Privacy-preserving and Communication-efficient Convolutional Neural Network Prediction Framework in Mobile Cloud Computing

Yanan Bai[1,2], Yong Feng[1], and Wenyuan Wu[1*]
[1]Chongqing Key Laboratory of Automated Reasoning and Cognition, Chongqing Institute of Green and Intelligent Technology, Chinese Academy Sciences
Chongqing, 400714, China
[2] University of Chinese Academy of Sciences.
Beijing, 100049, China
[e-mail:baiyanan@cigit.ac.cn]
*Corresponding author: Wenyuan Wu

Received September 26, 2021; accepted November 24, 2021;
published December 31, 2021

Abstract

Deep Learning as a Service (DLaaS), utilizing the cloud-based deep neural network models to provide customer prediction services, has been widely deployed on mobile cloud computing (MCC). Such services raise privacy concerns since customers need to send private data to untrusted service providers. In this paper, we devote ourselves to building an efficient protocol to classify users' images using the convolutional neural network (CNN) model trained and held by the server, while keeping both parties' data secure. Most previous solutions commonly employ homomorphic encryption schemes based on Ring Learning with Errors (RLWE) hardness or two-party secure computation protocols to achieve it. However, they have limitations on large communication overheads and costs in MCC. To address this issue, we present LeHE4SCNN, a scalable privacy-preserving and communication-efficient framework for CNN-based DLaaS. Firstly, we design a novel low-expansion rate homomorphic encryption scheme with packing and unpacking methods (LeHE). It supports fast homomorphic operations such as vector-matrix multiplication and addition. Then we propose a secure prediction framework for CNN. It employs the LeHE scheme to compute linear layers while exploiting the data shuffling technique to perform non-linear operations. Finally, we implement and evaluate LeHE4SCNN with various CNN models on a real-world dataset. Experimental results demonstrate the effectiveness and superiority of the LeHE4SCNN framework in terms of response time, usage cost, and communication overhead compared to the state-of-the-art methods in the mobile cloud computing environment.

Keywords: Privacy Preservation, Convolutional Neural Networks, Homomorphic Encryption, Mobile Cloud Computing, Deep Learning.

1. Introduction

Recent years have witnessed the tremendous success of deep learning (DL) in various domains, such as medical diagnosis [1], risk assessment [2], image recognition [3], due to its strong capability in solving many complex tasks while achieving superior performance. Meanwhile, with the exponential development of intelligent mobile devices, mobile cloud computing (MCC) emerged as a powerful computing paradigm that makes various complex tasks feasible on a single mobile device [4]. It has elastic scalability and broad accessibility characteristics and provides on-demand self-service and pay-as-you-go services [5]. In this context, cloud providers (e.g., Amazon, Google, Alibaba) are now providing deep learning as a service (DLaaS) that offers DL model training and prediction APIs for users, enabling them by paying specific fees to enjoy the intelligent and convenient benefits of life [6]. Take a healthy ecosystem in the MCC scenario, for example, with the help of the DLaaS-based healthcare service provided by Google[1] or Microsoft[2], patients upload their photos of uncomfortable body parts to obtain more accurate medical diagnoses via the healthcare APIs installed on their mobile devices. It reduces the time wasted on traveling to the hospital and waiting in the hospital. However, a natural problem about privacy preservation in such services is raised that the inputs and the analysis results are disclosed to the service provider, breaching the privacy of sensitive user data.

Although the DLaaS providers claim that they will never leak users' data for commercial purposes, the increasing number of user data breaches alert us that there is no guarantee of what they promised [7]. A clear answer to protect users' privacy is to give users the privilege to download models from the server and run them locally on their mobile devices. Nevertheless, this solution is unsatisfactory in the real world. First, these models are thought of as the intellectual property of the model-holders and must be confidential. Second, publishing models may reveal information about the underlying training dataset, which includes a massive amount of private data, e.g., a disease prediction model is trained by many individual medical records. Third, mobile devices are usually configured with limited processing resources, storage, and electric quantity, which may not normally support the DL model's running locally. Therefore, our work aims to design a privacy-preserving DLaaS framework in the MCC environment, where both service providers and mobile device users can provide their private data safely.

Several protocols or frameworks have been proposed based on various secure computing technologies, such as secret sharing (SS), homomorphic encryption (HE), Yao's Garbled Circuit (GC), Goldreich-Micali-Wigderson (GMW), etc., to tackle this problem. These methods offer a promising way to reduce the computational complexity significantly when developing the privacy-preserving CNN prediction model. They may lead to an enormous communication overhead since they heavily focus on employing the RLWE based HE schemes to perform the secure linear layers, and using two-party computation protocols to execute non-linear layers, which need immense rounds of interactions between user and service provider.

In the MCC environment, response time and usage cost are two important indexes to evaluate users' satisfaction with DLaaS services. They are mainly determined by both parties' computation latency and communication overhead, also limited to the unstable mobile data

[1] https://en.wikipedia.org/wiki/Google_Health
[2] https://health.d365industrypartners.com

transfer speed, which is decided by the status of network load and user's location. Unfortunately, the enormous communication overheads of the existing methods consume more bandwidth, leading to an additional and inevitable usage cost for mobile users [8], and extended response time degrade the service's user experience. They constitute barriers between mobile device users and cloud service providers.

This paper introduces LeHE4SCNN, a practical realization of privacy-preserving and communication-efficient CNN prediction framework. Typically, CNN is one of the most popular neural network architectures in DL. Convolutional and fully connected layers have linear properties, so we call them linear layers, while activation and pooling are non-linear layers. Our contributions are as follows:

- We propose a low-expansion rate homomorphic encryption scheme with novel packing and unpacking methods, based on the module learning with errors (MLWE) assumption. It allows homomorphic addition, inner product between two vectors, vector-matrix, and matrix-matrix multiplication operations. We derive the bound of accumulated noise and present several significant theoretical analyses and proofs, including correctness, security, and ciphertext expansion rate.
- We present LeHE4SCNN, a piracy-preserving and communication-efficient CNN prediction framework in the MCC environment, to ensure computation security and low communication overheads between mobile users and cloud service providers. It adopts the proposed LeHE scheme to secure the linear layers and exploits the data shuffling technique to secure the non-linear layers.
- We implement LeHE4SCNN with varied CNN models and test its performance on the industrial dataset. Most importantly, we evaluate the response time and usage cost of LeHE4SCNN with other competitors in the MCC environment. Experimental results show the superiority and effectiveness of LeHE4SCNN in the MCC environment.

The rest of this paper is organized as follows. Section 2 gives the related work and preliminaries. We describe the LeHE scheme with novel packing and unpacking methods in Section 3. Moreover, we discuss the overview and the design details of LeHE4SCNN in Section 4. Section 5 shows the experimental results and discussion. Finally, Section 6 concludes this paper.

2. Related Work

2.1 Related Work

Homomorphic Encryption Based: Homomorphic encryption (HE) is a form of encryption that can directly evaluate functions over encrypted texts without disclosing any data information. It includes multiple types of encryption schemes that perform different classes of computation over encrypted data. Some common types of homomorphic encryption are partially homomorphic encryption (PHE), somewhat homomorphic (SHE), leveled fully homomorphic (LHE), and fully homomorphic encryption (FHE). FHE allows arbitrarily unlimited operations on the encrypted data but has an expensive computation cost [9]. Compared with FHE, PHE enjoys a lower computation overhead but only supports the evaluation of circuits consisting of only one type of gate, e.g., addition or multiplication. SHE schemes can evaluate two types of gates, but only for a subset of circuits, while LHE supports depth-bound arithmetic circuits [10, 11].

