Secure Outsourced Matrix Computation and Application to Neural Networks

Xiaoqian Jiang[1], Miran Kim[1], Kristin Lauter[2], and Yongsoo Song[3]

[1] University of Texas, Health Science Center at Houston, USA
{Xiaoqian.Jiang, Miran.Kim}@uth.tmc.edu
[2] Microsoft Research, Redmond, USA
klauter@microsoft.com
[3] University of California, San Diego, USA
yongsoosong@ucsd.edu

Abstract. Homomorphic Encryption (HE) is a powerful cryptographic primitive to address privacy and security issues in outsourcing computation on sensitive data to an untrusted computation environment. Comparing to secure Multi-Party Computation (MPC), HE has advantages in supporting non-interactive operations and saving on communication costs. However, it has not come up with an optimal solution for modern learning frameworks, partially due to a lack of efficient matrix computation mechanisms.

In this work, we present a practical solution to encrypt a matrix homomorphically and perform arithmetic operations on encrypted matrices. Our solution includes a novel matrix encoding method and an efficient evaluation strategy for basic matrix operations such as addition, multiplication, and transposition. We also explain how to encrypt more than one matrix in a single ciphertext, yielding better amortized performance.

Our solution is generic in the sense that it can be applied to most of the existing HE schemes. It also achieves reasonable performance for practical use; for example, our implementation takes 9.21 seconds to multiply two encrypted square matrices of order 64 and 2.56 seconds to transpose a square matrix of order 64.

Our secure matrix computation mechanism has a wide applicability to our new framework E2DM, which stands for encrypted data and encrypted model. To the best of our knowledge, this is the first work that supports secure evaluation of the prediction phase based on both encrypted data and encrypted model, whereas previous work only supported applying a plain model to encrypted data. As a benchmark, we report an experimental result to classify handwritten images using convolutional neural networks (CNN). Our implementation on the MNIST dataset takes 28.59 seconds to compute ten likelihoods of 64 input images simultaneously, yielding an amortized rate of 0.45 seconds per image.

Keywords. Homomorphic encryption; matrix computation; machine learning; neural networks

1 Introduction

Homomorphic Encryption (HE) is an encryption scheme that allows for operations on encrypted inputs so that the decrypted result matches the outcome for the corresponding operations on the plaintext. This property makes it very attractive for secure outsourcing tasks, including financial model evaluation and genetic testing, which can ensure the privacy and security of data communication, storage, and computation [3, 47]. In biomedicine, it is extremely attractive due to the privacy concerns about patients' sensitive data [29, 48]. Recently deep neural network based models have been demonstrated to achieve great success in a number of health care applications [37], and a natural question is whether we can outsource such learned models to a third party and evaluate new samples in a secure manner?

There are several different scenarios depending on who owns the data and who provides the model. Assuming a few different roles including data owners (e.g. hospital, institution or

individuals), cloud computing service providers (e.g. Amazon, Google, or Microsoft), and machine learning model providers (e.g. researchers and companies), we can imagine the following situations: (1) the data owner trains a model and makes it available on a computing service provider to be used to make predictions on encrypted inputs from other data owners; (2) model providers encrypt their trained classifier models and upload them to a cloud service provider to make predictions on encrypted inputs from various data owners; and (3) a cloud service provider trains a model on encrypted data from some data owners and uses the encrypted trained model to make predictions on new encrypted inputs. The first scenario has been previously studied in CryptoNets [25] and subsequent follow-up work [10, 7]. The second scenario was considered by Makri et al. [36] based on Multi-Party Computation (MPC) using polynomial kernel support vector machine classification. However, the second and third scenarios with an HE system have not been studied yet. In particular, classification tasks for these scenarios rely heavily on efficiency of secure matrix computation on encrypted inputs.

1.1 Our Contribution

In this paper, we introduce a generic method to perform arithmetic operations on encrypted matrices using an HE system. Our solution requires $O(d)$ homomorphic operations to compute a product of two encrypted matrices of size $d \times d$, compared to $O(d^2)$ of the previous best method. We extend basic matrix arithmetic to some advanced operations: transposition and rectangular matrix multiplication. We also describe how to encrypt multiple matrices in a single ciphertext, yielding a better amortized performance per matrix.

We apply our matrix computation mechanism to a new framework E2DM, which takes encrypted data and encrypted machine learning model to make predictions securely. This is the first HE-based solution that can be applied to the prediction phase of the second and third scenarios described above. As a benchmark of this framework, we implemented an evaluation of *convolutional neural networks* (CNN) model on the MNIST dataset [34] to compute ten likelihoods of handwritten images.

1.2 Technical Details

After Gentry's first construction of a fully HE scheme [23], there have been several attempts to improve efficiency and flexibility of HE systems. For example, the ciphertext packing technique allows multiple values to be encrypted in a single ciphertext, thereby performing parallel

$$\begin{pmatrix} a_0 & a_1 & a_2 \\ a_3 & a_4 & a_5 \\ a_6 & a_7 & a_8 \end{pmatrix} \cdot \begin{pmatrix} b_0 & b_1 & b_2 \\ b_3 & b_4 & b_5 \\ b_6 & b_7 & b_8 \end{pmatrix} = \begin{pmatrix} a_0 & a_1 & a_2 \\ a_4 & a_5 & a_3 \\ a_8 & a_6 & a_7 \end{pmatrix} \odot \begin{pmatrix} b_0 & b_4 & b_8 \\ b_3 & b_7 & b_2 \\ b_6 & b_1 & b_5 \end{pmatrix} +$$

$$\begin{pmatrix} a_1 & a_2 & a_0 \\ a_5 & a_3 & a_4 \\ a_6 & a_7 & a_8 \end{pmatrix} \odot \begin{pmatrix} b_3 & b_7 & b_2 \\ b_6 & b_1 & b_5 \\ b_0 & b_4 & b_8 \end{pmatrix} + \begin{pmatrix} a_2 & a_0 & a_1 \\ a_3 & a_4 & a_5 \\ a_7 & a_8 & a_6 \end{pmatrix} \odot \begin{pmatrix} b_6 & b_1 & b_5 \\ b_0 & b_4 & b_8 \\ b_3 & b_7 & b_2 \end{pmatrix}$$

Fig. 1: Our matrix multiplication algorithm with $d = 3$

computation on encrypted vectors in a Single Instruction Multiple Data (SIMD) manner. In the current literature, most of practical HE schemes [9, 8, 20, 15] support their own packing methods to achieve better amortized complexity of homomorphic operations. Besides component-wise addition and multiplication on plaintext vectors, these schemes provide additional functionalities such as scalar multiplication and slot rotation. In particular, permutations on plaintext slots enable us to interact with values located in different plaintext slots.

A naive solution for secure multiplication between two matrices of size $d \times d$ is to use d^2 distinct ciphertexts to represent each input matrix (one ciphertext per one matrix entry) and apply pure SIMD operations (addition and multiplication) on encrypted vectors. This method consumes one level for homomorphic multiplication, but it takes $O(d^3)$ multiplications. Another approach is to consider a matrix multiplication as a series of matrix-vector multiplications. Halevi and Shoup [26] introduced a matrix encoding method based on its diagonal decomposition, putting the matrix in diagonal order and mapping each of them to a single ciphertext. So it requires d ciphertexts to represent the matrix and the matrix-vector multiplication can be computed using $O(d)$ rotations and multiplications. Therefore, the matrix multiplication takes $O(d^2)$ complexity and has a depth of a single multiplication.

We propose an efficient method to perform matrix operations by combining HE-friendly operations on packed ciphertexts such as SIMD arithmetics, scalar multiplication, and slot rotation. We first define a simple encoding map that identifies an arbitrary matrix of size $d \times d$ with a vector of dimension $n = d^2$ having the same entries. Let $\odot$ denote the component-wise product between matrices. Then matrix multiplication can be expressed as $A \cdot B = \sum_{i=0}^{d-1} A_i \odot B_i$ for some matrices A_i (resp. B_i) obtained from A (resp. B) by taking specific permutations. Figure 1 describes this equality for the case of $d = 3$. We remark that the initial matrix A_0 (resp. B_0) can be computed with $O(d)$ rotations, and that for any $1 \leq i < d$ the permuted matrix A_i (resp. B_i) can be obtained by $O(1)$ rotations from the initial matrix. Thus the total computational complexity is bounded by $O(d)$ rotations and multiplications. We refer to Table 1 for comparison of our method with prior work in terms of the number of input ciphertexts for a single matrix, complexity, and the required depth for implementation. We denote homomorphic multiplication and constant multiplication by `Mult` and `CMult`, respectively.

Methodology	Number of ciphertexts	Complexity	Required depth
Naive method	d^2	$O(d^3)$	1 `Mult`
Halevi-Shoup [28]	d	$O(d^2)$	1 `Mult`
Ours	1	$O(d)$	1 `Mult` + 2 `CMult`

Table 1: Comparison of secure d-dimensional matrix multiplication algorithms

Our basic solution is based on the assumption that a ciphertext can encrypt d^2 plaintext slots, but it can be extended to support matrix computation of an arbitrary size. When a ciphertext has more than d^2 plaintext slots, for example, we can encrypt multiple matrices in a single ciphertext and carry out matrix operations in parallel. On the other hand, if a matrix is too large to be encoded into one ciphertext, one can partition it into several sub-matrices and encrypt them individually. An arithmetic operation over large matrices can be expressed using block-wise operations, and the computation on the sub-matrices can be securely done using our

basic matrix algorithms. We will use this approach to evaluate an encrypted neural networks model on encrypted data.

Our implementation is publicly available at https://github.com/K-miran/HEMat. It is based on an HE scheme of Cheon et al. [15], which is optimized in computation over the real numbers. For example, it took 9.21 seconds to securely compute the product of two matrices of size 64×64 and 2.56 seconds to transpose a single matrix of size 64×64. For the evaluation of an encrypted CNN model, we adapted a similar network topology to CryptoNets: one convolution layer and two fully connected (FC) layers with square activation function. This model is obtained from the keras library [16] by training 60,000 images of the MNIST dataset, and used for the classification of handwriting images of size 28×28. It took 28.59 seconds to compute ten likelihoods of encrypted 64 input images using the encrypted CNN model, yielding an amortized rate of 0.45 seconds per image. This model achieves a prediction accuracy of 98.1% on the test set.

2 Preliminaries

The binary logarithm will be simply denoted by $\log(\cdot)$. We denote vectors in bold, e.g. $\mathbf{a}$, and every vector in this paper is a row vector. For a $d_1 \times d$ matrix A_1 and a $d_2 \times d$ matrix A_2, $(A_1; A_2)$ denotes the $(d_1 + d_2) \times d$ matrix obtained by concatenating two matrices in a vertical direction. If two matrices A_1 and A_2 have the same number of rows, $(A_1|A_2)$ denotes a matrix formed by horizontal concatenation. We let λ denote the security parameter throughout the paper: all known valid attacks against the cryptographic scheme under scope should take $\Omega(2^\lambda)$ bit operations.

