Improved Matrix Inversion with Packed Ciphertexts using Fully Homomorphic Encryption

Seunghu Kim[1], Seongbong Choi[1], and Hyung Tae Lee[1]

School of Computer Science and Engineering, Chung-Ang University, Republic of Korea
{rlatmdgn77, welq2st, hyungtaelee}@cau.ac.kr

Abstract. Matrix inversion is a fundamental operation, but performing it over encrypted matrices remains a significant challenge. This is mainly due to the fact that conventional inversion algorithms–such as Gaussian elimination–depend heavily on comparison and division operations, which are computationally expensive to perform under homomorphic encryption. To mitigate this, Ahn et al. (ESORICS 2023) introduced an inversion method based on iterative matrix multiplications. However, their approach encrypts matrices entry-wise, leading to poor scalability. A key limitation of prior work stems from the absence of an efficient matrix multiplication technique for matrix-packed ciphertexts, particularly one with low multiplicative depth.

In this paper, we present a novel homomorphic matrix multiplication algorithm optimized for matrix-packed ciphertexts, requiring only a multiplicative depth of two. Building on this foundation, we propose an efficient algorithm for homomorphic matrix inversion. Experimental results show that our method outperforms the state-of-the-art: for 8×8 matrices, it achieves a 6.8× speedup over the method by Ahn et al., and enables inversion of larger matrices that were previously infeasible. We further compare our homomorphic matrix multiplication technique against existing matrix-packed homomorphic matrix multiplication algorithms. When used for iterative inversion, our method consistently outperforms prior approaches. In particular, for 16×16 and 32×32 matrices, it achieves 1.88× and 1.43× speedups, respectively, over the algorithm by Aikata and Roy. Finally, we demonstrate the practical benefits of our method by applying it to privacy-preserving linear regression. For a dataset of 64 samples with 8 features, our approach achieves a 1.13× speedup in training time compared to the state-of-the-art homomorphic matrix inversion solution.

1 Introduction

As server-delegated computation becomes increasingly prevalent, concerns over potential breaches of data privacy continue to rise. Fully homomorphic encryption (FHE) has emerged as a promising solution to this challenge, since it enables secure outsourced computation by allowing arbitrary operations on encrypted data without revealing sensitive information. However, computation in the encrypted domain incurs substantial performance overheads, necessitating the development of more efficient FHE-friendly algorithms.

Matrix operations are among the most fundamental building blocks in computation, playing a critical role in applications including privacy-preserving machine learning and data analytics. Nevertheless, executing such operations efficiently over encrypted data remains challenging. In particular, ciphertext matrix inversion (C-MI) is a difficult task, since conventional inversion methods, such as Gaussian elimination, rely on division and comparison operations that are computationally expensive and difficult to support within the FHE framework.

Iterative methods, exemplified by Newton's method, offer a viable alternative in the context of FHE, since they rely solely on repeated ciphertext-ciphertext matrix multiplications (CC-MM). However, these methods typically require a large number of iterations, resulting in deep multiplicative circuits and increased computational costs. To mitigate this issue, Ahn et al. [1] proposed an FHE-friendly iterative matrix inversion algorithm that reduces the required number of iterations. Despite this advancement, their method encodes each matrix entry into a separate ciphertext, yielding d^3 ciphertexts for a $d \times d$ matrix and introducing severe scalability limitations.

A natural approach to improving scalability is to leverage ciphertext packing techniques [2], which allow multiple data elements to be encoded into a single ciphertext. Although various CC-MM methods have been

developed based on matrix-packed ciphertexts, they are often suboptimal for iterative applications due to their circuit depth. For example, the method by Jiang et al. [3] is efficient for single-step multiplications, but it incurs a circuit depth of three–consisting of one ciphertext-ciphertext multiplication and two ciphertext-plaintext multiplications–resulting in an overall circuit depth approximately three times the number of iterations when applied to iterative C-MI.

1.1 Our Contributions

In this paper, we propose a novel C-MI algorithm that operates on matrix-packed ciphertexts. As a core component, we propose an efficient CC-MI algorithm specifically designed for matrix-packed inputs, achieving a circuit depth of two. Building upon this CC-MM algorithm, we then construct an efficient C-MI algorithm for matrix-packed inputs. Finally, we demonstrate the applicability of our approach by employing it in a privacy-preserving linear regression task.

The key contributions of this work can be summarized as follows:

- We propose a novel CC-MM algorithm tailored for iterative C-MI, requiring a circuit depth of 2–consisting of one ciphertext-ciphertext multiplication and one ciphertext-plaintext multiplication–while maintaining competitive performance. To achieve this, we present a vectorized variant of the linear transformation technique originally introduced in HElib [4–6]. By fully utilizing the available slots in the ciphertext, the proposed vectorized approach significantly reduces computational complexity, thereby enhancing its applicability to a broad range of use cases involving linear transformations over large ring dimensions.
- Building on our CC-MM algorithm, we present a C-MI technique tailored for matrix-packed inputs, which offers significant improvements over prior non-packed approaches. Moreover, when compared with existing CC-MM algorithms that support matrix-packed ciphertexts, our method demonstrates superior performance for matrix dimensions up to $d = 32$. For example, for the inversion of a 16×16 matrix, our method takes 2004 seconds, achieving speedups of approximately $2.45\times$, $1.66\times$, and $1.88\times$ over the CC-MM algorithms from [3], [7], and [8], respectively, when used as subroutines. Consequently, our proposed C-MI delivers outstanding performance on matrices ranging from 4×4 to 32×32, which are commonly encountered in applications such as linear regression for medical prediction, where inverting a $d \times d$ matrix constructed from d input features is a key step.
- We present enhanced privacy-preserving linear regression training by leveraging our proposed C-MI algorithm as a foundational component. In contrast to prior work [1] which employs entry-wise encryption, our method supports packing multiple samples into a single ciphertext, thereby reducing both memory consumption and the number of homomorphic operations. In our experiments, training 64 samples with 8 features in the encrypted domain took 2767 seconds and achieved a mean squared error of 0.03 during inference. This corresponds to an approximate $1.13\times$ speedup compared to using the CC-MM method from [8].

1.2 Related Works

In this subsection, we review related work on CC-MM and C-MI algorithms in the context of FHE.

Ciphertext-Ciphertext Matrix Multiplication. An approach to CC-MM is to treat a matrix as a set of vectors and pack each vector individually. Lu et al. [9] employed row-wise packing, where each matrix row is encoded into a ciphertext, and designed a CC-MM algorithm based on the Fox algorithm [10]. Cha et al. [11] proposed shifted diagonal encoding which packs diagonal vectors into separate ciphertexts to reduce the number of key-switching operations in CC-MM. These methods require d ciphertexts to encrypt a $d \times d$ matrix, leaving slots unused when d is smaller than the number of available slots. For large d (e.g., $d = N$, where N is the ring dimension of the underlying FHE scheme), vector packing becomes a natural choice, since packing the entire matrix into a single ciphertext is no longer feasible. Recently, Park [12] proposed reducing CC-MM to four plaintext matrix multiplications, building on the techniques of [13,14] that leverage row-wise coefficient-packed ciphertexts.

Most research has focused on packing an entire matrix into a single ciphertext. In this setting, designing CC-MM algorithms often depends on whether the plaintext algebra is assumed to follow a hypercube structure. Halevi and Shoup introduced the hypercube structure in HElib [4–6], enabling multi-dimensional rotations through specific parameter choices. Building on this structure, Rathee et al. [15] extended the algorithm from [9] to support matrix-packed ciphertexts, which was further generalized to rectangular matrices in [16]. To the best of our knowledge, the method proposed in [17] represents the current state-of-the-art among hypercube-based approaches.

However, the hypercube structure is incompatible with the CKKS scheme [18] due to its characteristic-zero plaintext algebra. As a result, in CKKS, either row-wise or column-wise rotations require additional circuit depth, increasing the cost of CC-MM algorithms. CC-MM algorithms from [3, 7, 8] use row-major order encoding where the rows of a matrix are concatenated into a one-dimensional vector. This setting fits the CKKS scheme without relying on the hypercube structure. In this work, we focus on designing a CC-MM algorithm applicable to the CKKS scheme, since it is well-suited for real number computations and widely used in practical applications. Accordingly, we adopt the CC-MM algorithms in [3,7,8] as our primary baselines.

On the other hand, MHEAAN [19, 20] was proposed to support homomorphic matrix operations, but its security relies on the Multivariate RLWE (m-RLWE) assumption, which is non-standard. Other works [21–23] consider CC-MM algorithms for multiple matrices. Ma et al. [24] optimize the CC-MM algorithm in [3] for covariance matrix computation, and recent works [25, 26] introduce new CC-MM algorithms based on novel packing techniques.

Ciphertext Matrix Inversion. In [19,20], the authors present a C-MI method based on a matrix variant of the approximate multiplicative inversion technique from [27]. Specifically, the inverse matrix can be computed through matrix multiplications as follows: Let $\mathbf{A}$ be a $d \times d$ input matrix and $\mathbf{I}_d$ be the identity matrix of the same dimension, with $\|\overline{\mathbf{A}}\| \leq \epsilon < 1$, where $\overline{\mathbf{A}} = \mathbf{I}_d - \frac{1}{2^t}\mathbf{A}$ for a positive integer t, and $\|\cdot\|$ denotes the spectral norm and ϵ denotes a bound for ensuring convergence. The inverse is then given by

$$\frac{1}{2^t} \prod_{0 \leq i < r} \mathbf{A}(\mathbf{I}_d + \overline{\mathbf{A}}^{2^i}) = \mathbf{A}^{-1}(\mathbf{I}_d - \overline{\mathbf{A}}^{2^r}).$$

Additionally, in [28, 29], matrix inversion based on the Newton method [30] was employed. The Newton method iteratively updates $\mathbf{X}_k$ until it converges to $\mathbf{A}^{-1}$. For a dominant eigenvalue α of $\mathbf{A}\mathbf{A}^{\mathrm{T}}$ where $\mathbf{A}^{\mathrm{T}}$ denotes the transpose of a matrix $\mathbf{A}$, the updates proceed as follows:

$$\mathbf{X}_0 = \frac{1}{\alpha}\mathbf{A}^{\mathrm{T}} \quad , \quad \mathbf{X}_{k+1} = \mathbf{X}_k \left(2\mathbf{I}_d - \mathbf{A}\mathbf{X}_k\right).$$

Both approaches require successive matrix multiplications, leading to a significant circuit depth when implemented in the encrypted domain. To address this, Ahn et al. proposed a depth-optimized matrix inversion approach [1], which splits the matrix multiplication chain into separate matrices denoted as $\mathbf{A}$ and $\mathbf{Y}$. A full description of this algorithm is presented in Algorithm 1. We note that in Algorithm 1, $\mathrm{Tr}(\mathbf{A}\mathbf{A}^{\mathrm{T}})$ denotes the trace of the matrix $\mathbf{A}\mathbf{A}^{\mathrm{T}}$, defined as the sum of its diagonal entries.

However, their approach employs naive matrix multiplication, which does not utilize matrix-packing techniques. As a result, their experiments are limited to small matrix dimensions $d < 10$ and scalability remains a challenge.

1.3 Outline of the Paper

In Section 2, we present the preliminaries and notations. Section 3 presents our new CC-MM algorithm and its application to C-MI. Section 4 provides our implementation and comparative efficiency analysis of the proposed algorithm against existing methods. Section 5 discusses an application of C-MI, specifically for privacy-preserving linear regression training, and Section 6 provides concluding remarks.

Algorithm 1 $\mathsf{IterMatInv}(\mathbf{A}, r)$: Matrix Inversion in [1]

Input: A $d \times d$ matrix $\mathbf{A}$ and the number of iterations r
Output: An approximate inverse matrix $\mathbf{Y}_r$

$\lambda \leftarrow \mathrm{Tr}(\mathbf{A}\mathbf{A}^{\mathrm{T}})$
$\mathbf{Y}_0 \leftarrow \frac{1}{\lambda}\mathbf{A}^{\mathrm{T}}$
$\overline{\mathbf{A}}_0 \leftarrow \mathbf{I}_d - \frac{1}{\lambda}\mathbf{A}\mathbf{A}^{\mathrm{T}}$
for $i = 1$ to r **do**
    $\mathbf{Y}_i \leftarrow \mathbf{Y}_{i-1}(\mathbf{I}_d + \overline{\mathbf{A}}_{i-1})$
    $\overline{\mathbf{A}}_i \leftarrow \overline{\mathbf{A}}_{i-1}^2$
end for
return $\mathbf{Y}_r$

2 Preliminaries

In this section, we first present the general components of an FHE scheme. We then introduce basic arithmetic operations on packed ciphertexts, and subsequently focus on a more advanced evaluation procedure–linear transformation. These will serve as key components in our proposed CC-MM algorithm.

We begin by introducing the notations used throughout the paper.

Notations. For an $m \times n$ matrix $\mathbf{A}$, $\mathbf{A}_{ij}$ or $\mathbf{A}_{i,j}$ denotes its (i, j)-th entry, where $i = 1, \ldots, m$ and $j = 1, \ldots, n$. The row (resp., column) indices of $\mathbf{A}$ are assumed to range from 1 to m (resp., n). When performing computations involving row (resp., column) indices, modular arithmetic is applied with modulus m (resp., n). In this context, we use an adjusted modulus operation where the result lies in $\{1, \ldots, d\}$, instead of the conventional $\{0, \ldots, d-1\}$. All other index-related computations in this paper (e.g., vector indices or diagonal numbering) follow standard modular arithmetic with results in $\{0, \ldots, d-1\}$.