Leverages by LHE, Gilad et al. proposed CryptoNets [12] to evaluate a trained neural network in the ciphertext domain. As the main bottleneck of LHE is the computational cost, which grows dramatically with the number of levels of multiplication, CryptoNets' computation overheads are inevitably large. For performing the non-linear functions, it used low-degree polynomials to approximate the activation function, so the network structure must be retrained, hence it hurts the accuracy. Similar studies see E2DM [13], CryptoDL [14], Faster CryptoNets [15], etc.

Two-Party Computation Based: Two-party secure computation protocols e.g., Yao's garbled circuits (GC) and Goldreich-Micali-Wigderson(GMW), focus on dealing with two participants computing a function on personal input. These techniques have the advantage of being computationally efficient since these protocols rely on symmetric-key encryption schemes such as AES, and hardware support in the form of the Intel AES-NI instruction set. To overcome the limitations of HE-based methods, Rouhani et al. presented DeepSecure [16], which leveraged GC to perform secure privacy-preserving prediction. Nevertheless, DeepSecure required heavy communication overheads when executing multiplication operations. XONN [17] binarized the computations in neural networks and employed GC to get prediction results obliviously without leaking sensitive user data.

Mixed-protocol Based: Several studies proposed mixed-protocol-based solutions to trade off the advantages of different solutions. MiniONN [18] converted a neural network model into an oblivious form and evaluated it with secure two-party computation. Chameleon [19] used the GMW protocol for low-depth non-linear activation functions and GC for more complicated non-linear functions. For arithmetic operations such as addition and multiplication, it utilized SS-based methods to perform them. MiniONN and Chameleon required offline and online computation phases. They created correlated randomness or dot-product triplets in the offline phase to guarantee a fast online prediction phase. Besides, Chameleon needed a third party. GAZELLE [20] used an intricate combination of HE and GC to carry out the inference phase of the DL model, which applied the GC to perform the non-linear activation function and used lattice-based HE to execute linear operations. As a result, GAZELLE improved the runtime of private inference and reduced communication between the user and the cloud. FALCON [21] applied Fourier Transform to the multiplication operation of linear layers of a CNN model to make HE efficient and introduced a privacy-preserving protocol for the Softmax function. DELPHI [22] shifted expensive HE multiplication operation into offline phase to reduce online response time, and through navigating the performance accuracy trade-offs, developed a planner that automatically generated neural network architecture configurations.

2.2 Notations

Let $f(X)=X^n+1$, $n = 2^{n'}-1$ such that X^n+1 is the $2^{n'}$-th cyclotomic polynomial. We denote the rings $¢[X]/f(X)$ by R and $¢_q[X]/f(X)$ by R_q, respectively. For a polynomial $u(x) \in R_q$, we use u for $u(x)$, the infinite norm of the polynomial denotes by $||u||_\infty = \max|u_j|$, u_j is the jth coefficient of u. Infinite norm of a polynomial vector $\boldsymbol{p} = (p_1, p_2, \dots, p_n) \in R^n$ consisting of n polynomials is $||p||_\infty = \max(||p_i||_\infty)$. This study denotes the infinite norm by $||\cdot||$.

Vector is in the form of a column vector unless otherwise specified. For two vectors **a** and **b**, the inner product of two vectors can be defined as $<\mathbf{a}, \mathbf{b}>=\mathbf{a}^T\mathbf{b}$, where $\hat{a}_j$ is the jth component of **a**. Note that regular lower-case letters represent polynomials, bold lower-case letters represent vectors, while we denote matrices by bold upper-case letters, and use $\hat{a}_{ij}$ to represent the (i, j) element of a matrix. Furthermore, $\Pr[\cdot]$ represents the probability, *negl* means the negligible probability.

λ is the security parameter, denotes the scheme can resist 2^{λ} attacks. $k = k(\lambda)$ is the rank of a module determined by λ. q is a modulus parameter that determines the size of the finite field. η is noise distribution parameter. *dt*, *dp*, *du* and *dv* are public key compression, plaintext compression and ciphertext compression parameters respectively. After *mod q* operation, the coefficients of the polynomial are in the range $(-(q\text{-}1)/2, (q-1)/2]$. For $x \in ¤$, $round(x)$ is rounding x to the closest integer, $\lceil x \rceil$ and $\lfloor x \rfloor$ express rounding up and rounding down of x to the nearest integer. $a \leftarrow S$ denotes that a is chosen uniformly at random from a set S or a is chosen according to a probability distribution S. More preliminaries are present in the supplementary file.

3. LeHE Scheme with Packing and Unpacking Methods

3.1 Packing and Unpacking Methods

Single Instruction Multiple Data (SIMD) [24] is widely used in homomorphic encryption schemes such as Brakerski-Gentry-Vaikuntanathan (BGV) [10], Cheon-Kim-Kim-Song(CKKS) [25]. They adopt the Chinese Remainder Theorem (CRT) to number fields and partition the plaintext space into a vector of plaintext slots. As a result, a single homomorphic addition or multiplication of a pair of ciphertexts implicitly adds or multiplies (component-wise) an entire plaintext vector.

As we know, the CNN prediction task requires a lot of inner product operations. However, using the CRT-based SIMD schemes to execute this operation is time-consuming since rotating operations are required to sum up the results between slots. Different from the CTR-based SIMD, we construct the inner product terms by a delicate placement of coefficients. It packs several low-dimensional vectors into a high-degree polynomial and rearranges the coefficients of one polynomial. Therefore, these inner products arise in some coefficient terms of polynomial convolution.

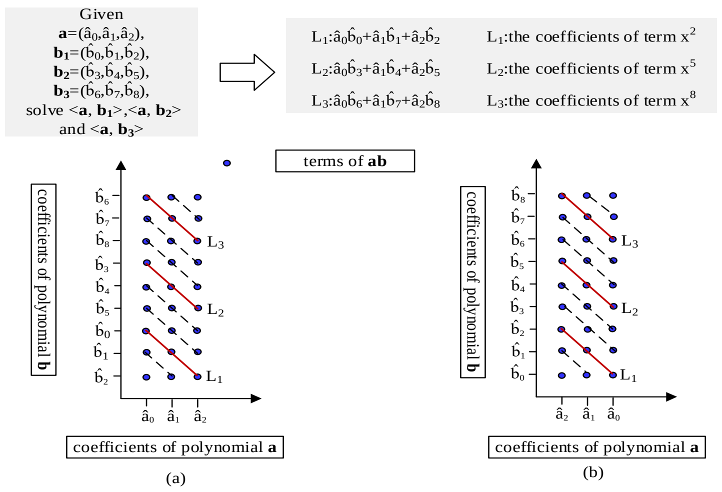

Given
$\mathbf{a}=(\hat{a}_0,\hat{a}_1,\hat{a}_2)$,
$\mathbf{b_1}=(\hat{b}_0,\hat{b}_1,\hat{b}_2)$,
$\mathbf{b_2}=(\hat{b}_3,\hat{b}_4,\hat{b}_5)$,
$\mathbf{b_3}=(\hat{b}_6,\hat{b}_7,\hat{b}_8)$,
solve $<\mathbf{a}, \mathbf{b_1}>$,$<\mathbf{a}, \mathbf{b_2}>$ and $<\mathbf{a}, \mathbf{b_3}>$
$L_1:\hat{a}_0\hat{b}_0+\hat{a}_1\hat{b}_1+\hat{a}_2\hat{b}_2$
$L_2:\hat{a}_0\hat{b}_3+\hat{a}_1\hat{b}_4+\hat{a}_2\hat{b}_5$
$L_3:\hat{a}_0\hat{b}_6+\hat{a}_1\hat{b}_7+\hat{a}_2\hat{b}_8$
L_1:the coefficients of term x^2
L_2:the coefficients of term x^5
L_3:the coefficients of term x^8
terms of **ab**
coefficients of polynomial **b**
$\hat{b}_6$ $\hat{b}_7$ $\hat{b}_8$ $\hat{b}_3$ $\hat{b}_4$ $\hat{b}_5$ $\hat{b}_0$ $\hat{b}_1$ $\hat{b}_2$
L_3
L_2
L_1
$\hat{a}_0$ $\hat{a}_1$ $\hat{a}_2$
coefficients of polynomial **a**
(a)
coefficients of polynomial **b**
$\hat{b}_8$ $\hat{b}_7$ $\hat{b}_6$ $\hat{b}_5$ $\hat{b}_4$ $\hat{b}_3$ $\hat{b}_2$ $\hat{b}_1$ $\hat{b}_0$
L_3
L_2
L_1
$\hat{a}_2$ $\hat{a}_1$ $\hat{a}_0$
coefficients of polynomial **a**
(b)