2.1 Homomorphic Encryption

HE is a cryptographic primitive that allows us to compute on encrypted data without decryption and generate an encrypted result which matches that of operations on plaintext [9, 20, 6, 15]. So it enables us to securely outsource computation to a public cloud. This technology has great potentials in many real-world applications such as statistical testing, machine learning, and neural networks [41, 31, 25, 32].

Let $\mathcal{M}$ and $\mathcal{C}$ denote the spaces of plaintexts and ciphertexts, respectively. An HE scheme $\Pi = (\mathtt{KeyGen}, \mathtt{Enc}, \mathtt{Dec}, \mathtt{Eval})$ is a quadruple of algorithms that proceeds as follows:

- $\mathtt{KeyGen}(1^\lambda)$. Given the security parameter λ, this algorithm outputs a public key pk, a public evaluation key evk and a secret key sk.
- $\mathtt{Enc}_{\mathsf{pk}}(\boldsymbol{m})$. Using the public key pk, the encryption algorithm encrypts a message $\boldsymbol{m} \in \mathcal{M}$ into a ciphertext $\mathsf{ct} \in \mathcal{C}$.
- $\mathtt{Dec}_{\mathsf{sk}}(\mathsf{ct})$. For the secret key sk and a ciphertext ct, the decryption algorithm returns a message $\boldsymbol{m} \in \mathcal{M}$.
- $\mathtt{Eval}_{\mathsf{evk}}(f; \mathsf{ct}_1, \ldots, \mathsf{ct}_k)$. Using the evaluation key evk, for a circuit $f : \mathcal{M}^k \to \mathcal{M}$ and a tuple of ciphertexts $(\mathsf{ct}_1, \ldots, \mathsf{ct}_k)$, the evaluation algorithm outputs a ciphertext $\mathsf{ct}' \in \mathcal{C}$.

An HE scheme Π is called *correct* if the following statements are satisfied with an overwhelming probability:

1. $\mathtt{Dec}_{\mathsf{sk}}(\mathsf{ct}) = \boldsymbol{m}$ for any $\boldsymbol{m} \in \mathcal{M}$ and $\mathsf{ct} \leftarrow \mathtt{Enc}_{\mathsf{pk}}(\boldsymbol{m})$.
2. $\mathtt{Dec}_{\mathsf{sk}}(\mathsf{ct}') = f(\boldsymbol{m}_1, \ldots, \boldsymbol{m}_k)$ with an overwhelming probability if $\mathsf{ct}' \leftarrow \mathtt{Eval}_{\mathsf{evk}}(f; \mathsf{ct}_1, \ldots, \mathsf{ct}_k)$ for an arithmetic circuit $f : \mathcal{M}^k \to \mathcal{M}$ and for some ciphertexts $\mathsf{ct}_1, \ldots, \mathsf{ct}_k \in \mathcal{C}$ such that $\mathtt{Dec}_{\mathsf{sk}}(\mathsf{ct}_i) = \boldsymbol{m}_i$.

An HE system can securely evaluate an arithmetic circuit f consisting of addition and multiplication gates. Throughout this paper, we denote by $\mathtt{Add}(\mathsf{ct}_1, \mathsf{ct}_2)$ and $\mathtt{Mult}_{\mathsf{evk}}(\mathsf{ct}_1, \mathsf{ct}_2)$ the homomorphic addition and multiplication between two ciphertexts ct_1 and ct_2, respectively. In addition, we let $\mathtt{CMult}_{\mathsf{evk}}(\mathsf{ct}; \boldsymbol{u})$ denote the multiplication of ct with a scalar $\boldsymbol{u} \in \mathcal{M}$. For simplicity, we will omit the subscript of the algorithms when it is clear from the context.

2.2 Ciphertext Packing Technique

Ciphertext packing technique allows us to encrypt multiple values into a single ciphertext and perform computation in a SIMD manner. After Smart and Vercauteren [46] first introduced a packing technique based on polynomial-CRT, it has been one of the most important features of HE systems. This method represents a native plaintext space $\mathcal{M}$ as a set of n-dimensional vectors in $\mathcal{R}^n$ over a ring $\mathcal{R}$ using appropriate encoding/decoding methods (each factor is called a *plaintext slot*). One can encode and encrypt an element of $\mathcal{R}^n$ into a ciphertext, and perform component-wise arithmetic operations over the plaintext slots at once. It enables us to reduce the expansion rate and parallelize the computation, thus achieving better performance in terms of amortized space and time complexity.

However, the ciphertext packing technique has a limitation that it is not easy to handle a circuit with some inputs in different plaintext slots. To overcome this problem, there have been proposed some methods to move data in the slots over encryption. For example, some HE schemes [24, 15] based on the ring learning with errors (RLWE) assumption exploit the structure of Galois group to implement the rotation operation on plaintext slots. That is, such HE schemes include the rotation algorithm, denoted by $\mathtt{Rot}(\mathsf{ct}; \ell)$, which transforms an encryption ct of $\boldsymbol{m} = (m_0, \ldots, m_{n-1}) \in \mathcal{M} = \mathcal{R}^n$ into an encryption of $\rho(\boldsymbol{m}; \ell) := (m_\ell, \ldots, m_{n-1}, m_0, \ldots, m_{\ell-1})$. Note that ℓ can be either positive or negative, and a rotation by $(-\ell)$ is the same as a rotation by $(n - \ell)$.

2.3 Linear Transformations

Halevi and Shoup [26] introduced a method to evaluate an arbitrary linear transformation on encrypted vectors. In general, an arbitrary linear transformation $L : \mathcal{R}^n \to \mathcal{R}^n$ over plaintext vectors can be represented as $L : \boldsymbol{m} \mapsto U \cdot \boldsymbol{m}$ for some matrix $U \in \mathcal{R}^{n \times n}$. We can express the matrix-vector multiplication by combining rotation and constant multiplication operations.

Specifically, for $0 \le \ell < n$, we define the *ℓ-th diagonal vector* of U by $\boldsymbol{u}_\ell = (U_{0,\ell}, U_{1,\ell+1}, \ldots, U_{n-\ell-1,n-1}, U_{n-\ell,0}, \ldots, U_{n-1,\ell-1}) \in \mathcal{R}^n$. Then we have

$$U \cdot \boldsymbol{m} = \sum_{0 \le \ell < n} (\boldsymbol{u}_\ell \odot \rho(\boldsymbol{m}; \ell)) \tag{1}$$

where $\odot$ denotes the component-wise multiplication between vectors. Given a matrix $U \in \mathcal{R}^{n \times n}$ and an encryption ct of the vector $\boldsymbol{m}$, Algorithm 1 describes how to compute a ciphertext of the desired vector $U \cdot \boldsymbol{m}$.

As shown in Algorithm 1, the computational cost of matrix-vector multiplication is about n additions, constant multiplications, and rotations. Note that rotation operation needs to perform a key-switching operation and thus is comparably expensive than the other two operations. So we can say that the complexity is asymptotically $O(n)$ rotations. It can be reduced when the number of nonzero diagonal vectors of U is relatively small.

Algorithm 1 Homomorphic linear transformation

procedure $\texttt{LinTrans}(\texttt{ct}; U)$
1: $\texttt{ct}' \leftarrow \texttt{CMult}(\texttt{ct}; \boldsymbol{u}_0)$
2: **for** $\ell = 1$ to $n-1$ **do**
3: $\quad \texttt{ct}' \leftarrow \texttt{Add}(\texttt{ct}', \texttt{CMult}(\texttt{Rot}(\texttt{ct}; \ell); \boldsymbol{u}_\ell))$
4: **end for**
5: **return** $\texttt{ct}'$

3 Secure Matrix Multiplication via Homomorphic Encryption

In this section, we propose a simple encoding method to convert a matrix into a plaintext vector in a SIMD environment. Based on this encoding method, we devise an efficient algorithm to carry out basic matrix operations over encryption.

3.1 Permutations for Matrix Multiplication

We propose an HE-friendly expression of the matrix multiplication operation. For a $d \times d$ square matrix $A = (A_{i,j})_{0 \le i,j < d}$, we first define useful permutations σ, τ, ϕ, and ψ on the set $\mathcal{R}^{d \times d}$. For simplicity, we identify $\mathbb{Z} \cap [0, d)$ as a representative of $\mathbb{Z}_d$ and write $[i]_d$ to denote the reduction of an integer i modulo d into that interval. All the indexes will be considered as integers modulo d.

- $\sigma(A)_{i,j} = A_{i,i+j}$.
- $\tau(A)_{i,j} = A_{i+j,j}$.
- $\phi(A)_{i,j} = A_{i,j+1}$.
- $\psi(A)_{i,j} = A_{i+1,j}$.

Note that ϕ and ψ represent the column and row shifting functions, respectively. Then for two square matrices A and B of order d, we can express their matrix product AB as follows:

$$A \cdot B = \sum_{k=0}^{d-1} \left(\phi^k \circ \sigma(A)\right) \odot \left(\psi^k \circ \tau(B)\right), \tag{2}$$

where $\odot$ denotes the component-wise multiplication between matrices. The correctness is shown in the following equality by computing the matrix component of the index (i, j):

$$\begin{aligned}
\sum_{k=0}^{d-1} \left(\phi^k \circ \sigma(A)\right)_{i,j} \cdot \left(\psi^k \circ \tau(B)\right)_{i,j} &= \sum_{k=0}^{d-1} \sigma(A)_{i,j+k} \cdot \tau(B)_{i+k,j} \\
&= \sum_{k=0}^{d-1} A_{i,i+j+k} \cdot B_{i+j+k,j} \\
&= \sum_{k=0}^{d-1} A_{i,k} \cdot B_{k,j} \\
&= (A \cdot B)_{i,j}.
\end{aligned}$$

Since Equation (2) consists of permutations on matrix entries and the Hadamard multiplication operations, we can efficiently evaluate it using an HE system with ciphertext packing method.

3.2 Matrix Encoding Method

We propose a row ordering encoding map to transform a vector of dimension $n = d^2$ into a matrix in $\mathcal{R}^{d\times d}$. For a vector $\boldsymbol{a} = (a_k)_{0\le k<n}$, we define the encoding map $\iota : \mathcal{R}^n \to \mathcal{R}^{d\times d}$ by

$$\iota : \boldsymbol{a} \mapsto A = (a_{d\cdot i+j})_{0\le i,j<d},$$

i.e., $\boldsymbol{a}$ is the concatenation of row vectors of A. It is clear that $\iota(\cdot)$ is an isomorphism between additive groups, which implies that matrix addition can be securely computed using homomorphic addition in a SIMD manner. In addition, one can perform multiplication by scalars by adapting a constant multiplication of an HE scheme. Throughout this paper, we identify two spaces $\mathcal{R}^n$ and $\mathcal{R}^{d\times d}$ with respect to the $\iota(\cdot)$, so a ciphertext will be called an encryption of A if it encrypts the plaintext vector $\boldsymbol{a} = \iota^{-1}(A)$.

