Secure and Efficient Outsourced Matrix Multiplication with Homomorphic Encryption

Aikata Aikata[0000−0003−0934−2982] and Sujoy Sinha Roy[0000−0002−9805−5389]

Graz University of Technology, Austria
{aikata,sujoy.sinharoy}@iaik.tugraz.at

Abstract. Fully Homomorphic Encryption (FHE) is a promising privacy-enhancing technique that enables secure and private data processing on untrusted servers, such as privacy-preserving neural network (NN) evaluations. However, its practical application presents significant challenges. Limitations in how data is stored within homomorphic ciphertexts and restrictions on the types of operations that can be performed create computational bottlenecks. As a result, a growing body of research focuses on optimizing existing evaluation techniques for efficient execution in the homomorphic domain.

One key operation in this space is matrix multiplication, which forms the foundation of most neural networks. Several studies have even proposed new FHE schemes specifically to accelerate this operation. The optimization of matrix multiplication is also the primary goal of our work. We leverage the Single Instruction Multiple Data (SIMD) capabilities of FHE to increase data packing and significantly reduce the KeySwitch operation count— an expensive low-level routine in homomorphic encryption. By minimizing KeySwitching, we surpass current state-of-the-art solutions, requiring only a minimal multiplicative depth of two.

The best-known complexity for matrix multiplication at this depth is $\mathcal{O}(d)$ for matrices of size $d \times d$. Remarkably, even the leading techniques that require a multiplicative depth of three still incur a KeySwitch complexity of $\mathcal{O}(d)$. In contrast, our method reduces this complexity to $\mathcal{O}(\log d)$ while maintaining the same level of data packing. Our solution broadly applies to all FHE schemes supporting Single Instruction Multiple Data (SIMD) operations. We further generalize the technique in two directions: allowing arbitrary packing availability and extending it to rectangular matrices. This versatile approach offers significant improvements in matrix multiplication performance and enables faster evaluation of privacy-preserving neural network applications.

Keywords: Fully Homomorphic Encryption · Secure Outsourced Matrix Multiplication · Arbitrary Packing · Privacy-enhancing Techniques

1 Introduction

The ability to process encrypted data without decryption has positioned Fully Homomorphic Encryption (FHE) as the "Holy Grail" of privacy-preserving data

storage and computation [28]. However, this promising technique faces significant challenges that hinder its widespread adoption, including substantial data expansion and high computational requirements. These issues have sparked numerous research directions aimed at addressing the computing limitations associated with FHE, such as hardware acceleration approaches [17,1,27,4] that seek to enhance server performance.

Most of the efficient and high-performing FHE schemes [12,7,15] are lattice-based. A key limitation of these schemes is their linear slot-wise ciphertext encoding, which can be conceptualized as a one-dimensional array where each plaintext element occupies a single index. This encoding restricts operations requiring permutation, as elements cannot be easily extracted from the array, unlike in plaintext operations. Consequently, performing permutations on homomorphic ciphertext necessitates costly multiplications with masks and rotations. While this is manageable for operations like approximate function evaluation that operate slot-wise, it poses a significant challenge for matrix multiplication.

Matrix multiplication is fundamental in advanced mathematics and is especially critical for secure data analysis and machine learning (ML), particularly within neural networks (NN) [21,22,30,20,33]. Network components, such as fully connected layers and filters/kernels, depend heavily on efficient matrix multiplication. Although efficient algorithms like Strassen's algorithm exist for plaintext operations, conducting matrix multiplication in the encrypted domain remains an emerging area of research. This field has garnered attention for its potential to facilitate encrypted ML training and inference using FHE schemes like CKKS [10], which support approximate arithmetic.

1.1 Prior Works.

The authors in [18] introduced the first technique for multiplying encrypted matrices and vectors, which was later extended to matrix-matrix multiplication by [25] and [31]. However, these methods require a substantial number of homomorphic multiplications and rotation operations. The literature [25,31,30,21,22,20,33] on encrypted matrix multiplication (referred to as matrix-matrix multiplication from here on) can be broadly categorized into three types, as summarized in Table 1.

The first type necessitates a multiplicative depth of two and employs a simple row-wise encoding for the initial input data. The work in [30] exemplifies this approach, presenting a technique for d^3 packing availability in the ciphertext. For a square matrix of dimensions $d \times d$, this method requires $2 \cdot d + 3 \cdot \log_2(d) - 2$ rotations and one ciphertext-ciphertext (ct-ct) multiplication. Importantly, these operations are costly, as a KeySwitch operation is required after each to maintain that the ciphertext is decryptable with the same secret key. Thus, the total KeySwitch complexity amounts to $2 \cdot d + 3 \cdot \log_2(d) - 1$. A significant limitation of this approach is its requirement for d^3 slots of packing availability, which limits scalability for large matrices. Additionally, it does not generalize well for lower packing availability.

Table 1. Comparison with secure d-dimensional matrix multiplication techniques. The division of works is done based on the three types of works discussed in Section 1.1. # Key-Switches = # ct-ct Mult + # Rotations.

Methodology	**Packing**	**# ct-ct Mult**	**# Rotations**	**Required Depth**†
Naive	1	$\mathcal{O}(d^3)$	-	2
[31,25]	d	$\mathcal{O}(d^2)$	$\mathcal{O}(d^2 \log_2 d)$	2
This Work	$\mathbf{d^2}$	$\mathcal{O}(\mathbf{d})$	$\mathcal{O}(\mathbf{d} \log_2 \mathbf{d})$	**2**
[30]	d^3	$\mathcal{O}(1)$	$\mathcal{O}(d)$	2
This Work	$\mathbf{d^3}$	$\mathcal{O}(\mathbf{1})$	$\mathcal{O}(\log_2 \mathbf{d})$	**2**
[22]	$d^2, d^{3\ddagger}$	$\mathcal{O}(d)$	$\mathcal{O}(d)$	3
[13,21,20,33]	d^2	$\mathcal{O}(d)$	$\mathcal{O}(d \log_2 d)$	2

† This includes (Plaintext-Ciphertext) pt-ct and ct-ct multiplications, which consume the same depth in Libraries like OpenFHE [2].
‡ With $d\times$ more packing, the number of rotations reduce from $3 \cdot d + 5 \cdot \sqrt{d}$ to $d + 2 \cdot \sqrt{d}$.

The second category, as described in [22], diverges slightly from previous methods by utilizing diagonal-packing for matrix multiplication. In this approach, matrices are packed diagonally rather than row- or column-wise. While this technique is highly complex for a multiplicative depth of two, it allows for some pre-processing at a higher multiplicative depth of three, reducing the complexity to $3d + 5\sqrt{d}$ rotations and d ct-ct multiplications. If d^3 slot packing is available, it can be further optimized to require $d + 2\sqrt{d}$ rotation computations. However, a major drawback is the necessity for three multiplicative depths. Multiplicative depth is the currency in FHE schemes; the less spent, the better, as more depth remains for remaining computations. Although the algorithms proposed in [22] can be adjusted to operate within a multiplicative depth of two, this significantly increases the number of rotations and ct-ct multiplications.

The third and final category of works, as discussed in [21,20,33], diverges significantly from the previous two types. These studies leverage a multivariate variant of the CKKS scheme (m-RLWE) [13], which enables the encoding of matrices into a hypercube structure (tensor packing) instead of a linear array-like structure typical of CKKS. This approach allows for more efficient rotations, making row-wise or column-wise transformations cheaper. Matrix multiplication using this scheme requires only a multiplicative depth of two, with the cost of transformations reduced to $2 \cdot d + 4\sqrt{d}$ rotations and d ct-ct multiplications. However, the multivariate CKKS [13] is incompatible with the original CKKS [10]. Furthermore, the parameters for multivariate CKKS are not standardized, and its initial proposal [29] was found to be insecure [5], limiting its adaptability for existing implementations.

This category also includes recent works [23,32] that propose altering the initial encoding of ciphertexts to facilitate faster multiplication on the server. Such modifications require changing the specifications of the FHE scheme or

necessitating client support for different encodings. When client data is already encrypted and stored on the server, these techniques become impractical, as the server would need to adjust the encoding to the desired form. For new computations, the FHE encoding that employs a Discrete Fourier Transform (DFT) is already resource-intensive for the FHE client. Consequently, we opted not to pursue this direction, as it merely shifts the computational burden from the server to the client, which has even less computational capacity.

1.2 Contributions.

In this work, we restrict our solution to a multiplicative depth of two and build on the first type of technique discussed above. We observe that the best-known technique in this direction does not fully utilize the SIMD processing capabilities and leaves significant scope for optimization. We bridge this gap and propose a technique that improves with higher packing availability. For d^3 packing, our technique requires only $5 \cdot \log_2 d$ rotation operations and one ct-ct multiplication. Thus, this work contributes an efficient homomorphic matrix-multiplication framework for privacy-preserving applications. Its features are as follows.

- The proposed framework fully exploits the SIMD processing capabilities provided by FHE schemes and their routines. It is generalized for various packing availabilities, ranging from d^2 to d^3 for square matrices of dimension d, with benefits increasing with increasing packing availability in the homomorphic ciphertext.
- The KeySwitch operation in FHE is the most resource-intensive low-level routine and serves as a benchmark for assessing the performance overhead of our proposed techniques. We demonstrate that for d^3 packing, our technique achieves the lowest KeySwitch complexity ($\mathcal{O}(\log_2 d)$) compared to all prior works in the literature. This is also illustrated in Table 1.
- While our initial proposal focuses on square matrices, we further generalize it to accommodate rectangular matrices. To this end, we introduce two techniques based on padding and divide-and-conquer strategies, enhancing the versatility of our framework for various neural network applications and layers, such as filter layers.
- Alongside our proposal, we provide validation artefacts for our technique, which can be accessed at [1]. Our approach leverages the open-source FHE library OpenFHE [2], allowing researchers and practitioners to easily integrate our matrix multiplication framework into their own projects.