Fig. 1. An example of packing and unpacking methods

Fig. 1 indicates a straightforward example. The coordinates depict the filling method of the polynomial coefficients from lower to higher degree terms. The cross point represents the convolutional result terms of two polynomials, and the oblique line represents similar terms needed to merge. Note that **Fig. 1(a)** and **(b)** are essentially the same, just two different ways of filling coefficients. Therefore, the coefficients of x^2, x^5 and x^8 are the required inner products. In the convolutional layers of the CNN prediction task, the size of the kernel (i.e., the vector dimension of solving the inner product) is usually much smaller than the degree of plaintext polynomial. It can load more vectors to solve more inner products. Therefore, the packing and unpacking methods are suitable for convolutional operations on CNN.

Packing matrix: The packing and unpacking methods can be used for solving homomorphic vector-matrix in parallel. Algorithm. S1 (found in the supplementary file) presents the packed encoding method of a matrix. For an input $\mathbf{M} = \{\hat{b}_{ij} \mid i \in [0,r), j \in [0,s)\}$, elements in each column are arranged in reverse order. Then, along the direction of the columns of the new matrix, the element values of $\lfloor n/r \rfloor$ columns are assembled into a packed plaintext polynomial, thus $\lceil sr/n \rceil$ polynomials can be obtained. Assume that a polynomial is represented as $m_z = \sum_{i=0}^{n-1} \hat{b}'_i x^i$, $z \in [0, \lceil sr/n \rceil)$ by the above two steps, a packed polynomial can be expressed as:

$$m_z = \sum_{i=0}^{r-1} \hat{b}'_i x^{r-1-i} + \sum_{i=r}^{2r-1} \hat{b}'_i x^{3r-1-i} + \sum_{i=2r}^{3r-1} \hat{b}'_i x^{5r-1-i} + \ldots + \sum_{i=(\lfloor n/r \rfloor -1)r}^{\lfloor n/r \rfloor r-1} \hat{b}'_i x^{2\lfloor n/r \rfloor r - r - 1 - i} \in R_q, \quad z \in [0, \lceil sr/n \rceil)$$

Result Unpacking: After finishing the homomorphic multiplication between vector **w** and input matrix $\mathbf{M}$, we need to unpack the resulting polynomial res_z, $z \in [0, \lceil sr/n \rceil)$ to get several inner products. Because:

$$GetCoeff(res_z, r-1) = \sum_{i=0}^{r-1} \hat{w}_i \hat{b}'_i, \quad GetCoeff(res_z, 2r-1) = \sum_{i=0}^{r-1} \hat{w}_i \hat{b}'_{i+r},$$

$$GetCoeff(res_z, 3r-1) = \sum_{i=0}^{r-1} \hat{w}_i \hat{b}'_{i+2r}, \ldots, \ldots, GetCoeff(res_z, \lfloor n/r \rfloor r - 1) = \sum_{i=0}^{r-1} \hat{w}_i \hat{b}'_{i+(\lfloor n/r \rfloor -1)r}$$

Where $GetCoeff(res_z, i)$, $0 \le i < n$ represents the coefficient of the term x^i in polynomial res_z, the $\lfloor n/r \rfloor$ coefficients of res_z are $\lfloor n/r \rfloor$ inner products between w and m_z. Algorithm. S2 in the supplementary file illustrates the corresponding steps. Consequently, $\lceil sr/n \rceil$ packed polynomials are multiplied by w to yield s inner products. Therefore, we acquire multiple inner products in parallel from the packing and unpacking methods.

Furthermore, a pair of multiplication matrices can also employ packing and unpacking to solve matrix-matrix multiplication. For the matrix $\mathbf{W}$'s row vectors and the matrix $\mathbf{M}$'s column vectors, these vectors are placed at appropriate positions of polynomials w and m, and one of the polynomials is placed in reverse order to construct the inner product terms in product polynomial.

3.2 Homomorphic Operations

In the task of privacy-preserving CNN prediction, the cloud holds model parameters and conducts inner product computations between the user's encrypted data and model parameters.

Hence, the proposed HE scheme should support the homomorphic inner product between plaintext and ciphertext vector, together with homomorphic addition. Whereas, the encryption scheme [23] did not provide any homomorphic inner product operation. Under these conditions, we modify the LeHE encryption scheme to support homomorphic addition, vector-vector multiplication, vector-matrix multiplication, where the vector is plaintext, and the second operand is ciphertext. The algorithms of addition and vector-vector multiplication are detailed as follows.

- LeHE.Add($\mathbf{c}_1$, $\mathbf{c}_2$):Input $\mathbf{c}_1$= LeHE.Enc(pk, m_1) = ($\mathbf{u}_1$, v_1)$\in R_q^2 \times R_q$, and $\mathbf{c}_2$= LeHE.Enc(pk, m_2) = ($\mathbf{u}_2$, v_2)$\in R_q^k \times R_q$, then compute $\mathbf{u}_1'$= Decompress$_q$($\mathbf{u}_1$, du), v_1'= Decompress$_q$(v, dv), the same operations to $\mathbf{u}_2$ and v_2, output $\mathbf{c}_1$+ $\mathbf{c}_2$ = ($\mathbf{u}_1'$ + $\mathbf{u}_2'$, v_1'+ v_2').
- LeHE.ConMul(w, $\mathbf{c}$): Input plaintext polynomial $w \in R_q$, and ciphertext $\mathbf{c}$ = LeHE.Enc(pk, m) = ($\mathbf{u}$, v). Compute $\mathbf{u}'$= Decompress$_q$ ($\mathbf{u}$, du), and compute $\mathbf{c}' = w \cdot \mathbf{c} = (w\mathbf{u}', wv) \in R_{q_2}^{k+1}$, then output $\mathbf{c}'$.

Based on the packed and unpacking methods introduced in section 3.1, together with the extended homomorphic algorithms, the homomorphic vector-matrix product of LeHE using packing and unpacking scheme is designed to resolve secure linear operation in CNN. Algorithm. S3 in the supplementary file presents the specific steps.

After the input vector $\mathbf{w}$ is assembled as a polynomial, the input matrix $\mathbf{M}$ calls LeHE.PackingEncode to pack into polynomials. Then the scheme executes LeHE.Enc to encrypt these polynomials. Next, it repeatedly calls LeHE.ConMul to finish the vector-matrix multiplication. Finally, it conducts the LeHE.Dec and LeHE.UnpackingDecode algorithms to acquire the desired inner products. This scheme contains implementations of the inner product of two vectors and the vector-matrix multiplication using packing and unpacking. Besides, the homomorphic matrix-matrix multiplication scheme is similar, except for the packed encoding method of two matrices.

3.3 Noise Analysis and Correctness

In this part, we derive the conditions of noise growth expression and decryption correctness after the ConMul(w, c) operation.