The transpose of a matrix $\mathbf{A}$ is denoted by $\mathbf{A}^{\mathrm{T}}$. For two matrices $\mathbf{A}$ and $\mathbf{B}$ of the same size, $\mathbf{A} \odot \mathbf{B}$ denotes their Hadamard (entry-wise) product. Similarly, for two vectors $\mathbf{v}$ and $\mathbf{w}$ of the same size, $\mathbf{v} \odot \mathbf{w}$ denotes their component-wise product. The i-th component of a vector $\mathbf{v}$ is written as $\mathbf{v}[i]$. An object prefixed with ct denotes its encrypted form; for example, $\mathsf{ct}.\mathbf{A}$ represents the ciphertext corresponding to the plaintext matrix $\mathbf{A}$. Finally, for an algorithm A, we write $a \leftarrow A$ to indicate that A outputs a.

2.1 Fully Homomorphic Encryption

We briefly review an FHE scheme which consists of the following four polynomial-time algorithms. Let $\mathcal{M}$ and $\mathcal{C}$ denote plaintext and ciphertext spaces of the FHE scheme, respectively.

- $\mathsf{KeyGen}(\lambda)$: For a given security parameter λ, it outputs a secret key sk, a public key pk, and a set of evaluation keys evk.
- $\mathsf{Enc}(pk, m)$: On input the public key pk and a message $m \in \mathcal{M}$, it outputs $ct \in \mathcal{C}$, which is an encryption of m.
- $\mathsf{Dec}(sk, ct)$: On input the secret key sk and the ciphertext ct, it outputs m' obtained by decrypting ct using sk.
- $\mathsf{Eval}(evk, f, ct_1, \ldots, ct_k)$: Given the set of evaluation keys evk, an arithmetic circuit $f : \mathcal{M}^k \to \mathcal{M}$ and a set of ciphertexts $ct_1, \ldots, ct_k$, which are encryptions of $m_1, \ldots, m_k$, respectively, it outputs $\overline{ct}$ which is an encryption of the evaluation result $\overline{m} = f(m_1, \ldots, m_k)$.

We say that an FHE scheme is *correct* if for any security parameter λ, any messages $m_i \in \mathcal{M}$ with $1 \le i \le k$, and any arithmetic circuit $f : \mathcal{M}^k \to \mathcal{M}$, it holds

$$\mathsf{Dec}(sk, \mathsf{Eval}(evk, f, ct_1, \ldots, ct_k)) = f(m_1, \ldots, m_k)$$

with overwhelming probability in λ where $(sk, pk, evk) \leftarrow \mathsf{KeyGen}(\lambda)$ and $ct_i \leftarrow \mathsf{Enc}(pk, m_i)$ for $1 \le i \le k$.

2.2 Basic Operations on Packed Ciphertexts

Next, we provide homomorphic evaluation algorithms for basic and specific operations. Some FHE schemes allow to encode multiple messages into a single ciphertext and perform single-instruction-multiple-data (SIMD) operations on ciphertexts through a packing technique [2]. Throughout this paper, we primarily concentrate on FHE schemes that support packing of messages and SIMD operations.

Let $\mathcal{M}$ be a message space of a basic FHE scheme and assume that the batch size of the FHE scheme is n. That is, it is assumed that the message space of FHE supporting packed techniques is $\mathcal{M}^n$. We consider the following basic homomorphic operations on packed ciphertexts:

- $\mathsf{Add}(ct_1, ct_2)$: It takes as inputs two ciphertexts ct_1, ct_2 of messages $\mathbf{m} = (m_1, \ldots, m_n)$ and $\overline{\mathbf{m}} = (\overline{m}_1, \ldots, \overline{m}_n)$, respectively, and outputs a ciphertext of message $\mathbf{m} + \overline{\mathbf{m}} = (m_1 + \overline{m}_1, \ldots, m_n + \overline{m}_n)$.
  We note that the input ciphertexts may be regarded as those of matrices $\mathbf{M}$, $\overline{\mathbf{M}}$ of the same size. In this case, the output of this algorithm is regarded as a ciphertext of plaintext matrix $\mathbf{M} + \overline{\mathbf{M}}$.
- $\mathsf{Mult}(ct_1, ct_2)$: It takes as inputs two ciphertexts ct_1, ct_2 of messages $\mathbf{m} = (m_1, \ldots, m_n)$ and $\overline{\mathbf{m}} = (\overline{m}_1, \ldots, \overline{m}_n)$, respectively, and outputs a ciphertext of message $(m_1 \cdot \overline{m}_1, \ldots, m_n \cdot \overline{m}_n)$, which is denoted by $\mathbf{m} \odot \overline{\mathbf{m}}$.
  As the Add algorithm, the input ciphertexts may be regarded as those of matrices $\mathbf{M}$, $\overline{\mathbf{M}}$ of the same size. In this case, the output of this algorithm is regarded as a ciphertext of plaintext matrix $\mathbf{M} \odot \overline{\mathbf{M}}$.
- $\mathsf{PMult}(\mathbf{m}, ct)$: On inputs a plaintext $\mathbf{m} = (m_1, \ldots, m_n)$ and a ciphertext ct, which is a ciphertext of $\overline{\mathbf{m}} = (\overline{m}_1, \ldots, \overline{m}_n)$, it outputs a ciphertext of $\mathbf{m} \odot \overline{\mathbf{m}}$.
  We note that if two inputs of this algorithm are a plaintext matrix $\mathbf{M}$ and a ciphertext of plaintext matrix $\overline{\mathbf{M}}$, respectively, it outputs a ciphertext of plaintext matrix $\mathbf{M} \odot \overline{\mathbf{M}}$.

A rotation operation can be employed to enable evaluations across different slots.

- $\mathsf{Rot}(ct, l)$: It takes a ciphertext ct which is a ciphertext of message $(m_1, \ldots, m_n)$ and outputs a ciphertext of message $(m_l, \ldots, m_n, m_1, \ldots, m_{l-1})$. Here, a positive (resp., negative) l implies a left (resp., right) rotation. It is worth noting that distinct evaluation keys are required to rotate a ciphertext by different indices l.

2.3 Packing Technique in the CKKS scheme

In this subsection, we delve into the structure of packed plaintexts with a particular focus on the CKKS scheme [18]. Let $\mathcal{P} = \mathbb{Q}[X]/(X^N + 1)$, where $N > 2$ is a power-of-2 ring dimension, and define $\omega = \exp(\frac{\pi i}{N})$ as a $(2N)$-th primitive root of unity in $\mathbb{C}$. For $0 \leq k < N/2$, ω_k is given by ω^{5^k}. A map $\tau : \mathcal{P} \to \mathbb{C}^{N/2}$ is defined as $\tau : m(X) \mapsto \mathbf{m} = (m_0, \ldots, m_{N/2-1})$ for a plaintext polynomial $m(X) \in \mathcal{P}$, where $m_j = m(\omega_j)$. Consequently, τ^{-1} may be employed in packing to encode a length-$(N/2)$ vector of complex numbers into a polynomial.

When a smaller vector is sufficient, sparse packing can be utilized as discussed in [31]. Let n be a divisor of N, and define $Y = X^{N/n}$ and $\tilde{\omega}_k = \tilde{\omega}^{5^k}$ for $0 \leq k < n/2$, where $\tilde{\omega} = \exp(\frac{\pi i}{n})$. This allows an $(n/2)$-length vector $\tilde{\mathbf{m}} = (\tilde{m}_0, \ldots, \tilde{m}_{n/2-1})$ to be encoded in a polynomial $\overline{m}(Y) \in \mathbb{Q}[Y]/(Y^n + 1)$, such that $\overline{m}(\tilde{\omega}_j) = \tilde{m}_j$. This encoding can be interpreted as $\tilde{m}(X) = \overline{m}(X^{N/n})$ by treating Y as $X^{N/n}$. Decoding $\tilde{m}(X)$ using $\{\omega_j\}_{0 \leq j < N/2}$ yields $\tau(\tilde{m}) = (\hat{m}_0, \ldots, \hat{m}_{N/2-1})$, where

$$\hat{m}_j = \tilde{m}(\omega_j) = \overline{m}(\omega_j^{N/n}) = \overline{m}(\tilde{\omega}_j) = \tilde{m}_{j \bmod (n/2)}.$$

This demonstrates that the sparsely packed vector $\tilde{\mathbf{m}}$ internally corresponds to (N/n)-times replication of itself. Hence, in this paper, the matrices are considered sparsely packed when $d^2 < N/2$, with d being a power-of-2, and are interpreted as an $((N/2)/d^2)$-fold concatenated matrix.

2.4 Linear Transformation

We introduce a method for evaluating a linear transformation, which can be conceptualized as either a matrix-vector multiplication or a vector-matrix multiplication. The approach introduced in HElib [4] involves utilizing the diagonals of the matrix. For a $d \times d$ matrix $\mathbf{M}$, let $\mathbf{d}_k$ be its k-th diagonal, where $\mathbf{d}_k[i-1] = \mathbf{M}_{i(i+k)}$ for $0 \leq k < d$ and $0 < i \leq d$. That is, the k-th diagonal vector is given as

$$\mathbf{d}_k = \left[\mathbf{M}_{1(1+k)}\ \mathbf{M}_{2(2+k)}\ \cdots\ \mathbf{M}_{d(d+k)}\right]^{\mathrm{T}}$$

where the indices are taken modulo d. Then, the matrix-vector multiplication $\mathbf{w} = \mathbf{M}\mathbf{v}$ can be computed as follows:

$$\mathbf{w} = \sum_{0 \leq i < d} \mathbf{d}_i \odot \rho(\mathbf{v}, i) \tag{1}$$

where $\rho(\mathbf{v}, l)$ represents the vector obtained by circularly rotating the components of column vector $\mathbf{v}$ to the upward by l positions. For example, if $\mathbf{v} = (v_1, v_2, v_3, v_4)^{\mathrm{T}}$, then $\rho(\mathbf{v}, 3) = (v_4, v_1, v_2, v_3)^{\mathrm{T}}$.

For row vectors $\mathbf{x}$ and $\mathbf{y}$, the vector-matrix multiplication $\mathbf{x} = \mathbf{y}\mathbf{M}$ can be represented as $\mathbf{x}^{\mathrm{T}} = \mathbf{M}^{\mathrm{T}}\mathbf{y}^{\mathrm{T}}$. Let $\overline{\mathbf{d}}_k$ denote the k-th diagonal vector of $\mathbf{M}^{\mathrm{T}}$, defined as $\overline{\mathbf{d}}_k[i-1] = \mathbf{M}_{(i+k)i}$. Then, $\mathbf{x} = \mathbf{y}\mathbf{M}$ can be computed as follows:

$$\mathbf{x}^{\mathrm{T}} = \sum_{0 \leq i < d} \overline{\mathbf{d}}_i \odot \rho(\mathbf{y}^{\mathrm{T}}, i). \tag{2}$$

In our algorithms, we require a matrix-vector multiplication where the matrix is unencrypted while the vector is encrypted. When $\mathbf{v}$ (resp., $\mathbf{y}$) is given as a packed ciphertext of a vector and $\mathbf{d}_i$'s (resp., $\overline{\mathbf{d}}_i$'s) are given as plaintexts of vectors, the addition, rotation, and Hadamard product operations in Eq. (1) (resp., (2)) can be homomorphically evaluated using Add, Rot, and PMult operations, respectively, which were introduced in Section 2.2.

3 Our Proposed Methods for CC-MM and C-MI

In this section, we present an improved C-MI algorithm in which the entire matrix is packed into a single ciphertext. To that end, we propose a novel CC-MM algorithm optimized for nested multiplications. The proposed method requires only a circuit depth of two and maximizes computational efficiency by fully utilizing the large batch sizes afforded by a high ring dimension of the underlying FHE scheme. We begin by introducing the core concept of our approach and design a tailored subroutine to support the overall algorithm. We then formally describe the complete CC-MM algorithm. Based on this construction, we develop an improved C-MI algorithm optimized for matrix-packed inputs.

In our design, matrices are packed into a single ciphertext using row-major order encoding, as in [3,7,8]. This approach concatenates the rows of a matrix sequentially and encrypts the resulting vector as a single ciphertext. We note that our algorithm is also compatible with column-major order packing, in which the columns are concatenated instead.

3.1 Main Idea of the Proposed CC-MM Method

We first introduce the main idea of our proposed CC-MM algorithm. For simplicity, we assume that the two input matrices are $d \times d$ square matrices.

Let $\mathbf{A}$, $\mathbf{B}$, and $\mathbf{C}$ be $d \times d$ matrices where $\mathbf{C}$ is the product of $\mathbf{A}$ and $\mathbf{B}$, i.e., $\mathbf{C} = \mathbf{A}\mathbf{B}$. When $\mathbf{b}_j$ and $\mathbf{c}_j$ are the j-th columns of $\mathbf{B}$ and $\mathbf{C}$, respectively, the following relationship holds:

$$\mathbf{c}_j = \mathbf{A}\mathbf{b}_j = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1d} \\ a_{21} & a_{22} & \cdots & a_{2d} \\ \vdots & \vdots & \ddots & \vdots \\ a_{d1} & a_{d2} & \cdots & a_{dd} \end{bmatrix} \begin{bmatrix} b_{1j} \\ b_{2j} \\ \vdots \\ b_{dj} \end{bmatrix} = \begin{bmatrix} a_{11}b_{1j} + a_{12}b_{2j} + \cdots + a_{1d}b_{dj} \\ a_{21}b_{1j} + a_{22}b_{2j} + \cdots + a_{2d}b_{dj} \\ \vdots \\ a_{d1}b_{1j} + a_{d2}b_{2j} + \cdots + a_{dd}b_{dj} \end{bmatrix} \tag{3}$$

where $\mathbf{A} = [a_{ij}]$ and $\mathbf{B} = [b_{ij}]$ for $i, j = 1, \ldots, d$.