1.3 Roadmap

The paper is organized as follows: In Section 2, we provide an overview of the FHE routines and decribe prior state-of-the-art matrix multiplication techniques. Section 3 details our proposed technique, explaining how it achieves improved

[1] `https://anonymous.4open.science/r/MatMul-0568`

runtime complexity. This section also discusses the generalized approach for arbitrary packing availability within the range of d^2 to d^3. In Section 4, we extend this technique to accommodate arbitrary rectangular matrix multiplication. The experimental evaluation is presented in Section 5, where we assess the performance of our approach. Section 6 explores scenarios involving simultaneous matrix multiplications, and Section 7 concludes the paper.

2 Background

Notations Let $\mathbb{Z}_Q$ represent the ring of integers in the $[0, Q-1]$ range. $\mathcal{R}_{Q,N} = \mathbb{Z}_Q[x]/(x^N+1)$ refers to polynomial ring containing polynomials of degree at most $N-1$ and coefficients in $\mathbb{Z}_Q$. In the Residue Number System (RNS) [16] representation, Q is a composite modulus comprising co-prime moduli, $Q = \prod_{i=0}^{L-1} q_i$. The RNS representation divides a big computation modulo Q into much smaller computations modulo q_i such that the small computations can be carried out in parallel. With the application of RNS, a polynomial $a \in \mathcal{R}_{Q,N}$ becomes a vector, say $\boldsymbol{a}$, of residue polynomials. Let the i-th residue polynomial within $\boldsymbol{a}$ be denoted as $a^i \in \mathcal{R}_{q_i,N}$. $\langle , \rangle$ denotes the dot-product between two ring elements. Matrices are denoted using capital letters. For simplicity, we assume throughout that the values in fractions are divisible.

2.1 FHE Schemes and Routines

Several Fully Homomorphic Encryption (FHE) schemes are documented in the literature, including BFV [15], BGV [7], CGGI [14], CKKS [12,11]. While BGV and BFV encrypt integers, CKKS is designed for fixed-point numbers, making it particularly well-suited for machine learning applications [19,24]. Consequently, this work focuses on the RNS (Residue Number System) variant of CKKS [11]. Below, we briefly outline the main procedures within the RNS CKKS framework [11], where ciphertexts operate at level l (indicating a multiplicative depth of $l-1$), with $l < L$. A CKKS ciphertext is represented as $\mathsf{c} = (\boldsymbol{c}_0, \boldsymbol{c}_1)$, where $\boldsymbol{c}_0$ and $\boldsymbol{c}_1$ are polynomial vectors.

Two important terms- Depth and Packing, are used throughout the paper to assess the importance of this work. Multiplicative depth refers to the complexity of the computations that the FHE scheme can support. More specifically, it denotes the maximum number of operations (like multiplications) that can be performed on encrypted data before noise in the ciphertext grows too large and prevents decryption. Understanding depth is crucial for assessing the practicality of FHE in computational tasks. Every ciphertext initially starts with full-depth L. After multiple computations, the noise growth is significant, and scaling is done to reduce the noise, which also reduces the computational depth. Thus, the lower the computation depth of a function the more computation can be performed on the data, before the depth is refreshed via the expensive Bootstrapping operation.

Table 2. CKKS Parameters

Parameter	Definition
$N, n\ (\leq \frac{N}{2})$	Polynomial size, maximum slots packed
Q, q_i	Coefficient modulus, RNS bases $Q = \prod_{i=0}^{L} q_i$
L, l	Multiplicative depth (#RNS bases - 1) $l < L$
P, p_i	Special modulus and its RNS base
L_{boot}, L_{eff}	Multiplicative depth of/after bootstrapping

Packing, also known as batching, is a technique that significantly improves the efficiency of FHE schemes, particularly those based on RLWE (Ring Learning with Error). Instead of encrypting a single plaintext value into a single ciphertext, packing allows multiple plaintext values to be encoded into a single ciphertext. This is especially valuable for applications requiring parallel computations (SIMD), such as matrix operations, machine learning, or data analytics. In RLWE-based schemes, packing leverages the structure of the underlying ring. Typically, plaintexts are elements of a polynomial ring, and packing encodes several plaintext slots into a single polynomial. Each slot can then store an individual message, enabling the system to simultaneously perform parallel homomorphic operations (like addition and multiplication) across all packed slots. This reduces the number of ciphertexts needed and increases throughput. The CKKS parameters are summarized in Table 2.

1. `CKKS.KeyGen( )`: This routine generates secret key $\mathtt{sk} = (1, s)$, public key $\mathtt{pk} = (-a \cdot s + e, a) \in \mathcal{R}^2_{Q_L,N}$, and several key-switching keys $\mathtt{ksk}_i = (-a \cdot s + e + P \cdot s', a) \in \mathcal{R}^2_{PQ_L,N}$ for $i \in [0, L)$, where a is uniformly random and s' is a secret polynomial square or permutation, depending on the type of key.
2. `CKKS.Enc`$(m, \mathtt{pk})$: It encrypts message m , and returns ciphertext $\mathtt{c} = (\boldsymbol{c}_0, \boldsymbol{c}_1) = v \cdot \mathtt{pk} + (m + e, e) \in \mathcal{R}^2_{Q_L,N}$, where e is refreshed after every computation.
3. `CKKS.Dec`$(\mathtt{c}, \mathtt{sk})$: The ciphertext `c` is decrypted using the secret key `sk` to return message $m' = \langle \mathtt{c}, \mathtt{sk} \rangle$.
4. `CKKS.Add`$(\mathtt{c}, \mathtt{c}')$: It takes two input ciphertexts `c` and $\mathtt{c}'$ and adds them to compute $\mathtt{c}_{\text{add}} = (\boldsymbol{d}_0, \boldsymbol{d}_1) = (\boldsymbol{c}_0 + \boldsymbol{c}'_0, \boldsymbol{c}_1 + \boldsymbol{c}'_1)$.
5. `CKKS.Mult`$(\mathtt{c}, \mathtt{c}')$: It multiplies the two input ciphertexts (`c`, $\mathtt{c}'$), and computes the non-linear ciphertext $\mathtt{d} = (\boldsymbol{d}_0, \boldsymbol{d}_1, \boldsymbol{d}_2) = (\boldsymbol{c}_0 \cdot \boldsymbol{c}'_0, \boldsymbol{c}_0 \cdot \boldsymbol{c}'_1 + \boldsymbol{c}_1 \cdot \boldsymbol{c}'_0, \boldsymbol{c}_1 \cdot \boldsymbol{c}'_1)$. Subsequently, `CKKS.KeySwitch` is employed to transform `d` into a linear ciphertext. It is the most expensive routine.
6. `CKKS.KeySwitch`$(\mathtt{d}, \mathtt{ksk})$: It uses a KeySwitch or 'evaluation key `ksk` to homomorphically transform a ciphertext decryptable under one key into a new ciphertext decryptable under another key. It computes $\mathtt{c}''$ where $\boldsymbol{c}''_0 = \sum_{i=0}^{l-1} d_2^i \cdot ksk_0^i \in \mathcal{R}_{PQ_l,N}$ and $\boldsymbol{c}''_1 = \sum_{i=0}^{l-1} d_2^i \cdot ksk_1^i \in \mathcal{R}_{PQ_l,N}$. This is followed by $\mathtt{c} = \big((d_0, d_1) + (\mathtt{CKKS.ModDown}(\mathtt{c}'')\big) \in \mathcal{R}^2_{Q_l,N}$. `CKKS.ModDown`() scales down the modulus (PQ_l to Q_l).
7. `CKKS.Rotate`$(\mathtt{c}, rot, \mathtt{ksk}_{\mathtt{rot}})$: It rotates the plaintext slots within `c` by *rot*. First, a permutation ρ is applied to the ciphertext polynomial coefficients. This permutation is called automorphism and is determined by the Galoi

element $gle = 5^{rot} \bmod 2N$. Finally, the permuted ciphertext is processed by `CKKS.Keyswitch` using the rotation key $\texttt{ksk}_{\texttt{rot}}$.

8. `CKKS.Bootstrap`: It refreshes a noisy ciphertext [6,8,9] by producing a new ciphertext with a higher depth or lower noise. As bootstrapping itself consumes a certain number of depths, the depth of a bootstrapped ciphertext, say L_{eff}, is smaller than the initial depth L after fresh encryption.

The fundamental FHE operations include addition, multiplication, and rotation. Among the low-level routines, ModDown and KeySwitch, the KeySwitch operation is the most resource-intensive and is required after every rotation and ct-ct multiplication. Consequently, the frequency of these routines significantly influences the complexity of matrix multiplication. In contrast, bootstrapping is a high-level routine that employs all basic and low-level routines.