3.3 Matrix Multiplication on Packed Ciphertexts

An arbitrary permutation operation on $\mathcal{R}^{d\times d}$ can be understood as a linear transformation $L : \mathcal{R}^n \to \mathcal{R}^n$ such that $n = d^2$. In general, its matrix representation $U \in \{0,1\}^{n\times n} \subseteq \mathcal{R}^{n\times n}$ has n number of nonzero diagonal vectors. So if we directly evaluate the permutations $A \mapsto \phi^k \circ \sigma(A)$ and $B \mapsto \psi^k \circ \tau(B)$ for $1 \le k < d$, each of them requires $O(d^2)$ homomorphic operations and thus the total complexity is $O(d^3)$. We provide an efficient algorithm to perform the matrix multiplication on packed ciphertexts by combining Equation (2) and our matrix encoding map.

3.3.1 Tweaks of Permutations

We focus on the following four permutations σ, τ, ϕ, and ψ described above. We let U^σ, U^τ, V, and W denote the matrix representations corresponding to these permutations, respectively. Firstly, the matrix representations U^σ and U^τ of σ and τ are expressed as follows:

$$U^\sigma_{d\cdot i+j,\ell} = \begin{cases} 1 & \text{if } \ell = d\cdot i + [i+j]_d; \\ 0 & \text{otherwise;} \end{cases}$$

$$U^\tau_{d\cdot i+j,\ell} = \begin{cases} 1 & \text{if } \ell = d\cdot [i+j]_d + j; \\ 0 & \text{otherwise,} \end{cases}$$

for $0 \le i, j < d$ and $0 \le \ell < d^2$. Similarly, for $1 \le k < d$, the matrix representations of ϕ^k and ψ^k can be computed as follows:

$$V^k_{d\cdot i+j,\ell} = \begin{cases} 1 & \text{if } \ell = d\cdot i + [j+k]_d; \\ 0 & \text{otherwise;} \end{cases}$$

$$W^k_{d\cdot i+j,\ell} = \begin{cases} 1 & \text{if } \ell = d\cdot [i+k]_d + j; \\ 0 & \text{otherwise,} \end{cases}$$

for $0 \le i, j < d$ and $0 \le \ell < d^2$.

As described in Equation (1), we employ the diagonal decomposition of the matrix representations for multiplications with encrypted vectors. Let us count the number of diagonal vectors

to estimate the complexity. We use the same notation $\boldsymbol{u}_\ell$ to write the ℓ-th diagonal vector of a matrix U. For simplicity, we identify $\boldsymbol{u}_{d^2-\ell}$ with $\boldsymbol{u}_{-\ell}$. The matrix U^σ has exactly $(2d-1)$ number of nonzero diagonal vectors, denoted by $\boldsymbol{u}_k^\sigma$ for $k \in \mathbb{Z} \cap (-d, d)$. The ℓ-th diagonal vector of U^τ is nonzero if and only if ℓ is divisible by the integer d, so U^τ has d nonzero diagonal vectors. For any $1 \leq k < d$, the matrix V^k has two nonzero diagonal vectors $\boldsymbol{v}_k$ and $\boldsymbol{v}_{k-d}$. Similarly, the matrix W^k has the only nonzero diagonal vector $\boldsymbol{w}_{d\cdot k}$. Therefore, homomorphic evaluations of the permutations σ and τ require $O(d)$ rotations while it takes $O(1)$ rotations to compute ψ^k or ϕ^k for any $1 \leq k < d$.

3.3.2 Homomorphic Matrix Multiplication

Suppose that we are given two ciphertexts $\mathsf{ct}.A$ and $\mathsf{ct}.B$ that encrypt matrices A and B of size $d \times d$, respectively. In the following, we describe an efficient evaluation strategy for homomorphic matrix multiplication.

Step 1-1: This step performs the linear transformation U^σ on the input ciphertext $\mathsf{ct}.A$. As mentioned above, the matrix U^σ is a sparse matrix with $(2d-1)$ number of nonzero diagonal vectors $\boldsymbol{u}_k^\sigma$ for $k \in \mathbb{Z} \cap (-d, d)$, so we can represent the linear transformation as

$$U^\sigma \cdot \boldsymbol{a} = \sum_{-d<k<d} (\boldsymbol{u}_k^\sigma \odot \rho(\boldsymbol{a}; k)) \tag{3}$$

where $\boldsymbol{a} = \iota^{-1}(A) \in \mathcal{R}^n$ is the vector representation of A. If $k \geq 0$, the k-th diagonal vector is computed by

$$\boldsymbol{u}_k^\sigma[\ell] = \begin{cases} 1 & \text{if } 0 \leq \ell - d \cdot k < (d-k); \\ 0 & \text{otherwise,} \end{cases}$$

where $\boldsymbol{u}_k^\sigma[\ell]$ denotes the ℓ-th component of $\boldsymbol{u}_k^\sigma$. In the other cases $k < 0$, it is computed by

$$\boldsymbol{u}_k^\sigma[\ell] = \begin{cases} 1 & \text{if } -k \leq \ell - (d+k) \cdot d < d; \\ 0 & \text{otherwise.} \end{cases}$$

Then Equation (3) can be securely computed as

$$\sum_{-d<k<d} \mathtt{CMult}\left(\mathtt{Rot}(\mathsf{ct}.A; k); \boldsymbol{u}_k^\sigma\right),$$

resulting the encryption of the plaintext vector $U^\sigma \cdot \boldsymbol{a}$, denoted by $\mathsf{ct}.A^{(0)}$. Thus, the computational cost is about $2d$ additions, constant multiplications, and rotations.

Step 1-2: This step is to evaluate the linear transformation U^τ on the input ciphertext $\mathsf{ct}.B$. As described above, the matrix U^τ has d nonzero diagonal vectors so we can express this matrix-vector multiplication as

$$U^\tau \cdot \boldsymbol{b} = \sum_{0 \leq k < d} (\boldsymbol{u}_{d\cdot k}^\tau \odot \rho(\boldsymbol{b}; d \cdot k)), \tag{4}$$

where $\boldsymbol{b} = \iota^{-1}(B)$ and $\boldsymbol{u}_{d\cdot k}^\tau$ is the $(d \cdot k)$-th diagonal vector of the matrix U^τ. We note that for any $0 \leq k < d$, the vector $\boldsymbol{u}_{d\cdot k}^\tau$ contains one in the $(k + d \cdot i)$-th component for $0 \leq i < d$ and zeros in all the other entries. Then Equation (4) can be securely computed as

$$\sum_{0 \leq k < d} \mathtt{CMult}\left(\mathtt{Rot}(\mathsf{ct}.B; d \cdot k); \boldsymbol{u}_{d\cdot k}^\tau\right),$$

resulting the encryption of the plaintext vector $U^\tau \cdot \boldsymbol{b}$, denoted by $\mathsf{ct}.B^{(0)}$. The complexity of this procedure is roughly half of the Step 1-1: d additions, constant multiplications, and rotations.

Step 2: This step securely computes the column and row shifting operations of $\sigma(A)$ and $\tau(B)$, respectively. For $1 \leq k < d$, the column shifting matrix V^k has two nonzero diagonal vectors $\boldsymbol{v}_k$ and $\boldsymbol{v}_{k-d}$ that are computed by

$$\boldsymbol{v}_k[\ell] = \begin{cases} 1 & \text{if } 0 \leq [\ell]_d < (d-k); \\ 0 & \text{otherwise;} \end{cases}$$

$$\boldsymbol{v}_{k-d}[\ell] = \begin{cases} 1 & \text{if } (d-k) \leq [\ell]_d < d; \\ 0 & \text{otherwise.} \end{cases}$$

Then we get an encryption $\mathsf{ct}.A^{(k)}$ of the matrix $\phi^k \circ \sigma(A)$ by adding two ciphertexts $\texttt{CMult}(\texttt{Rot}(\mathsf{ct}.A^{(0)}; k); \boldsymbol{v}_k)$ and $\texttt{CMult}(\texttt{Rot}(\mathsf{ct}.A^{(0)}; k-d); \boldsymbol{v}_{k-d})$. In the case of the row shifting permutation, the corresponding matrix W^k has exactly one nonzero diagonal vector $\boldsymbol{w}_{d \cdot k}$ whose entries are all one. Thus we can obtain an encryption of the matrix $\psi^k \circ \tau(B)$ by computing $\mathsf{ct}.B^{(k)} \leftarrow \texttt{Rot}(\mathsf{ct}.B^{(0)}; d \cdot k)$. The computational cost of this procedure is about d additions, $2d$ constant multiplications, and $3d$ rotations.

Step 3: This step computes the Hadamard multiplication between the ciphertexts $\mathsf{ct}.A^{(k)}$ and $\mathsf{ct}.B^{(k)}$for $0 \leq k < d$, and finally aggregates all the resulting ciphertexts. As a result, we get an encryption $\mathsf{ct}.AB$ of the matrix AB. The running time of this step is d homomorphic multiplications and additions.

In summary, we can perform the homomorphic matrix multiplication operation as described in Algorithm 2.

Algorithm 2 Homomorphic matrix multiplication

procedure $\texttt{HE-MatMult}(\mathsf{ct}.A, \mathsf{ct}.B)$

[Step 1-1]:

$\mathsf{ct}.A^{(0)} \leftarrow \texttt{LinTrans}(\mathsf{ct}.A; U^\sigma)$

[Step 1-2]:

$\mathsf{ct}.B^{(0)} \leftarrow \texttt{LinTrans}(\mathsf{ct}.B; U^\tau)$

[Step 2]:

for $k = 1$ to $d-1$ **do**
    $\mathsf{ct}.A^{(k)} \leftarrow \texttt{LinTrans}(\mathsf{ct}.A^{(0)}; V^k)$
    $\mathsf{ct}.B^{(k)} \leftarrow \texttt{LinTrans}(\mathsf{ct}.B^{(0)}; W^k)$
end for

[Step 3]:

$\mathsf{ct}.AB \leftarrow \texttt{Mult}(\mathsf{ct}.A^{(0)}, \mathsf{ct}.B^{(0)})$
for $k = 1$ to $d-1$ **do**
    $\mathsf{ct}.AB \leftarrow \texttt{Add}(\mathsf{ct}.AB, \texttt{Mult}(\mathsf{ct}.A^{(k)}, \mathsf{ct}.B^{(k)}))$
end for
return $\mathsf{ct}.AB$

3.3.3 Further Improvements

This implementation of matrix multiplication takes about $5d$ additions, $5d$ constant multiplications, $6d$ rotations, and d multiplications. The complexity of Steps 1-1 and 1-2 can be reduced by applying the idea of baby-step/giant-step algorithm. Given an integer $k \in (-d, d)$, we can write $k = \sqrt{d} \cdot i + j$ for some $-\sqrt{d} < i < \sqrt{d}$ and $0 \leq j < \sqrt{d}$. It follows from [27, 28] that Equation (3) can be expressed as

$$U^\sigma \cdot \boldsymbol{a} = \sum_{\substack{-\sqrt{d}<i<\sqrt{d} \\ 0 \leq j < \sqrt{d}}} \left(\boldsymbol{u}^\sigma_{\sqrt{d}\cdot i + j} \odot \rho(\boldsymbol{a}; \sqrt{d} \cdot i + j) \right) = \sum_{-\sqrt{d}<i<\sqrt{d}} \rho \left(\sum_{0 \leq j < \sqrt{d}} \boldsymbol{a}_{i,j};\ \sqrt{d} \cdot i \right)$$

where $\boldsymbol{a}_{i,j} = \rho(\boldsymbol{u}^\sigma_{\sqrt{d}\cdot i+j}; -\sqrt{d} \cdot i) \odot \rho(\boldsymbol{a}; j)$. We first compute encryptions of baby-step rotations $\rho(\boldsymbol{a}; j)$ for $0 \leq j < \sqrt{d}$. We use them to compute the ciphertexts of $\boldsymbol{a}_{i,j}$'s using only constant multiplications. After that, we perform $\sqrt{d}$ additions, $\sqrt{d}$ constant multiplications, and a single rotation for each i. In total, Step 1-1 can be homomorphically evaluated with $2d$ additions, $2d$ constant multiplications, and $3\sqrt{d}$ rotations. Step 1-2 can be computed in a similar way using d additions, d constant multiplications, and $2\sqrt{d}$ rotations.