In terms of linear transformations, the above relation can be expressed using the diagonals of the input matrices. Specifically, let $\mathbf{d}_i$ be the i-th diagonal vector of the matrix $\mathbf{A}$, i.e.,

$$\mathbf{d}_i = \left[a_{1(1+i)}\ a_{2(2+i)}\ \cdots\ a_{d(d+i)}\right]^{\mathrm{T}}$$

for $i = 0, \ldots, d-1$. Then, Eq. (3) can be reformulated as

$$\begin{aligned}
\mathbf{c}_j &= \begin{bmatrix} a_{11}b_{1j} + a_{12}b_{2j} + \cdots + a_{1d}b_{dj} \\ a_{21}b_{1j} + a_{22}b_{2j} + \cdots + a_{2d}b_{dj} \\ \vdots \\ a_{d1}b_{1j} + a_{d2}b_{2j} + \cdots + a_{dd}b_{dj} \end{bmatrix} = \begin{bmatrix} a_{11}b_{1j} + a_{12}b_{2j} + \cdots + a_{1d}b_{dj} \\ a_{22}b_{2j} + a_{23}b_{3j} + \cdots + a_{21}b_{1j} \\ \vdots \\ a_{dd}b_{dj} + a_{d1}b_{1j} + \cdots + a_{d(d-1)}b_{(d-1)j} \end{bmatrix} \\
&= \begin{bmatrix} a_{11} \\ a_{22} \\ \vdots \\ a_{dd} \end{bmatrix} \odot \begin{bmatrix} b_{1j} \\ b_{2j} \\ \vdots \\ b_{dj} \end{bmatrix} + \begin{bmatrix} a_{12} \\ a_{23} \\ \vdots \\ a_{d1} \end{bmatrix} \odot \begin{bmatrix} b_{2j} \\ b_{3j} \\ \vdots \\ b_{1j} \end{bmatrix} + \cdots + \begin{bmatrix} a_{1d} \\ a_{21} \\ \vdots \\ a_{d(d-1)} \end{bmatrix} \odot \begin{bmatrix} b_{dj} \\ b_{1j} \\ \vdots \\ b_{(d-1)j} \end{bmatrix} \\
&= \mathbf{d}_0 \odot \rho(\mathbf{b}_j, 0) + \mathbf{d}_1 \odot \rho(\mathbf{b}_j, 1) + \cdots + \mathbf{d}_{d-1} \odot \rho(\mathbf{b}_j, d-1) \\
&= \sum_{0 \le i < d} \mathbf{d}_i \odot \rho(\mathbf{b}_j, i).
\end{aligned}$$

Next, we define the $d \times d$ matrix $\mathsf{repdiag}(\mathbf{A}, i)$ whose columns are all identical copies of the i-th diagonal of $\mathbf{A}$, as follows:

$$\mathsf{repdiag}(\mathbf{A}, i) = \overbrace{\left[\mathbf{d}_i\ \mathbf{d}_i\ \cdots\ \mathbf{d}_i\right]}^{d}.$$

Then, the matrix multiplication $\mathbf{C} = \mathbf{AB}$ can be expressed as a sum of Hadamard products between the replicated diagonals of $\mathbf{A}$ and rotations of $\mathbf{B}$, as shown below:

$$\begin{aligned}
\mathbf{C} &= \left[\mathbf{c}_1\ \mathbf{c}_2\ \cdots\ \mathbf{c}_d\right] \\
&= \left[\sum_{0 \le i < d} \mathbf{d}_i \odot \rho(\mathbf{b}_1, i) \cdots \sum_{0 \le i < d} \mathbf{d}_i \odot \rho(\mathbf{b}_d, i)\right] \\
&= \sum_{0 \le i < d} \left[\mathbf{d}_i \odot \rho(\mathbf{b}_1, i) \cdots \mathbf{d}_i \odot \rho(\mathbf{b}_d, i)\right] \\
&= \sum_{0 \le i < d} \left[\mathbf{d}_i \cdots \mathbf{d}_i\right] \odot \left[\rho(\mathbf{b}_1, i) \cdots \rho(\mathbf{b}_d, i)\right] \\
&= \sum_{0 \le i < d} \mathsf{repdiag}(\mathbf{A}, i) \odot \varphi(\mathbf{B}, i) \qquad (4)
\end{aligned}$$

where φ denotes the row shifting operation. More precisely, $\varphi(\mathbf{B}, i)$ circularly rotates each column vector of $\mathbf{B}$ upward by i positions. For example, $\varphi(\mathbf{B}, 1) = \left[\rho(\mathbf{b}_1, 1) \cdots \rho(\mathbf{b}_d, 1)\right]$.

To help the readers understanding, we present a toy example that illustrates the decomposition of the matrix multiplication of 3×3 matrices into a summation of Hadamard products in Figure 1. We note that the matrix multiplication algorithm in Eq. (4), as shown in Figure 1, is described in a column-based manner. Similarly, it can be formulated in a row-based manner, although the details are omitted in this paper.

To evaluate Eq. (4) with encrypted matrices $\mathsf{ct}.\mathbf{A}$ and $\mathsf{ct}.\mathbf{B}$, the required rotations of $\mathsf{ct}.\mathbf{B}$, corresponding to $\varphi(\mathbf{B}, i)$ for $0 \le i < d$, can be readily obtained using at most d Rot operations via a naive method. More precisely, given a row-major order packed and encrypted matrix $\mathsf{ct}.\mathbf{B}$, $\mathsf{Rot}(\mathsf{ct}.\mathbf{B}, d \cdot i)$ yields $\varphi(\mathbf{B}, i)$ in the underlying plaintext.

In the following subsections, we mainly discuss how to efficiently compute the $\mathsf{repdiag}$ function.

$$
\begin{aligned}
\mathbf{C} = \mathbf{AB} &= \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} \begin{bmatrix} b_{11} & b_{12} & b_{13} \\ b_{21} & b_{22} & b_{23} \\ b_{31} & b_{32} & b_{33} \end{bmatrix} \\
&= \begin{bmatrix} a_{11} & a_{11} & a_{11} \\ a_{22} & a_{22} & a_{22} \\ a_{33} & a_{33} & a_{33} \end{bmatrix} \odot \begin{bmatrix} b_{11} & b_{12} & b_{13} \\ b_{21} & b_{22} & b_{23} \\ b_{31} & b_{32} & b_{33} \end{bmatrix} + \begin{bmatrix} a_{12} & a_{12} & a_{12} \\ a_{23} & a_{23} & a_{23} \\ a_{31} & a_{31} & a_{31} \end{bmatrix} \odot \begin{bmatrix} b_{21} & b_{22} & b_{23} \\ b_{31} & b_{32} & b_{33} \\ b_{11} & b_{12} & b_{13} \end{bmatrix} + \begin{bmatrix} a_{13} & a_{13} & a_{13} \\ a_{21} & a_{21} & a_{21} \\ a_{32} & a_{32} & a_{32} \end{bmatrix} \odot \begin{bmatrix} b_{31} & b_{32} & b_{33} \\ b_{11} & b_{12} & b_{13} \\ b_{21} & b_{22} & b_{23} \end{bmatrix} \\
&= \begin{bmatrix} \mathbf{d}_0 & \mathbf{d}_0 & \mathbf{d}_0 \end{bmatrix} \odot \varphi(\mathbf{B}, 0) + \begin{bmatrix} \mathbf{d}_1 & \mathbf{d}_1 & \mathbf{d}_1 \end{bmatrix} \odot \varphi(\mathbf{B}, 1) + \begin{bmatrix} \mathbf{d}_2 & \mathbf{d}_2 & \mathbf{d}_2 \end{bmatrix} \odot \varphi(\mathbf{B}, 2) \\
&= \mathsf{repdiag}(\mathbf{A}, 0) \odot \varphi(\mathbf{B}, 0) + \mathsf{repdiag}(\mathbf{A}, 1) \odot \varphi(\mathbf{B}, 1) + \mathsf{repdiag}(\mathbf{A}, 2) \odot \varphi(\mathbf{B}, 2)
\end{aligned}
$$

Fig. 1. Toy example of decomposing 3×3 matrix multiplication $\mathbf{C} = \mathbf{AB}$ into three parts using Hadamard products

$$
\mathbf{W}^{(0)} = \left[\begin{array}{ccc|ccc|ccc} 1&0&0&0&0&0&0&0&0 \\ 1&0&0&0&0&0&0&0&0 \\ 1&0&0&0&0&0&0&0&0 \\ \hline 0&0&0&0&1&0&0&0&0 \\ 0&0&0&0&1&0&0&0&0 \\ 0&0&0&0&1&0&0&0&0 \\ \hline 0&0&0&0&0&0&0&0&1 \\ 0&0&0&0&0&0&0&0&1 \\ 0&0&0&0&0&0&0&0&1 \end{array}\right], \quad \mathbf{W}^{(1)} = \left[\begin{array}{ccc|ccc|ccc} 0&1&0&0&0&0&0&0&0 \\ 0&1&0&0&0&0&0&0&0 \\ 0&1&0&0&0&0&0&0&0 \\ \hline 0&0&0&0&0&1&0&0&0 \\ 0&0&0&0&0&1&0&0&0 \\ 0&0&0&0&0&1&0&0&0 \\ \hline 0&0&0&0&0&0&1&0&0 \\ 0&0&0&0&0&0&1&0&0 \\ 0&0&0&0&0&0&1&0&0 \end{array}\right], \quad \mathbf{W}^{(2)} = \left[\begin{array}{ccc|ccc|ccc} 0&0&1&0&0&0&0&0&0 \\ 0&0&1&0&0&0&0&0&0 \\ 0&0&1&0&0&0&0&0&0 \\ \hline 0&0&0&1&0&0&0&0&0 \\ 0&0&0&1&0&0&0&0&0 \\ 0&0&0&1&0&0&0&0&0 \\ \hline 0&0&0&0&0&0&0&1&0 \\ 0&0&0&0&0&0&0&1&0 \\ 0&0&0&0&0&0&0&1&0 \end{array}\right]
$$

Fig. 2. Example of matrix $\mathbf{W}^{(k)}$ with $0 \le k < 3$ for a 3×3 matrix $\mathbf{A}$

3.2 Homomorphic Generation of d-fold Replicated Diagonals

The main challenge at this point is the evaluation of the $\mathsf{repdiag}$ function in the encrypted domain. Our approach builds on the linear transformation introduced in Section 2.4.

We first define a $d^2 \times d^2$ matrix $\mathbf{W}^{(k)}$ that generates the replicated k-th diagonal as follows:

$$
\mathbf{W}^{(k)}_{i,j} = \begin{cases} 1 & \text{if } j = d \cdot \lfloor \frac{(i-1)}{d} \rfloor + 1 + \left(\lfloor \frac{(i-1)}{d} \rfloor + k \bmod d \right) \\ 0 & \text{otherwise.} \end{cases} \tag{5}
$$

An example of $\mathbf{W}^{(k)}$ with $0 \le k < 3$ for a 3×3 matrix $\mathbf{A}$ is illustrated in Figure 2.

Let $\mathbf{A}$ be a vector of length d^2, obtained by row-major order encoding of the $d \times d$ matrix. Then, the replicated diagonals can be obtained by evaluating

$$
\mathsf{repdiag}(\mathbf{A}, k) = \mathbf{W}^{(k)}\mathbf{A}.
$$

Let $\mathsf{Diag}(\mathbf{W}^{(k)}, l)$ denote the l-th diagonal of $\mathbf{W}^{(k)}$, which is a vector of length d^2. Then, the following holds:

$$
\mathsf{ct.repdiag}(\mathbf{A}, k) = \sum_{-d<l<d} \mathsf{Diag}(\mathbf{W}^{(k)}, l) \odot \mathsf{Rot}(\mathsf{ct.A}, l)
$$

where $\sum$ and $\odot$ denote shorthand notations for the Add and PMult operations, respectively.

For example, when $\mathsf{ct.A}$ is a ciphertext of vector $(a_{11}, a_{12}, a_{13}, a_{21}, a_{22}, a_{23}, a_{31}, a_{32}, a_{33})$ for a 3×3 matrix $\mathbf{A} = [a_{ij}]$, $\mathsf{ct.repdiag}(\mathbf{A}, k)$ is an encryption of length-9 vector

$$
\begin{aligned}
&(a_{11}, a_{11}, a_{11}, a_{22}, a_{22}, a_{22}, a_{33}, a_{33}, a_{33}) \ \text{ if } k = 0, \\
&(a_{12}, a_{12}, a_{12}, a_{23}, a_{23}, a_{23}, a_{31}, a_{31}, a_{31}) \ \text{ if } k = 1, \text{ and} \\
&(a_{13}, a_{13}, a_{13}, a_{21}, a_{21}, a_{21}, a_{32}, a_{32}, a_{32}) \ \text{ if } k = 2.
\end{aligned}
$$

This requires approximately $2d$ operations each of Add, PMult, and Rot when using the naive linear transformation since the matrix $\mathbf{W}^{(k)}$ in Eq. (5) has $2d - 1$ nonzero diagonals from Theorem 1. Thus,

evaluating $\mathsf{ct.repdiag}(\mathbf{A}, k)$ for all $0 \le k < d$ incurs a total of roughly $2d^2$ Add, PMult, and Rot operations each.