2.2 Matrix Multiplication Technique

In this section, we will introduce the state-of-the-art technique for matrix multiplication at multiplicative depth two, presented in [30]. This technique leverages the available packing capability of d^3. To simplify this, let us take an example of a processing system which only operates on the array. The proposal in the work utilizes arrays which can store d^3 elements, where each matrix to be multiplied is $d \times d$. In this approach, the first step is to decide how to effectively pack a matrix in the array so that it facilitates multiplication. Suppose we have two matrices, A and B, of dimension 4×4 ($d = 4$) for multiplication. This means our array must accommodate 64 elements.

$$\text{A}= \begin{bmatrix} a_0 & a_1 & a_2 & a_3 \\ a_4 & a_5 & a_6 & a_7 \\ a_8 & a_9 & a_{10} & a_{11} \\ a_{12} & a_{13} & a_{14} & a_{15} \end{bmatrix} \quad \text{B}= \begin{bmatrix} b_0 & b_1 & b_2 & b_3 \\ b_4 & b_5 & b_6 & b_7 \\ b_8 & b_9 & b_{10} & b_{11} \\ b_{12} & b_{13} & b_{14} & b_{15} \end{bmatrix}$$

Packing Strategy. To facilitate efficient multiplication, the authors needed to adopt a packing strategy that organizes the elements of matrices A and B into the array. One common approach is to fill the array in a row-major or column-major order, depending on the operations we intend to perform. They chose a row-major order, where the elements of matrix A would be packed into the first 16 slots of the array. Similarly, matrix B would be packed in the first 16 slots of another array, as shown below.

$$A = [a_0a_1 \quad a_2a_3 \quad a_4a_5 \quad a_6a_7 \quad a_8a_9 \quad a_{10}a_{11} \quad a_{12}a_{13} \quad a_{14}a_{15}] \quad (1)$$
$$B = [b_0b_1 \quad b_2b_3 \quad b_4b_5 \quad b_6b_7 \quad b_8b_9 \quad b_{10}b_{11} \quad b_{12}b_{13} \quad b_{14}b_{15}] \quad (2)$$

Matrix Multiplication Strategy. The authors in [30] employ an efficient multiplication technique that enhances the performance of matrix multiplication in the homomorphic setting. In this approach, the elements of matrix A are

duplicated column-wise, while the elements of matrix B are duplicated row-wise. For example, the first column of A and the first row of B are as follows.

$$\mathrm{A}_1 = \begin{bmatrix} a_0 \\ a_4 \\ a_8 \\ a_{12} \end{bmatrix} \quad \mathrm{B}_1 = \begin{bmatrix} b_0 & b_1 & b_2 & b_3 \end{bmatrix}$$

Once the rows and columns are duplicated, the proposed technique gives the multiplied result A·B as follows.

$$\begin{aligned} \mathrm{A}\cdot\mathrm{B} = & \begin{bmatrix} a_0 & a_0 & a_0 & a_0 \\ a_4 & a_4 & a_4 & a_4 \\ a_8 & a_8 & a_8 & a_8 \\ a_{12} & a_{12} & a_{12} & a_{12} \end{bmatrix} \odot \begin{bmatrix} b_0 & b_1 & b_2 & b_3 \\ b_0 & b_1 & b_2 & b_3 \\ b_0 & b_1 & b_2 & b_3 \\ b_0 & b_1 & b_2 & b_3 \end{bmatrix} + \begin{bmatrix} a_1 & a_1 & a_1 & a_1 \\ a_5 & a_5 & a_5 & a_5 \\ a_9 & a_9 & a_9 & a_9 \\ a_{13} & a_{13} & a_{13} & a_{13} \end{bmatrix} \odot \begin{bmatrix} b_4 & b_5 & b_6 & b_7 \\ b_4 & b_5 & b_6 & b_7 \\ b_4 & b_5 & b_6 & b_7 \\ b_4 & b_5 & b_6 & b_7 \end{bmatrix} \\ & + \begin{bmatrix} a_2 & a_2 & a_2 & a_2 \\ a_6 & a_6 & a_6 & a_6 \\ a_{10} & a_{10} & a_{10} & a_{10} \\ a_{14} & a_{14} & a_{14} & a_{14} \end{bmatrix} \odot \begin{bmatrix} b_8 & b_9 & b_{10} & b_{11} \\ b_8 & b_9 & b_{10} & b_{11} \\ b_8 & b_9 & b_{10} & b_{11} \\ b_8 & b_9 & b_{10} & b_{11} \end{bmatrix} + \begin{bmatrix} a_3 & a_3 & a_3 & a_3 \\ a_7 & a_7 & a_7 & a_7 \\ a_{11} & a_{11} & a_{11} & a_{11} \\ a_{15} & a_{15} & a_{15} & a_{15} \end{bmatrix} \odot \begin{bmatrix} b_{12} & b_{13} & b_{14} & b_{15} \\ b_{12} & b_{13} & b_{14} & b_{15} \\ b_{12} & b_{13} & b_{14} & b_{15} \\ b_{12} & b_{13} & b_{14} & b_{15} \end{bmatrix} \end{aligned}$$

This technique requires d column-wise and row-wise duplications of the matrices. With $d^3 (= 64)$ packing available, the matrices are stored as follows (their matrix form visualization is provided afterwards in Equation 4). Notably, if we assume that matrices A and B can be packed into two ciphertexts (one for A and one for B), as illustrated in Equation 3, then only one ct-ct multiplication is needed.

$$A \cdot B = [a_0\ a_0\ a_0\ a_0\ a_4\ a_4\ \cdots\ a_{15}\ a_{15}] \odot [b_0\ b_1\ b_2\ b_3\ b_0\ b_1\ \cdots\ b_{14}\ b_{15}] \quad (3)$$

There are two challenges associated with the packing format assumption. The first is transforming the input data into the required form. In prior works, all inputs (matrices A and B) are encoded in a row-wise format (row_enc), as shown in Equations 1 and 2. Therefore, it is necessary to convert this row-wise packing into the desired encoding for our approach. The second challenge is how to accumulate the resulting multiplication results. The methods for addressing these two challenges distinguish the work presented in [30] from our proposal, which is discussed in the next section. The technique proposed in [30] is detailed in Algorithm 1.

$$A \cdot B = \begin{bmatrix} a_0 & a_0 & a_0 & a_0 \\ a_4 & a_4 & a_4 & a_4 \\ a_8 & a_8 & a_8 & a_8 \\ a_{12} & a_{12} & a_{12} & a_{12} \\ \hline a_1 & a_1 & a_1 & a_1 \\ a_5 & a_5 & a_5 & a_5 \\ a_9 & a_9 & a_9 & a_9 \\ a_{13} & a_{13} & a_{13} & a_{13} \\ \hline a_2 & a_2 & a_2 & a_2 \\ a_6 & a_6 & a_6 & a_6 \\ a_{10} & a_{10} & a_{10} & a_{10} \\ a_{14} & a_{14} & a_{14} & a_{14} \\ \hline a_3 & a_3 & a_3 & a_3 \\ a_7 & a_7 & a_7 & a_7 \\ a_{11} & a_{11} & a_{11} & a_{11} \\ a_{15} & a_{15} & a_{15} & a_{15} \end{bmatrix} \odot \begin{bmatrix} b_0 & b_1 & b_2 & b_3 \\ b_0 & b_1 & b_2 & b_3 \\ b_0 & b_1 & b_2 & b_3 \\ b_0 & b_1 & b_2 & b_3 \\ \hline b_4 & b_5 & b_6 & b_7 \\ b_4 & b_5 & b_6 & b_7 \\ b_4 & b_5 & b_6 & b_7 \\ b_4 & b_5 & b_6 & b_7 \\ \hline b_8 & b_9 & b_{10} & b_{11} \\ b_8 & b_9 & b_{10} & b_{11} \\ b_8 & b_9 & b_{10} & b_{11} \\ b_8 & b_9 & b_{10} & b_{11} \\ \hline b_{12} & b_{13} & b_{14} & b_{15} \\ b_{12} & b_{13} & b_{14} & b_{15} \\ b_{12} & b_{13} & b_{14} & b_{15} \\ b_{12} & b_{13} & b_{14} & b_{15} \end{bmatrix} \tag{4}$$

Plaintext Masks. The authors define two plaintext masks π_i and ψ_i, such that multiplication with π_i causes all expected elements of column i to become zero, while multiplication with ψ_i results in all expected elements of row i becoming zero, as illustrated above. This is achieved by placing '1' and '0' in the desired positions within another array of size d^2. Although this multiplication is plaintext-to-ciphertext (pt-ct), it requires an additional multiplicative depth. Consequently, the total multiplicative depth for the overall technique is two.

$$\begin{aligned}
\texttt{cMult}(A, \pi_0) &= [a_0 0 \quad 00 \quad a_4 0 \quad 00 \quad a_8 0 \quad 00 \quad a_{12} 0 \quad 00] \\
\texttt{cMult}(A, \pi_1) &= [0 a_1 \quad 00 \quad 0 a_5 \quad 00 \quad 0 a_9 \quad 00 \quad 0 a_{13} \quad 00] \\
\texttt{cMult}(A, \pi_2) &= [00 \quad a_2 0 \quad 00 \quad a_6 0 \quad 00 \quad a_{10} 0 \quad 00 \quad a_{14} 0] \\
\texttt{cMult}(A, \pi_3) &= [00 \quad 0 a_3 \quad 00 \quad 0 a_7 \quad 00 \quad 0 a_{11} \quad 00 \quad 0 a_{15}]
\end{aligned}$$

$$\begin{aligned}
\texttt{cMult}(B, \psi_0) &= [b_0 b_1 \quad b_2 b_3 \quad 00 \quad 00 \quad 00 \quad 00 \quad 00 \quad 00] \\
\texttt{cMult}(B, \psi_1) &= [00 \quad 00 \quad b_4 b_5 \quad b_6 b_7 \quad 00 \quad 00 \quad 00 \quad 00] \\
\texttt{cMult}(B, \psi_2) &= [00 \quad 00 \quad 00 \quad 00 \quad b_8 b_9 \quad b_{10} b_{11} \quad 00 \quad 00] \\
\texttt{cMult}(B, \psi_3) &= [00 \quad 00 \quad 00 \quad 00 \quad 00 \quad 00 \quad b_{12} b_{13} \quad b_{14} b_{15}]
\end{aligned}$$

After obtaining these results, the values are first right-aligned using the rotations specified in Step 3 and Step 10 for matrices A and B. Once aligned, these matrices are appended next to each other, which requires $d - 1$ rotations for both A and B (Steps 4-5 and 11-12). A duplication step is then performed to replicate these values and fill in the zeros in the ciphertexts, as shown in Steps 6-7 and 13-14. After multiplication, the results are accumulated again using $\log_2 d$ rotations, detailed in Steps 16-17. Overall, this technique requires $2 \cdot d$