On the other hand, we can further reduce the number of constant multiplications in Step 2 by leveraging two-input multiplexers. The sum of $\rho(\boldsymbol{v}_k; -k)$ and $\rho(\boldsymbol{v}_{k-d}; d-k)$ generates a plaintext vector that has 1's in all the slots, which implies that

$$\begin{aligned} \texttt{CMult}(\texttt{Rot}(\texttt{ct}.A^{(0)}; k-d); \boldsymbol{v}_{k-d}) &= \texttt{Rot}(\texttt{CMult}(\texttt{ct}.A^{(0)}; \rho(\boldsymbol{v}_{k-d}; d-k)); k-d) \\ &= \texttt{Rot}(\texttt{ct}.A^{(0)} - \texttt{CMult}(\texttt{ct}.A^{(0)}; \rho(\boldsymbol{v}_k, -k)); k-d). \end{aligned}$$

For each $1 \leq k < d$, we compute $\texttt{CMult}(\texttt{ct}.A^{(0)}; \rho(\boldsymbol{v}_k, -k))$. Then, using the fact that

$$\texttt{CMult}(\texttt{Rot}(\texttt{ct}.A^{(0)}; k); \boldsymbol{v}_k) = \texttt{Rot}(\texttt{CMult}(\texttt{ct}.A^{(0)}; \rho(\boldsymbol{v}_k, -k)); k),$$

we obtain the desired ciphertext $\texttt{ct}.A^{(k)}$ with addition and rotation operations.

Table 2 summarizes the complexity and the required depth of each step of Algorithm 2 with the proposed optimization techniques.

Step	Add	CMult	Rot	Mult	Depth
1-1	$2d$	$2d$	$3\sqrt{d}$	-	1 CMult
1-2	d	d	$2\sqrt{d}$	-	
2	$2d$	d	$3d$	-	1 CMult
3	d	-	-	d	1 Mult
Total	$6d$	$4d$	$3d + 5\sqrt{d}$	d	1 Mult + 2 CMult

Table 2: Complexity and required depth of Algorithm 2

4 Advanced Operations for Homomorphic Matrix Computation

In this section, we introduce a method to transpose a matrix over an HE system. We also present a faster rectangular matrix multiplication by employing the ideas from Algorithm 2. We can further extend our algorithms to parallel matrix computation without additional cost.

4.1 Matrix Transposition on Packed Ciphertexts

Let U^t be the matrix representation of the transpose map $A \mapsto A^t$ on $\mathcal{R}^{d\times d} \cong \mathcal{R}^n$. For $0 \le i, j < d$, its entries are given by

$$U^t_{d\cdot i+j,k} = \begin{cases} 1 & \text{if } k = d\cdot j + i; \\ 0 & \text{otherwise.} \end{cases}$$

The k-th diagonal vector of U^t is nonzero if and only if $k = (d-1)\cdot i$ for some $i \in \mathbb{Z} \cap (-d, d)$, so the matrix U^t is a sparse matrix with $(2d-1)$ nonzero diagonal vectors. We can represent this linear transformation as

$$U^t \cdot \boldsymbol{a} = \sum_{-d<i<d} (\boldsymbol{t}_{(d-1)\cdot i} \odot \rho(\boldsymbol{a}; (d-1)\cdot i))$$

where $\boldsymbol{t}_{(d-1)\cdot i}$ denotes the nonzero diagonal vector of U^t. The ℓ-th component of the vector $\boldsymbol{t}_{(d-1)\cdot i}$ is computed by

$$\boldsymbol{t}_{(d-1)\cdot i}[\ell] = \begin{cases} 1 & \text{if } \ell - i = (d+1)\cdot j, 0 \le j < d-i; \\ 0 & \text{otherwise,} \end{cases}$$

if $i \ge 0$, or

$$\boldsymbol{t}_{(d-1)\cdot i}[\ell] = \begin{cases} 1 & \text{if } \ell + i = (d+1)\cdot j, 0 \le j < d+i; \\ 0 & \text{otherwise,} \end{cases}$$

if $i < 0$. The total computational cost is about $2d$ rotations and the baby-step/giant-step approach can be used to reduce the complexity; the number of automorphism can be reduced down to $3\sqrt{d}$.

4.2 Rectangular Matrix Multiplication

In this section, we design an efficient algorithm for rectangular matrix multiplication such as $\mathcal{R}^{\ell\times d} \times \mathcal{R}^{d\times d} \to \mathcal{R}^{\ell\times d}$ or $\mathcal{R}^{d\times d} \times \mathcal{R}^{d\times \ell} \to \mathcal{R}^{d\times \ell}$. For convenience, let us consider the former case that A has a smaller number of rows than columns (i.e., $\ell < d$). A naive solution is to generate a square matrix by padding zeros in the bottom of the matrix A and perform the homomorphic matrix multiplication algorithm in Section 3.3, resulting in running time of $O(d)$ rotations and multiplications. However, we can further optimize the complexity by manipulating its matrix multiplication representation using a special property of permutations described in Section 3.1.

4.2.1 Refinements of Rectangular Matrix Multiplication

Suppose that we are given an $\ell \times d$ matrix A and a $d \times d$ matrix B such that ℓ divides d. Since σ and ϕ are defined as row-wise operations, the restrictions to the rectangular matrix A are well-defined permutations on A. By abuse of notation, we use the same symbols σ and ϕ to denote the restrictions. We also use $C_{\ell_1:\ell_2}$ to denote the $(\ell_2 - \ell_1) \times d$ submatrix of C formed by extracting from ℓ_1-th row to the $(\ell_2 - 1)$-th row of C. Then their matrix product AB has shape $\ell \times d$ and it can be expressed as follows:

$$\begin{aligned} A \cdot B &= \sum_{0 \le k < d} (\phi^k \circ \sigma(A)) \odot \left((\psi^k \circ \tau(B))_{0:\ell} \right) \\ &= \sum_{0 \le i < \ell} \sum_{0 \le j < d/\ell} (\phi^{j \cdot \ell + i} \circ \sigma(A)) \odot \left((\psi^{j \cdot \ell + i} \circ \tau(B))_{0:\ell} \right). \end{aligned}$$

Our key observation is the following lemma, which gives an idea of a faster rectangular matrix multiplication algorithm.

Lemma 1. *Two permutations σ and ϕ are commutative. In general, we have $\phi^k \circ \sigma = \sigma \circ \phi^k$ for $k > 0$. Similarly, we obtain $\psi^k \circ \tau = \tau \circ \psi^k$ for $k > 0$.*

Now let us define a $d \times d$ matrix $\bar{A}$ containing (d/ℓ) copies of A in a vertical direction (i.e., $\bar{A} = (A; \ldots; A)$). Lemma 1 implies that

$$\begin{aligned} (\phi^i \circ \sigma(\bar{A}))_{j \cdot \ell : (j+1) \cdot \ell} &= \phi^i \circ (\sigma(\bar{A})_{j \cdot \ell : (j+1) \cdot \ell}) \\ &= \phi^i \circ \sigma \circ \phi^{j \cdot \ell}(A) \\ &= \phi^{j \cdot \ell + i} \circ \sigma(A). \end{aligned}$$

Similarly, using the commutative property of τ and ψ, it follows

$$(\psi^i \circ \tau(B))_{j \cdot \ell : (j+1) \cdot \ell} = (\psi^{j \cdot \ell + i} \circ \tau(B))_{0:\ell}.$$

Therefore, the matrix product AB is written as follows:

$$A \cdot B = \sum_{0 \le j < d/\ell} \left(\sum_{0 \le i < \ell} (\phi^i \circ \sigma(\bar{A})) \odot (\psi^i \circ \tau(B)) \right)_{j \cdot \ell : (j+1) \cdot \ell}$$

4.2.2 Homomorphic Rectangular Matrix Multiplication

Suppose that we are given two ciphertexts $\mathsf{ct}.\bar{A}$ and $\mathsf{ct}.B$ that encrypt matrices $\bar{A}$ and B, respectively. We first apply the baby-step/giant-step algorithm to generate the encryptions of $\sigma(\bar{A})$ and $\tau(B)$ as in Section 3.3.3. Next, we can securely compute $\sum_{i=0}^{\ell-1} (\phi^i \circ \sigma(\bar{A})) \odot (\psi^i \circ \tau(B))$ in a similar way to Algorithm 2, say the output is $\mathsf{ct}.\bar{A}B$. Finally, we perform aggregation and rotation operations to get the final result: $\sum_{j=0}^{d/\ell-1} \mathtt{Rot}(\mathsf{ct}.\bar{A}B; j \cdot \ell \cdot d)$. This step can be evaluated using a repeated doubling approach, yielding a running time of $\log(d/\ell)$ additions and rotations. See Algorithm 3 for an explicit description of homomorphic rectangular matrix multiplication.

Table 3 summarizes the total complexity of Algorithm 3. Even though we need additional computation for Step 4, we can reduce the complexities of Step 2 and 3 to $O(\ell)$ rotations and ℓ multiplications, respectively. We also note that the final output $\mathsf{ct}.AB$ encrypts a $d \times d$ matrix containing (d/ℓ) copies of the desired matrix product AB in a vertical direction.

This resulting ciphertext has the same form as a rectangular input matrix of Algorithm 3, so it can be reusable for further matrix computation without additional cost.