Theorem 1. *The matrix* $\mathbf{W}^{(k)}$ *defined in Eq.* (5) *has* $2d-1$ *nonzero diagonals.*

Proof. Let $\delta = j - i$ for $i, j \in \{1, \ldots, d^2\}$. Then, $(\delta \bmod d^2) \in \{0, \ldots, d^2 - 1\}$ indicates the diagonal on which the (i, j)-th entry lies.

The row index i can be expressed as

$$i = dp + q + 1 \quad \text{for} \quad p, q \in \{0, \ldots, d-1\}.$$

Then, $\lfloor \frac{(i-1)}{d} \rfloor = p$. Based on the condition that the entry equals 1, the indices of nonzero diagonals are computed as

$$\begin{aligned} \delta &= j - i \\ &= (dp + 1 + ((p+k) \bmod d)) - (dp + q + 1) \\ &= ((p+k) \bmod d) - q. \end{aligned}$$

Given a fixed k, we have $((p+k) \bmod d) \in \{0, 1, \ldots, d-1\}$. Thus, for $i \in \{1, \ldots, d^2\}$, δ must lie in the range $\{-d+1, \ldots, d-1\}$, and all values in this range are attainable. This implies that, among the d^2 possible diagonals, exactly $2d-1$ distinct diagonals contain nonzero entries for each k.

3.3 Optimization via Vectorized Linear Transformation

In this subsection, we introduce a vectorized method that enables efficient evaluation of repetitive linear transformation. Let $s = (N/2)/d^2$. The input encrypted $d \times d$ matrix $\mathsf{ct}.\mathbf{A}$ exhibits an internal structure corresponding to an s-fold replication of itself, as introduced in Section 2.3. In other words, $\mathsf{ct}.\mathbf{A}$ can be interpreted as a flattened encrypted vector $\mathsf{ct}.\mathbf{A}^{\exp}$ of length $(d^2 \cdot s)$,

$$\mathsf{ct}.\mathbf{A} \approx \mathsf{ct}.\mathbf{A}^{\exp} = \mathsf{ct}.\overbrace{\Big[\underbrace{\mathbf{A}}_{d^2}\ \mathbf{A} \cdots \mathbf{A}\Big]}^{d^2 \cdot s}$$

where the row-major flattened version of $\mathbf{A}$ (of length d^2) is repeated s times. This structure allows the Rot operation to be performed in a SIMD-like manner as follows:

$$\mathsf{Rot}(\mathsf{ct}.\mathbf{A}^{\exp}, l) = \mathsf{ct}.\big[\rho(\mathbf{A}, l) \cdots \rho(\mathbf{A}, l)\big] \tag{6}$$

where ρ denotes a one-dimensional circular shift applied to the row-major encoding of $\mathbf{A}$.

For clarity, we begin by considering the case $s = d$, and then generalize to arbitrary s. Recall that $\mathbf{W}^{(k)}$ contains $2d-1$ nonzero diagonals from Theorem 1, indexed by $-d < l = j - i < d$, independent of k. This allows us to vectorize the linear transformation for $\mathsf{repdiag}$ based on the d identical rotations of $\mathbf{A}$, as follows:

$$\mathsf{ct}.\begin{bmatrix} \mathsf{repdiag}(\mathbf{A}, 0) \\ \mathsf{repdiag}(\mathbf{A}, 1) \\ \vdots \\ \mathsf{repdiag}(\mathbf{A}, d-1) \end{bmatrix} = \sum_{-d<l<d} \begin{bmatrix} \mathsf{Diag}(\mathbf{W}^{(0)}, l) \\ \mathsf{Diag}(\mathbf{W}^{(1)}, l) \\ \vdots \\ \mathsf{Diag}(\mathbf{W}^{(d-1)}, l) \end{bmatrix} \odot \mathsf{ct}.\begin{bmatrix} \rho(\mathbf{A}, l) \\ \rho(\mathbf{A}, l) \\ \vdots \\ \rho(\mathbf{A}, l) \end{bmatrix}. \tag{7}$$

From Eq. (7), we can compute d instances of the $\mathsf{repdiag}$ function simultaneously with only $2d$ operations each of Add, PMult and Rot. We refer to this vectorized result (of length d^3) as $\mathbf{A}^{\mathrm{vec}}$. For brevity, we write the vectorized diagonals term as $\mathsf{Diags}(\{\mathbf{W}^{(i)}\}_{i=0}^{d-1}, l)$. So, Eq. (7) can be rewritten as

$$\mathsf{ct}.\mathbf{A}^{\mathrm{vec}} = \sum_{-d<l<d} \mathsf{Diags}(\{\mathbf{W}^{(i)}\}_{i=0}^{d-1}, l) \odot \mathsf{Rot}(\mathsf{ct}.\mathbf{A}^{\exp}, l). \tag{8}$$

Subsequently, by vectorizing the rotations of the matrix $\mathbf{B}$, we can complete the multiplication between $\mathbf{A}$ and $\mathbf{B}$. Specifically, we define a concatenated rotation as

$$\mathsf{ct}.\mathbf{B}^{\mathrm{rots}} = \mathsf{ct}.\left[\mathbf{B}\ \varphi(\mathbf{B},1)\ \cdots\ \varphi(\mathbf{B},d)\right],$$

where each $\varphi(\mathbf{B},i)$ denotes a circular upward rotation of every column vector in $\mathbf{B}$ by i positions, as defined in Eq. (4). The resulting ciphertext $\mathsf{ct}.\mathbf{B}^{\mathrm{rots}}$ contains d rotated matrices concatenated into a single ciphertext, occupying a total of d^3 slots. This can be computed entirely in the encrypted domain, given the input ciphertext $\mathsf{ct}.\mathbf{B}$ and masking vector. For $k = 0, \ldots, d^3-1$, the i-th masking vector mask_i is defined as

$$\mathsf{mask}_i[k] = \begin{cases} 1 & \text{if } i \cdot d^2 \le k < (i+1)\cdot d^2 \\ 0 & \text{otherwise.} \end{cases} \tag{9}$$

The detailed procedure is given in Algorithm 2. We note that Eq. (4) can be evaluated with a single homomorphic multiplication, given $\mathsf{ct}.\mathbf{A}^{\mathrm{vec}}$ and $\mathsf{ct}.\mathbf{B}^{\mathrm{rots}}$ generated by Algorithm 2.

Algorithm 2 VecRots($\mathsf{ct}.\mathbf{B}$)

Input: $\mathsf{ct}.\mathbf{B}$, which can be interpreted as a flattened encrypted vector $\mathsf{ct}.\mathbf{B}^{\mathrm{exp}}$
Output: $\mathsf{ct}.\mathbf{D}$, an encryption of $\mathbf{B}^{\mathrm{rots}}$

1: $\mathsf{ct}.\mathbf{B}^{\mathrm{rots}} \leftarrow \mathsf{ct}.\mathbf{0}$
2: **for** $l = 0$ to $d-1$ **do**
3: $\quad \mathsf{ct}.\mathbf{B}' \leftarrow \mathsf{Rot}(\mathsf{ct}.\mathbf{B}^{\mathrm{exp}}, d\cdot l)$
4: $\quad \mathsf{ct}.\mathbf{B}'' \leftarrow \mathsf{PMult}(\mathsf{ct}.\mathbf{B}', \mathsf{mask}_l)$
5: $\quad \mathsf{ct}.\mathbf{D} \leftarrow \mathsf{Add}(\mathsf{ct}.\mathbf{B}^{\mathrm{rots}}, \mathsf{ct}.\mathbf{B}'')$
6: **end for**
7: **return** $\mathsf{ct}.\mathbf{D}$

The following lemma and theorem show the correctness of Algorithm 2.

Lemma 1. *Let $\mathbf{B}$ be a $d \times d$ matrix encoded in row-major order. Then, for any integer $0 \le l < d$, the one-dimensional rotation $\rho(\mathbf{B}, d\cdot l)$ corresponds to the row-shifting $\varphi(\mathbf{B}, l)$.*

Proof. Let $\mathbf{B}^* \leftarrow \rho(\mathbf{B}, d\cdot l)$. Then, for all $i, j \in \{0,1,\ldots,d-1\}$, we have

$$\mathbf{B}^*[d\cdot i + j] = \mathbf{B}[(d\cdot(i+l)+j) \bmod d^2].$$

This implies that the entries of the $(i+l)$-th row of $\mathbf{B}$ are circularly shifted upward into the i-th row of $\mathbf{B}^*$, which corresponds exactly to the row-shifting operation $\varphi(\mathbf{B}, l)$.

Theorem 2. *Given a ciphertext $\mathsf{ct}.\mathbf{B}$ that can be interpreted as $\mathsf{ct}.\mathbf{B}^{\mathrm{exp}}$, Algorithm 2 correctly outputs an encryption of $\mathbf{B}^{\mathrm{rots}}$.*

Proof. We interpret $\mathsf{ct}.\mathbf{B}$ as its d-fold replicated form $\mathsf{ct}.\mathbf{B}^{\mathrm{exp}}$. By Eq. (6) and Lemma 1, we obtain

$$\begin{aligned} \mathsf{ct}.\mathbf{B}' &= \mathsf{ct}.\left[\rho(\mathbf{B}, d\cdot l)\ \cdots\ \rho(\mathbf{B}, d\cdot l)\right] \\ &= \mathsf{ct}.\left[\varphi(\mathbf{B}, l)\ \cdots\ \varphi(\mathbf{B}, l)\right]. \end{aligned}$$

Since PMult performs a component-wise multiplication on the underlying plaintext and mask_i is defined as in Eq. (9), we have

$$\mathsf{ct}.\mathbf{B}'' = \mathsf{ct}.\left[\mathbf{0}^{l\cdot d^2}\ \varphi(\mathbf{B}, l)\ \mathbf{0}^{(d-l-1)\cdot d^2}\right]$$

where $\mathbf{0}^{\ell}$ denotes a zero vector of length ℓ. By applying Add over $l = 0, \ldots, d-1$, we obtain

$$\mathsf{ct}.\mathbf{D} = \mathsf{ct}.\left[\varphi(\mathbf{B}, 0)\ \varphi(\mathbf{B}, 1)\ \cdots\ \varphi(\mathbf{B}, d-1)\right] = \mathsf{ct}.\mathbf{B}^{\mathrm{rots}}.$$

We now analyze the efficiency of Algorithm 2. A naive implementation requires approximately d Rot, PMult, and Add operations. However, by applying the baby-step giant-step optimization, the number of Rot operations can be reduced to around $2\sqrt{d}$. Specifically, let n_b be the number of baby-step rotations such that n_b divides d. The core idea is to reuse a set of precomputed rotations $\{\beta_j = \mathsf{Rot}(\mathsf{ct}.\mathbf{B}, d \cdot j)\}_{j=0}^{n_b-1}$. Using these, VecRots can be computed as

$$\sum_{i=0}^{(d/n_b)-1} \mathsf{Rot}\Big(\sum_{j=0}^{n_b-1} \beta_j \odot \rho(\mathsf{mask}_{n_b \cdot i+j}, -n_b \cdot d \cdot i), n_b \cdot d \cdot i \Big) \tag{10}$$

where $\sum$ and $\odot$ denote shorthand notations for the Add and PMult operations, respectively. The total number of Rot operations in Eq. (10) is $n_b + \frac{d}{n_b}$. By choosing $n_b \approx \sqrt{d}$, this cost is minimized to about $2\sqrt{d}$.

We proceed to present the optimized CC-MM algorithm, as detailed in Algorithm 3. In the description, the computation corresponding to Eq. (8) is denoted by VecRepDiag.

Algorithm 3 $\mathsf{MatMult}(\mathsf{ct}.\mathbf{A}, \mathsf{ct}.\mathbf{B})$: CC-MM when $s = d$

Input: $\mathsf{ct}.\mathbf{A}$ and $\mathsf{ct}.\mathbf{B}$, which can be interpreted as flattened encrypted vectors $\mathsf{ct}.\mathbf{A}^{\exp}$ and $\mathsf{ct}.\mathbf{B}^{\exp}$, respectively
Output: $\mathsf{ct}.\mathbf{C}$, where $\mathbf{C} = \mathbf{AB}$

1: $\mathsf{ct}.\mathbf{A}^{\mathrm{vec}} \leftarrow \mathsf{VecRepDiag}(\mathsf{ct}.\mathbf{A})$
2: $\mathsf{ct}.\mathbf{B}^{\mathrm{rots}} \leftarrow \mathsf{VecRots}(\mathsf{ct}.\mathbf{B})$
3: $\mathsf{ct}.\mathbf{C} \leftarrow \mathsf{Mult}(\mathsf{ct}.\mathbf{A}^{\mathrm{vec}}, \mathsf{ct}.\mathbf{B}^{\mathrm{rots}})$
4: **for** $i = 1$ to $\log_2 d$ **do** ▷ Matrix Folding Sum
5: $\quad \mathsf{ct}.\mathbf{C} \leftarrow \mathsf{Add}(\mathsf{ct}.\mathbf{C}, \mathsf{Rot}(\mathsf{ct}.\mathbf{C}, d^3/2^i))$
6: **end for**
7: **return** $\mathsf{ct}.\mathbf{C}$

The algorithm performs vectorized linear transformations on matrix $\mathbf{A}$ to replicate its diagonals, while simultaneously generating a vector of rotated versions of matrix $\mathbf{B}$. Using these two vectorized structures, the component-wise product can be computed via a single homomorphic multiplication. The following illustrates an example for the case $d = 3$:

$$\mathsf{ct}.\mathbf{C} \leftarrow \mathsf{Mult}(\mathsf{ct}.\begin{bmatrix} a_{11} & a_{11} & a_{11} \\ a_{22} & a_{22} & a_{22} \\ a_{33} & a_{33} & a_{33} \\ a_{12} & a_{12} & a_{12} \\ a_{23} & a_{23} & a_{23} \\ a_{31} & a_{31} & a_{31} \\ a_{13} & a_{13} & a_{13} \\ a_{21} & a_{21} & a_{21} \\ a_{32} & a_{32} & a_{32} \end{bmatrix}, \mathsf{ct}.\begin{bmatrix} b_{11} & b_{12} & b_{13} \\ b_{21} & b_{22} & b_{23} \\ b_{31} & b_{32} & b_{33} \\ b_{21} & b_{22} & b_{23} \\ b_{31} & b_{32} & b_{33} \\ b_{11} & b_{12} & b_{13} \\ b_{31} & b_{32} & b_{33} \\ b_{11} & b_{12} & b_{13} \\ b_{21} & b_{22} & b_{23} \end{bmatrix}).$$

The final result is obtained by summing the vector elements, an operation referred to as the matrix folding sum.