Algorithm 1 `Matrix.Mult` [30]

Require: $A, B \leftarrow \text{row\_enc}(\mathtt{A}_{d\times d}, \mathtt{B}_{d\times d})$
Out: $C = \text{row\_enc}(\mathtt{A}_{d\times d} \times \mathtt{B}_{d\times d})$

// Preprocess A
for $j = 0$ to $d-1$ **do**
  $\tilde{A}[j] \leftarrow \mathtt{cMult}(A, \pi_j)$ ▷ Splitting A cols
  $\tilde{A}[j] \leftarrow \mathtt{Rot}(\tilde{A}[j], -j)$ ▷ Right align all cols
for $i = 1$ to $d-1$ **do** ▷ Add the cols
  $\tilde{A}[0] += \mathtt{Rot}(\tilde{A}[j], -i(d^2 - i))$
for $i = 0$ to $\log_2 d - 1$ **do** ▷ Replicate cols
  $\tilde{A}[0] += \mathtt{Rot}(\tilde{A}[0], -2^i)$

// Preprocess B
for $j = 0$ to $d-1$ **do**
  $\tilde{B}[j] \leftarrow \mathtt{cMult}(B, \psi_j)$ ▷ Splitting B rows
  $\tilde{B}[j] \leftarrow \mathtt{Rot}(\tilde{B}[j], -j \cdot d)$ ▷ Top align all rows
for $i = 1$ to $d-1$ **do** ▷ Add the rows
  $\tilde{B}[0] += \mathtt{Rot}(\tilde{B}[j], -(d^2 - d)$
for $i = 1$ to $d-1$ **do** ▷ Replicate the rows
  $\tilde{B}[0] += \mathtt{Rot}(\tilde{B}[0], -d \cdot 2^i)$

// Compute C
$C = \mathtt{Mult}(\tilde{A}[0], \tilde{B}[0])$
for $j = 0$ to $\log_2 d$ **do** ▷ Result Accumulation
  $C = C + \mathtt{Rot}(C, d^3/2^i)$

pt-ct multiplications, one ct-ct multiplication, and $2 \cdot d + 3\log_2 d - 2$ rotations, while consuming a multiplicative depth of two. The authors limit this proposal to square matrices and do not extend it to rectangular matrices or arbitrary packing.

In contrast, other techniques in the field [22] that utilize diagonal-based packing for matrix multiplication incur significantly higher rotation costs due to the required transformations. These techniques perform poorly at a multiplicative depth of two. However, by employing some pre-generation at the expense of additional multiplicative depth, they can reduce the complexity to $\mathcal{O}(d)$ rotations for lower packing availability $\mathcal{O}(d^2)$; unfortunately, this advantage does not translate effectively to higher packing availabilities.

3 Proposed Matrix Multiplication Technique

In this work, we optimize the technique outlined in [30]. We still perform ct-ct multiplication using Equation 3 (as discussed in Section 2). However, what sets our approach apart is how we achieve the packing format required for Equation 3. Before presenting our technique for d^3 packing, we note a gap in the literature: no existing technique addresses matrix multiplication complexity using Equa-

tion 3 for d^2 packing. This requirement highlighted in [30] poses a limitation for medium- to large-sized matrices.

Furthermore, prior techniques are constrained to specific packing availabilities and do not generalize to accommodate arbitrary slot availability. This limitation restricts their applicability, particularly when the available packing varies with the matrix size across different layers. This flexibility is essential for effectively handling diverse matrix dimensions in practical applications.

For instance, in a CKKS ciphertext, the available packing ranges from $2^{13} - 2^{15}$ depending on the polynomial degree. Consequently, the technique from [30] can only be applied to matrices of dimensions upto 2^5. This limitation makes it unsuitable for processing larger datasets, such as high-resolution images of size 64×64 [3,26] or 100×100 [26]. In contrast, a technique that utilizes d^2 packing could facilitate fast matrix multiplications for data sizes up to 2^7. Therefore, we begin with a technique designed for d^2 packing and demonstrate how it can be extended to accommodate higher packing availability up to d^3. While packing more than d^3 is possible, it does not enhance the complexity of a single matrix multiplication.

3.1 Technique for d^2 packing

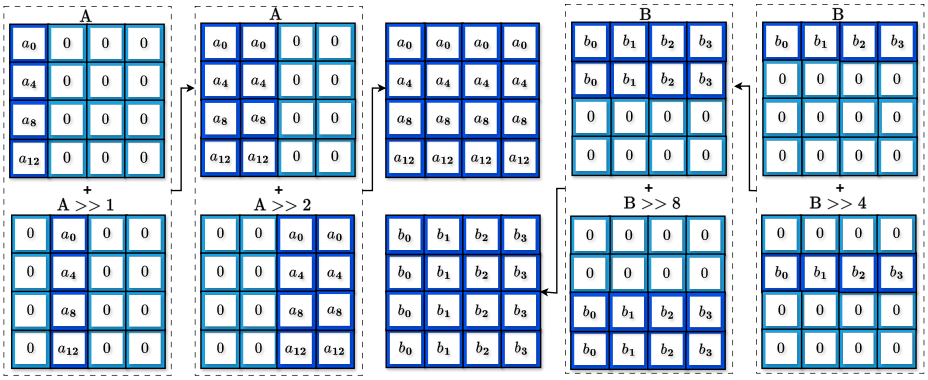

Fig. 1. Matrix Column and Row duplication for A and B respectively.

Each matrix consists of d^2 elements, which means that one ciphertext which has d^2 packing capability, can only pack a single matrix entirely. To ensure compatibility with previous works, we assume these ciphertexts are initially row-wise encoded and that d is a power of two.

Our proposed algorithm is outlined in Algorithm 2. We initiate the process of multiplying by splitting matrix A column-wise in Step 2, which allows for more efficient handling of its elements. This is followed by immediate left-alignment

in Step 3 to ensure that the data is properly structured for subsequent operations. Similarly, matrix B is split row-wise in Step 7 and left-aligned in Step 8, maintaining consistency in data organization. The duplication of these matrices necessitates $\log_2 d$ steps, as depicted in Figure 1. This duplication process occurs in Steps 4-5 for matrix A and Steps 9-10 for matrix B, allowing us to leverage the SIMD capabilities of the encryption scheme. Since we create a distinct ciphertext for each row/column split, we can perform multiplication and addition separately, as shown in Steps 11-12.

Algorithm 2 `Matrix.Mult_d`^{2}

Require: $A, B \leftarrow \text{row\_enc}(\mathtt{A}_{d\times d}, \mathtt{B}_{d\times d})$
Out: $C = \text{row\_enc}(\mathtt{A}_{d\times d} \times \mathtt{B}_{d\times d})$

// Preprocess A
1: **for** $j = 0$ to $d-1$ **do**
2: $\quad \tilde{A}[j] \leftarrow \mathtt{cMult}(A, \pi_j)$ ▷ Splitting A cols
3: $\quad \tilde{A}[j] \leftarrow \mathtt{Rot}(\tilde{A}[j], j)$ ▷ Right align all cols
4: $\quad$ **for** $i = 0$ to $\log_2(d) - 1$ **do** ▷ Replicate cols
5: $\quad\quad \tilde{A}[j] += \mathtt{Rot}(\tilde{A}[j], -2^i)$

// Preprocess B
6: **for** $j = 0$ to $d-1$ **do**
7: $\quad \tilde{B}[j] \leftarrow \mathtt{cMult}(B, \psi_j)$ ▷ Splitting B rows
8: $\quad \tilde{B}[j] \leftarrow \mathtt{Rot}(\tilde{B}[j], j \cdot d)$ ▷ Top align all rows
9: $\quad$ **for** $i = 0$ to $\log_2(d) - 1$ **do** ▷ Replicate the rows
10: $\quad\quad \tilde{B}[j] += \mathtt{Rot}(\tilde{B}[j], -2^i \cdot d)$

// Compute C
11: **for** $j = 0$ to $d-1$ **do**
12: $\quad C += \mathtt{cMult}(\tilde{A}[j], \tilde{B}[j])$

Since one ciphertext can only pack data from one row-wise or column-wise split, a post-multiplication transform is unnecessary, streamlining the process. Notably, no left alignment is required in the algorithms when $j = 0$. As a result, this technique demands $2 \cdot d$ pt-ct multiplications for the necessary transformations, d ct-ct multiplications to combine the results, and $2 \cdot d(1 + \log_2 d) - 2$ rotations to align the data correctly for the final multiplication. This efficient use of resources reduces the overall computational complexity. These details are clearly summarized in Table 3.

3.2 Technique for $2 \cdot d^2$ packing.

In the previous case, we focused on d^2 packing, where one ciphertext could only encode a single matrix. However, for smaller matrices, it is possible that more slots are available— specifically, $2 \cdot d^2$, which allows for packing two matrices within the same ciphertext. To accommodate this scenario, we modify the algorithm outlined in the previous section and describe it in Algorithm 3.