Algorithm 3 Homomorphic rectangular matrix multiplication

procedure HE-RMatMult($\mathsf{ct}.\bar{A}, \mathsf{ct}.B$)

[Step 1]:

$\mathsf{ct}.A^{(0)} \leftarrow \mathtt{LinTrans}(\mathsf{ct}.\bar{A}; U^{\sigma})$

$\mathsf{ct}.B^{(0)} \leftarrow \mathtt{LinTrans}(\mathsf{ct}.B; U^{\tau})$

[Step 2]:

for $k = 1$ to $\ell - 1$ **do**

  $\mathsf{ct}.A^{(k)} \leftarrow \mathtt{LinTrans}(\mathsf{ct}.A^{(0)}; V^k)$

  $\mathsf{ct}.B^{(k)} \leftarrow \mathtt{LinTrans}(\mathsf{ct}.B^{(0)}; W^k)$

end for

[Step 3]:

$\mathsf{ct}.\bar{A}B \leftarrow \mathtt{Mult}(\mathsf{ct}.A^{(0)}, \mathsf{ct}.B^{(0)})$

for $k = 1$ to $\ell - 1$ **do**

  $\mathsf{ct}.\bar{A}B \leftarrow \mathtt{Add}(\mathsf{ct}.\bar{A}B, \mathtt{Mult}(\mathsf{ct}.A^{(k)}, \mathsf{ct}.B^{(k)}))$

end for

[Step 4]:

$\mathsf{ct}.AB \leftarrow \mathsf{ct}.\bar{A}B$

for $k = 0$ to $\log(d/\ell) - 1$ **do**

  $\mathsf{ct}.AB \leftarrow \mathtt{Add}(\mathsf{ct}.AB, \mathtt{Rot}(\mathsf{ct}.AB; \ell \cdot d \cdot 2^k))$

end for

return $\mathsf{ct}.AB$

4.3 Parallel Matrix Computation

Throughout Sections 3 and 4, we have identified the message space $\mathcal{M} = \mathcal{R}^n$ with the set of matrices $\mathcal{R}^{d \times d}$ under the assumption that $n = d^2$. However, most of HE schemes [9, 20, 15] have a quite large number of plaintext slots (e.g. thousands) compared to the matrix dimension in some real-world applications, i.e., $n \gg d^2$. If a ciphertext can encrypt only one matrix, most of plaintext slots would be wasted. This subsection introduces an idea that allows multiple matrices to be encrypted in a single ciphertext, thereby performing parallel matrix computation in a SIMD manner.

For simplicity, we assume that n is divisible by d^2 and let $g = n/d^2$. We modify the encoding map in Section 3.2 to make a 1-to-1 correspondence ι_g between $\mathcal{R}^n$ and $(\mathcal{R}^{d \times d})^g$, which transforms an n-dimensional vector into a g-tuple of square matrices of order d. Specifically, for an input vector $\boldsymbol{a} = (a_\ell)_{0 \le \ell < n}$, we define ι_g by

$$\iota_g : \boldsymbol{a} \mapsto \left(A_k = (a_{g \cdot (d \cdot i + j) + k})\right)_{0 \le k < g}.$$

The components of $\boldsymbol{a}$ with indexes congruent to k modulo g are corresponding to the k-th matrix A_k.

We note that for an integer $0 \le \ell < d^2$, the rotation operation $\rho(\boldsymbol{a}; g \cdot \ell)$ represents the matrix-wise rotation by ℓ positions. It can be naturally extended to the other matrix-wise operations including scalar linear transformation and matrix multiplication. For example, we can encrypt g number of $d \times d$ matrices into a single ciphertext and perform the matrix multiplication operations between g pairs of matrices at once by applying our matrix multiplication algorithm

Step	Add	CMult	Rot	Mult
1	$3d$	$3d$	$5\sqrt{d}$	-
2	ℓ	2ℓ	3ℓ	-
3	ℓ	-	-	ℓ
4	$\log(d/\ell)$	-	$\log(d/\ell)$	
Total	$3d + 2\ell + \log(d/\ell)$	$3d + 2\ell$	$3\ell + 5\sqrt{d} + \log(d/\ell)$	ℓ

Table 3: Complexity of Algorithm 3

on two ciphertexts. The total complexity remains the same as Algorithm 2, which results in a less amortized computational complexity of $O(d/g)$ per matrix.

5 Implementation of Homomorphic Matrix Operations

In this section, we report the performance of our homomorphic matrix operations and analyze the performance of the implementation. For simplicity, we assume that d is a power-of-two integer. In general, we can pad zeros at the end of each row to set d as a power of two.

In our implementation, we employ a special encryption scheme suggested by Cheon et al. [15], which supports approximate computation over encrypted data, called HEAAN. A unique property of the HE scheme is the *rescaling* procedure to truncate a ciphertext into a smaller modulus, which leads to rounding of the plaintext. This plays an important role in controlling the magnitude of messages, and thereby achieving efficiency of approximate computation. For further details, we refer to [15] and [12].

All the experiments were performed on a Macbook Pro laptop with an Intel Core i7 running with 4 cores rated at 2.9 GHz. Our implementation is based on the HEAAN library [14] with C++ based Shoup's NTL library [44].

5.1 Parameter Setting

We present how to select parameters of our underlying HE scheme to support homomorphic matrix operations described in Section 3 and 4. Our underlying HE scheme is based on the RLWE assumption over the cyclotomic ring $\mathcal{R} = \mathbb{Z}[X]/(X^N + 1)$ for a power-of-two integer N. Let us denote by $[\cdot]_q$ the reduction modulo q into the interval $(-q/2, q/2] \cap \mathbb{Z}$ of the integer. We write $\mathcal{R}_q = \mathcal{R}/q\mathcal{R}$ for the residue ring of $\mathcal{R}$ modulo an integer q. The native plaintext space is represented as a set of $(N/2)$-dimensional complex vectors.

Suppose that all the elements are scaled by a factor of an integer p and then converted into the nearest integers for quantization. If we are multiplying a ciphertext by a plaintext vector, we assume that the constant vector is scaled by a factor of an integer p_c to maintain the precision. Thus, the rescaling procedure after homomorphic multiplication reduces a ciphertext modulus by p while the rescaling procedure after a constant multiplication reduces a modulus by p_c. For example, Algorithm 2 has depth of two constant multiplications for Step 1 and 2, and has additional depth of a single homomorphic multiplication for Step 3. This implies that an input ciphertext modulus is reduced by $(2 \log p_c + \log p)$ bits after the matrix multiplication algorithm.

As a result, we obtain the following lower bound on the bit length of a fresh ciphertext modulus, denoted by $\log q$:

$$\log q = \begin{cases} \log q_0 & \text{for } \texttt{HE-MatAdd}; \\ 2\log p_{\mathsf{c}} + \log p + \log q_0 & \text{for } \texttt{HE-MatMult}; \\ \log p_{\mathsf{c}} + \log q_0 & \text{for } \texttt{HE-MatTrans}, \end{cases}$$

where q_0 is the output ciphertext modulus. The final ciphertext represents the desired vector but is scaled by a factor of p, which means that $\log q_0$ should be larger than $\log p$. In our implementation, we take $\log p = 24, \log p_{\mathsf{c}} = 15$, and $\log q_0 = 32$.

We use the discrete Gaussian distribution of standard deviation $\sigma = 3.2$ to sample error polynomials. The secret-key polynomials were sampled from the discrete ternary uniform distribution over $\{0, \pm 1\}^N$. The cyclotomic ring dimension is chosen as $N = 2^{13}$ to achieve a 80-bit security level against the known attacks of the LWE problem based on the estimator of Albrecht et al. [2]. In short, we present three parameter sets and sizes of the corresponding fresh ciphertexts as follows:

$$(N, q, \text{size}) = \begin{cases} (2^{13}, 2^{32}, 64 \text{ KB}) & \text{for } \texttt{HE-MatAdd}; \\ (2^{13}, 2^{86}, 172 \text{ KB}) & \text{for } \texttt{HE-MatMult}; \\ (2^{13}, 2^{47}, 94 \text{ KB}) & \text{for } \texttt{HE-MatTrans}. \end{cases}$$

5.2 Performance of Matrix Operations

Table 4 presents timing results for matrix addition, multiplication, and transposition for various matrix sizes from 4 to 64 where the *throughput* means the number of matrices being processed in parallel. We provide three distinct implementation variants: *single-packed*, *sparsely-packed*, and *fully-packed*. The single-packed implementation is that a ciphertext represents only a single matrix; two other implementations are to encode several matrices into sparsely or fully packed plaintext slots. We use the same parameters for all variants, and thus each ciphertext can hold up to $N/2 = 2^{12}$ plaintext values. For example, if we consider 4×4 matrices, we can process operations over $2^{12}/(4 \cdot 4) = 256$ distinct matrices simultaneously. In the case of dimension 16, a ciphertext can represent up to $2^{12}/(16 \cdot 16) = 16$ distinct matrices.

Dim	Throughput	Message size	Expansion rate	Encoding+ Encryption	Decoding+ Decryption	Relative time per matrix		
						`HE-MatAdd`	`HE-MatMult`	`HE-MatTrans`
4	1	0.47 KB	3670	34 ms	9 ms	0.62 ms	779 ms	363 ms
	16	0.75 KB	229	41 ms	12 ms	0.05 ms	47 ms	18 ms
	256	12.0 KB	14.3	95 ms	81 ms	0.03 ms	3 ms	1 ms
16	1	0.75 KB	229	33 ms	13 ms	0.62 ms	2501 ms	847 ms
	4	3.0 KB	57.3	48 ms	27 ms	0.19 ms	649 ms	211 ms
	16	12.0 KB	14.3	97 ms	78 ms	0.04 ms	162 ms	49 ms
64	1	12.0 KB	14.3	108 ms	76 ms	0.62 ms	9208 ms	2557 ms

Table 4: Benchmarks of homomorphic matrix operations

Ciphertext sizes. As mentioned above, a ciphertext could hold 2^{12} different plaintext slots, and thus we can encrypt one 64×64 matrix into a fully-packed ciphertext. We assumed that all the inputs had $\log p = 24$ bits of precision, which means that an input matrix size is bounded by $64 \times 64 \times 24$ bits or 12 KB. Since a single ciphertext is at most 172 KB for an evaluation of matrix multiplication, the encrypted version is $172/12 \approx 14$ times larger than the unencrypted version. In Table 4, the third column gives the size of input matrices and the fourth column gives an expansion ratio of the generated ciphertext to the input matrices.

Timing results. We conducted experiments over ten times and measured the average running times for all the operations. For the parameter setting in Section 5.1, the key generation takes about 1.277 seconds. In Table 4, the fifth column gives timing for encoding input matrices and encrypting the resulting plaintext slots. Since matrix multiplication requires the largest fresh ciphertext modulus and takes more time than others, we just report the encryption timing results for the case. In the sixth column, we give timing for decrypting the output ciphertext and decoding to its matrix form. Note that encryption and decryption timings are similar each other; but encoding and decoding timings depend on the throughput. The last three columns give amortized time per matrix for homomorphic matrix computation. The entire execution time, called *latency*, is similar between the three variant implementations so the parallel matrix computation provides roughly a speedup as a factor of the throughput.

Performance of rectangular matrix multiplication. We present the performance of Algorithm 3 described in Section 4.2. As a concrete example, we consider the rectangular matrix multiplication $\mathbb{R}^{16\times 64} \times \mathbb{R}^{64\times 64} \to \mathbb{R}^{16\times 64}$. As we described above, our optimized method has a better performance than a simple method exploiting Algorithm 2 for the multiplication between 64×64 matrices.

To be precise, the first step of Algorithms 2 or 3 generates two ciphertexts $\mathsf{ct}.A^{(0)}$ and $\mathsf{ct}.B^{(0)}$ by applying the linear transformations of U^{σ} and U^{τ}, thus both approaches have almost the same computational complexity. Next, in the second and third steps, two algorithms apply the same operations to the resulting ciphertexts but with different numbers: Algorithm 2 requires approximately four times more operations compared to Algorithm 3. As a result, Step 2 turns out to be the most time consuming part in Algorithm 2, whereas it is not the dominant procedure in Algorithm 3. Finally, Algorithm 3 requires some additional operations for Step 4, but we need only $\log(64/16) = 2$ automorphisms.