Theorem 3. *Given ciphertexts* $\mathsf{ct}.\mathbf{A}$ *and* $\mathsf{ct}.\mathbf{B}$*, which can be interpreted as* $\mathsf{ct}.\mathbf{A}^{\exp}$ *and* $\mathsf{ct}.\mathbf{B}^{\exp}$*, respectively, Algorithm 3 correctly outputs an encryption of* $\mathbf{C} = \mathbf{AB}$*.*

Proof. We begin by recalling Eq. (4):

$$\mathbf{C} = \mathbf{AB} = \sum_{0 \le i < d} \mathsf{repdiag}(\mathbf{A}, i) \odot \varphi(\mathbf{B}, i).$$

From Eq. (8) and the correctness of Algorithm 2, after Step 2, the ciphertexts $\mathsf{ct}.\mathbf{A}^{\mathrm{vec}}$ and $\mathsf{ct}.\mathbf{B}^{\mathrm{rots}}$ correspond to the following plaintext matrices:

$$\mathbf{A}^{\mathrm{vec}} = \left[\mathsf{repdiag}(\mathbf{A}, 0) \ \ldots \ \mathsf{repdiag}(\mathbf{A}, d-1)\right],$$
$$\mathbf{B}^{\mathrm{rots}} = \left[\varphi(\mathbf{B}, 0) \ \ldots \ \varphi(\mathbf{B}, d-1)\right].$$

Since Mult performs component-wise multiplication on the underlying plaintext vectors, Step 3 yields a ciphertext of

$$\mathbf{C} = \begin{bmatrix} \mathsf{repdiag}(\mathbf{A}, 0) \odot \varphi(\mathbf{B}, 0) \\ \vdots \\ \mathsf{repdiag}(\mathbf{A}, d-1) \odot \varphi(\mathbf{B}, d-1) \end{bmatrix}.$$

Finally, Steps 4 and 5 apply the matrix folding sum to aggregate the d blocks, each of length d^2, resulting in a ciphertext of

$$\left[\sum_{i=0}^{d-1} \mathsf{repdiag}(\mathbf{A}, i) \odot \varphi(\mathbf{B}, i)\right]^d,$$

which corresponds to a d-fold replication of $\mathbf{AB}$. Therefore, Algorithm 3 correctly outputs a ciphertext of $\mathbf{C} = \mathbf{AB}$ in the form of its d-fold replication.

While the naive evaluation of $\mathsf{VecRepDiag}$ requires $2d$ operations each of Add, PMult, and Rot, the baby-step giant-step optimization reduces the number of Rot operations to approximately $2\sqrt{2d}$, in a manner similar to that used in Algorithm 2. The details of this optimization for $\mathsf{VecRepDiag}$ will be discussed in the following subsection in a more general form. Consequently, Algorithm 3 requires approximately $3d + \log_2 d$ Add operations, $3d$ PMult operations, $2\sqrt{d} + 2\sqrt{2d} + \log_2 d$ Rot operations, and a single Mult operation.

3.4 Generalization of Our CC-MM Algorithm

The aforementioned algorithm requires d^3 slots. For large matrices where d^3 exceeds the maximum batch size $N/2$, the computation is divided into multiple batches. Specifically, we define $s = \min\{d,\ (N/2)/d^2\}$, where s denotes the number of blocks processed in parallel. The total number of required batches is then $B = d/s$.

Algorithm 2 and Eq. (8) can be generalized using a starting index i_s and an ending index $i_s + s$. Specifically,

$$\mathsf{ct}.\begin{bmatrix} \mathsf{repdiag}(\mathbf{A}, i_s) \\ \vdots \\ \mathsf{repdiag}(\mathbf{A}, i_s + s - 1) \end{bmatrix} \leftarrow \mathsf{VecRepDiag}(\mathsf{ct}.\mathbf{A}, i_s) \quad , \quad \mathsf{ct}.\begin{bmatrix} \varphi(\mathbf{B}, i_s) \\ \vdots \\ \varphi(\mathbf{B}, i_s + s - 1) \end{bmatrix} \leftarrow \mathsf{VecRots}(\mathsf{ct}.\mathbf{B}, i_s),$$

resulting in s-block structures occupying $d^2 \cdot s$ slots.

Our generalized CC-MM algorithm, applicable to matrix sizes satisfying $d^2 \leq N/2$, is described in Algorithm 4.

Theorem 4. *Given ciphertexts* $\mathsf{ct}.\mathbf{A}$ *and* $\mathsf{ct}.\mathbf{B}$*, which can be interpreted as* $\mathsf{ct}.\mathbf{A}^{\mathrm{exp}}$ *and* $\mathsf{ct}.\mathbf{B}^{\mathrm{exp}}$*, respectively, Algorithm 4 correctly outputs an encryption of* $\mathbf{C} = \mathbf{AB}$*.*

Proof. For the iteration index $i = 0, \ldots, B-1$, the ciphertext resulting from $\mathsf{Mult}(\mathsf{ct}.\mathbf{A}^{\mathrm{vec}}, \mathsf{ct}.\mathbf{B}^{\mathrm{rots}})$ in Step 6 corresponds to the plaintext matrix

$$\mathbf{C}^{(i)} = \begin{bmatrix} \mathsf{repdiag}(\mathbf{A}, i \cdot s) \odot \varphi(\mathbf{B}, i \cdot s) \\ \vdots \\ \mathsf{repdiag}(\mathbf{A}, i \cdot s + s - 1) \odot \varphi(\mathbf{B}, i \cdot s + s - 1) \end{bmatrix},$$

Algorithm 4 $\mathsf{MatMult}(\mathsf{ct}.\mathbf{A}, \mathsf{ct}.\mathbf{B})$: Generalized CC-MM

Input: $\mathsf{ct}.\mathbf{A}$ and $\mathsf{ct}.\mathbf{B}$, which can be interpreted as flattened encrypted vectors $\mathsf{ct}.\mathbf{A}^{\exp}$ and $\mathsf{ct}.\mathbf{B}^{\exp}$, respectively
Output: $\mathsf{ct}.\mathbf{C}$, where $\mathbf{C} = \mathbf{AB}$

$s = \min\{d,\ (N/2)/d^2\}$, $B = d/s$
$\mathsf{ct}.\mathbf{C} \leftarrow \mathbf{0}$
for $i = 0$ to $B - 1$ **do**
    $\mathsf{ct}.\mathbf{A}^{\mathrm{vec}} \leftarrow \mathsf{VecRepDiag}(\mathsf{ct}.\mathbf{A}, i \cdot s)$
    $\mathsf{ct}.\mathbf{B}^{\mathrm{rots}} \leftarrow \mathsf{VecRots}(\mathsf{ct}.\mathbf{B}, i \cdot s)$
    $\mathsf{ct}.\mathbf{C} \leftarrow \mathsf{Add}(\mathsf{ct}.\mathbf{C}, \mathsf{Mult}(\mathsf{ct}.\mathbf{A}^{\mathrm{vec}}, \mathsf{ct}.\mathbf{B}^{\mathrm{rots}}))$
end for
for $i = 1$ to $\log_2 s$ **do** ▷ Matrix Folding Sum
    $\mathsf{ct}.\mathbf{C} \leftarrow \mathsf{Add}(\mathsf{ct}.\mathbf{C}, \mathsf{Rot}(\mathsf{ct}.\mathbf{C}, (d^2 \cdot s)/2^i))$
end for
return $\mathsf{ct}.\mathbf{C}$

based on the correctness of the VecRepDiag and VecRots evaluations. After B iterations, the aggregated ciphertext $\mathsf{ct}.\mathbf{C}$ is updated to a ciphertext of

$$\mathbf{C} = \begin{bmatrix} \sum_{i=0}^{B-1} \mathsf{repdiag}(\mathbf{A}, i \cdot s) \odot \varphi(\mathbf{B}, i \cdot s) \\ \vdots \\ \sum_{i=0}^{B-1} \mathsf{repdiag}(\mathbf{A}, i \cdot s + s - 1) \odot \varphi(\mathbf{B}, i \cdot s + s - 1) \end{bmatrix}.$$

The matrix folding sum is then applied to aggregate the s blocks, updating $\mathsf{ct}.\mathbf{C}$ to the ciphertext of plaintext matrix

$$\begin{aligned} \mathbf{C} &= \left[\sum_{i=0}^{B-1} \sum_{j=0}^{s-1} \mathsf{repdiag}(\mathbf{A}, i \cdot s + j) \odot \varphi(\mathbf{B}, i \cdot s + j)\right]^s \\ &= \left[\sum_{i=0}^{d-1} \mathsf{repdiag}(\mathbf{A}, i) \odot \varphi(\mathbf{B}, i)\right]^s, \end{aligned}$$

which corresponds to the s-fold replicated form of $\mathbf{AB}$. Therefore, Algorithm 4 correctly outputs a ciphertext of $\mathbf{C} = \mathbf{AB}$ in the form of its s-fold replication.

We note that Algorithm 4 makes B calls to VecRepDiag. To further optimize this, we present a baby-step giant-step method tailored for the multiple-batch setting. Let n_b be the number of baby-step rotations, and define the set $\{\alpha_j = \mathsf{Rot}(\mathbf{A}^{\exp}, j)\}_{j=0}^{n_b-1}$. Then, $\mathsf{VecRepDiag}(\mathsf{ct}.\mathbf{A}, i_s)$ can be reformulated as

$$\sum_{-d/n_b < i < d/n_b} \mathsf{Rot}\Big(\sum_{j=0}^{n_b-1} \gamma_{i,j} \odot \alpha_j, n_b \cdot i\Big),$$

where $\gamma_{i,j} = \rho(\mathsf{Diags}(\{\mathbf{W}^{(k)}\}_{k=i_s}^{i_s+s-1}, n_b \cdot i + j), -n_b \cdot i)$, and $\sum$ and $\odot$ represent Add and PMult operations, respectively.

It is worth noting that the rotations α_j can be reused across all B iterations, where $i_s = 0, s, \ldots, s(B-1)$. Thus, the total number of Rot operations is approximately $n_b + B \cdot (2d/n_b)$.

Theorem 5. *The optimal number of* Rot *operations required for evaluating* VecRepDiag *in Algorithm 4 is* $\lceil 2\sqrt{2Bd} \rceil$*, where* $B = d/s$ *and* $s = \min\{d,\ (N/2)/d^2\}$ *with* N *denoting the ring dimension of the underlying FHE scheme.*

Proof. For positive integers n_b and d, the expression $n_b + B \cdot 2d/n_b$ is minimized when $n_b = \sqrt{2Bd}$, according to the arithmetic–geometric mean inequality. Therefore, the optimal number of Rot operations is $\lceil 2\sqrt{2Bd} \rceil$.

Consequently, our CC-MM algorithm performs B calls each to VecRots and VecRotDiag, along with B Mult operations, $B + \log_2 s$ Add operations, and $\log_2 s$ Rot operations. A more detailed analysis is provided in Table 1.

3.5 Improved Matrix Inversion Algorithm

In this subsection, we present an improved C-MI algorithm that builds on the approach of Ahn et al. [1] and leverages our proposed CC-MM algorithm. Before describing the full C-MI algorithm, we introduce the relevant sub-algorithms.

Matrix Transposition. We employ the ciphertext matrix transposition (C-MT) algorithm proposed in [3]. Specifically, $\mathsf{MatTrans}(\mathsf{ct}.\mathbf{A})$ takes as input a ciphertext of a $d \times d$ matrix $\mathbf{A}$, and returns a ciphertext of its transpose $\mathbf{A}^{\mathrm{T}}$.

Trace Evaluation. Computing the trace of a matrix involves two steps: (1) extracting the main diagonal and (2) summing all values in the corresponding slots. To extract the main diagonal, we use the masking vector $\mathsf{mask}_{\mathrm{Tr}}$ defined as follows: for $k = 0, \ldots, d^2 - 1$,

$$\mathsf{mask}_{\mathrm{Tr}}[k] = \begin{cases} 1 & \text{if } k \equiv 0 \mod (d+1) \\ 0 & \text{otherwise.} \end{cases}$$

We then define the $\mathsf{EvalSum}$ algorithm, which computes the sum of all slot values in a ciphertext. Given a ciphertext c of message $(m_1, \ldots, m_n)$, the algorithm $\mathsf{EvalSum}(c)$ returns a ciphertext of $(\sum_{i=1}^{n} m_i, \ldots, \sum_{i=1}^{n} m_i)$, which can be efficiently implemented using Rot and Add operations.

