# FHE4DMM: A Low-Latency Distributed Matrix Multiplication with Fully Homomorphic Encryption

Yi Chen, Qiang-Sheng Hua, Zixiao Hong, Lin Zhu, and Hai Jin

***Abstract*—Fully Homomorphic Encryption (FHE) is a promising technology for secure, non-interactive outsourced computation. One notable method to increase the throughput of FHE-based outsourcing is batching, which typically involves large-scale matrix-matrix multiplications (MM). However, the substantial overhead inherent in existing FHE schemes poses a major challenge for processing these large-scale tasks, often resulting in insufficient memory or prolonged delays on a single machine, making it practically unviable.**

**Utilizing multi-machine parallelism in cloud clusters for outsourced computation offers a natural solution to these obstacles. In this work, we propose FHE4DMM, a distributed algorithm that provides a unified view on encrypted matrices, accommodating various FHE schemes and any matrix dimensions, to accelerate large-scale encrypted MM. A key innovation is its reuse optimizations for parallelized homomorphic computations, which can offer valuable insights for broader FHE-based applications.**

**We utilized FHE4DMM to conduct large-scale square $(4096 \times 4096)$ and rectangular $(32768 \times 32768, 32768 \times 16)$ matrix multiplications on 256 machines, achieving computation time of 172.2s and 76.1s, respectively, while ensuring a 128-bit security level. For scalability, the experiments demonstrate that FHE4DMM achieves linear speedup for $2^i$ ($i$ is from $0$ to $6$) machines across various matrix dimension cases. In addition, within the range of matrix dimensions that the state-of-the-art (SOTA) distributed FHE-MM algorithm [1] can handle, FHE4DMM attains a maximum speedup of 16.62x. To assess its practical performance, FHE4DMM is applied in a basic multi-layer feedforward network. We used 64 machines to perform secure outsourced inference on MNIST and CIFAR-10 datasets with encrypted models and data. Compared to using the SOTA, our method achieved speedups of up to 3.54x and 4.22x respectively, with the MM module obtaining a 4.09x and 4.87x speedup.**

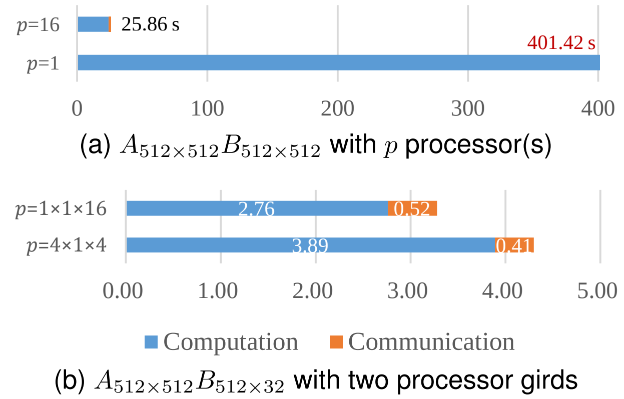
***Index Terms*—Homomorphic encryption, Distributed algorithm, Matrix-matrix multiplication.**

## I. Introduction

FULLY Homomorphic Encryption (FHE) stands out as the most promising technology for ensuring the security of data in use. However, its status as the preferred choice in practical scenarios is restrained by the considerable computational and memory resource demands during computing on encrypted data, thereby posing challenges for real-world applications. Over the past decade following Gentry's breakthrough [2] in FHE, there has been a substantial enhancement in the efficiency of homomorphic encryption schemes. Currently, schemes like BFV/BGV/CKKS [3]–[5], which integrate powerful packing techniques, and FHEW/TFHE [6], [7], supporting fast bootstrapping, become the mainstream. Moreover, a well-practiced arithmetic framework has been devised to handle high-precision integers using the Residue Number System (RNS) and accelerate polynomial multiplication through the Fast Fourier Transform (FFT).

On this foundation, numerous FHE libraries [8]–[10] have surfaced, alongside accelerated implementations on GPUs [11]–[13], FPGAs [14], and ASICs [15], [16]. Notably, their main focus is single-node optimization, specifically tailoring the deep optimization of underlying polynomial arithmetic to enhance the resource utilization of individual hardware. However, the substantial performance gap of $10^5$ between ciphertext and plaintext computation presents an obstacle to executing large-scale applications on a single node. When considering computation-intensive applications such as machine learning (ML), the overhead in privacy-preserving ML (PPML) becomes more pronounced. Meanwhile, the inherent support of FHE for secure outsourced computation aligns seamlessly with cloud computing clusters. Thus, it is important to investigate efficient distributed FHE algorithms.