Table 5 summarizes an experimental result based on the same parameter as in the above section. The total running times of two algorithms are 9.21 seconds and 4.29 seconds, respectively; therefore, Algorithm 3 achieves a speedup of 2X compared to Algorithm 2.

Algorithm	Step 1	Step 2	Step 3	Step 4	Total
`HE-MatMult`	2.355s	5.694s	1.161s	-	9.208s
`HE-RMatMult`		1.529s	0.354s	0.047s	4.288s
Speedup	-	3.714	3.269	-	2.147

Table 5: Performance comparison of homomorphic square and rectangular matrix multiplications

6 E2DM: Making Prediction based on Encrypted Data and Model

In this section, we propose a novel framework E2DM to test encrypted convolutional neural networks model on encrypted data. We consider a new service paradigm where model providers offer encrypted trained classifier models to a public cloud and the cloud server provides on-line prediction service to data owners who uploaded their encrypted data. In this inference, the cloud should learn nothing about private data of the data owners, nor about the trained models of the model providers.

6.1 Neural Networks Architecture

The first viable example of CNN on image classification was AlexNet by Krizhevsky et al. [33] and it was dramatically improved by Simonyan et al. [45]. It consists of a stack of *linear* and *non-linear* layers. The linear layers can be convolution layers or FC layers. The non-linear layers can be max pooling (i.e., compute the maximal value of some components of the feeding layer), mean pooling (i.e., compute the average value of some components of the feeding layer), ReLu functions, or sigmoid functions.

Specifically, the convolution operator forms the fundamental basis of the convolutional layer. The convolution has kernels, or windows, of size $k \times k$, a stride of (s, s), and a mapcount of h. Given an image $I \in \mathbb{R}^{w \times w}$ and a kernel $K \in \mathbb{R}^{k \times k}$, we compute the convolved image $\mathtt{Conv}(I, K) \in \mathbb{R}^{d_K \times d_K}$ by

$$\mathtt{Conv}(I, K)_{i',j'} = \sum_{0 \le i,j < k} K_{i,j} \cdot I_{s \cdot i' + i, s \cdot j' + j}$$

for $0 \le i', j' < d_K = \lceil (w-k)/s \rceil + 1$. Here $\lceil \cdot \rceil$ returns the least integer greater than or equal to the input. It can be extended to multiple kernels $\mathcal{K} = \{K^{(k)}\}_{0 \le k < h}$ as

$$\mathtt{Conv}(I, \mathcal{K}) = (\mathtt{Conv}(I, K^{(0)}), \cdots, \mathtt{Conv}(I, K^{(h-1)})) \in \mathbb{R}^{d_K \times d_K \times h}.$$

On the other hand, FC layer connects n_I nodes to n_O nodes, or equivalently, it can be specified by the matrix-vector multiplication of an $n_O \times n_I$ matrix. Note that the output of convolution layer has a form of tensor so it should be flatten before FC layer. Throughout this paper, we concatenate the rows of the tensor one by one and output a column vector, denoted by $\mathtt{FL}(\cdot)$.

6.2 Homomorphic Evaluation of CNN

We present an efficient strategy to evaluate CNN prediction model on the MNIST dataset. Each image is a 28×28 pixel array, where the value of each pixel represents a level of gray. After an arbitrary number of hidden layers, each image is labeled with 10 possible digits. The training set has 60,000 images and the test set has 10,000 images. We assume that a neural network is trained with the plaintext dataset in the clear. We adapted a similar network topology to CryptoNets: one convolution layer and two FC layers with square activation function. Table 6 describes our neural networks to the MNIST dataset and summarizes the hyperparameters.

The final step of neural networks is usually to apply the softmax activation function for a purpose of probabilistic classification. We note that it is enough to obtain an index of maximum values of outputs in a prediction phase.

In the following, we explain how to securely test encrypted model on encrypted multiple data. In our implementation, we take $N = 2^{13}$ as a cyclotomic ring dimension so each plaintext vector is allowed to have dimension less than 2^{12} and one can predict 64 images simultaneously in a SIMD manner. We describe the parameter selection in more detail below.

Layer	Description
Convolution	Input image 28×28, kernel size 7×7, stride size of 3, number of output channels 4
1^{st} square	Squaring 256 input values
FC-1	Fully connecting with 256 inputs and 64 outputs: $\mathbb{R}^{64\times256} \times \mathbb{R}^{256\times1} \to \mathbb{R}^{64\times1}$
2^{nd} square	Squaring 64 input values
FC-2	Fully connecting with 64 inputs and 10 outputs: $\mathbb{R}^{10\times64} \times \mathbb{R}^{64\times1} \to \mathbb{R}^{10\times1}$

Table 6: Description of our CNN to the MNIST dataset

6.2.1 Encryption of Images

At the encryption phase, the data owner encrypts the data using the public key of an HE scheme. Suppose that the data owner has a two-dimensional image $I \in \mathbb{R}^{28\times28}$. For $0 \le i', j' < d_K = 8$, let us define an extracted image feature $I[i', j']$ formed by taking the elements $I_{3\cdot i'+i, 3\cdot j'+j}$ for $0 \le i, j < 7$. That is, a single image can be represented as the 64 image features of size 7×7. It can be extended to multiple images $\mathcal{I} = \{I^{(k)}\}_{0\le k<64}$. For each $0 \le i, j < 7$, the dataset is encoded into a matrix consisting of the (i, j)-th components of 64 features over 64 images and it is encrypted as follows:

$$\mathsf{ct}.I_{i,j} = \mathtt{Enc}\begin{bmatrix} I^{(0)}[0,0]_{i,j} & I^{(1)}[0,0]_{i,j} & \cdots & I^{(63)}[0,0]_{i,j} \\ I^{(0)}[0,1]_{i,j} & I^{(1)}[0,1]_{i,j} & \cdots & I^{(63)}[0,1]_{i,j} \\ \vdots & \vdots & \ddots & \vdots \\ I^{(0)}[7,7]_{i,j} & I^{(1)}[7,7]_{i,j} & \cdots & I^{(63)}[7,7]_{i,j} \end{bmatrix}.$$

The resulting ciphertexts $\{\mathsf{ct}.I_{i,j}\}_{0\le i,j<7}$ are sent to the public cloud and stored in their encrypted form.

6.2.2 Encryption of Trained Model

The model provider encrypts the trained prediction model values such as multiple convolution kernels' values $\mathcal{K} = \{K^{(k)}\}_{0\le k<4}$ and weights (matrices) of FC layers. The provider begins with a procedure for encrypting each of the convolution kernels separately. For $0 \le i, j < 7$ and $0 \le k < 4$, the (i, j)-th component of the kernel matrix $K^{(k)}$ is copied into plaintext slots and the model provider encrypts the plaintext vector into a ciphertext, denoted by $\mathsf{ct}.K^{(k)}_{i,j}$.

Next, the first FC layer is specified by a 64×256 matrix and it can be divided into four square sub-matrices of size 64×64. For $0 \le k < 4$, we write W_k to denote the k-th sub-matrix. Each matrix is encrypted into a single ciphertext using the matrix encoding method in Section 3.2, say the output ciphertext $\mathsf{ct}.W_k$.

For the second FC layer, it can be expressed by a 10×64 matrix. The model provider pads zeros in the bottom to obtain a matrix V of size 16×64 and then generates a 64×64 matrix $\bar{V}$ containing four copies of V vertically, say the output ciphertext $\mathsf{ct}.V$. Finally, the model provider transmits three distinct types of ciphertexts to the cloud: $\mathsf{ct}.K^{(k)}_{i,j}$, $\mathsf{ct}.W_k$, and $\mathsf{ct}.V$.

6.2.3 Homomorphic Evaluation of Neural Networks

At the prediction phase, the public cloud takes ciphertexts of the images from the data owner and the neural network prediction model from the model provider. Since the data owner uses a SIMD technique to batch 64 different images, the first FC layer is specified as a matrix multiplication: $\mathbb{R}^{64\times 256} \times \mathbb{R}^{256\times 64} \to \mathbb{R}^{64\times 64}$. Similarly, the second FC layer is represented as a matrix multiplication: $\mathbb{R}^{10\times 64} \times \mathbb{R}^{64\times 64} \to \mathbb{R}^{10\times 64}$.

Homomorphic convolution layer. The public cloud takes the ciphertexts $\mathsf{ct}.I_{i,j}$ and $\mathsf{ct}.K_{i,j}^{(k)}$ for $0 \le i, j < 7$ and $0 \le k < 4$. We apply pure SIMD operations to efficiently compute dot-products between the kernel matrices and the extracted image features. For each $0 \le k < 4$, the cloud performs the following computation on ciphertexts:

$$\mathsf{ct}.C_k \leftarrow \sum_{0\le i,j<7} \mathtt{Mult}(\mathsf{ct}.I_{i,j}, \mathsf{ct}.K_{i,j}^{(k)}).$$

By the definition of the convolution, the resulting ciphertext ct_k represents a square matrix C_k of order 64 such that

$$C_k = \begin{bmatrix} \vdots & & \vdots \\ \mathtt{FL}(\mathtt{Conv}(I^{(0)}, K^{(k)})) & \cdots & \mathtt{FL}(\mathtt{Conv}(I^{(63)}, K^{(k)})) \\ \vdots & & \vdots \end{bmatrix}.$$

That is, it is an encryption of the matrix C_k having the i-th column as the flatten convolved result between the the i-th image $I^{(i)}$ and the k-th kernel $K^{(k)}$.

The first square layer. This step applies the square activation function to all the encrypted output images of the convolution in a SIMD manner. That is, for each $0 \le k < 4$, the cloud computes as follows:

$$\mathsf{ct}.S_k^{(1)} \leftarrow \mathtt{SQR}(\mathsf{ct}.C_k)$$

where $\mathtt{SQR}(\cdot)$ denotes the squaring operation of an HE scheme. Note that $\mathsf{ct}.S_k^{(1)}$ is an encryption of the matrix $C_k \odot C_k$.

The FC-1 layer. This procedure requires a matrix multiplication between a 64×256 weight matrix $W = (W_0|W_1|W_2|W_3)$ and a 256×64 input matrix $C = (C_0 \odot C_0;\ C_1 \odot C_1;\ C_2 \odot C_2;\ C_3 \odot C_3)$. The matrix product $W \cdot C$ is formed by combining the blocks in the same way, that is,

$$W \cdot C = \sum_{0\le k<4} (W_k \cdot (C_k \odot C_k)).$$

Thus the cloud performs the following computation:

$$\mathsf{ct}.F \leftarrow \sum_{0\le k<4} \mathtt{HE\text{-}MatMult}(\mathsf{ct}.W_k, \mathsf{ct}.S_k^{(1)}).$$

The second square layer. This step applies the square activation function to all the output nodes of the first FC layer:

$$\mathsf{ct}.S^{(2)} \leftarrow \mathtt{SQR}(\mathsf{ct}.F).$$

The FC-2 layer. This step performs the rectangular multiplication algorithm between the weight ciphertext $\mathsf{ct}.V$ and the output ciphertext $\mathsf{ct}.S^{(2)}$ of the second square layer:

$$\mathsf{ct.out} \leftarrow \texttt{HE-RMatMult}(\mathsf{ct}.V, \mathsf{ct}.S^{(2)}).$$

6.2.4 The Threat Model

Suppose that one can ensure the IND-CPA security of an underlying HE scheme, which means that ciphertexts of any two messages are computationally indistinguishable. Since all the computations on the public cloud are performed over encryption, the cloud learns nothing from the encrypted data so we can ensure the confidentiality of the data against such a semi-honest server.