Algorithm 5 outputs a ciphertext in which all slots contain the trace of $\mathbf{A}$, denoted as $\mathrm{Tr}(\mathbf{A})$.

Algorithm 5 $\mathsf{EvalTrace}(\mathsf{ct}.\mathbf{A})$

Input: $\mathsf{ct}.\mathbf{A}$ which is a ciphertext of a $d \times d$ matrix $\mathbf{A}$
Output: $\mathsf{ct}.\mathbf{T}$, which is a ciphertext in which all slots contain $\mathrm{Tr}(\mathbf{A})$
1: $\mathsf{ct}.\mathbf{A}' \leftarrow \mathsf{PMult}(\mathsf{ct}.\mathbf{A}, \mathsf{mask}_{\mathrm{Tr}})$
2: $\mathsf{ct}.\mathbf{T} \leftarrow \mathsf{EvalSum}(\mathsf{ct}.\mathbf{A}')$
3: **return** $\mathsf{ct}.\mathbf{T}$

Theorem 6. *Given a ciphertext* $\mathsf{ct}.\mathbf{A}$ *encrypting a* $d \times d$ *matrix* $\mathbf{A}$*, Algorithm 5 correctly outputs a ciphertext in which every slot holds the trace value* $Tr(\mathbf{A})$*.*

Proof. For $\mathbf{A} = \left[a_{ij}\right]$, the trace is the sum of the main diagonal entries where $i = j$. When $\mathbf{A}$ is encoded in row-major order, the main diagonal entries are located at positions

$$\mathbf{A}[d \cdot i + j] = \mathbf{A}[d \cdot i + i] = \mathbf{A}[i \cdot (d+1)]$$

for $i, j \in \{0, 1, \ldots, d-1\}$. The vector $\mathsf{mask}_{\mathrm{Tr}}$ selects these positions, so applying PMult preserves the main diagonal and zeroes out the rest. Subsequent application of $\mathsf{EvalSum}$ then aggregates the nonzero slots, resulting in a ciphertext where each slot contains the trace value $\mathrm{Tr}(\mathbf{A})$.

Homomorphic Division. We denote the division algorithm by $\mathsf{EvalDivide}$. Let c_1 and c_2 be ciphertexts of messages $(a_1, \ldots, a_d)$ and $(b_1, \ldots, b_d)$, respectively. Then, $\mathsf{EvalDivide}(c_1, c_2)$ returns a ciphertext of the component-wise quotient $(\frac{a_1}{b_1}, \ldots, \frac{a_d}{b_d})$. This operation can be implemented by approximating the reciprocal function $1/x$ using Chebyshev polynomials.

Building upon the aforementioned sub-algorithms, the complete C-MI algorithm is presented in Algorithm 6. Here, $\mathbf{I}_d$ denotes the $d \times d$ plaintext identity matrix. We denote the homomorphic subtraction operation by Sub, which performs component-wise subtraction on the underlying plaintext vectors.

Our improved C-MI algorithm adopts the iterative framework of [1], while replacing the sub-algorithms with matrix-packed variants. This substitution significantly reduces the number of required ciphertexts and Mult operations, at the cost of increased circuit depth in the CC-MM step. The algorithm performs r iterations of CC-MM and outputs the ciphertext of $\mathbf{Y}_r$, which approximates the inverse of the input matrix $\mathbf{A}$. The correctness of Algorithm 6 directly follows from that of the original framework proposed by Ahn et al. [1], along with the correctness of the underlying sub-algorithms employed in our construction. We refer readers to [1] for a detailed convergence analysis.

Algorithm 6 $\mathsf{MatInv}(\mathsf{ct}.\mathbf{A}, r)$: Improved C-MI

Input: $\mathsf{ct}.\mathbf{A}$ and r, where $\mathsf{ct}.\mathbf{A}$ is a ciphertext of $d \times d$ matrix $\mathbf{A}$ and r is the number of iterations
Output: $\mathsf{ct}.\mathbf{Y}_r$ which is an encryption of $\mathbf{A}^{-1}$

$\mathsf{ct}.\mathbf{A}^{\mathrm{T}} \leftarrow \mathsf{MatTrans}(\mathsf{ct}.\mathbf{A})$
$\mathsf{ct}.\mathbf{A}\mathbf{A}^{\mathrm{T}} \leftarrow \mathsf{MatMult}(\mathsf{ct}.\mathbf{A}, \mathsf{ct}.\mathbf{A}^{\mathrm{T}})$
$\mathsf{ct}.\mathbf{Tr} \leftarrow \mathsf{EvalTrace}(\mathsf{ct}.\mathbf{A}\mathbf{A}^{\mathrm{T}})$
$\mathsf{ct}.\mathbf{Y}_0 \leftarrow \mathsf{EvalDivide}(\mathsf{ct}.\mathbf{A}^{\mathrm{T}}, \mathsf{ct}.\mathbf{Tr})$
$\mathsf{ct}.\mathbf{A}_0 \leftarrow \mathsf{Sub}(\mathbf{I}_d, \mathsf{EvalDivide}(\mathsf{ct}.\mathbf{A}\mathbf{A}^{\mathrm{T}}, \mathsf{ct}.\mathbf{Tr}))$
for $i = 1$ to r **do**
    $\mathsf{ct}.\mathbf{Y}_i \leftarrow \mathsf{MatMult}(\mathsf{ct}.\mathbf{Y}_{i-1}, \mathsf{Add}(\mathbf{I}_d, \mathsf{ct}.\mathbf{A}_{i-1}))$
    $\mathsf{ct}.\mathbf{A}_i \leftarrow \mathsf{MatMult}(\mathsf{ct}.\mathbf{A}_{i-1}, \mathsf{ct}.\mathbf{A}_{i-1})$
end for
return $\mathsf{ct}.\mathbf{Y}_r$

4 Efficiency Comparison

Our improvement stems from the use of a CC-MM method optimized for repeated matrix multiplications. Since alternative CC-MM methods can also be employed within Algorithm 6, this section summarizes such alternatives and compares their efficiency. We first analyze the complexity of CC-MM methods, followed by experimental results with concrete performance comparisons.

4.1 Complexity Comparison with Existing CC-MM Methods

The naive approach adopted in [1] encrypts each matrix entry into a separate ciphertext, enabling CC-MM with a single circuit depth. However, this method incurs significant resource overhead, requiring $O(d^2)$ ciphertexts and $O(d^3)$ Mult operations.

Single-depth CC-MM can also be achieved using vector-wise packing, which reduces the number of required ciphertexts to $O(d)$. As a baseline for comparison, we introduce a diagonal-based method–denoted as DP in this paper–that encrypts the diagonal vectors of the input matrices into separate ciphertexts. The detailed algorithm is provided in Appendix A.

The main baselines considered in this paper utilize matrix-packed ciphertexts. The method proposed in [3] is a widely used CC-MM technique based on linear transformations, but it requires a circuit depth of 3, making it unsuitable for chained multiplications. The algorithm in [7] reduces the depth to 2 but requires a batch size of d^3. To overcome this limitation, the authors of [8] extended the method of [7] to support larger matrices with $d^3 > N/2$ by introducing a matrix duplication technique that improves overall efficiency. Nevertheless, this duplication mechanism relies on zero-padded ciphertexts, while the resulting outputs are not zero-padded, hindering their direct use in successive CC-MM operations. In contrast, our proposed algorithm achieves a depth of 2 using only one Mult and one PMult, and is fully compatible with chained multiplications.

Table 1 summarizes the computational complexity, the number of required ciphertexts, and circuit depth of various CC-MM techniques. For clarity and brevity, constant terms in operation counts are omitted. In the table, s denotes the number of matrices that can be packed into a single ciphertext, evaluated as $s = \min\{d, (N/2)/d^2\}$.

We now compare the relatively expensive operations, namely Rot and Mult, which involve key-switching. When $s = d$, our method requires fewer operations and lower circuit depth than the algorithm of [3]. For example, under the CKKS scheme with a ring dimension of 2^{16} (i.e., 2^{15} available slots), the condition $s = d$ holds for matrix sizes up to $d \leq 32$. The approach in [7] is applicable only when $s = d$, and under this setting, our method incurs fewer operations while maintaining the same circuit depth. Compared to the algorithm in [8], our method has comparable complexity for small values of d, but its complexity increases with larger matrix sizes. Nonetheless, in iterative multiplication scenarios, our method remains advantageous over [8] since it avoid one PMult per iteration, which would otherwise be required to convert intermediate results into a zero-padded format for subsequent CC-MM computations.

Table 1. Complexity Comparison of Matrix Multiplication Algorithms for Encrypted $d \times d$ Matrices

	Naive	DP	[3]	[7]	[8]	Ours†
Rot	–	d^2	$3d + 5\sqrt{d}$	$2d + 3\log_2 d$	$\frac{2d}{s}(\log_2 d + 1) + 3\log_2 s$	$\boxed{2\sqrt{d}}^{(a)} + \boxed{\frac{2\sqrt{2}d}{\sqrt{s}}}^{(b)} + \boxed{\log_2 s}^{(c)}$
PMult	–	–	$4d$	$2d$	$\frac{2d}{s}$	$\boxed{d}^{(a)} + \boxed{\frac{2d^2}{s}}^{(b)}$
Add	$d^2(d-1)$	$d(d-1)$	$6d$	$2d + 3\log_2 d$	$\frac{d}{s}(2\log_2 d + 1) + 3\log_2 s$	$\boxed{d}^{(a)} + \boxed{\frac{2d^2}{s}}^{(b)} + \boxed{\log_2 s}^{(c)} + \frac{d}{s}$
Mult	d^3	d^2	d	1	d/s	d/s
Required Ctxt	$2d^2$	$2d$	2	2	2	2
Required Depth	1Mult	1Mult	1Mult + 2PMult	1Mult + 1PMult	1Mult + 1PMult	1Mult + 1PMult

$^{*}s = \min\{d,\ (N/2)/d^2\}$ where N is a ring dimension of the underlying FHE scheme

$^{\dagger}(a)$ cost incurred by VecRots, (b) cost incurred by VecRotDiag, (c) cost incurred by the matrix folding sum

On the other hand, when $s < d$, our method loses its advantage due to increased ciphertext overhead. Specifically, the required batch size of d^3 exceeds the number of available slot capacity of a single ciphertext when $s < d$, i.e., $d^3 > (N/2)$, resulting in reduced efficiency. This explains why our method achieves the best performance up to $d = 32$ (where $d^3 \leq (N/2)$), but its efficiency degrades for $d = 64$ (where $d^3 > N/2$), in Table 2. See Section 4.2 for further details.

4.2 Concrete Performance Comparison

In this subsection, we implement the CC-MM algorithms listed in Section 4.1 and compare their empirical performance through comparative analysis.

Experimental Setting and Implementation Details In our implementation[1], we employed the OpenFHE library [32] to instantiate FHE schemes. We conducted all experiments on a virtual machine running Ubuntu 20.04, hosted on a system consisting of an Intel(R) Xeon(R) Silver 4215R CPU at 3.2 GHz and 128 GB RAM, operating on Windows Server 2019. All parameters in our experiments were selected for achieving 128-bit security.

The implementation details are summarized as follows:

- When implementing the algorithm in [7] for the cases when $d^3 > N/2$, we employed the Strassen algorithm [33] to partition the input matrix and pack the resulting submatrices into separate ciphertexts.
- For the algorithm in [8], we artificially zeroed out the non-padding regions after each multiplication by applying an additional PMult operation to maintain the zero-padding structure required for consecutive matrix multiplications.
- The baby-step giant-step parameters of our proposed algorithm were configured in accordance with Theorem 5.
- All algorithms used power-of-2 rotation keys with binary decomposition to enable arbitrary rotations.

Comparison of Running Time We conducted two primary experiments: matrix squaring and matrix inversion over encrypted inputs. Additionally, detailed performance benchmarks for single CC-MM are provided in Appendix B.

Matrix Squaring Experiment. We measured the execution time of 15 consecutive ciphertext matrix squaring operations. To this end, the initial ciphertext level was configured to accommodate the circuit depth required for 15 rounds of CC-MM. The scaling factor Δ was set to 2^{50} for all methods.

Table 2 presents the results. When $d = 4$, the DP method achieves the best performance, primarily due to its smaller ring dimension and ciphertext modulus. For $d = 8, 16, 32$, our proposed algorithm outperforms the alternatives, demonstrating superior efficiency in scenarios involving chained multiplications. However, for $d = 64$, the method from [8] yields better performance despite its higher circuit depth, owing to its more favorable complexity for large d.

[1] Code available at `https://github.com/oksuman/matrix-mult-fhe`

Table 2. Running Time of 15 matrix squaring with respect to d (unit: sec)

d	DP	[3]	[7]	[8]	Ours
4	**73**	289	190	200	109
8	347	741	511	401	**190**
16	1625	1447	1310	646	**325**
32	7348	3127	3254	942	**627**
64	33248	6453	21343‡	**1425**	3056
N	2^{16}	2^{17}	2^{17}	2^{17}	2^{17}
Δ	2^{50}	2^{50}	2^{50}	2^{50}	2^{50}
L	15	45	30	45	30

‡ Using Strassen's algorithm with reduction to 32×32 matrices

Matrix Inversion Experiment. We evaluated C-MI for matrix dimensions ranging from $d = 4$ to 64, with the number of iterations (r) increasing proportionally with d. For each dimension, we analyzed $10,000$ randomly generated matrices and recorded the number of iterations required to achieve convergence, defined as reaching an element-wise error below 10^{-4} in at least 95% of the cases. The corresponding r values used for each d are listed in Table 3.