Algorithm 3 `Optimized.Matrix.Mult_2` $\cdot \mathtt{d}^2$

Require: $A, B \leftarrow \text{row\_enc}(\mathtt{A}_{d\times d}, \mathtt{B}_{d\times d})$
Out: $C = \text{row\_enc}(\mathtt{A}_{d\times d} \times \mathtt{B}_{d\times d})$

// Preprocess A
$A+ = \mathtt{Rot}(A, -d^2 + 1)$
for $j = 0$ to $(d/2) - 1$ **do**
  $\tilde{A}[j] \leftarrow \mathtt{cMult}(A, \pi_j)$ ▷ Splitting A cols
  $\tilde{A}[j] \leftarrow \mathtt{Rot}(\tilde{A}[j], 2j)$ ▷ Right align all cols
  for $i = 0$ to $\log_2(d) - 1$ **do** ▷ Replicate cols
    $\tilde{A}[j]+ = \mathtt{Rot}(\tilde{A}[j], -2^i)$

// Preprocess B
$B+ = \mathtt{Rot}(B, -d^2 + d)$
for $j = 0$ to $(d/2) - 1$ **do**
  $\tilde{B}[j] \leftarrow \mathtt{cMult}(B, \psi_j)$ ▷ Splitting B rows
  $\tilde{B}[j] \leftarrow \mathtt{Rot}(\tilde{B}[j], 2 \cdot j \cdot d)$ ▷ Top align all rows
  for $i = 0$ to $\log_2(d) - 1$ **do** ▷ Replicate the rows
    $\tilde{B}[j]+ = \mathtt{Rot}(\tilde{B}[j], -2^i \cdot d)$

// Compute C
for $j = 0$ to $(d/2) - 1$ **do**
  $C+ = \mathtt{cMult}(\tilde{A}[j], \tilde{B}[j])$
$C+ = \mathtt{Rot}(c, d^2)$

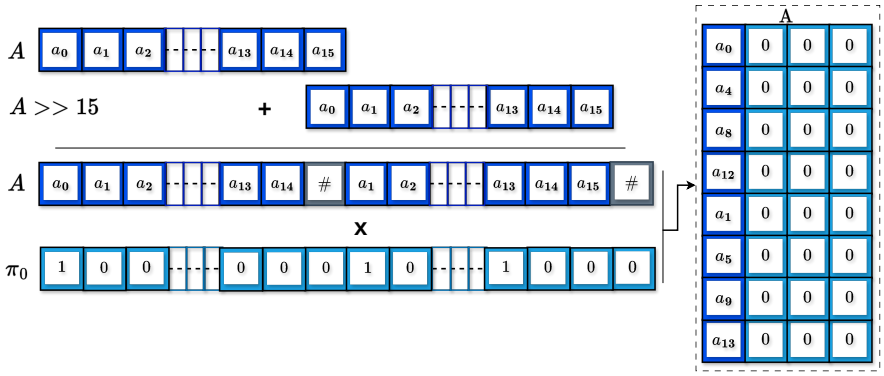

Fig. 2. Matrix duplication and alignment done in one step using rotation. First, Matrix A is rotated and added to the original A. Then, a mask is multiplied to remove the unwanted elements for the upcoming column-wise duplication step.

Table 3. Complexity of the proposed secure d-dimensional matrix multiplications

Packing	**# pt-ct Mult**	**# ct-ct Mult**	**# Rotations**	**Required Depth**[†]
d^2	$2 \cdot d$	d	$2 \cdot d(1 + \log_2 d) - 2$	2
$2 \cdot d^2$	d	$\frac{d}{2}$	$d(1 + \log_2 d) + 1$	2
$4 \cdot d^2$	$\frac{d}{2}$	$\frac{d}{4}$	$\frac{d}{2}(1 + \log_2 d) + 4$	2
d^3	2	1	$5 \cdot \log_2 d$	2
$d^{3\ddagger}$	1	1	$3 \cdot \log_2 d$	2

† This includes pt-ct and ct-ct multiplications, which consume the same depth in Libraries like OpenFHE [2].
‡ If one of the matrix is unencrypted and can be defined n any form by the server.

Instead of directly initiating row-wise or column-wise matrix decomposition, we propose packing two copies of the matrices within the same ciphertext. This critical step (Steps 1 and 7) is where our technique diverges from all prior works. The key innovation here is that rather than simply duplicating matrix A, the second ciphertext is shifted by one value to the left, as illustrated in Figure 2.

After performing multiplication with the plaintext mask, we achieve duplicated and left-aligned columns as required for the subsequent steps, eliminating the need for an additional rotation for left alignment. Thus, this adjustment allows us to obtain two columns with a single multiplication and rotation. Similarly, the B matrix is duplicated by keeping d values to the left, ensuring a seamless integration of the matrices. Consequently, this technique requires only d pt-ct multiplications, $\frac{d}{2}$ ct-ct multiplications, and $d \cdot (1 + \log_2 d) + 1$ rotations, as outlined in Table 3. A final rotation is necessary for accumulating the two partial results packed in the same ciphertext (Step 15).

3.3 Generalization to arbitrarily high packing complexity ($> d^2$)

If we estimate the cost for $4 \cdot d^2$ packing availability, we can initially pack four matrices per ciphertext. Hence, the required pt-ct and ct-ct multiplications, as well as the rotations for duplication and alignment, will decrease by a factor of four. This complexity is also outlined in Table 3. For generalization we observe that we can quantify the complexities based on the required steps and packing availability. Let s be the number of matrices that can be packed in a ciphertext. We can formulate the complexity in five parts, as follows.

1. Matrix Duplication. This part consists of matrix duplication depending on s. The duplication of this type has logarithmic complexity, as illustrated in Figure 1. The cost of each matrix duplication is $\log_2 s$ rotations and additions. For example, when $s = 1$, no duplication is possible, resulting in a cost of 0. When $s = 2$, one duplication is possible per matrix, yielding a cost of 1. Therefore, the total cost for this step is $2 \log_2 s$ rotations and additions for both matrices.

2. RC Extraction. Once the matrices have been duplicated, their row/column-wise (RC) extraction requires plaintext multiplication with the appropriate masks,

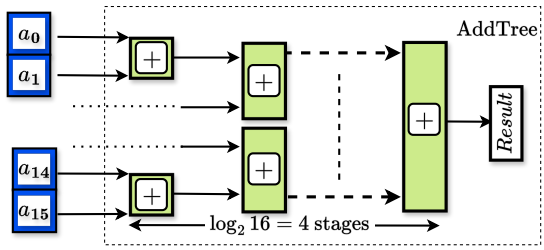

Fig. 3. The adder tree depicting final result accumulation within a ciphertext. Every stage with t elements performs an addition (in SIMD) and results in $\frac{t}{2}$ resultant elements, which are then added in the next stage. This process continues until all the elements are accumulated.

π_i and ψ_i. This extraction is performed $\frac{d}{s}\times$ per matrix, as duplication within the ciphertext offers s times more rows/columns per multiplication. Thus, the total pt-ct multiplication count amounts to $\frac{2 \cdot d}{s}$. While the first extraction does not require alignment, every subsequent extraction necessitates left alignment via rotations. Therefore, the total number of required rotations for alignment is $2(\frac{d}{s} - 1)$ for both the rows and columns of matrices A and B.

3. RC Duplication. After alignment, the rows and columns require duplication. Each duplication incurs a fixed cost of $\log_2 d$ rotations and additions. However, its frequency changes depending on the number of duplications needed, $\frac{2 \cdot d}{s}$. Thus, the total number of required rotations and additions for this step is $\frac{2 \cdot d}{s}(\log_2 d)$ for both row and column duplications.

4. Multiplication. We proceed to their multiplication once we have the row and column-wise duplicated matrices. This process is directly related to the number of ciphertexts that must be multiplied, resulting in $\frac{d}{s}$ ct-ct multiplications.

5. Accumulation. Finally, we must accumulate the results packed within a ciphertext. This accumulation process is not an issue for d^2 packing since no matrix copies can be stored in a single ciphertext. However, accumulation becomes necessary for any higher packing availability. The accumulation is analogous to duplication but in the reverse direction, as depicted in Figure 3. It is important to note that elements cannot be accessed prior to rotation in an FHE ciphertext; hence, each $\log_2 s$ stage in the addition tree requires a rotation. Overall, the complexity of this stage is $\log_2 s$ rotations and additions.

Finally, we arrive at the following generalized operation requirements for matrix multiplication at depth two. The algorithm for this process is outlined in Algorithm 4. By substituting the appropriate value of s into the formulas below, we can derive the operation complexities listed in Table 3. As previously mentioned, Key Switching is only necessary following the ct-ct multiplications and rotations. Therefore, while pt-ct multiplication reduces the multiplicative depth, it does not significantly impact the overall time consumption.

Algorithm 4 `Generalized.Matrix.Mult` (for arbitrary s matrix packing)

Require: $A, B \leftarrow \text{row\_enc}(\mathtt{A}_{d\times d}, \mathtt{B}_{d\times d})$
Out: $C = \text{row\_enc}(\mathtt{A}_{d\times d} \times \mathtt{B}_{d\times d})$

// Preprocess A
for $i = 0$ to $\log_2 s - 1$ **do**
  $A+ = \mathtt{Rot}(A, -d^2 \cdot 2^i + 2^i)$ ▷ Matrix Duplication
for $j = 0$ to $\frac{d}{s} - 1$ **do**
  $\tilde{A}[j] \leftarrow \mathtt{cMult}(A, \pi_j)$ ▷ Extracting A column-wise
  $\tilde{A}[j] \leftarrow \mathtt{Rot}(\tilde{A}[j], s \cdot j)$ ▷ Left align all ciphertexts
  for $i = 0$ to $\log_2(d) - 1$ **do** ▷ Column Duplication
    $\tilde{A}[j]+ = \mathtt{Rot}(\tilde{A}[j], -2^i)$

// Preprocess B
for $i = 0$ to $\log_2 s - 1$ **do**
  $B+ = \mathtt{Rot}(B, -d^2 \cdot 2^i + d \cdot 2^i)$ ▷ Matrix Duplication
for $j = 0$ to $\frac{d}{s} - 1$ **do**
  $\tilde{B}[j] \leftarrow \mathtt{cMult}(B, \psi_j)$ ▷ Extracting B row-wise
  $\tilde{B}[j] \leftarrow \mathtt{Rot}(\tilde{B}[j], s \cdot j \cdot d)$ ▷ Left align all ciphetexts
  for $i = 0$ to $\log_2(d) - 1$ **do** ▷ Row Duplication
    $\tilde{B}[j]+ = \mathtt{Rot}(\tilde{B}[j], -2^i \cdot d)$

// Compute C
for $j = 0$ to $\frac{d}{s} - 1$ **do**
  $C+ = \mathtt{cMult}(\tilde{A}[j], \tilde{B}[j])$ ▷ Matrix Multiplication
for $i = 0$ to $\log_2 s - 1$ **do**
  $C+ = \mathtt{Rot}(C, d^2 \cdot 2^i)$ ▷ Accumulation

- # **pt-ct Multiplications:** $\frac{2\cdot d}{s}$
- # **ct-ct Multiplications:** $\frac{d}{s}$
- # **Rotations:** $\frac{2\cdot d}{s}(\log_2 d) + 2(\frac{d}{s} - 1) + 3 \cdot \log_2 s$

The d^3 Packing Case.

