACM SIGSAC Conf. on Computer and Communications Security*, Association for Computing Machinery, New York, NY, USA, pp. 308–318, 2016.

[8] L. Yu, L. Liu, C. Pu, M. E. Gursoy and S. Truex, "Differentially private model publishing for deep learning," in *2019 IEEE Sym. on Security and Privacy (SP)*, pp. 332–349, 2019.

[9] B. D. Rouhani, M. S. Riazi and F. Koushanfar, "Deepsecure: scalable provably-secure deep learning," in *Proc. of the 55th Annual Design Automation Conf. (DAC'18)*, Association for Computing Machinery, New York, NY, USA, pp. 1–6, 2018.

[10] K. Nandakumar, N. K. Ratha, S. Pankanti and S. Halevi, “Towards deep neural network training on encrypted data,” in *IEEE Conf. on Computer Vision and Pattern Recognition Workshops*, Long Beach, CA, USA, pp. 40–48, 2019.

[11] R. Xu, J. Joshi and C. Li, “NN-EMD: Efficiently training neural networks using encrypted multi-sourced datasets,” [Online]. Available: https://arxiv.org/abs/2012.10547, 2020.

[12] P. Mohassel and Y. Zhang, “SecureML: A system for scalable privacy-preserving machine learning,” in *IEEE Sym. on Security and Privacy*, San Jose, CA, USA, pp. 19–38, 2017.

[13] R. Xu, J. B. D. Joshi and C. Li, “CryptoNN: Training neural networks over encrypted data,” in *39th IEEE Int. Conf. on Distributed Computing Systems*, Dallas, TX, USA, pp. 1199–1209, 2019.

[14] Q. Lou, B. Feng, G. C. Fox and L. Jiang, “Glyph: Fast and accurately training deep neural networks on encrypted data,” in *Advances in Neural Information Processing Systems 33: Annual Conf. on Neural Information Proc. Systems*, 2020.

[15] A. Madi, R. Sirdey and O. Stan, “Computing neural networks with homomorphic encryption and verifiable computing,” in *Int. Conf. on Applied Cryptography and Network Security*, Rome, Italy, vol. 12418, pp. 295–317, 2020.

[16] S. Lee, H. Ko, J. Kim and H. Oh, “vCNN: Verifiable convolutional neural network based on zk-SNARKs,” in *Cryptology ePrint Archive: Report 2020/584*, 2020.

[17] L. Zhao, Q. Wang, C. Wang, Q. Li, C. Shen *et al.*, “VeriML: Enabling integrity assurances and fair payments for machine learning as a service,” *IEEE Transactions on Parallel and Distributed Systems*, vol. 32, no. 10, pp. 2524–2540, 2021.

[18] E. Hesamifard, H. Takabi, M. Ghasemi and R. N. Wright, “Privacy-preserving machine learning as a service,” in *Proceedings on Privacy Enhancing Technologies*, vol. 2018, no. 3, pp. 123–142, 2018.

[19] J. L. H. Crawford, C. Gentry, S. Halevi, D. Platt and V. Shoup, “Doing real work with FHE: The case of logistic regression,” in *Proc. of the 6th Workshop on Encrypted Computing & Applied Homomorphic Cryptography (WAHC’18)*, Association for Computing Machinery, New York, NY, USA, pp. 1–12, 2018.

[20] 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.

[21] Z. Ghodsi, T. Gu and S. Garg, “SafetyNets: Verifiable execution of deep neural networks on an untrusted cloud,” in *Advances in Neural Information Processing Systems 30 (NIPS 2017)*, Long Beach, CA, USA, pp. 4672–4681, 2017.

[22] J. H. Cheon, A. Kim, M. Kim and Y. Song, “Homomorphic encryption for arithmetic of approximate numbers,” in *Int. Conf. on the Theory and Application of Cryptology and Information Security*, vol. 10624, pp. 409–437, 2017.

[23] C. Juvekar, V. Vaikuntanathan and A. Chandrakasan, “GAZELLE: A low latency framework for secure neural network inference,” in *27th USENIX Security Sym.*, Baltimore, MD, USA, pp. 1651–1669, 2018.

[24] L. Lamport, R. E. Shostak and M. C. Pease, “The Byzantine generals problem,” in *Concurrency: The Works of Leslie Lamport*, pp. 203–226, 2019.

[25] M. Castro and B. Liskov, “Practical byzantine fault tolerance,” in *Proc. of the Third USENIX Sym. on Operating Systems Design and Implementation (OSDI)*, New Orleans, Louisiana, USA, pp. 173–186, 2019.

[26] B. Li and D. Micciancio, “On the security of homomorphic encryption on approximate numbers,” in A. Canteaut, F. X. Standaert (eds.), *Advances in Cryptology–EUROCRYPT 2021. Lecture Notes in Computer Science*, Springer, Cham, vol. 12696, 2021.