Table 3. The number of iterations for convergence with regard to d

Matrix Dimension (d)	4	8	16	32	64
Iterations (r)	18	21	25	28	31

Table 4. Iterative Matrix Inversion over Encrypted Matrix (unit: sec)

Algorithm		Matrix Dimension (d)				
		4	8	16	32	64
Naive	Δ	2^{50}	2^{50}	-	-	-
	L	27	30	-	-	-
	Time	900	8975	-	-	-
DP	Δ	2^{50}	2^{50}	2^{50}	-	-
	L	27	30	34	-	-
	Time	413	2399	15283	-	-
[3]	Δ	2^{59}	2^{59}	2^{59}	2^{59}	2^{59}
	L	29	29	29	29	29
	Time	2376	3314	4909	7804	13578
[7]	Δ	2^{50}	2^{59}	2^{59}	2^{59}	-
	L	48	29	29	29	-
	Time	584	1690	3318	7190	-
[8]	Δ	2^{59}	2^{59}	2^{59}	2^{59}	2^{59}
	L	29	29	29	29	29
	Time	2268	2898	3770	4734	**6164**
Ours	Δ	2^{50}	2^{59}	2^{59}	2^{59}	2^{59}
	L	48	29	29	29	29
	Time	**375**	**1305**	**2004**	**3304**	11680

The total circuit depth for C-MI is determined by r times the CC-MM depth, with additional depth required for C-MT and trace evaluation. For comparison with existing methods, we also implemented a naive CC-MM approach in which each matrix entry is packed into a separate ciphertext. All experiments were conducted using a ring dimension of $N = 2^{17}$. Bootstrapping was applied when necessary, and for such cases, the scaling factor was set to $\Delta = 2^{59}$ to ensure adequate numerical precision.

As shown in Table 4, our method extends the applicability of C-MI to larger matrix dimensions d, which were practically infeasible in [1], while also offering improved performance for smaller values of d. In particular, our CC-MM technique yields the most efficient results for $d = 4, 8, 16$, and 32, benefiting from its lower computational complexity and reduced circuit depth, as discussed in earlier section. For $d = 64$, however, the method based on the CC-MM technique from [8] achieves superior performance.

5 Application To Privacy-Preserving Linear Regression

In this section, we illustrate the application of C-MI, with a focus on privacy-preserving linear regression. Specifically, we consider the secure outsourcing of linear regression model training. We first define our system model and describe the training procedure in detail. Finally, we compare the execution times of outsourced training using our proposed CC-MM technique and that of [8], both employed as subroutines within the C-MI framework.

5.1 System Model

Securely outsourced training addresses two critical challenges in modern machine learning: the computational burden of model training and the need to utilize large-scale datasets while preserving data privacy. To enable secure delegation of such computation-intensive tasks, numerous works have explored FHE [18, 34–36] for privacy-preserving model training, including linear regression [9, 15, 37, 38].

Our system model for privacy-preserving linear regression follows the framework described in [39–42], as illustrated in Figure 3. In this paper, we focus on the single-key scenario, where a single data owner ($n = 1$) serves as both the data provider and decryptor. For collaborative training scenarios involving multiple data owners ($n > 1$), threshold FHE schemes [43–46] can be adopted to enable secure multiparty computation.

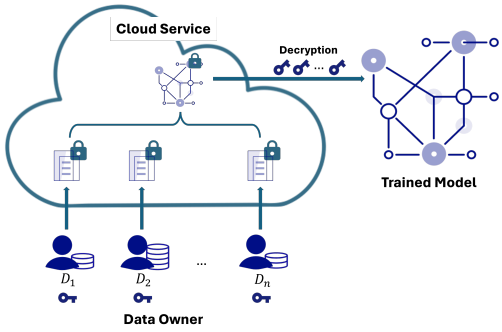

Fig. 3. Our System Model: Data owners $(D_1, \ldots, D_n)$ provide training datasets encrypted with either their individual public keys or a shared group public key. Model training is performed in the cloud service, where all model parameters remain encrypted throughout the computation. Upon completion, the trained model is decrypted by an authorized decryptor holding the corresponding private keys.

5.2 Secure Outsourced Training of Linear Regression

Let $\tilde{\mathbf{x}} \in \mathbb{R}^n$ be a feature vector, where n is the number of features. Given a model parameter $\mathbf{w} \in \mathbb{R}^n$, the linear regression model predicts the output $\tilde{\mathbf{y}} = \tilde{\mathbf{x}}^{\mathrm{T}}\mathbf{w}$, where $\tilde{\mathbf{y}} \in \mathbb{R}$. To train $\mathbf{w}$, the training feature matrix $\mathbf{X} \in \mathbb{R}^{s \times n}$ and the corresponding label vector $\mathbf{y} \in \mathbb{R}^s$ are required, where s is the number of training samples. If the matrix $\mathbf{X}^{\mathrm{T}}\mathbf{X}$ is invertible, $\mathbf{w}$ can be trained as $\mathbf{w}^* = (\mathbf{X}^{\mathrm{T}}\mathbf{X})^{-1}\mathbf{X}^{\mathrm{T}}\mathbf{y}$ [47].

Algorithm 7 describes the procedure for training a linear regression model over encrypted datasets. The Combine function homomorphically aggregates horizontally partitioned data from multiple data owners into a single encrypted dataset. The Rebatch function reshapes the intermediate result into a row-major packed ciphertext, adjusting the number of slots. Note that the required batch size changes from $s \times n$ to $s \times s$. Finally, the MatVecMult function performs homomorphic matrix-vector multiplication using matrix-packed ciphertexts.

We omit the implementation details of these auxiliary functions and instead focus on the C-MI algorithms, which constitute the primary computational bottleneck in the secure training workflow.

Algorithm 7 Secure Training of Linear Regression

Input: $\mathsf{Enc}(\mathbf{X}_1), \ldots, \mathsf{Enc}(\mathbf{X}_n), \mathsf{Enc}(\mathbf{y}_1), \ldots, \mathsf{Enc}(\mathbf{y}_n)$, where each $\mathbf{X}_i$ and $\mathbf{y}_i$ are training dataset from D_i.
Output: $\mathsf{Enc}(\mathbf{w}^*)$

Step 0:
$\mathsf{Enc}(\mathbf{X}) \leftarrow \mathsf{Combine}(\mathbf{X}_1, \ldots, \mathbf{X}_n)$
$\mathsf{Enc}(\mathbf{y}) \leftarrow \mathsf{Combine}(\mathbf{y}_1, \ldots, \mathbf{y}_n)$

Step 1:
$\mathsf{Enc}(\mathbf{X}^{\mathrm{T}}) \leftarrow \mathsf{MatTrans}(\mathsf{Enc}(\mathbf{X}))$
$\mathsf{Enc}(\mathbf{X}^{\mathrm{T}}\mathbf{X}) \leftarrow \mathsf{MatMult}(\mathsf{Enc}(\mathbf{X}^{\mathrm{T}}), \mathsf{Enc}(\mathbf{X}))$
$\mathsf{Rebatch}(\mathsf{Enc}(\mathbf{X}^{\mathrm{T}}\mathbf{X}))$

Step 2:
$\mathsf{Enc}((\mathbf{X}^{\mathrm{T}}\mathbf{X})^{-1}) \leftarrow \mathsf{MatInv}(\mathsf{Enc}(\mathbf{X}^{\mathrm{T}}\mathbf{X}))$

Step 3:
$\mathsf{Enc}(\mathbf{X}^{\mathrm{T}}\mathbf{y}) \leftarrow \mathsf{MatVecMult}(\mathsf{Enc}(\mathbf{X}^{\mathrm{T}}), \mathsf{Enc}(\mathbf{y}))$

Step 4:
$\mathsf{Enc}(\mathbf{w}^*) \leftarrow \mathsf{MatVecMult}(\mathsf{Enc}((\mathbf{X}^{\mathrm{T}}\mathbf{X})^{-1}), \mathsf{Enc}(\mathbf{X}^{\mathrm{T}}\mathbf{y}))$

return $\mathsf{Enc}(\mathbf{w}^*)$

5.3 Performance Comparison

In our experiment, we used a Kaggle dataset comprising 8 features related to university admission outcomes[2]. Since the experiment was conducted under the single-key scenario, **Step 0** of the training procedure was excluded. A total of 64 data samples were encrypted and used for model training.

Table 5. Running times of linear regression training (unit: sec)

	Step 1	**Step 2**	**Step 3**	**Step 4**	**Total**	**MSE**
[8]	801	1660	647	6	3114	0.030
Ours	1018	1109	634	6	2767	0.006

[2] `https://www.kaggle.com/datasets/mohansacharya/graduate-admissions`

Table 5 presents the execution times for **Steps 1–4**. For comparison, we selected the CC-MM algorithm proposed in [8], which demonstrated the best overall performance among the baseline algorithms. The approach in [7] was excluded, since it cannot efficiently handle 64×64 matrices under the chosen FHE parameter setting, which adheres to the conventional 128-bit security level. The benchmark was conducted with parameters $\Delta = 2^{59}$, $L = 31$, and $N = 2^{17}$.

Our improved C-MI, which leverages our CC-MM, achieved lower execution time in **Step 2**, enhancing the overall training efficiency. In particular, **Step 2** involves the inversion of an 8×8 matrix, where our method significantly outperformed the baseline approach.

However, **Step 1** incurs relatively high latency, since it involves a 64×64 CC-MM under a large ciphertext modulus. Since our CC-MM method incurs greater overhead for $d = 64$, its performance was inferior to that of the baseline in this step. This limitation could be mitigated by adopting the CC-MM technique from [8] specifically for large-d operations.

After training, we evaluated inference using a separate set of 64 samples. The resulting mean squared error (MSE) was approximately 0.006 with our method, compared to 0.03 with the baseline [8]. It is worth noting that the CC-MM algorithm from [8] requires three circuit depths per iteration in C-MI, leading to more frequent bootstrapping and consequently higher bootstrapping-related errors during training.

6 Conclusion

In this paper, we presented an enhanced C-MI algorithm based on the work of Ahn et al. [1], which computes an approximate matrix inverse over encrypted input matrices. While the original method [1] introduced a novel FHE-friendly approach for matrix inversion, it is assumed that each matrix entry was encrypted in a separate ciphertext. Although optimal in terms of circuit depth for matrix multiplication, this design incurs substantial memory overhead due to the large number of ciphertexts required.

To overcome this limitation, we extended Ahn et al.'s algorithm to support matrix-packed ciphertexts by developing a new CC-MM algorithm tailored for this setting. Our proposed CC-MM method operates with a circuit depth of only 2 and is particularly well-suited for iterative matrix multiplications, a core requirement in C-MI. The key idea behind our CC-MM construction is a vectorized linear transformation that minimizes the number of homomorphic operations, especially when large batch sizes are available. Through comprehensive experiments, we demonstrated that our method outperforms existing CC-MM approaches for matrix dimensions $d \leq 32$ when used as a subroutine in C-MI, making it particularly effective for low-dimensional settings. As a representative application, we applied our improved C-MI to privacy-preserving linear regression training, where it achieved superior performance compared to state-of-the-art alternatives. These results confirm the practical relevance of our approach, especially for machine learning tasks involving 10 to 30 features–common in domains such as healthcare and finance.

References

1. T. M. Ahn, K. H. Lee, J. S. Yoo, and J. W. Yoon, "Cheap and fast iterative matrix inverse in encrypted domain," in *ESORICS 2023, Part I*, ser. LNCS, G. Tsudik, M. Conti, K. Liang, and G. Smaragdakis, Eds., vol. 14344. Springer, 2023, pp. 334–352.
2. N. P. Smart and F. Vercauteren, "Fully homomorphic SIMD operations," *Des. Codes Cryptogr.*, vol. 71, no. 1, pp. 57–81, 2014.
3. X. Jiang, M. Kim, K. E. Lauter, and Y. Song, "Secure outsourced matrix computation and application to neural networks," in *ACM CCS 2018*, D. Lie, M. Mannan, M. Backes, and X. Wang, Eds. ACM, 2018, pp. 1209–1222.
4. S. Halevi and V. Shoup, "Algorithms in HElib," in *CRYPTO 2014, Part I*, ser. LNCS, J. A. Garay and R. Gennaro, Eds., vol. 8616. Springer, 2014, pp. 554–571.
5. ——, "Faster homomorphic linear transformations in HElib," in *CRYPTO 2018, Part I*, ser. LNCS, H. Shacham and A. Boldyreva, Eds., vol. 10991. Springer, 2018, pp. 93–120.
6. ——, "Design and implementation of HElib: a homomorphic encryption library," *IACR Cryptol. ePrint Arch.*, p. 1481, 2020.