From the Thm 1 in Appendix part of the supplementary file, the ciphertext noise ε := $\mathbf{e}^T\mathbf{r}$ + e_2 + c_v + $\mathbf{c_t}^T\mathbf{r}$ -$\mathbf{s}^T\mathbf{e}_1$ + $\mathbf{s}^T\mathbf{c_u}$, where ($\mathbf{s}$,$\mathbf{e}$,$\mathbf{e}_1$,e_2,r) ← $(\beta_\eta^k \times \beta_\eta^k \times \beta_\eta^k \times \beta_\eta \times \beta_\eta^k)$, $\mathbf{c_t}$, $\mathbf{c_u}$, c_v are the noise are the noises generated by compression and decompression, and ε satisfies (1).

$$\Pr(\|\varepsilon\| > 12 \cdot \sqrt{\frac{2 \cdot nk \cdot \eta}{6} \cdot (\frac{q}{2^{du+1}} + \eta)^2} + \frac{q}{2^{dv+1}}) = negl. \quad (1)$$

To make the inner product result correct after modulo operation, the plaintexts w and m need to meet (2).

$$\| w \| \cdot \| m \| \cdot n < \frac{2^{dp}}{2} \quad (2)$$

$$\begin{aligned} m' &= LeHE.Dec(\mathbf{c}, sk) = round((v - \mathbf{s}^T u) \cdot \frac{2^{dp}}{q}) mod\, 2^{dp} \\ &= round(m + \varepsilon \cdot \frac{2^{dp}}{q}) mod\, 2^{dp} \end{aligned} \quad (3)$$

$$h = w \cdot m' = round(w \cdot m + w \cdot \varepsilon \cdot \frac{2^{dp}}{q}) mod\, 2^{dp} \tag{4}$$

Therefore, from (4), the noise term after LeHE.ConMul(w, **c**) is:

$$\varepsilon_{sum} := w \cdot \varepsilon \cdot \frac{2^{dp}}{q} \tag{5}$$

To eliminate ε_{sum} by rounding operation, $\| \varepsilon_{sum} \|$ should satisfy:

$$Pr(\| \varepsilon_{sum} \| > 1/2) = negl \tag{6}$$

From (1) and (5), we get:

$$\Pr(\| \varepsilon_{sum} \| > (12 \cdot \sqrt{\frac{2 \cdot nk \cdot \eta}{6} \cdot (\frac{q}{2^{du+1}} + \eta)^2} + \frac{q}{2^{dv+1}}) \cdot w \cdot \frac{2^{dp}}{q} = negl \tag{7}$$

Hence, when the parameter settings satisfy (2), (6), and (7) simultaneously, the ciphertext decryption can be correct.

3.4 Security

The security of the encryption scheme defined above can be guaranteed by **Lemma 1**.

Lemma 1: The LeHE scheme is IND-CPA secure under the MLWE assumption.

Proof: We use a game-based proof method. The notation $Adv_{Game}(\mathsf{A})$ represents the advantage of an adversary A in a game.

- **Game 0**: The adversary A is executed in the IND-CPA security experiment referred to as Game 0. The advantage of the adversary is $|Pr[b = b'$ in Game 1 $] - Pr[b = b'$ in Game 0 $]| „ Adv_{MLWE}(\mathsf{B})$.
- **Game 1:** In Game 1, the value $\mathbf{t}' := \mathbf{A}\mathbf{s} + \mathbf{e}$, used in KeyGen algorithm, is substituted by a uniformly random value $\mathbf{t}' \leftarrow R_q^k$. It is possible to verify that there exists an adversary B with the same running time as that of A such that $|Pr[b = b'$ in Game 1 $] - Pr[b = b'$ in Game 0 $]| „ Adv_{MLWE}(\mathsf{B})$.
- **Game 2**: In Game 2, the values $\mathbf{u} := \mathbf{A}^T\mathbf{r} + e_1$, and $v := \mathbf{t}^T\mathbf{r} + e_2$ used in the generation of the challenge ciphertext are simultaneously substituted with uniformly random values $(\mathbf{u}', v') \leftarrow R_q^k \times R_q$. Again, it is possible to verify that there exists an adversary B with the same running time as that of A such that $|Pr[b = b'$ in Game 2 $] - Pr[b = b'$ in Game 1 $]| „ Adv_{MLWE}(\mathsf{B})$.
- **Game 3**: In Game 3, we use the challenge ciphertext generation method in Game 2, sample $w \leftarrow R_q$ uniformly, and compute $(w \cdot \mathbf{u}', w \cdot v')$. So it is possible to verify that there exists an adversary B with the same running time as that of A such that $|Pr[b = b'$ in Game 3 $] - Pr[b = b'$ in Game 2 $]| „ Adv_{MLWE}(\mathsf{B})$.
- **Game 4**: In Game 4, we use the challenge ciphertext generation method in Game 3 to generate two challenge ciphertexts $(w_1 \cdot \mathbf{u}_1', w_1 \cdot v_1')$ and $(w_2 \cdot \mathbf{u}_2', w_2 \cdot v_2')$. Then compute $(w_1 \cdot \mathbf{u}_1', w_1 \cdot v_1') + (w_2 \cdot \mathbf{u}_2', w_2 \cdot v_2')$. Therefore, there exists an adversary B with the same running time as that of A such that $|Pr[b = b'$ in Game 4 $] - Pr[b = b'$ in Game 3

$]|„\ Adv_{MLWE}(\mathsf{B})$. Since the values of the two challenge ciphertexts are independent of *b*, i.e., $Pr[b = b'$ in Game 4 $] = 1/2$. In sum , $Adv_{cpa}(\mathsf{A})„\ 4 \cdot Adv_{MLWE}(\mathsf{B})$. □

4. LeHE4SCNN

4.1 Overview

Consider such an intelligent healthcare scenario in MCC that: in COVID-19 outbreak period, a user may want to know his health condition via uploading a photo of tongue surface to the healthcare API in a smart device, then the result of lung inflammation with the corresponding probability can be received from the cloud server. For example, the output label "Lung Inflammation" with probability 0.3 and 0.7 have different meanings for treatment. In this section, we outline the execution flow of LeHE4SCNN. The basic idea of LeHE4SCNN is to perform the secure neural network evaluation between smart device *D* and untrusted cloud server *S* by the proposed homomorphic encryption and data shuffling technologies jointly.

Hypothesis: Both *D* and *S* are *semi-honest* [26], presumed to follow the LeHE4SCNN protocol, and never deviate from it, although they might attempt to infer more information based on the data they receive and transmit. Specifically, the device leaks no information about the input contents, intermediate calculation result, and classified results to the cloud. The input data is factual, never using fake data. For the cloud server, LeHE4SCNN protects the parameters of the whole CNN model, and it does not hide the model architecture.

There are two parts to the framework: the preprocessing phase and the computation phase. In the preprocessing phase, the cloud randomly generates permutation matrices depending on the architecture of the CNN, except for the first layer. Then it multiplies the corresponding permutation matrices by weight vectors to yield the permuted model **W**. In the computation phase, the user (i.e., smart device) and the cloud server work together to complete the following tasks of linear and non-linear operations:

- **Evaluate linear layers (i.e., Conv and FC layer)**: For linear layers, *S* utilizes the LeHE with packing and unpacking scheme to execute matrix-matrix multiplication in convolutional layers and vector-matrix multiplication fully connected layers. Take Conv layer, for instance, *D* feeds the convolutional layer with an encrypted input matrix **C**=Enc (*pk*, **M**), *S* computes **RS**=**W**·**C**. In detail, *pk* is the public key of *D*, **M** is the packed polynomial matrix of the original input matrix, and **W** is the filters matrices. The fully connected layer is similar except for homomorphic vector-matrix multiplication.
- **Evaluate non-linear functions (i.e., activation and pooling layer):** *S* and *D* perform designed secure computation protocols to keep data secure. Concretely, *S* first computes shuffled result **RS**'=**RS** × **P**, where **P** is a permutation matrix. Next, it sends **RS**' to the intelligent device *D* to decrypt, and then it carries out ReLU and pooling operations. We use the shuffling algorithm stated in section 2.3.2 to keep data confidential.