For d^3 packing, we note that $s = d$, resulting in two required pt-ct multiplications and one ct-ct multiplication, yielding a constant time complexity. Additionally, the number of rotations required is $5 \log_2 d$, which is logarithmic in terms of the matrix dimension d. This performance surpasses all prior works, including those operating at multiplicative depth three [22].

In specific scenarios where the server has control over the model, and only one matrix is encrypted, this operation count can be further reduced to $3 \log_2 d$ rotations (as shown in Table 3). If we assume that the client can format the data as needed during the encoding step before encryption, the rotation operations can be minimized to $\log_2 d$, and the multiplicative depth requirement becomes

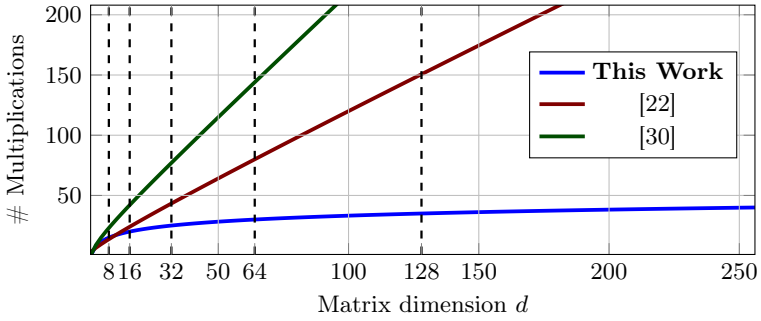

Fig. 4. Graph comparing the rotation count reduction for d^3 packing.

one. Therefore, the proposed technique is versatile and can be optimized for various scenarios.

A comparison of actual operation counts for encrypted matrix-matrix multiplication with prior works is illustrated in Figure 4, specifically for d^3 packing availability. It underscores the efficiency and effectiveness of our approach in enhancing secure matrix multiplication tasks within untrusted environments.

4 Generalization to Rectangular Matrices

4.1 Padding based technique

The above technique can also be adapted for rectangular matrices of the form- $A_{l\times d} \cdot B_{d\times t}$. Previous works [30] employ zero-padding to the rows and columns of the matrices, transforming them into square matrices. This approach allows them to leverage techniques designed specifically for square matrices. We extend this idea to our method as well. By adding appropriate padding to the matrices A and B, we can ensure they fit the necessary dimensions for our algorithm.

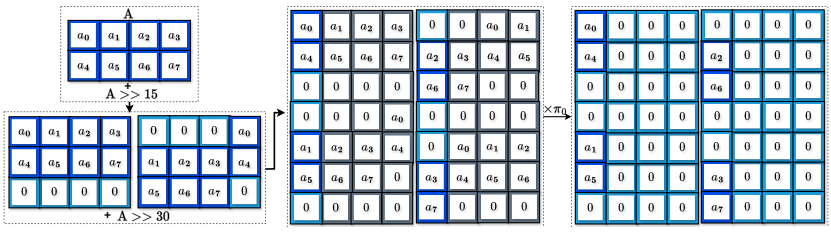

Fig. 5. The $A_{2\times 4}$ matrix duplication transformation considering padding. The rectangular matrix undergoes two rotated accumulations to result in the middle ciphertext. This ciphertext behaves like a square matrix with zero padding. The grey elements are not needed and are removed via multiplication with the mask. After this step, column-wise duplication follows.

This transformation can be incorporated into the application logic, ensuring that shifts consistently account for the factor d, as shown in Figure 5. Notably, most complexities remain unchanged except for row-wise and column duplication steps. Specifically, the columns of A will now need to be duplicated t times, while the rows of B require l duplications. Thus, the complexity for the RC Duplication step is adjusted to $\frac{d}{s}(\log_2 l + \log_2 t)$. This allows us to perform the same efficient operations while accommodating the rectangular structure of the matrices. The zero-padding process does not significantly impact the overall complexity, as it effectively maintains the logarithmic characteristics of our approach.

The technique discussed can also be generalized for rectangular matrices by adopting the approach from [30]. Specifically, when $d < t$, $t - d$ zero-padding columns are added to matrix A, and conversely, when $d > t$, zero-padding columns are appended to matrix B. This method is bounded by the naive technique that ensures both matrices become square, with dimensions $k \times k$, where $k = \max(l, d, t)$. As a result, the final rotation complexity for this approach is $5 \cdot \log_2 k$, necessitating k^3 packing.

4.2 Common Divide-and-Conquer Technique

The previous technique necessitates high packing availability, a requirement that becomes increasingly challenging when there is a significant difference between d and l or t. To address this, we explore an alternate matrix-splitting and divide-and-conquer approach that can be applied in all cases, irrespective of whether $l \neq d$ or $t \neq d$. This method is illustrated in Figure 6, showcasing two scenarios: one where $d < l, t$ and another where $d > l, t$.

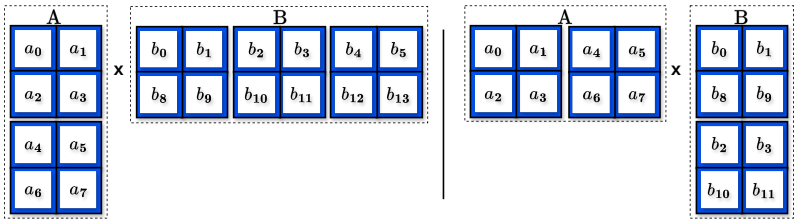

Fig. 6. Rectangular matrices division for the divide-and-conquer technique.

The essence of this technique lies in dividing the larger matrix into smaller $d \times d$ matrices, which mitigates the need for excessive padding. For instance, in the first scenario (where $l = 4$, $d = 2$, and $t = 6$), instead of requiring 6^3 packings, we only need $6 \cdot 2^2$ slots per ciphertext. This reduces the overall packing requirement significantly. Additionally, the prior rotation complexity of $5 \cdot \log_2 6$ simplifies to $5 \cdot \log_2 2$, as no accumulation is necessary. In the second scenario (where $l, t = 2, d4$), instead of needing 4^3 packing, we again require only $4 \cdot 2^2$ slots per ciphertext. Here, the rotation requirement changes from $5 \cdot \log_2 4$

to $5 \cdot \log_2 2 + \log_2 2$, as one accumulation is required. Consequently, the overall packing requirement can be generalized as $k \cdot p^2$, where $k = \max(l, d, t)$ and $p = \min(l, d, t)$. This alternate approach reduces the need for extensive padding.

Next, the $p \times p$ matrix chunks are appended. These chunks need to be duplicated $\frac{t}{d}\times$ for A and $\frac{l}{p}\times$ for B. After the initial duplication, all the $p \times p$ matrices are packed in one ciphertext. Hence, $3 \cdot (\log_2 p)$ rotations are required for row-wise and column-wise duplication and result accumulation for a $p \times p$ matrix. $\log_2 \frac{d}{p}$ rotations are required for the final accumulation if d is greater than l, t (the second case in Figure 6). There is no increase in ct-ct multiplications, and the multiplicative depth stays at two. The rotation complexity for this approach can be expressed as: $(\frac{l}{p} + \frac{t}{p}) \log_2 p + (\log_2 \frac{t}{p} + \log_2 \frac{l}{p}) + 3 \cdot (\log_2 p) + \log_2 \frac{d}{p}$. The breakdown of each component is as follows.

- *Initial Matrix Duplication.* The term $(\frac{l}{p} + \frac{t}{p}) \log_2 p$ accounts for the number of rotations required to duplicate the initial matrices. This arises from the need to replicate the smaller matrix chunks across the dimensions of l, t.
- *Duplication*$_{p\times p}$. The $p \times p$ matrix chunks need to be duplicated $\frac{t}{d}\times$ for matrix A and $\frac{l}{p}\times$ for matrix B. This contributes to the subsequent rotation complexity of $\log_2 \frac{t}{p} + \log_2 \frac{l}{p}$ operations.
- *Accumulation*$_{p\times p}$. Once the $p \times p$ matrices are packed into a single ciphertext, an additional $3 \cdot (\log_2 p)$ rotations are required for the row-wise and column-wise duplication and the accumulation of results for all the packed $p \times p$ matrix separately.
- *Final Accumulation.* The term $\log_2 \frac{d}{p}$ is included if d exceeds both l and t, indicating the rotations required for the final accumulation of results.

There is no increase in ct-ct multiplications throughout this process, and the multiplicative depth remains at two. This method effectively optimizes both the packing requirements and the rotation complexity.

5 Experimental Evaluation

We utilize the CKKS [12] FHE scheme for our experimental evaluation. Prior works also adopt this scheme due to its ability to perform computations over approximate arithmetic, making it suitable for various applications, including Neural Networks. For our benchmarks, we employ the open-source OpenFHE [2] library, ensuring compatibility with the setup provided by the FHERMA matrix multiplication challenge[2]. Our proposed matrix multiplication technique was tested in this challenge, where the available packing was $2 \cdot d^2$, yielding the best results[3]. Our artefacts for the general solution are available at[4].