6.3 Performance Evaluation of E2DM

We evaluated our E2DM framework to classify encrypted handwritten images of the MNIST dataset. We used the library keras [16] with Tensorflow [1] to train the CNN model from 60,000 images of the dataset by applying the ADADELTA optimization algorithm [51].

6.3.1 Optimization Techniques

Suppose that we are given an encryption $\mathsf{ct}.A$ of a $d \times d$ matrix A. Recall from Section 3 that we apply homomorphic liner transformations to generate the encryption $\mathsf{ct}.A^{(\ell)}$ of a matrix $\phi^\ell \circ \sigma(A)$ for $0 \leq \ell < d$. Sometimes one can pre-compute $\phi^\ell \circ \sigma(A)$ in the clear and generate the corresponding ciphertexts for free. Thus this approach gives us a space/time trade-off: although it requires more space for d ciphertexts rather than a single ciphertext, it reduces the overhead of rotation operations from $(3d + 5\sqrt{d})$ to $(d + 2\sqrt{d})$, achieving a better performance. This method has another advantage, in that an input ciphertext modulus is reduced by $(\log p + \log p_{\mathsf{c}})$ bits after matrix multiplication while $(\log p + 2\log p_{\mathsf{c}})$ in the original method. This is because the encryptions of $\phi^k \circ \sigma(A)$ are given as fresh ciphertexts and it only requires additional depths to generate the encryptions of $\psi^k \circ \tau(B)$.

We can apply this idea to the FC layers. For each $0 \leq k < 4$ and $0 \leq \ell < 64$, the model provider generates a ciphertext $\mathsf{ct}.W_k^{(\ell)}$ representing the matrix $\phi^\ell \circ \sigma(W_k)$ of the first FC layer. For the second FC layer, the provider generates an encryption $\mathsf{ct}.V^{(\ell)}$ of the matrix $\phi^\ell \circ \sigma(\bar{V})$ for $0 \leq \ell < 16$.

6.3.2 Parameters

The convolution layer and the square activation layers have a depth of one homomorphic multiplication. As discussed before, the FC layers have depth of one homomorphic multiplication and one constant multiplication by applying the pre-computation optimization technique. Therefore, the lower bound on the bit length of a fresh ciphertext modulus is $5\log p + 2\log p_{\mathsf{c}} + \log q_0$. In our implementation, we assume that all the inputs had $\log p = 24$ bits of precision and set the bit length of the output ciphertext modulus as $\log q_0 = \log p + 8$. In addition, we set $\log p_{\mathsf{c}} = 15$ for the bit precision of constant values. We could actually obtain the bit length of the largest ciphertext modulus around 182 and took the ring dimension $N = 2^{13}$ to ensure 80 bits of security. This security was chosen to be consistent with other performance number reported from CryptoNets. Note that a fresh ciphertext has 0.355 MB under this parameter setting.

6.3.3 Ciphertext Sizes

Each image is a 28×28 pixel array, where each pixel is in the range from 0 to 255. The data owner first chooses 64 images in the MNIST dataset, normalizes the data by dividing by the maximum value 255, and generates the ciphertexts $\{\mathsf{ct}.I_{i,j}\}_{0 \le i,j < 7}$. The total size of ciphertexts is $0.355 \times 49 \approx 17.417$ MB and a single ciphertext contains informations of 64 images, and therefore the total ciphertext size per image is $17.417/64 \approx 0.272$ MB or 278 KB. Since each image has approximately $28 \times 28 \times 24$ bits, it is 121 times smaller than the encrypted one. Meanwhile, the model provider generates three distinct types of ciphertexts:

- $\mathsf{ct}.K_{i,j}^{(k)}$ for $0 \le i, j < 7$ and $0 \le k < 4$;
- $\mathsf{ct}.W_{k}^{(\ell)}$ for $0 \le k < 4$ and $0 \le \ell < 64$;
- $\mathsf{ct}.V^{(\ell)}$ for $0 \le \ell < 16$.

The total size of ciphertexts is $0.355 \times 468 \approx 166.359$ MB. After the homomorphic evaluation of E2DM, the cloud sends only a single ciphertext to an authority who is the legitimate owner of the secret key of the HE scheme. The ciphertext size is around 0.063 MB and the size per image is 0.063/64 MB $\approx$ 1 KB. Table 7 summarizes the numbers in the second and third columns.

	Ciphertext size	Size per instance
Data owner → Cloud	17.417 MB	278 KB
Model provider → Cloud	166.359 MB	-
Cloud → Authority	0.063 MB	1 KB

Table 7: Ciphertext sizes of E2DM

6.3.4 Implementation Details

The key generation takes about 1.38 seconds for the parameters setting in Section 6.3.2. The data owner takes about 1.56 seconds to encrypt 64 different number of images. Meanwhile, the model provider takes about 12.33 seconds to generate the encrypted prediction models. This procedure takes more time than the naive method but it is an one-time process before data outsourcing and so it is a negligible overhead.

In Table 8, we report timing results for the evaluation of E2DM. The third column gives timings for each step and the fourth column gives the relative time per image (if applicable). The dominant cost of evaluating the framework is that of performing the first FC layer. This step requires four matrix multiplication operations over encrypted 64×64 matrices so it expects to take about $9.21 \times 4 \approx 36.84$ seconds from the result of Table 4. We further take advantage of the pre-computation method described in Section 6.3.1, and thereby it only took about 20.79 seconds to evaluate the layer (1.8 times faster). Similarly, we could apply this approach to the second FC layer, which leads to 1.97 seconds for the evaluation. In total, it took about 28.59 seconds to classify encrypted images from the encrypted training model, yielding an amortized rate of 0.45 seconds per image.

After the evaluation, the cloud returns only a single packed ciphertext that is transmitted the authority. Then the output can be decrypted with the secret key and the authority computes

	Stage	Latency (s)	Amortized time per image (ms)
Data owner	Encoding + Encryption	1.56	24.42
Model provider	Encoding + Encryption	12.33	-
Cloud	Convolution	5.68	88.75
	1^{st} square	0.10	1.51
	FC-1	20.79	324.85
	2^{nd} square	0.06	0.96
	FC-2	1.97	30.70
	Total	**28.59**	**446.77**
Authority	Decoding +Decryption	0.07	1.14

Table 8: Performance results of E2DM for MNIST

the argmax of 10 scores for each image to obtain the prediction. These procedures take around 0.07 seconds, yielding an amortized time of 1.14 milliseconds per image. In the end, the data owner gets the results from the authority.

This model achieves an accuracy of 98.1% on the test set. The accuracy is the same as the one obtained by the evaluation of the model in the clear, which implies that there is no precision loss from the approximate homomorphic encryption.

6.4 Comparison with Previous Work

Table 9 compares our benchmark result for the MNIST dataset with the state-of-the-art frameworks: CryptoNets [25], MiniONN [35], and GAZELLE [30]. The first column indicates the framework and the second column denotes the method used for preserving privacy. The last columns give running time and communication costs required for image classification.

HE-based frameworks. We used a similar network topology to CryptoNets (only different numbers of nodes in the hidden layers) but considered different scenario and underlying cryptographic primitive. CryptoNets took 570 seconds to perform a single prediction, yielding in an amortized rate of 0.07 seconds. In our case, data is represented in a matrix form and applied to the evaluation of neural networks using homomorphic matrix operations. As a result, E2DM achieves 20-fold reduction in latency and 34-fold reduction in message sizes. CryptoNets allows more SIMD parallelism, so it could give better amortized running time. However, this implies that CryptoNets requires a very large number of prediction to yield better amortized complexity, so its framework turns out to be less competitive in practice.

Mixed protocol frameworks. Liu et al. [35] presented MiniONN framework of privacy-preserving neural networks by employing a ciphertext packing technique as a pre-processing tool. Recently, Juvekar et al. [30] presented GAZELLE that deploys automorphism structure of an underlying HE scheme to perform matrix-vector multiplication, thereby improving the performance significantly. It took 30 milliseconds to classify one image from the MNIST dataset and has an online bandwidth cost of 0.5 MB. Even though these mixed protocols achieve relatively fast run-time, they require interaction between protocol participants, resulting in high bandwidth usage.

Framework	Method	Runtime (s)				Communication (MB)			
		Offline	Online	Total	Amortized	Offline	Online	Total	Per instance
CryptoNets	HE	-	-	570	0.07	-	-	595.5	0.07
MiniONN	HE, MPC	0.88	0.40	1.28	1.28	3.6	44	47.6	47.6
GAZELLE	HE, MPC	0	0.03	0.03	0.03	0	0.5	0.5	0.5
E2DM	HE	-		28.59	0.45	-	-	17.48	0.27

Table 9: MNIST Benchmark of privacy-preserving neural network frameworks

7 Related Works

7.1 Secure Outsourced Matrix Computation

Matrix multiplication can be performed using a series of inner products. Wu and Haven [49] suggested the first secure inner product method in a SIMD environment. Their approach is to encrypt each row or column of a matrix into an encrypted vector and obtain component-wise product of two input vectors by performing a single homomorphic multiplication. However, it should aggregate all the elements over the plaintext slots in order to get the desired result and this procedure requires at least $\log d$ automorphisms. Since one can apply this solution to each row of A and each column of B, the total complexity of secure matrix multiplication is about d^2 multiplications and $d^2 \log d$ automorphisms.

Recently, several other approaches have been considered by applying the encoding methods of Lauter et al. [41] and Yasuda et al. [50] on an RLWE-based HE scheme. Duong et al. [19] proposed a method to encode a matrix as a constant polynomial in the native plaintext space. Then secure matrix multiplication requires only one homomorphic multiplication over packed ciphertexts. This method was later improved in [38]. However, this solution has a serious drawback for practical use: the resulting ciphertext has non-meaningful terms in its coefficients, so for more computation, it should be decrypted and re-encoded by removing the terms in the plaintext polynomial.

Most of related works focus on verifiable secure outsourcing of matrix computation [11, 4, 39, 21]. In their protocols, a client delegates a task to an untrusted server and the server returns the computation result with a proof of the correctness of the computation. There are general results [22, 17, 21] of verifiable secure computation outsourcing by applying a fully HE scheme with Yao's Garbled circuit or pseudo-random functions. However, it is still far from practical to apply these theoretical approaches to real-world applications.