This paper focuses on matrix-matrix multiplication (MM), a fundamental building block in ML and scientific computing. Based on insights gained from the following preliminary experiments, we are motivated to delve further into the distributed algorithm in FHE.



(a) $A_{512\times512}B_{512\times512}$ with $p$ processor(s)

(b) $A_{512\times512}B_{512\times32}$ with two processor girds

Fig. 1. Time breakdown(s) of FHE-MM.

**Single Machine Cannot Handle Large FHE Parameters.** Employing the state-of-the-art FHE-based general MM algorithm [17] (FHE-MM), we conduct tests on a **1TB** large memory machine to perform a MM with two 512 × 512 square matrices. The experimental findings reveal a significant

The authors are with the National Engineering Research Center for Big Data Technology and System, Services Computing Technology and System Lab, School of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan 430074, P. R. China (email: qshua@hust.edu.cn).

(Corresponding author: Qiang-Sheng Hua.)

Manuscript received July 19, 2024; revised August 16, XX.

memory usage of **546GB**, surpassing the memory size of the server-side common processors.

**Medium-Sized FHE-MM Suffices to Exhaust the Computational Resources on a Single Machine.** Adopting block matrix multiplication (BMM) is a natural idea to avoid using large FHE parameters. Therefore we decompose the original MM into $4^3$ small MMs with a block size of $128 \times 128$ (ensuring the 128-bit security level). We executed these smaller MMs concurrently on a single machine and collaboratively on multiple machines, respectively. Each small MM is also computed in parallel internally. The outcomes in Fig. 1(a) show an execution time of **401.42s** on a single machine, which remains impractical; however, the observed linear speedup of **15.52x** across multiple machines implies that single-machine parallelism is insufficient for accelerating these medium-sized FHE-MMs due to limited resources.

The parallel general matrix multiplication (PGEMM) algorithm on unencrypted data has been studied quite maturely. This well-established research provides an optimistic insight that leveraging multiple machines can significantly accelerate FHE-based applications. We then execute FHE-PGEMM using two distinct processor grids with identical FHE parameters: one derived from the state-of-the-art PGEMM algorithm [18] (i.e., $[4, 1, 4]$) and another using an arbitrary partitioning $[1, 1, 16]$ (see Fig. 1(b)). However, both configurations show comparable communication costs, with the former incurring more computations. These observations suggest that homomorphic computation is related to raw data partitioning. This realization emphasizes the need to optimize FHE-PGEMM by considering both communication and computation costs.

**Our Contribution.** We propose FHE4DMM, a low-latency FHE-based distributed MM for any matrix dimension and processor number. Our key insight is that certain potential redundancies in homomorphic computations, obscured by plaintext-perspective algorithm designs, consume significant time. FHE4DMM contributes optimizations in "higher dimensions" compared to a traditional distributed 3DMM algorithm [19] (3DMM). The additional dimension aims to address the above issues, including the careful selection of a fixed data layout to ensure optimal computation before task partitioning and eliminating redundant computations.

Combined with highly optimized single-node FHE-MM at both algorithmic and hardware resource utilization levels, FHE4DMM achieves excellent scalability, demonstrated by its linear speedup across various matrix dimensions and weak scalability across different problem sizes. Using 64 machines, we applied FHE4DMM for secure outsourced inference on MNIST and CIFAR-10 datasets, representing two types of large-scale applications, with varying batch sizes. The experimental results show that FHE4DMM achieves a greater speedup for larger data scales.

## II. Background

We first provide an overview of existing FHE schemes based on learning with errors over rings (RLWE). We then differentiate between PGEMM algorithms with and without FHE. Following this, we introduce our system model and outline our optimization goals. Table I summarizes commonly used symbols in this paper.

TABLE I
Notations and their definitions

| Symbol | Definition |
|---|---|
| $m, n, k$ | Dimensions of matrix multiplication $A_{m\times k}B_{k\times n}$ |
| $d_1, d_2, d_3$ | $d_1 = \max\{m, n, k\}, d_2 = \text{median}\{m, n, k\}$,<br>$d_3 = \min\{m, n, k\}$ |
| $[I : I + x]$ | Index set $\{I, \ldots, I + x - 1\}$ (shorthanded as $[x]$ when $I = 0$. Multiple dimensions separated by ";") |
| $P_{ijt}$ | Processor index under the processor grid<br>$p = p_m \times p_n \times p_k$ ($i \in [p_m], j \in [p_n], t \in [p_k]$) |
| $q, t$ | Ciphertext modulus and plaintext modulus |
| $N$ | Number of polynomial coefficients |
| $S$ | Number of plaintext slots ($S_o$: the optimal one) |
| $H$ | Word size of a ciphertext |
| CGEMM(A,B) or<br>CGEMM$(m, n, k)$ | Ciphertext-Ciphertext matrix multiplication with encryption of two matrices $A, B$ |