In the following layer, *D* repeatedly executes the step of evaluating linear layers, as the filter matrix in the current layer is permuted in the preprocessing phase, and the input is rearranged in the same way, so *S* will get the same convolutional or fully connected results as without using shuffled technology.

After implementing the above two steps repeatedly until the last hidden layers, to protect the model data in the current layer, *S* adds encryption of a random number $r \in ¢_{dp}$ to the linear computation results and sends it to the *D*. As the non-linear operation of the last hidden layer usually is Softmax function, and because

$$y_i = \frac{e^{z_i+r}}{\sum_{j=1}^{num\_out} e^{z_j+r}} = \frac{e^{z_i}}{\sum_{j=1}^{num\_out} e^{z_j}} \tag{8}$$

where z_i is the ith $i \in [1, num\_out]$ input elements of the output layer. D decrypts the ciphertext and gets the prediction result directly.

4.2 Design Details

4.2.1 Preprocessing Phase

The cloud server *S* is required to perform three tasks in the preprocessing phase. 1) Preprocess the model parameters. Covert the float numbers of model data into integer matrices based on the plaintext range, then change the integer matrices into polynomial matrices according to packed encode methods. 2) Generate shuffling matrices. In the convolutional layer, *S* generates two types of shuffling matrices, one for shuffling channels and the other for shuffling the elements in the columns of the extended input matrix. For a fully connected layer, *S* generates a shuffling matrix. 3) The rearranged model is also produced by multiplying by these matrices.

4.2.2 Secure Linear Layers

The smart device performs an input validity check before being served. Some trivial inputs (such as all zeroes or standard basis vectors) are forbidden, and the following operations are completed on the input that passes the check.

An input image to a convolutional layer represents a tuple ($c_i \times h \times w$), where c_i is the number of input channels, *h* and *w* are the height and width of the image. The smart device *D* carries out the following three steps:

1) According to the size of the filter, input is zero-padded depending on the VALID mode or SAME mode. For example, in the SAME mode, for each input channel, *D* expands input into an extended matrix **M** with the size of $(c_i f_h f_w + 1) \times hw$ via stepping a kernel matrix $f_h \times f_w$ across the input image and connects the row vector $[1, 1,\ldots, 1]^{hw}$ into the first row.
2) For the extended matrix **M**, *D* chooses the appropriate elements of **M** for packing to get $\mathbf{M}^{(1)'}$: it performs steps 1 to 4 of Algorithm. S1 first, and then transforms **M** into a matrix with $h(c_i f_h f_w + 1) \times w$. It packs the elements in a pooling window into a polynomial so that the shuffled technique does not separate them. *D* can compute up to $z = \lfloor n / (c_i \cdot f_h \cdot f_w + 1) \rfloor$ inner products at a time, so at most packs $\lfloor z / (pool\_h \cdot pool\_w) \rfloor$ elements of pooling window at once, where $pool\_h$ and $pool\_w$ are the height and width of the pooling window in the following pooling operation.
3) *D* encrypts $\mathbf{M}^{(1)'}$ to get Enc(*pk*, $\mathbf{M}^{(1)'}$) and passes it to *S*.

The cloud *S* holds c_o filters with the size of $c_i \times f_h \times f_w$. In the preprocessing phase, it first switches the filter into a matrix **W** with size of $c_o \times c_i f_h f_w$, and concatenates bias vectors to the first column of **W**. Next, it integrates each matrix row as a polynomial. Thus, $\mathbf{W}^{(1)} = \{w_i, i \in [0, c_o)\}$ is a polynomial matrix. In the online phase, it involves the steps below to accomplish the linear layers:

1) *S* calls LeHE.ConMul($\mathbf{W}^{(1)}$, Enc(*pk*, $\mathbf{M}^{(1)'}$) to output $\mathbf{RS}^{(1)}=\mathbf{W}^{(1)}$ Enc(*pk*, $\mathbf{M}^{(1)'}$), where each matrix element involves (*k*+1) polynomials.
2) *S* rearranges $\mathbf{RS}^{(1)}$ using the ciphertext shuffling technique. Suppose the next linear layer is a convolutional layer, *S* multiplies every column vector in the extended matrix by permuted matrices. Then it messes up the arrangement of the channels via multiplying $\mathbf{RS}^{(1)}$ by another permutation matrix. Finally, it sends $\mathbf{RS}^{(1)'}$ to *D*. If the next linear layer is a fully connected layer, *S* performs the FLATTEN operation on $\mathbf{RS}^{(1)}$ to create a vector, then it multiplies the components by the prepared permutation matrix to obtain $\mathbf{RS}^{(1)'}$, and transmits $\mathbf{RS}^{(1)'}$ to *D*. By these operations, the inner products between the out-of-order input and the corresponding rearranged model remain unchanged, as if it is not using the shuffling method in the next hidden layer.

4.2.3 Secure Non-linear Layers

The end device *D* conducts the computation of non-linear functions. Once *D* finishes the decryption by calling LeHE.Dec(**RS**', *sk*) and unpacks the results by Algorithm. S2, it executes activation functions such as ReLU or sigmoid function on each inner product. Then it performs the pooling function. Afterward, *D* outputs the out-of-order result and feeds into the next linear layer.

When *D* and *S* complete all hidden layers' operations, *S* firstly generates a random integer number $r \in ¢_{dp}$, assuming that the linear result of the last hidden is $\mathbf{RS}^{(l)}$. After calling LeHE.Enc(*pk*, *r*), it then performs LeHE.Add(Enc(*pk*, *r*), $\mathbf{RS}^{(l)}$). Finally, the result is sent to *D*. *D* decrypts it, then it runs the Softmax function to output the prediction result.

4.3 Security Analysis of LeHE4SCNN

For the intelligent mobile device side, the security is guaranteed by the LeHE scheme when it encrypts the input image. For the cloud side, the hardness of obtaining the plaintexts of input and calculated results (including the intermediate and final results) is equal to solving the MLWE hard problem. The cloud protects its model by shuffling technique, and the analysis is following:

- Take the CNN architecture in **Table 1**, for instance. Since in the pooling window, $pool_h \cdot pool_w$ elements are rearranged at random and need not be restored. The probability that the end device can guess the correct arrangement to solve the correct model is $1/(4!\cdot 6!\cdot 37!) \approx 1/2^{157}$ in the Conv-1 layer, which can resist 2^{157} attacks. In Conv-2, the number of input channels and the output ciphertext is all 16, so it is $1/(4!\cdot 16!\cdot 16!) \approx 1/2^{93}$. And $1/(120!) \approx 1/2^{660}$ in FC-1, $1/(84!) \approx 1/2^{420}$ in FC-2. Therefore, the end device cannot solve the correct rearrangement of the inner products, namely cannot acquire model parameters. As S adds the mask r to the multiplication results in the output layer, the plaintext result's distribution is not distinguished from the uniform distribution.
- The smart device is forbidden for some special input such as standard basis or its integer multiples, designed to defend against the model inversion attack. However, in the intelligent healthcare scenario, the inputs of a device are rarely all zeroes or standard basis. Hence, the limitations do not affect the availability of service.

Table 1. Layers description of CNN

layers	Description
[Conv-1]	Input image: 28×28, kernel size: 5×5, stride (1, 1), number of output channels 6, padding = same, Activation = ReLU
[maxpooling-1]	Input size: $6\times28\times28$, window size: 1×2×2, output size: $6\times14\times14$
[Conv-2]	Input size: 6×14 ×14, kernel size 6×6, stride (2, 2), number of output channels 16, and output size:16×7×7, padding=same, Activation = ReLU.
[maxpooling-2]	Input size: 16×7×7, window size: 1×2×2, output size: 16×4×4.
[FC-1]	Fully connecting with 16×4×4 = 256 inputs and 120 outputs, Activation = ReLU.
[FC-2]	Fully connecting with 20 inputs and 84 outputs, Activation = ReLU.
[FC-3]	Fully connecting with 84 inputs and 10 outputs, Activation = Softmax.