We take ring-degree $N = 2^{16}$, which enables us to fully pack 32×32 matrices, duplicated 32 times. For our benchmarks, we evaluate matrix sizes ranging from

[2] `https://fherma.io/challenges/652bf669485c878710fd020b/overview`
[3] Winner of the competition.
[4] `https://anonymous.4open.science/r/MatMul-0568`

Table 4. Runtime evaluation of privacy-preserving matrix multiplication.

Matrix Dimension		Utilized Packing	Runtime	Slot Usage
A	B		(s)	
2×2	2×2	$d^3 = 2^3$	1.54	0.02%
2×4	4×4	$d^3 = 2^6$	1.79	0.20%
4×4	4×2	$d^3 = 2^6$	1.80	0.20%
4×4	4×4	$d^3 = 2^6$	1.94	0.20%
8×8	8×4	$d^3 = 2^9$	2.21	1.56%
8×8	8×8	$d^3 = 2^9$	2.31	1.56%
16×16	16×4	$d^3 = 2^{12}$	3.12	12.5%
16×16	16×16	$d^3 = 2^{12}$	3.23	12.5%
32×32	32×8	$d^3 = 2^{15}$	7.54	100%
32×32	32×32	$d^3 = 2^{15}$	7.88	100%
64×64	64×64	$8 \cdot d^2 = 2^{15}$	17.42	100%
128×128	128×128	$2 \cdot d^2 = 2^{15}$	157.1	100%

2×2 to 128×128 to demonstrate the scalability and efficiency of the technique. We observe that the proposed method is highly parallelizable. Therefore, we leverage the available parallelization using the '*pragma omp parallel*' routine. The runtimes reported in Table 4 are based on benchmarks executed on a 12th Gen Intel® Core™ i7-1260P processor with 16 threads and 32 GB RAM. We also provide benchmarks for the rectangular matrix case (Case 1) where padding is used. Additionally, the runtime for the divide-and-conquer approach can be extrapolated from the results reported for square matrices.

6 Discussion on the Case of Packed Multiplications.

The proposed technique and results may raise the question: for an application requiring several simultaneous matrix multiplications (r), is it more efficient to duplicate a matrix within the same ciphertext or pack r distinct matrices for separate multiplications? In the case of d^2 packing, our technique necessitates $2 \cdot d(1 + \log_2 d - 2)$ rotations. If s packing slots are available, we could pack s distinct matrices into one ciphertext instead of duplicating the same matrix. This approach can be applied $\frac{r}{s}$ times if $r > s$. The ct-ct multiplication count remains d, and the number of rotations required for processing s packed matrix multiplications is $2 \cdot d(1 + \log_2 d) - 2$. However, packing s distinct matrices in one ciphertext requires $s-1$ additional rotations and additions per matrix. Therefore, the total rotation requirement for $r > s$ becomes- $\frac{r}{s}(2 \cdot d(1 + \log_2 d) + 2 \cdot s - 4)$.

For $r \leq s$, the complexity simplifies to:

$$2 \cdot d(1 + \log_2 d) + 2 \cdot r - 4$$

In the prior work [30], similar to our technique for d^3 packing, the approach involves packing (or duplicating) the same matrix s times within a single ciphertext. The total number of rotations required to process $r \leq s$ matrix multiplica-

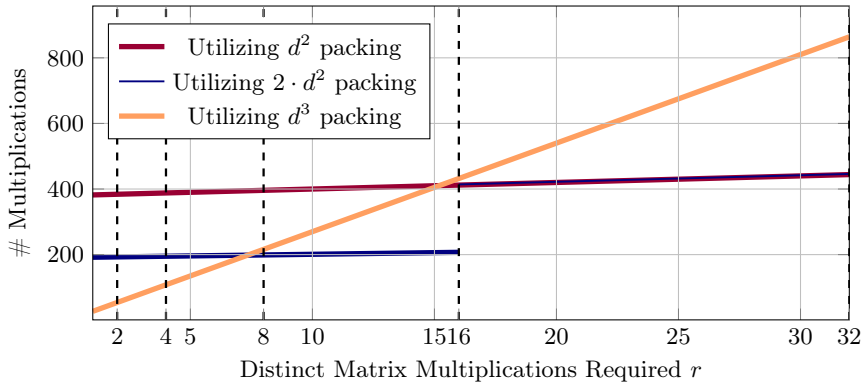

Fig. 7. Graph comparing the rotation count for $d^3(d = 32)$ packing available peer ciphertext with increasing distinct simultaneous matrix multiplication requirement r.

tions using this technique are as follows.

$$2 \cdot r \cdot d(1 + \log_2 d) + 2 \cdot r(\frac{d}{s} - 1) + 3 \cdot r \cdot \log_2 s$$

For the case where $r = s = 1$, both our proposed technique and the technique from [30] offer the same computational complexity. However, as r increases, the complexity of the latter technique grows much more rapidly. This makes our proposed technique for d^2 packing more suitable for scenarios where multiple matrix multiplications need to be performed simultaneously, as it lowers computational costs. The choice of technique ultimately depends on the available matrix packing capacity (s) and the required number of matrix multiplications (r). Users can make informed decisions that align with their specific application requirements by providing a detailed complexity breakdown regarding these values.

In Figure 7, we illustrate the number of required rotations for the above two techniques for matrices of dimension $d = 32$ and packing availability $d^3 = 2^{15}$. When required r is 15 or higher, the former technique, utilizing d^2 packed computation with distinct matrix packing, results in a lower rotation requirement. However, the d^3 packing technique per ciphertext is better for lower simultaneous matrix multiplication requirements. A similar analysis can be done for more combinations of s and r with different packing utilization per matrix multiplication to make it comprehensive.

7 Conclusion

In this work, we introduced a novel technique for secure matrix multiplication using homomorphic encryption that significantly outperforms all previous approaches. By optimizing the required key-switch operation complexity from $\mathcal{O}(d)$

to $\mathcal{O}(\log_2 d)$ for square matrices of dimension d, our approach enhances computational efficiency. We also extended this technique to handle rectangular matrices by proposing two methods: one based on padding and another employing a divide-and-conquer strategy. These methods generalize our approach beyond square matrices, making it adaptable to arbitrary packing between d^2 and d^3. This adaptability is particularly beneficial in secure neural network applications, where the matrix dimensions and the available packing per matrix change at each layer. By thoroughly analyzing the complexities and trade-offs of these techniques, we offer insights that can guide future research in privacy-preserving computation. This work lays a strong foundation for future exploration in optimizing secure matrix operations using homomorphic encryption.

Acknowledgement

This work was supported by the State Government of Styria, Austria – Department Zukunftsfonds Steiermark. We also extend our gratitude to the organizers of the FHERMA challenges for motivating this work.

References

1. Aikata, Mert, A.C., Kwon, S., Deryabin, M., Roy, S.S.: REED: chiplet-based scalable hardware accelerator for fully homomorphic encryption. IACR Cryptol. ePrint Arch. p. 1190 (2023), https://eprint.iacr.org/2023/1190
2. Al Badawi, A., Bates, J., Bergamaschi, F., Cousins, D.B., Erabelli, S., Genise, N., Halevi, S., Hunt, H., Kim, A., Lee, Y., Liu, Z., Micciancio, D., Quah, I., Polyakov, Y., R.V., S., Rohloff, K., Saylor, J., Suponitsky, D., Triplett, M., Vaikuntanathan, V., Zucca, V.: OpenFHE: Open-Source Fully Homomorphic Encryption Library. In: Proceedings of the 10th Workshop on Encrypted Computing & Applied Homomorphic Cryptography. pp. 53–63. WAHC'22, Association for Computing Machinery, New York, NY, USA (2022). https://doi.org/10.1145/3560827.3563379, https://doi.org/10.1145/3560827.3563379
3. angelolmg: Textile texture database (tilda) for defect detection (2023), https://www.kaggle.com/datasets/angelolmg/tilda-400-64x64-patches
4. Beirendonck, M.V., D'Anvers, J., Turan, F., Verbauwhede, I.: FPT: A fixed-point accelerator for torus fully homomorphic encryption. In: Meng, W., Jensen, C.D., Cremers, C., Kirda, E. (eds.) Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security, CCS 2023, Copenhagen, Denmark, November 26-30, 2023. pp. 741–755. ACM (2023). https://doi.org/10.1145/3576915.3623159, https://doi.org/10.1145/3576915.3623159
5. Bootland, C., Castryck, W., Vercauteren, F.: On the security of the multivariate ring learning with errors problem. IACR Cryptol. ePrint Arch. p. 966 (2018), https://eprint.iacr.org/2018/966
6. Bossuat, J., Mouchet, C., Troncoso-Pastoriza, J.R., Hubaux, J.: Efficient Bootstrapping for Approximate Homomorphic Encryption with Non-sparse Keys. In: Canteaut, A., Standaert, F. (eds.) Advances in Cryptology - EUROCRYPT 2021 - 40th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zagreb, Croatia, October 17-21, 2021, Proceedings, Part

I. Lecture Notes in Computer Science, vol. 12696, pp. 587–617. Springer (2021). https://doi.org/10.1007/978-3-030-77870-5_21, https://doi.org/10.1007/978-3-030-77870-5_21