### A. RLWE-based FHE Schemes

**RNS Variants.** Current FHE schemes all mask plaintext in noise terms. When the noise exceeds a certain threshold, the original plaintext becomes irretrievable. In RLWE-based schemes, this critical threshold is almost determined by $\log(q/t)$. The modulus $q$ is derived from a polynomial ring $R_q = \mathbb{Z}_q[X]/(f(X))$ within RLWE, where $f(X)$ is a cyclotomic polynomial of degree $N - 1$.

To accommodate more noise, $\log q$ is normally set from $10^2$ to $10^3$. Current FHE libraries adopt the RNS variants to handle large integer operations with higher parallelism. In the RNS, computation over $R_q$ is decomposed into multiple computations over $R_{q_i}$, where $q = \Pi_{i=1}^{l} q_i$ and each $q_i$ is a small prime. Compared to the relatively inexpensive (i.e., $O(l \cdot N)$) ciphertext-ciphertext addition (ADD) and plaintext-ciphertext multiplication (PMULT), ciphertext-ciphertext multiplication (MULT) and ciphertext rotation (ROT) are more costly (i.e., $O(l^2 \cdot N \log N)$) due to key-switching [20].

**Packing Techniques.** Packing techniques allow homomorphic operations to imply some specific computation on batches of raw data. We focus mainly on SIMD encoding (see Fig. 2), using $S$ to denote the number of fully packed plaintext slots to unify different schemes (i.e., for CKKS, $S = N/2$; for BFV/BGV, which supports hypercube encoding [21], $S = N$).

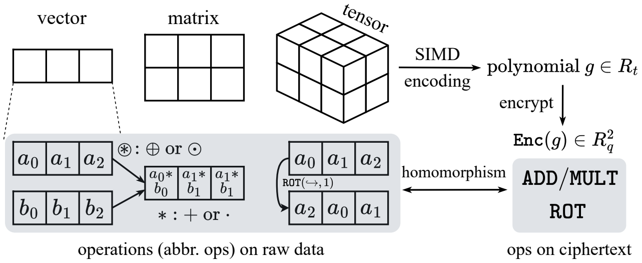


Fig. 2. SIMD encoding and SIMD homomorphic operations

### B. Parallel General Matrix Multiplication

**GEMM.** There exist well-established linear algebra libraries [22]–[24] that support high-performance GEMMs on unencrypted data. However, SIMD computations vary across different applications, and there is no standardized FHE-MM yet. In this paper, we focus on relatively general MM with two encrypted matrices (aka, CGEMM).

We adopt the state-of-the-art efficient CGEMM algorithm [17] applicable to any matrix dimensions. The complexity of this algorithm is summarized in Table II. Note that different FHE schemes and optimizations (e.g., Baby-step/Giant-step (BSGS)) may introduce some variations, but these do not affect the overall analysis.

TABLE II
THE COMPLEXITY OF [17] DESCRIBED BY THE NUMBER OF HOMOMORPHIC OPERATIONS ($C_{\texttt{ADD}} = 2k + d_3$; $C_{\texttt{ROT}} = 2k + 2d_3$)

| Matrix Type | `ADD` | `PMULT` | `ROT` | `MULT` |
|---|---|---|---|---|
| $k = d_1$ | $C_{\texttt{ADD}} + \log \frac{k^3}{d_2 d_3^2}$ | $2k$ | $C_{\texttt{ROT}} + \log \frac{k^3}{d_2 d_3^2}$ | $d_3$ |
| $k = d_2$ or $k = d_3$ | $C_{\texttt{ADD}} + \log \frac{d_1 d_2}{d_3^2}$ | $2k$ | $C_{\texttt{ROT}} + \log \frac{d_1 d_2}{d_3^2}$ | $d_3$ |

**PGEMM.** PGEMM is commonly employed for efficient large-scale MMs. Regardless of the matrix partitioning scheme employed, PGEMM ensures that the total computation of $O(mnk)$ multiplications and additions remain constant[1]. For modern processors with peak performance over $10^{12}$ FLOPS, practical latency is significantly impacted by data transfers. The recent works such as COSMA [26] and CA3DMM [18] both have optimal or near-optimal communication costs for all matrix dimensions and any number of processors through properly partitioning data across processors.