5. Experimental Results and Discussion

In this section, we evaluate the implementation of LeHE4SCNN on the industrial dataset. Then, compared with the state-of-the-art methods, we test the response time and usage cost of LeHE4SCNN with different CNN models in the MCC environment. The performance of LeHE with the packing and unpacking scheme is report in supplementary file.

5.1 Experimental Settings

Testbed: We conduct our implementations on a cloud server and a laptop hosted by our lab. The cloud server is equipped with one Intel Xeon(R) E5-2680 2.4 GHz processor with 24 GB RAM, while the laptop is configured with an Intel i5-7500 CPU with 8GB RAM. Their operating system is Ubuntu 16.04.2LTS and enables support for the AES-NI instruction set. We implement LeHE and LeHE4SCNN in C++. In our experiments, the cloud server works as DLaaS services provider, while the laptop works as a mobile device user. We perform all the experiments in sequence without adopting any parallel hardware. Moreover, we repeat each experiment 50 times, the mean value is reported as the final result to attain a fair result.

Dataset: LeHE4SCNN framework is evaluated on CNN to classify encrypted handwritten images of the MNIST dataset (Modified National Institute of Standards and Technology database). It is a dataset of images representing digits handwritten by more than 500 different writers. It contains 60000 training images and 10000 testing images. The format of the images is 28×28, and the integer value of each pixel represents a level of grey with a range of 0 to 255. Moreover, each image is labeled with the digit it depicts.

Parameter selection: We set $n = n(\lambda) = 1024$, $\eta = 5$ and $k = 2$. In the LeHE scheme, supposed that $||w||, ||m|| \leq 2^9$, $||<w, m>|| \leq 2^{29}/2$, hence $dp = 29$. From the correct analysis of section 3.3, we set $du = dt = dv = 49$, $q \approx 2^{52}$ (the next prime of 2^{52}). With the selected parameters, LeHE scheme can achieve a 102-bit security level. We implement LeHE4SCNN on a CNN with eight layers, the details of each layer are present in **Table 1**. Since the LeHE scheme only supports the operations on integer space, the model parameters (i.e., **W** and **b**) are enlarged by 50 times and are rounded to the nearest integer. As a result, $\|w\|, \|b\| < 2^6$, and $\|m\| < 2^8$. Thus, $dp = 25$, $q = 140737488355333 \approx 2^{47}$ and $du = dv = 44$, q_2=9223372036854775837 $\approx 2^{63}$, and the selected parameters of the scheme can achieve a 128-bit security level.

5.2 Performance of LeHE4SCNN on Real Network

In this part, we test the performance of LeHE4SCNN on the real network in terms of computation and communication overhead, respectively.

Computation overheads: Table 2 shows the time overheads of each layer in LeHE4SCNN. The dominant cost of evaluating the framework is that of performing the convolution layer. Take the Conv-1 layer for example, the mobile device first takes about 1.98s to encode and encrypt the target image using the encryption scheme. Then the cloud spends 1.003s to execute the homomorphic multiplication (including shuffling operation) for the convolution layer. After that, the convolution result is sent to the device again. It requires 1.414s to decrypt and decode the result. Then it only costs almost 0.112ms and 0.038ms for ReLU and Maxpool. Hence, the total computation overhead of executing the Conv-1 layer, including the non-linear layers between the mobile device and the cloud, is 4.4s. For Conv-2 and non-linear layer 2, the computation overhead is about 3.33s. The following layers cost 0.58s, 0.17s, and 0.037s to finish the corresponding fully connected and non-linear operations.

Table 2. The computation overheads of the CNN on each image of dataset MNIST (ms)

Layers	**User**			**Cloud**
	Pac.Encode+Enc	**Dec+Unpac.decode**	**Comput.**	**Eval.**
Conv-1	1983.16	1414.68	–	1003.44
Relu-1	–	–	0.112	–
Maxpool-1	–	–	0.038	–
Conv-2	335.95	1900.46	–	1095.74
Relu-2	–	–	0.044	–
Maxpool-2	–	–	0.014	–
FC-1	26.71	287.28	–	269.13
Relu-3	–	–	0.012	–
FC-2	19.52	78.74	–	75.39
Relu-4	–	–	0.005	–
FC-3	19.61	7.24	–	10.65
Softmax	–	–	0.025	–
Total	2384.95	3688.4	0.25	2454.35

Using the packing and unpacking methods, the homomorphic evaluation on the cloud is not the most time-consuming operation for classifying a single image, which accounts for 28.78% of total computation overhead. In contrast, encryption and decryption take more time, which takes 27.96% and 43.25%, respectively. The computation overheads on the device, including ReLU, Maxpool, and Softmax, are the least time-consuming operations. It only accounts for nearly 0.01% of time cost. The time it takes for the LeHE4SCNN framework to predict a picture is about 8.53s. The prediction accuracy reaches 99%.

Communication overheads: By the parameters selected above, each coefficient of the ciphertext polynomials requires 47 bits, since using compression and decompression functions, the actual space occupied by a coefficient of the ciphertext is 44 bit/8=5.5 bytes(B).

In the Conv-1 layer, D packs eight columns of the extended matrix into a polynomial for solving eight inner products at a time. Hence, 14×7=98 ciphertexts are needed to pass to the cloud. Since one plaintext polynomial corresponds to three ciphertexts, the total message delivered to the cloud is $98\times 1024\times 5.5\times 3$ bits or 1617KB. After the cloud completes the homomorphic matrix-matrix multiplications, the result costs $2\times 14\times 7\times 1024\times 7.875\times 3$B, where $q_2 \approx 2^{63}$, make up 7.875B, and the filter matrix is packed into two ciphertexts, so it requires 4630.5KB in total.

In the Conv-2 layer, the input image size is $6\times 14\times 14$. D transforms it into an extended matrix with stride (2, 2) and packs every four columns of the extended matrix into a polynomial, so the size of the matrix plaintext to be encrypted is 4×4. Since each polynomial is encrypted into one ciphertext, the total space required is 264KB. After homomorphic multiplication calculations, the ciphertext size obtained by the cloud is $16 \times 4 \times 4$ bytes, and the communication overhead transmitted by the cloud to the device is 6048KB.

Table 3. Communication Overheads of the CNN on each image of dataset MNIST (KB)

Layers	**Device → Cloud**	**Cloud → Device**	**Total of each layer**
Conv-1	1617	4630.50	6247.50
Conv-2	264	6048	6312
FC-1	16.50	945.2	961.7
FC-2	16.50	259.93	276.43
FC-3	16.50	23.63	40.13
Total	1930.5	11907.26	13837.76KB ≈ 13.51MB

In the subsequent three FC layers, the input is encrypted into one ciphertext, occupying a space of 16.5KB. Through homomorphic multiplication, the cloud generates 40 ciphertexts in the FC-1 layer, 11 ciphertexts in the FC-2 layer, and one ciphertext in the FC-3 layer, respectively, the corresponding communication overheads sent by the device to the cloud are 945.2KB, 259.93KB, and 23.63KB. **Table 3** presents the communication overhead of using the compression and decompression functions. The size of the message sent by the device to the cloud is 1930.5KB, the size of the message sent by the cloud to the device is 11647KB, and the total overhead is 13837.76KB or 13.51MB.