7. Brakerski, Z., Gentry, C., Vaikuntanathan, V.: Fully homomorphic encryption without bootstrapping. Electron. Colloquium Comput. Complex. p. 111 (2011), https://eccc.weizmann.ac.il/report/2011/111
8. Chen, H., Chillotti, I., Song, Y.: Improved Bootstrapping for Approximate Homomorphic Encryption. In: Ishai, Y., Rijmen, V. (eds.) Advances in Cryptology - EUROCRYPT 2019 - 38th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Darmstadt, Germany, May 19-23, 2019, Proceedings, Part II. Lecture Notes in Computer Science, vol. 11477, pp. 34–54. Springer (2019). https://doi.org/10.1007/978-3-030-17656-3_2, https://doi.org/10.1007/978-3-030-17656-3_2
9. Cheon, J.H., Han, K., Kim, A., Kim, M., Song, Y.: Bootstrapping for Approximate Homomorphic Encryption. In: Nielsen, J.B., Rijmen, V. (eds.) Advances in Cryptology - EUROCRYPT 2018 - 37th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Tel Aviv, Israel, April 29 - May 3, 2018 Proceedings, Part I. Lecture Notes in Computer Science, vol. 10820, pp. 360–384. Springer (2018). https://doi.org/10.1007/978-3-319-78381-9_14, https://doi.org/10.1007/978-3-319-78381-9_14
10. Cheon, J.H., Han, K., Kim, A., Kim, M., Song, Y.: A full RNS variant of approximate homomorphic encryption. In: Cid, C., Jr., M.J.J. (eds.) Selected Areas in Cryptography - SAC 2018 - 25th International Conference, Calgary, AB, Canada, August 15-17, 2018, Revised Selected Papers. Lecture Notes in Computer Science, vol. 11349, pp. 347–368. Springer (2018). https://doi.org/10.1007/978-3-030-10970-7_16, https://doi.org/10.1007/978-3-030-10970-7_16
11. Cheon, J.H., Han, K., Kim, A., Kim, M., Song, Y.: A full RNS variant of approximate homomorphic encryption. In: Cid, C., Jr., M.J.J. (eds.) Selected Areas in Cryptography - SAC 2018 - 25th International Conference, Calgary, AB, Canada, August 15-17, 2018, Revised Selected Papers. Lecture Notes in Computer Science, vol. 11349, pp. 347–368. Springer (2018). https://doi.org/10.1007/978-3-030-10970-7_16, https://doi.org/10.1007/978-3-030-10970-7_16
12. Cheon, J.H., Kim, A., Kim, M., Song, Y.S.: Homomorphic encryption for arithmetic of approximate numbers. In: Takagi, T., Peyrin, T. (eds.) Advances in Cryptology - ASIACRYPT 2017 - 23rd International Conference on the Theory and Applications of Cryptology and Information Security, Hong Kong, China, December 3-7, 2017, Proceedings, Part I. Lecture Notes in Computer Science, vol. 10624, pp. 409–437. Springer (2017). https://doi.org/10.1007/978-3-319-70694-8_15, https://doi.org/10.1007/978-3-319-70694-8_15
13. Cheon, J.H., Kim, A., Yhee, D.: Multi-dimensional packing for HEAAN for approximate matrix arithmetics. IACR Cryptol. ePrint Arch. p. 1245 (2018), https://eprint.iacr.org/2018/1245
14. Chillotti, I., Gama, N., Georgieva, M., Izabachène, M.: TFHE: fast fully homomorphic encryption over the torus. Journal of Cryptology **33**(1), 34–91 (2020)
15. Fan, J., Vercauteren, F.: Somewhat practical fully homomorphic encryption. IACR Cryptol. ePrint Arch. p. 144 (2012), http://eprint.iacr.org/2012/144
16. Garner, H.L.: The Residue Number System. IRE Trans. Electron. Comput. **8**(2), 140–147 (1959). https://doi.org/10.1109/TEC.1959.5219515, https://doi.org/10.1109/TEC.1959.5219515

17. Geelen, R., Beirendonck, M.V., Pereira, H.V.L., Huffman, B., McAuley, T., Selfridge, B., Wagner, D., Dimou, G.D., Verbauwhede, I., Vercauteren, F., Archer, D.W.: BASALISC: programmable hardware accelerator for BGV fully homomorphic encryption. IACR Trans. Cryptogr. Hardw. Embed. Syst. **2023**(4), 32–57 (2023). https://doi.org/10.46586/TCHES.V2023.I4.32-57, https://doi.org/10.46586/tches.v2023.i4.32-57
18. Halevi, S., Shoup, V.: Algorithms in helib. In: Garay, J.A., Gennaro, R. (eds.) Advances in Cryptology - CRYPTO 2014 - 34th Annual Cryptology Conference, Santa Barbara, CA, USA, August 17-21, 2014, Proceedings, Part I. Lecture Notes in Computer Science, vol. 8616, pp. 554–571. Springer (2014). https://doi.org/10.1007/978-3-662-44371-2_31, https://doi.org/10.1007/978-3-662-44371-2_31
19. Han, K., Hong, S., Cheon, J.H., Park, D.: Logistic regression on homomorphic encrypted data at scale. In: The Thirty-Third AAAI Conference on Artificial Intelligence, AAAI 2019, The Thirty-First Innovative Applications of Artificial Intelligence Conference, IAAI 2019, The Ninth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2019, Honolulu, Hawaii, USA, January 27 - February 1, 2019. pp. 9466–9471. AAAI Press (2019). https://doi.org/10.1609/aaai.v33i01.33019466, https://doi.org/10.1609/aaai.v33i01.33019466
20. Huang, H., Zong, H.: Secure matrix multiplication based on fully homomorphic encryption. J. Supercomput. **79**(5), 5064–5085 (2023). https://doi.org/10.1007/S11227-022-04850-4, https://doi.org/10.1007/s11227-022-04850-4
21. Jang, J., Lee, Y., Kim, A., Na, B., Yhee, D., Lee, B., Cheon, J.H., Yoon, S.: Privacy-preserving deep sequential model with matrix homomorphic encryption. In: Suga, Y., Sakurai, K., Ding, X., Sako, K. (eds.) ASIA CCS '22: ACM Asia Conference on Computer and Communications Security, Nagasaki, Japan, 30 May 2022 - 3 June 2022. pp. 377–391. ACM (2022). https://doi.org/10.1145/3488932.3523253, https://doi.org/10.1145/3488932.3523253
22. Jiang, X., Kim, M., Lauter, K.E., Song, Y.: Secure outsourced matrix computation and application to neural networks. In: Lie, D., Mannan, M., Backes, M., Wang, X. (eds.) Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, CCS 2018, Toronto, ON, Canada, October 15-19, 2018. pp. 1209–1222. ACM (2018). https://doi.org/10.1145/3243734.3243837, https://doi.org/10.1145/3243734.3243837
23. Ju, J.H., Park, J., Kim, J., Kim, D., Ahn, J.H.: Neujeans: Private neural network inference with joint optimization of convolution and bootstrapping. CoRR **abs/2312.04356** (2023). https://doi.org/10.48550/ARXIV.2312.04356, https://doi.org/10.48550/arXiv.2312.04356
24. Kim, A., Song, Y., Kim, M., Lee, K., Cheon, J.: Logistic regression model training based on the approximate homomorphic encryption. BMC Medical Genomics **11** (10 2018). https://doi.org/10.1186/s12920-018-0401-7
25. Lu, W., Kawasaki, S., Sakuma, J.: Using fully homomorphic encryption for statistical analysis of categorical, ordinal and numerical data. In: 24th Annual Network and Distributed System Security Symposium, NDSS 2017, San Diego, California, USA, February 26 - March 1, 2017. The Internet Society (2017)
26. Mavi, A.: Sign language digits dataset (2017). https://doi.org/10.34740/KAGGLE/DSV/11071, https://www.kaggle.com/dsv/11071
27. Mert, A.C., Aikata, Kwon, S., Shin, Y., Yoo, D., Lee, Y., Roy, S.S.: Medha: Microcoded Hardware Accelerator for computing on Encrypted data. IACR Trans. Cryptogr. Hardw. Embed. Syst. **2023**(1), 463–500 (2023). https://doi.org/10.

46586/tches.v2023.i1.463-500, https://doi.org/10.46586/tches.v2023.i1.463-500
28. Micciancio, D.: A first glimpse of cryptography's holy grail. Commun. ACM **53**(3), 96 (2010). https://doi.org/10.1145/1666420.1666445, https://doi.org/10.1145/1666420.1666445
29. Pedrouzo-Ulloa, A., Troncoso-Pastoriza, J.R., Pérez-González, F.: Multivariate lattices for encrypted image processing. In: 2015 IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP 2015, South Brisbane, Queensland, Australia, April 19-24, 2015. pp. 1707–1711. IEEE (2015). https://doi.org/10.1109/ICASSP.2015.7178262, https://doi.org/10.1109/ICASSP.2015.7178262
30. Rizomiliotis, P., Triakosia, A.: On matrix multiplication with homomorphic encryption. In: Regazzoni, F., van Dijk, M. (eds.) Proceedings of the 2022 on Cloud Computing Security Workshop, CCSW 2022, Los Angeles, CA, USA, 7 November 2022. pp. 53–61. ACM (2022). https://doi.org/10.1145/3560810.3564267, https://doi.org/10.1145/3560810.3564267
31. Wang, S., Huang, H.: Secure outsourced computation of multiple matrix multiplication based on fully homomorphic encryption. KSII Trans. Internet Inf. Syst. **13**(11), 5616–5630 (2019). https://doi.org/10.3837/TIIS.2019.11.019, https://doi.org/10.3837/tiis.2019.11.019
32. Zheng, X., Li, H., Wang, D.: A new framework for fast homomorphic matrix multiplication. IACR Cryptol. ePrint Arch. p. 1649 (2023), https://eprint.iacr.org/2023/1649
33. Zhu, L., Hua, Q., Chen, Y., Jin, H.: Secure outsourced matrix multiplication with fully homomorphic encryption. In: Tsudik, G., Conti, M., Liang, K., Smaragdakis, G. (eds.) Computer Security - ESORICS 2023 - 28th European Symposium on Research in Computer Security, The Hague, The Netherlands, September 25-29, 2023, Proceedings, Part I. Lecture Notes in Computer Science, vol. 14344, pp. 249–269. Springer (2023). https://doi.org/10.1007/978-3-031-50594-2_13, https://doi.org/10.1007/978-3-031-50594-2_13