7.2 Privacy-preserving Neural Networks Predictions

Privacy-preserving deep learning prediction models were firstly considered by Gilad-Bachrach et al. [25]. The authors presented the private evaluation protocol CryptoNets for CNN. A number of subsequent works have improved it by normalizing weighted sums prior to applying the approximate activation function [10], or by employing a fully HE to apply an evaluation of an arbitrary deep neural networks [7].

There are other approaches for privacy-preserving deep learning prediction based on MPC [5, 42] or its combination with (additively) HE. The idea behind such hybrid protocols is to evaluate

scalar products using HE and compute activation functions (e.g. threshold or sigmoid) using MPC technique. Mohassel and Zhang [40] applied the mixed-protocol framework of [18] to implement neural networks training and evaluation in a two-party computation setting. Liu et al. [35] presented MiniONN to transform an existing neural network to an oblivious neural network by applying a SIMD batching technique. Riazi et al. [43] designed Chameleon, which relies on a trusted third-party. Their frameworks were later improved in [30] by leveraging effective use of packed ciphertexts. Even though these hybrid protocols could improve efficiency, they result in high bandwidth and long network latency.

8 Conclusion and Future Work

In this paper, we presented a practical solution for secure outsourced matrix computation. We did demonstrate its applicability by presenting a novel framework E2DM for secure evaluation of encrypted neural networks on encrypted data. Our benchmark shows that E2DM achieves lower messages sizes and latency than the solution of CryptoNets.

Our secure matrix computation primitive can be applied to various computing applications such as genetic testing and machine learning. In particular, we can investigate financial model evaluation based on our E2DM framework. Our another future work is to extend the matrix computation mechanism for more advanced operations.

Another direction to explore is to employ a variant of HEAAN based on a Residue Number System (RNS) [13]. This scheme achieves a significant performance gain from its full RNS implementation compared to the original HEAAN scheme requiring high-precision arithmetic. We can switch the underlying HE scheme, thereby improving the performance of matrix operations and evaluation of neural networks.

References

1. M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, et al. Tensorflow: Large-scale machine learning on heterogeneous distributed systems, 2015. `https://www.tensorflow.org`.
2. M. R. Albrecht, R. Player, and S. Scott. On the concrete hardness of learning with errors. *Journal of Mathematical Cryptology*, 9(3):169–203, 2015.
3. C. S. Alliance. Security guidance for critical areas of focus in cloud computing, 2009. `http://www.cloudsecurityalliance.org.`
4. M. J. Atallah and K. B. Frikken. Securely outsourcing linear algebra computations. In *Proceedings of the 5th ACM Symposium on Information, Computer and Communications Security*, pages 48–59. ACM, 2010.
5. M. Barni, C. Orlandi, and A. Piva. A privacy-preserving protocol for neural-network-based computation. In *Proceedings of the 8th workshop on Multimedia and security*, pages 146–151. ACM, 2006.
6. J. W. Bos, K. Lauter, J. Loftus, and M. Naehrig. Improved security for a ring-based fully homomorphic encryption scheme. In *Cryptography and Coding*, pages 45–64. Springer, 2013.
7. F. Bourse, M. Minelli, M. Minihold, and P. Paillier. Fast homomorphic evaluation of deep discretized neural networks. Cryptology ePrint Archive, Report 2017/1114, 2017. `https://eprint.iacr.org/2017/1114`.
8. Z. Brakerski. Fully homomorphic encryption without modulus switching from classical GapSVP. In *Advances in Cryptology–CRYPTO 2012*, pages 868–886. Springer, 2012.
9. Z. Brakerski, C. Gentry, and V. Vaikuntanathan. (Leveled) fully homomorphic encryption without bootstrapping. In *Proc. of ITCS*, pages 309–325. ACM, 2012.
10. H. Chabanne, A. de Wargny, J. Milgram, C. Morel, and E. Prouff. Privacy-preserving classification on deep neural network. Cryptology ePrint Archive, Report 2017/035, 2017. `https://eprint.iacr.org/2017/035`.
11. D. Chaum and T. P. Pedersen. Wallet databases with observers. In *Annual International Cryptology Conference*, pages 89–105. Springer, 1992.
12. J. H. Cheon, K. Han, A. Kim, M. Kim, and Y. Song. Bootstrapping for approximate homomorphic encryption. In *Advances in Cryptology–EUROCRYPT 2018*, pages 360–384. Springer, 2018.

13. J. H. Cheon, K. Han, A. Kim, M. Kim, and Y. Song. A full rns variant of approximate homomorphic encryption. In *To appear in International Conference on Selected Areas in Cryptography*. Springer, 2018.
14. J. H. Cheon, A. Kim, M. Kim, and Y. Song. Implementation of HEAAN, 2016. `https://github.com/kimandrik/HEAAN`.
15. J. H. Cheon, A. Kim, M. Kim, and Y. Song. Homomorphic encryption for arithmetic of approximate numbers. In *Advances in Cryptology–ASIACRYPT 2017: 23rd International Conference on the Theory and Application of Cryptology and Information Security*, pages 409–437. Springer, 2017.
16. F. Chollet et al. Keras, 2015. `https://github.com/keras-team/keras`.
17. K.-M. Chung, Y. T. Kalai, F.-H. Liu, and R. Raz. Memory delegation. In *Annual Cryptology Conference*, pages 151–168. Springer, 2011.
18. D. Demmler, T. Schneider, and M. Zohner. ABY-a framework for efficient mixed-protocol secure two-party computation. In *NDSS*, 2015.
19. D. H. Duong, P. K. Mishra, and M. Yasuda. Efficient secure matrix multiplication over lwe-based homomorphic encryption. *Tatra Mountains Mathematical Publications*, 67(1):69–83, 2016.
20. J. Fan and F. Vercauteren. Somewhat practical fully homomorphic encryption. Cryptology ePrint Archive, Report 2012/144, 2012. `https://eprint.iacr.org/2012/144`.
21. D. Fiore and R. Gennaro. Publicly verifiable delegation of large polynomials and matrix computations, with applications. In *Proceedings of the 2012 ACM conference on Computer and communications security*, pages 501–512. ACM, 2012.
22. R. Gennaro, C. Gentry, and B. Parno. Non-interactive verifiable computing: Outsourcing computation to untrusted workers. In *Annual Cryptology Conference*, pages 465–482. Springer, 2010.
23. C. Gentry et al. Fully homomorphic encryption using ideal lattices. In *STOC*, volume 9, pages 169–178, 2009.
24. C. Gentry, S. Halevi, and N. P. Smart. Homomorphic evaluation of the AES circuit. In *Advances in Cryptology–CRYPTO 2012*, pages 850–867. Springer, 2012.
25. 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 *International Conference on Machine Learning*, pages 201–210, 2016.
26. S. Halevi and V. Shoup. Algorithms in HElib. In *Advances in Cryptology–CRYPTO 2014*, pages 554–571. Springer, 2014.
27. S. Halevi and V. Shoup. Bootstrapping for HElib. In *Advances in Cryptology–EUROCRYPT 2015*, pages 641–670. Springer, 2015.
28. S. Halevi and V. Shoup. Faster homomorphic linear transformations in HElib. Cryptology ePrint Archive, Report 2018/244, 2018. `https://eprint.iacr.org/2018/244`.
29. X. Jiang, Y. Zhao, X. Wang, B. Malin, S. Wang, L. Ohno-Machado, and H. Tang. A community assessment of privacy preserving techniques for human genomes. *BMC Med. Inform. Decis. Mak.*, 14 Suppl 1(Suppl 1):S1, Dec. 2014.
30. C. Juvekar, V. Vaikuntanathan, and A. Chandrakasan. GAZELLE: A low latency framework for secure neural network inference. In *27th USENIX Security Symposium (USENIX Security 18)*, Baltimore, MD, 2018. USENIX Association.
31. M. Kim and K. Lauter. Private genome analysis through homomorphic encryption. *BMC medical informatics and decision making*, 15(Suppl 5):S3, 2015.
32. M. Kim, Y. Song, S. Wang, Y. Xia, and X. Jiang. Secure logistic regression based on homomorphic encryption: Design and evaluation. *JMIR medical informatics*, 6(2), 2018.
33. A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.
34. Y. LeCun. The mnist database of handwritten digits. *http://yann. lecun. com/exdb/mnist/*, 1998.
35. J. Liu, M. Juuti, Y. Lu, and N. Asokan. Oblivious neural network predictions via minionn transformations. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 619–631. ACM, 2017.
36. E. Makri, D. Rotaru, N. P. Smart, and F. Vercauteren. Pics: Private image classification with SVM. Cryptology ePrint Archive, Report 2017/1190, 2017. `https://eprint.iacr.org/2017/1190`.
37. R. Miotto, F. Wang, S. Wang, X. Jiang, and J. T. Dudley. Deep learning for healthcare: review, opportunities and challenges. *Brief. Bioinform.*, May 2017.
38. P. K. Mishra, D. H. Duong, and M. Yasuda. Enhancement for secure multiple matrix multiplications over ring-lwe homomorphic encryption. In *International Conference on Information Security Practice and Experience*, pages 320–330. Springer, 2017.
39. P. Mohassel. Efficient and secure delegation of linear algebra. Cryptology ePrint Archive, Report 2011/605, 2011. `https://eprint.iacr.org/2011/605`.

40. P. Mohassel and Y. Zhang. SecureML: A system for scalable privacy-preserving machine learning. In *Security and Privacy (SP), 2017 IEEE Symposium on*, pages 19–38. IEEE, 2017.
41. M. Naehrig, K. Lauter, and V. Vaikuntanathan. Can homomorphic encryption be practical? In *Proceedings of the 3rd ACM workshop on Cloud computing security workshop*, pages 113–124. ACM, 2011.
42. C. Orlandi, A. Piva, and M. Barni. Oblivious neural network computing via homomorphic encryption. *EURASIP Journal on Information Security*, 2007(1):037343, 2007.
43. 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. *arXiv preprint arXiv:1801.03239*, 2018.
44. V. Shoup et al. NTL: A library for doing number theory, 2001.
45. K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.
46. N. P. Smart and F. Vercauteren. Fully homomorphic SIMD operations. Cryptology ePrint Archive, Report 2011/133, 2011. `https://eprint.iacr.org/2011/133`.
47. H. Takabi, J. B. Joshi, and G.-J. Ahn. Security and privacy challenges in cloud computing environments. *IEEE Security & Privacy*, 8(6):24–31, 2010.
48. S. Wang, X. Jiang, H. Tang, X. Wang, D. Bu, K. Carey, S. O. M. Dyke, D. Fox, C. Jiang, K. Lauter, and Others. A community effort to protect genomic data sharing, collaboration and outsourcing. *npj Genomic Medicine*, 2(1):33, 2017.
49. D. Wu and J. Haven. Using homomorphic encryption for large scale statistical analysis. Technical report, Technical Report: cs. stanford. edu/people/dwu4/papers/FHESI Report. pdf, 2012.
50. M. Yasuda, T. Shimoyama, J. Kogure, K. Yokoyama, and T. Koshiba. New packing method in somewhat homomorphic encryption and its applications. *Security and Communication Networks*, 8(13):2194–2213, 2015.
51. M. D. Zeiler. Adadelta: an adaptive learning rate method. *arXiv preprint arXiv:1212.5701*, 2012.