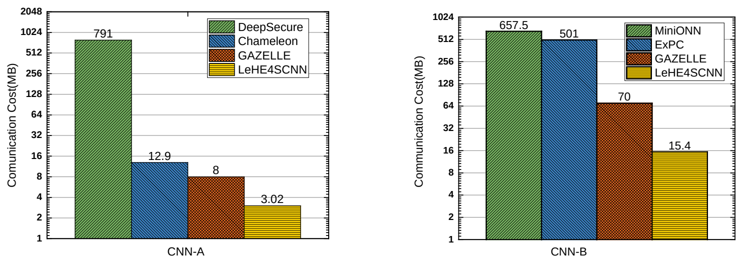
5.3 Comparison with Previous Work in MCC

In this part, we test the performance of LeHE4SCNN in the MCC environment, compared with several previous solutions. We are mainly concerned with the response time and usage cost of LeHE4SCNN compared to other competitors since both are two key indexes to reflect the users' satisfaction on service, which directly affect the DLaaS service widespread applications in MCC. We conduct comparisons based on two published and more complex CNNs used for privacy prediction tasks.

CNN-A: 1-Conv and 2-FC layers with ReLU activation from [12][16].

CNN-B: 2-Conv and 2-FC layers with ReLU activation and Maxpool from [18].

Communication overhead: **Fig. 2** is the communication overhead of different solutions on dataset MNIST. We observe that compared to the other mix-protocol-based solutions (e.g., MiniONN, GAZELLE), LeHE4SCNN enjoys the lowest bandwidth cost either in CNN-A or CNN-B. As shown in **Fig. 2(a)**, DeepSecure requires a bandwidth cost of 791 MB to classify one image from the MNIST dataset, which is $262\times$ than that of LeHE4SCNN. One of the main reasons for this performance gap is that DeepSecure is one of the secure two-party computation-based solutions, which requires extensive communication overhead performing secure privacy-preserving prediction. GAZELLE takes 70MB to finish one image classification in CNN-B. Compared to it, LeHE4SCNN still enjoys a lower bandwidth with only 15.4MB. The possible inner reasons are that GAZELLE employs the RLWE based HE scheme to secure the linear operations, which may lead to a high expansion rate of the ciphertext. LeHE4SCNN uses the proposed LeHE scheme to secure the linear operations, which has the advantage of a low expansion rate. Furthermore, GAZELLE requires several rounds of interaction between user and server to finish a non-linear operation, increasing communication cost. Whereas LeHE4SCNN adopts the data shuffling method to achieve the non-linear operation with one round of interaction per layer.



(a) Results in CNN-A (b) Results in CNN-B

Fig. 2. Communication overhead comparison of different solutions on MNIST.

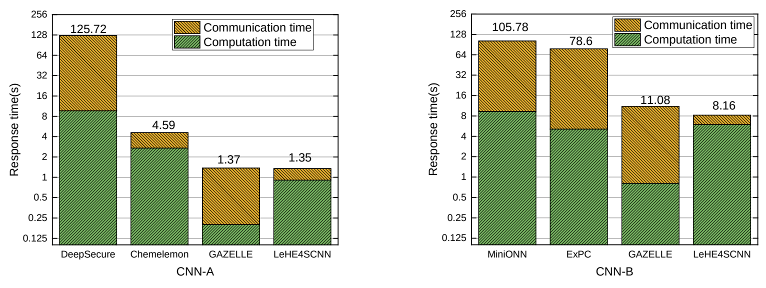
Response time analysis in MCC: For MCC, response time is a pivotal index to evaluate the quality of DLaaS services. The response time is defined as the duration time between the request the mobile device user sends and the result the user receives from the cloud server. Thus, the response time is the sum result of computation time and transmission time. Assume that the global average mobile data transfer speed is 54.53Mb/s (this data is collected from the website speedtest[3] reported in August 2021). **Fig. 3** shows
the results. From **Fig. 3(b)**, GAZELLE takes the lowest computation time of 0.81s to perform an image classification task, compared to the other solutions. However, GAZELLE also spends considerable time on the data transmission (10.27s). Thus LeHE4SCNN enjoys a lower response time than the other frameworks. It indicates that the communication overhead is a dominant factor to impact the quality of mobile service because the mobile data transfer speed is usually unstable and limited to network load and the location. Therefore, it demonstrates that our proposed LeHE4SCNN is a practical solution in the MCC environment.

Cost analysis in MCC: The usage cost of the cloud-based services in the MCC environment comprises service fees and data transfer fees. For the DLaaS services, billed per query becomes an increasingly popular pay mode, no matter what services framework they used, so the data transfer cost is the critical factor in determining the usage cost of different solutions.

In this experiment, we employ the wireless data-only plan of AT&A as the data transfer charging standard. The total amount of this plan is $50 with 25 GB and $10 per 2 GB for the overage part[4]. **Table 4** indicates a comparison of transfer fees in usage cost with different privacy-preserving solutions on CNN-B. The third column shows the transfer fees per query using different solutions, and the fourth column shows the usage cost per query when overage. The fifth column is the maximum number of queries allowed supported in the plan. We can observe that compared to other solutions, LeHE4SCNN can support about 1662 queries at most in the plan, which is $4\times$ than that of the best-known solutions (i.e., FALCON). Therefore, it demonstrates that LeHE4SCNN is a cost-effective solution in the MCC environment. For example, the proposed framework can provide early warning services of emergencies for the elderly who suffer from chronic diseases and whose health conditions require continuous monitoring. Especially during the COVID-19 epidemic, many suspected

[3] https://www.speedtest.net/global-index
[4] https://www.att.com/support/article/wireless/KM1048698/

patients are isolated at home, and they need to monitor their health conditions at all times and frequently send inquiries to determine whether they have suffered pneumonia. In that case, the framework we have presented is one of the best alternatives, as it can obtain multiple query services at a small cost to users.



(a) Results in CNN-A (b) Results in CNN-B

Fig. 3. Response time comparison of different solutions on MNIST.

Table 4. Data transfer cost of different solutions in the MCC environment

Frameworks	Communication overhead per query (MB)	Cost per query in the plan ($)	Cost per query out of the plan ($)	Number of queries supported in the plan
MiniONN	657.5	1.28	3.21	39
GAZELLE	70	0.14	0.34	357
FALCON	62.1	0.12	0.30	412
LeHE4SCNN	15.4	0.03	0.08	1662

6. Conclusion

The emerging topic of privacy-preserving deep learning as a service has attracted increasing attention in recent years. This study focuses on constructing the low communication and cost-efficient framework for secure convolutional neural network prediction in the mobile cloud computing environment. The increased communication overhead can be accomplished with more bandwidth consumption, leading to the extra cost for mobile users. Thus, the communication cost of privacy-preserving solutions is a crucial factor affecting their widespread applications in the MCC environment. To address it, we first propose the LeHE with the novel packing and unpacking methods to support the fast vector-matrix homomorphic operations. The apparent advantage of LeHE is that it can effectively bound the expansion ratio of the generated ciphertext. We also give the bound of accumulated noise, correctness, and security analysis. Finally, we present LeHE4SCNN, a privacy-preserving and communication-efficient framework for the convolutional neural network prediction task. LeHE4SCNN applies a judicious combination of LeHE and data shuffling methods to obtain low communication costs. Extensive experiments on the real network show that LeHE4SCNN achieves better response time and usage cost than the state-of-the-art methods in the mobile cloud computing environment. The subsequent work will study how to remove semi-honest assumptions so that the service framework can resist malicious attacks from internal participants and external attackers to expand the application scenarios.

Acknowledgement

This work was supported in parts by the National Key Research and Development Project (2020YFA0712303), in parts by Chongqing Research Program (cstc2019yszx-jcyjX0003, cstc2020yszx-jcyjX0005, cstc2021yszx-jcyjX0004), in parts by Guizhou Science and Technology Program ([2020]4Y056). in parts by National Natural Science Foundation of China (11771421), in parts by Youth Innovation Promotion Association of CAS (2018419).