7. P. Rizomiliotis and A. Triakosia, "On matrix multiplication with homomorphic encryption," in *Proceedings of the 2022 on Cloud Computing Security Workshop, CCSW 2022*, F. Regazzoni and M. van Dijk, Eds. ACM, 2022, pp. 53–61.
8. A. Aikata and S. S. Roy, "Secure and efficient outsourced matrix multiplication with homomorphic encryption," in *INDOCRYPT 2024, Part I*, ser. LNCS, S. Mukhopadhyay and P. Stanica, Eds., vol. 15495. Springer, 2024, pp. 51–74.
9. W. Lu, S. Kawasaki, and J. Sakuma, "Using fully homomorphic encryption for statistical analysis of categorical, ordinal and numerical data," in *NDSS 2017*. The Internet Society, 2017.
10. G. C. Fox, S. W. Otto, and A. J. G. Hey, "Matrix algorithms on a hypercube I: matrix multiplication," *Parallel Comput.*, vol. 4, no. 1, pp. 17–31, 1987.
11. H. Cha, I. Hwang, S. Min, J. Seo, and Y. Song, "Matrigear: Accelerating authenticated matrix triple generation with scalable prime fields via optimized HE packing," in *SP 2025*. IEEE, 2025, pp. 2453–2471.
12. J. H. Park, "Ciphertext-ciphertext matrix multiplication: Fast for large matrices," in *EUROCRYPT 2025, Part VIII*, ser. LNCS, S. Fehr and P. Fouque, Eds., vol. 15608. Springer, 2025, pp. 153–180.
13. J. Liu and L. F. Zhang, "Privacy-preserving and publicly verifiable matrix multiplication," *IEEE Trans. Serv. Comput.*, vol. 16, no. 3, pp. 2059–2071, 2023.
14. Y. Bae, J. H. Cheon, G. Hanrot, J. H. Park, and D. Stehlé, "Plaintext-ciphertext matrix multiplication and FHE bootstrapping: Fast and fused," in *CRYPTO 2024, Part III*, ser. LNCS, L. Reyzin and D. Stebila, Eds., vol. 14922. Springer, 2024, pp. 387–421.
15. D. Rathee, P. K. Mishra, and M. Yasuda, "Faster PCA and linear regression through hypercubes in HElib," in *WPES@CCS 2018*, D. Lie, M. Mannan, and A. Johnson, Eds. ACM, 2018, pp. 42–53.
16. H. Huang and H. Zong, "Secure matrix multiplication based on fully homomorphic encryption," *J. Supercomput.*, vol. 79, no. 5, pp. 5064–5085, 2023.
17. L. Zhu, Q. Hua, Y. Chen, and H. Jin, "Secure outsourced matrix multiplication with fully homomorphic encryption," in *ESORICS 2023, Part I*, ser. LNCS, G. Tsudik, M. Conti, K. Liang, and G. Smaragdakis, Eds., vol. 14344. Springer, 2023, pp. 249–269.
18. J. H. Cheon, A. Kim, M. Kim, and Y. S. Song, "Homomorphic encryption for arithmetic of approximate numbers," in *ASIACRYPT 2017, Part I*, ser. LNCS, T. Takagi and T. Peyrin, Eds., vol. 10624. Springer, 2017, pp. 409–437.
19. J. H. Cheon, A. Kim, and D. Yhee, "Multi-dimensional packing for HEAAN for approximate matrix arithmetics," *IACR Cryptol. ePrint Arch.*, p. 1245, 2018.
20. J. Jang, Y. Lee, A. Kim, B. Na, D. Yhee, B. Lee, J. H. Cheon, and S. Yoon, "Privacy-preserving deep sequential model with matrix homomorphic encryption," in *ASIA CCS '22*, Y. Suga, K. Sakurai, X. Ding, and K. Sako, Eds. ACM, 2022, pp. 377–391.
21. P. K. Mishra, D. H. Duong, and M. Yasuda, "Enhancement for secure multiple matrix multiplications over ring-lwe homomorphic encryption," in *ISPEC 2017*, ser. LNCS, J. K. Liu and P. Samarati, Eds., vol. 10701. Springer, 2017, pp. 320–330.
22. S. Wang and H. Huang, "Secure outsourced computation of multiple matrix multiplication based on fully homomorphic encryption," *KSII Trans. Internet Inf. Syst.*, vol. 13, no. 11, pp. 5616–5630, 2019.
23. P. K. Mishra, D. Rathee, D. H. Duong, and M. Yasuda, "Fast secure matrix multiplications over ring-based homomorphic encryption," *Inf. Secur. J. A Glob. Perspect.*, vol. 30, no. 4, pp. 219–234, 2021.
24. X. Ma, C. Ma, Y. Jiang, and C. Ge, "Improved privacy-preserving PCA using optimized homomorphic matrix multiplication," *Comput. Secur.*, vol. 138, p. 103658, 2024.
25. X. Zheng, H. Li, and D. Wang, "A new framework for fast homomorphic matrix multiplication," *IACR Cryptol. ePrint Arch.*, p. 1649, 2023.
26. J. Chen, L. Yang, W. Wu, Y. Liu, and Y. Feng, "Homomorphic matrix operations under bicyclic encoding," *IEEE Trans. Inf. Forensics Secur.*, vol. 20, pp. 1390–1404, 2025.
27. G. S. Çetin, Y. Doröz, B. Sunar, and W. J. Martin, "An investigation of complex operations with word-size homomorphic encryption," *IACR Cryptol. ePrint Arch.*, p. 1195, 2015.
28. M. D. Cock, R. Dowsley, A. C. A. Nascimento, and S. C. Newman, "Fast, privacy preserving linear regression over distributed datasets based on pre-distributed data," in *AISec 2015*, I. Ray, X. Wang, K. Ren, C. Dimitrakakis, A. Mitrokotsa, and A. Sinha, Eds. ACM, 2015, pp. 3–14.
29. N. Mital, C. Ling, and D. Gündüz, "Secure distributed matrix computation with discrete fourier transform," *IEEE Trans. Inf. Theory*, vol. 68, no. 7, pp. 4666–4680, 2022.
30. C. Guo and N. J. Higham, "A schur-newton method for the matrix p'th root and its inverse," *SIAM J. Matrix Anal. Appl.*, vol. 28, no. 3, pp. 788–804, 2006.
31. J. H. Cheon, K. Han, A. Kim, M. Kim, and Y. Song, "Bootstrapping for approximate homomorphic encryption," in *EUROCRYPT 2018, Part I*, ser. LNCS, J. B. Nielsen and V. Rijmen, Eds., vol. 10820. Springer, 2018, pp. 360–384.

32. A. A. Badawi, J. Bates, F. Bergamaschi, D. B. Cousins, S. Erabelli, N. Genise, S. Halevi, H. Hunt, A. Kim, Y. Lee, Z. Liu, D. Micciancio, I. Quah, Y. Polyakov, R. V. Saraswathy, K. Rohloff, J. Saylor, D. Suponitsky, M. Triplett, V. Vaikuntanathan, and V. Zucca, "OpenFHE: Open-source fully homomorphic encryption library," in *Proceedings of the 10th Workshop on Encrypted Computing & Applied Homomorphic Cryptography*, M. Brenner, A. Costache, and K. Rohloff, Eds. ACM, 2022, pp. 53–63.
33. V. STRASSEN, "Gaussian elimination is not optimal," *Numerische Mathematik*, vol. 13, pp. 354–356, 1969.
34. Z. Brakerski, "Fully homomorphic encryption without modulus switching from classical gapsvp," in *CRYPTO 2012*, ser. LNCS, R. Safavi-Naini and R. Canetti, Eds., vol. 7417. Springer, 2012, pp. 868–886.
35. J. Fan and F. Vercauteren, "Somewhat practical fully homomorphic encryption," *IACR Cryptol. ePrint Arch.*, p. 144, 2012.
36. Z. Brakerski, C. Gentry, and V. Vaikuntanathan, "(leveled) fully homomorphic encryption without bootstrapping," in *Innovations in Theoretical Computer Science 2012*, S. Goldwasser, Ed. ACM, 2012, pp. 309–325.
37. K. Arimitsu and K. Otsuka, "Privacy-preserving fast and exact linear equations solver with fully homomorphic encryption," *IACR Cryptol. ePrint Arch.*, p. 272, 2020.
38. W. Xu, B. Wang, J. Liu, Y. Chen, P. Duan, and Z. Hong, "Toward practical privacy-preserving linear regression," *Inf. Sci.*, vol. 596, pp. 119–136, 2022.
39. M. Kim, Y. Song, S. Wang, Y. Xia, and X. Jiang, "Secure logistic regression based on homomorphic encryption," *IACR Cryptol. ePrint Arch.*, p. 74, 2018.
40. K. Han, S. Hong, J. H. Cheon, and D. Park, "Logistic regression on homomorphic encrypted data at scale," in *AAAI 2019*. AAAI Press, 2019, pp. 9466–9471.
41. K. Nandakumar, N. K. Ratha, S. Pankanti, and S. Halevi, "Towards deep neural network training on encrypted data," in *CVPR Workshops 2019*. Computer Vision Foundation / IEEE, 2019, pp. 40–48.
42. M. Iezzi, "Practical privacy-preserving data science with homomorphic encryption: An overview," in *IEEE BigData 2020*, X. Wu, C. Jermaine, L. Xiong, X. Hu, O. Kotevska, S. Lu, W. Xu, S. Aluru, C. Zhai, E. Al-Masri, Z. Chen, and J. Saltz, Eds. IEEE, 2020, pp. 3979–3988.
43. J. Park, "Homomorphic encryption for multiple users with less communications," *IEEE Access*, vol. 9, pp. 135 915–135 926, 2021.
44. D. Boneh, R. Gennaro, S. Goldfeder, A. Jain, S. Kim, P. M. R. Rasmussen, and A. Sahai, "Threshold cryptosystems from threshold fully homomorphic encryption," in *CRYPTO 2018, Part I*, ser. LNCS, H. Shacham and A. Boldyreva, Eds., vol. 10991. Springer, 2018, pp. 565–596.
45. C. Mouchet, J. R. Troncoso-Pastoriza, J. Bossuat, and J. Hubaux, "Multiparty homomorphic encryption from ring-learning-with-errors," *Proc. Priv. Enhancing Technol.*, vol. 2021, no. 4, pp. 291–311, 2021.
46. C. Mouchet, E. Bertrand, and J. Hubaux, "An efficient threshold access-structure for rlwe-based multiparty homomorphic encryption," *J. Cryptol.*, vol. 36, no. 2, p. 10, 2023.
47. G. A. Seber and A. J. Lee, *Linear regression analysis*. John Wiley & Sons, 2012.

A Diagonal-Packing Based Matrix Multiplication

In this section, we present a CC-MM algorithm that utilizes $2d$ ciphertexts, where the diagonals of the input matrices are individually packed into separate ciphertexts. Specifically, we denote the i-th encrypted diagonal of matrix $\mathbf{A}$ as $\mathsf{ct}.\mathbf{D}_i^{\mathbf{A}}$ for $0 \leq i < d$, and apply the same packing structure to matrix $\mathbf{B}$. This symmetric diagonal-wise packing facilitates efficient evaluation of consecutive matrix multiplications. The complete description of the algorithm is detailed in Algorithm 8.

Algorithm 8 $\mathsf{MatMult_DP}(\{\mathsf{ct}.\mathbf{D}_i^{\mathbf{A}}\}_{0\leq i<d}, \{\mathsf{ct}.\mathbf{D}_i^{\mathbf{B}}\}_{0\leq i<d})$

Input: $\{\mathsf{ct}.\mathbf{D}_i^{\mathbf{A}}\}_{0\leq i<d}$ and $\{\mathsf{ct}.\mathbf{D}_i^{\mathbf{B}}\}_{0\leq i<d}$, which are encrypted diagonals of $d \times d$ matrices $\mathbf{A}$ and $\mathbf{B}$, respectively
Output: $\{\mathsf{ct}.\mathbf{D}_i^{\mathbf{C}}\}_{0\leq i<d}$, where $\mathsf{ct}.\mathbf{D}_i^{\mathbf{C}}$ is the encryption of the i-th diagonal of $d \times d$ matrix $\mathbf{C}(= \mathbf{AB})$

```
for i = 0 to d − 1 do
    ct.D_i^C ← 0
    for j = 0 to d − 1 do
        ct.R_j ← Mult(ct.D_j^A, Rot(ct.D_{i−j}^B, j))
        ct.D_i^C ← Add(ct.D_i^C, ct.R_j)
    end for
end for
return {ct.D_i^C}_{0≤i<d}
```

B Additional Experimental Results

To address potential interest from readers, we provide additional benchmarks, although this lies somewhat outside the main scope of the paper. Table 6 presents experimental results for a single multiplication across several algorithms. All experiments were conducted under parameter settings targeting a 128-bit security level, with a scaling factor of $\Delta = 2^{50}$. As discussed in the main body of the paper, our method is optimized for larger batch sizes satisfying $d^3 < (N/2)$, where d denotes the matrix dimension and N is the ring dimension of the underlying FHE scheme. Consequently, our method outperforms the alternatives when $d = 4, 8, 16$. However, while our method also outperforms others for $d = 32$ in the experiments of Table 2, it is outperformed by the method in [8] in this single-multiplication setting. This is because the parameter set for Table 6 is configured to support only a single multiplication, resulting in a smaller ring dimension N than that used in Table 2. To enable a fair comparison under settings with larger ring dimension N, which allows for supporting a greater circuit depth, the execution times reported in Table 2 may be normalized by dividing them by 15.

Table 6. Running Time of 1 multiplication with respect to d (unit: sec)

d	DP	[3]	[7]	[8]	Ours
4	0.58	0.46	0.43	0.29	0.16
8	3.43	1.36	1.49	0.81	0.39
16	17.30	3.11	4.60	1.37	0.83
32	82.00	6.58	32.50[§]	2.33	3.92
64	389.00	13.10	-	14.70	38.50
N	2^{14}	2^{14}	2^{14}	2^{14}	2^{14}
L	1	3	2	2	2

[§] Using Strassen's algorithm with reduction to 16×16 matrices