References

[1] D. Shen, G. Wu, and H.-Il. Suk, "Deep learning in medical image analysis," *Annual review of biomedical engineering*, vol. 19, pp. 221–248, 2017. Article (CrossRef Link)

[2] M. Tavana, A.-R. Abtahi, D. Di Caprio, and M. Poortarigh, "An artificial neural network and bayesian network model for liquidity risk assessment in banking," *Neurocomputing*, vol. 275, pp. 2525–2554, 2018. Article (CrossRef Link)

[3] W. Zheng, L. Yan, C. Gou, and F.-Y. Wang, "Fighting fire with fire: A spatial–frequency ensemble relation network with generative adversarial learning for adversarial image classification," *International Journal of Intelligent Systems*, vol. 36, no. 5, pp. 2081–2121, 2021. Article (CrossRef Link)

[4] N. Fernando, S. W. Loke, and W. Rahayu, "Mobile cloud computing: A survey," *Future generation computer systems*, vol. 29, no. 1, pp. 84–106, 2013. Article (CrossRef Link)

[5] B. Zhou and R. Buyya, "Augmentation techniques for mobile cloud computing: A taxonomy, survey, and future directions," *ACM Computing Surveys (CSUR)*, vol. 51, no. 1, pp. 1–38, 2019. Article (CrossRef Link)

[6] M. Ribeiro, K. Grolinger, and M. A. Capretz, "Mlaas: Machine learning as a service," in *Proc. of the IEEE 14th Intnational Conference. Machine Learning and Applications (ICMLA 2015)*, Miami, FL, USA, pp. 896–902, 2015. Article (CrossRef Link)

[7] M. S. Riazi, B. D. Rouani, and F. Koushanfar, "Deep learning on private data," *IEEE Security & Privacy*, vol. 17, no. 6, pp. 54–63, 2019. Article (CrossRef Link)

[8] T. H. Noor, S. Zeadally, A. Alfazi, and Q. Z. Sheng, "Mobile cloud computing: Challenges and future research directions," *Journal of Network and Computer Applications*, vol. 115, pp. 70–85, 2018. Article (CrossRef Link)

[9] C. Gentry, "Fully homomorphic encryption using ideal lattices," in *Proc. of the forty-first annual ACM symposium on Theory of computing (STOC'09)*, Bethesda, MD, USA, pp.169–178, 2009. Article (CrossRef Link)

[10] Z. Brakerski, C. Gentry, and V. Vaikuntanathan, "(Leveled) fully homomorphic encryption without bootstrapping," *ACM Transactions on Computation Theory (TOCT 2014)*, vol. 6, no. 3, pp. 1–36, 2014. Article (CrossRef Link)

[11] J. Fan and F. Vercauteren, "Somewhat practical fully homomorphic encryption," *IACR Cryptol. ePrint Arch.*, vol. 2012, pp. 144, 2012.

[12] R. Gilad-Bachrach, N. Dowlin, K. Laine, K. Lauter, M. Naehrig, and J. Wernsing, "Cryptonets: Applying neural networks to encrypted data with high throughput and accuracy," in *Proc. of the 33rd International Conference machine learning(PMLR 2016)*, New York, NY, USA, pp. 201–210, 2016.

[13] X. Jiang, M. Kim, K. Lauter, and Y. Song, "Secure outsourced matrix computation and application to neural networks," in *Proc. of the 2018 ACM SIGSAC Conference on Computer and Communications Security (CCS 2018)*, Toronto, Canada, pp. 1209–1222, 2018. Article (CrossRef Link)

[14] E. Hesamifard, H. Takabi, M. Ghasemi, and R. N. Wright, "Privacy-preserving machine learning as a service," in *Proc. of on Privacy Enhancing Technologies (PETS 2018)*, Barcelona, Spain, vol. 2018, no. 3, pp. 123–142, 2018. Article (CrossRef Link)

[15] E. Chou, J. Beal, D. Levy, S. Yeung, A. Haque, and L. Fei-Fei, "Faster cryptonets: Leveraging sparsity for real-world encrypted inference," *arXiv preprint arXiv:1811.09953*, 2018.

[16] B. D. Rouhani, M. S. Riazi, and F. Koushanfar, "Deepsecure: Scalable provably-secure deep learning," in *Proc. of the 55th Annual Design Automation Conference. (DAC 2018)*, San Francisco, CA, USA, pp. 1–6, 2018. Article (CrossRef Link)

[17] M. S. Riazi, M. Samragh, H. Chen, K. Laine, K. Lauter, and F. Koushanfar, "XONN: Xnor-based oblivious deep neural network inference," in *Proc. of the 28th USENIX Security Symposium (USENIX Security 2019)*, Santa Clara, CA, USA, pp.1501–1518, 2019.

[18] J. Liu, M. Juuti, Y. Lu, and N. Asokan, "Oblivious neural network predictions via minionn transformations," in *Proc. of the 2017 ACM SIGSAC Conference on Computer and Communications Security (CCS 2017)*, Dallas Texas, USA, pp. 619–631, 2017. Article (CrossRef Link)

[19] M. S. Riazi, C. Weinert, O. Tkachenko, E. M. Songhori, T. Schneider, and F. Koushanfar, "Chameleon: A hybrid secure computation framework for machine learning applications," in *Proc. of the 2018 on Asia Conference on Computer and Communications Security (ASIACCS 2018)*, Incheon, Korea, pp.707–721, 2018. Article (CrossRef Link)

[20] C. Juvekar, V. Vaikuntanathan, and A. Chandrakasan, "GAZELLE: A low latency framework for secure neural network inference," in *Proc. of the 27th USENIX Security Symposium (USENIX Security 2018)*, Baltimore, MD, USA, pp.1651–1669, 2018.

[21] S. Li, K. Xue, B. Zhu, C. Ding, X. Gao, D. Wei, and T. Wan, "Falcon: A fourier transform based approach for fast and secure convolutional neural network predictions," in *Proc. of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR 2020)*, Virtual, pp. 8705–8714, 2020. Article (CrossRef Link)

[22] P. Mishra, R. Lehmkuhl, A. Srinivasan, W. Zheng, and R. A. Popa, "Delphi: A cryptographic inference service for neural networks," in *Proc. of the 29th USENIX Security Symposium (USENIX Security 2020)*, pp. 2505–2522, 2020.

[23] C. Ke, W. Wu, and Y. Feng, "Low expansion rate encryption algorithm based on mlwe," *Computer Science*, vol. 46, no. 4, pp. 144–150, 2019. Article (CrossRef Link)

[24] N. P. Smart and F. Vercauteren, "Fully homomorphic SIMD operations," *Designs, codes and cryptography*, vol. 71, no. 1, pp. 57–81, 2014. Article (CrossRef Link)

[25] J. H. Cheon, A. Kim, M. Kim, and Y. Song, "Homomorphic encryption for arithmetic of approximate numbers," in *Proc. of Conference on the Theory and Application of Cryptology and Information Security(Asiacrypt 2017)*, Hong Kong, China, pp. 409–437, 2017. Article (CrossRef Link)

[26] A. Paverd, A. Martin, and I. Brown, "Modelling and automatically analysing privacy properties for honest-but-curious adversaries," Univ. Oxford, Oxford, UK, Tech. Rep, 2014.

Yanan Bai, received the M.S. degree in computer science from Guizhou University, Guizhou, China, in 2010. She joined the University of Ping dingshan, Henan, China, in 2010. She is currently a Ph.D. candidate in computer science from the University of Chinese Academy of Sciences, Chongqing, China. Her research interests are information security and artificial intelligence.

Yong Feng, born in 1965, Ph.D. supervisor, professor. Vice President of Chongqing society of electronics, chief scientist of automatic reasoning and its application in the field of high and new technology. His research interests are zero error calculation in automatic reasoning and information security.

Wenyuan Wu, born in 1972. Ph.D. supervisor, professor. His main research interests include cryptography theory, symbolic computation, and zero error calculation, automated reasoning.