In contrast, when using packing techniques, FHE-PGEMM must account for *homomorphic data movement* (e.g., `ROT` and `PMULT`). Moreover, encoding is related to FHE parameters, which can significantly impact performance. Therefore, optimization methods for traditional distributed algorithms [18], [26], [27] may not always be appropriate for FHE-PGEMM.

### C. System Model

To perform distributed matrix multiplication with FHE, a client initially encrypts the two input matrices. These encrypted matrices and evaluation keys (i.e., part of the public key) are then distributed to semi-honest [28], interconnected servers. The servers then conduct collaborative matrix multiplication through inter-server communication and homomorphic operations on local ciphertexts. Finally, the client receives the encrypted results from the servers and then decrypts them.

We model a parallel machine as described in [29]. Traditional distributed algorithms often focus on the number of synchronizations and communication costs $W$. This is because data partitioning does not affect the sum of computation cost $F$ across all processors, denoted as $F^*$. However, partitioning can influence computation in FHE, potentially altering $F^*$.

**Optimization Goals.** The overarching objective is to minimize the total latency in the design of FHE4DMM. In the current FHE-based application ecosystem, we prioritize reducing the total computation cost, especially for homomorphic data movement. We then aim to devise a scheduling strategy that ensures optimal or near-optimal communication costs under such fixed computations.

## III. Trivial Distributed MM with FHE

In this section, we employ the state-of-the-art CGEMM algorithm to design a distributed FHE-MM directly (§ III-A). Subsequently, we analyze its issues (§ III-B), which lead us to introduce the block intermediate representation (§ III-C).

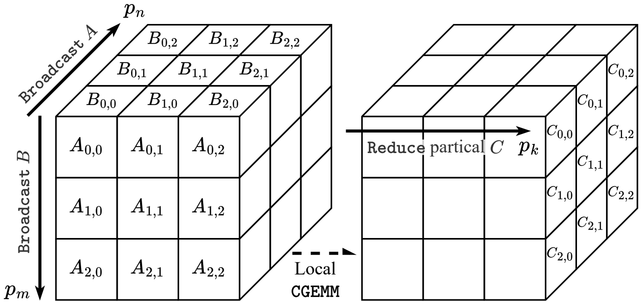


Fig. 3. Distributed 3DMM with FHE

### A. A Trivial FHE3DMM Algorithm

3DMM is a classic distributed MM algorithm designed to minimize synchronization and communication costs. Moreover, [18] demonstrated that 3DMM can achieve optimal or near-optimal communication costs through appropriate processor grid partitioning. Utilizing these benefits is advantageous for efficient FHE-PGEMM.

The general 3DMM organizes $p$ processors into a 3D grid $p_m \times p_n \times p_k$ (see Fig. 3), indexed by $P_{ijt}$. A trivial FHE3DMM can then be described in the following three steps:
**Step 1.** $P_{i0t}$ broadcasts encrypted $A_{i,t}$ to the processor group $P_{i:t}$, and $P_{0jt}$ broadcasts encrypted $B_{t,j}$ to the group $P_{:jt}$.
**Step 2.** Subsequently, each processor $P_{ijt}$ possesses both $A_{i,t}$ and $B_{t,j}$, and then performs $\texttt{CGEMM}(A_{i,t}, B_{t,j})$ locally.
**Step 3.** When local computation is completed, the processor group $P_{ij:}$ collectively reduces the partial result to $P_{ij0}$.

### B. Issues of Trivial FHE3DMM

Trivial FHE3DMM treats `CGEMM` as a black box, directly partitioning data at the raw data layer. Its failure to deep dive into FHE makes it impractical for real-world applications.

**Limited Packing.** FHE parameters should be determined solely by noise capacity and security level rather than the amount of encoding. Otherwise, encrypting each matrix block in Fig. 3 as one ciphertext would result in a large $N$ (or $S$), which presents potential issues for existing FHE schemes: (1) overflow of the security level; (2) significant increases in precomputation time and memory usage in FHE libraries, as well as (3) the overhead of each homomorphic operation.

**Data Transfer and Computation Modifications in the Rectangular Matrix Case.** Rectangular matrices are frequently encountered in practical applications. However, the

[1] This paper considers only the conventional MM without utilizing the fast matrix multiplication [25] (aka, Strassen's matrix multiplication).

constraints imposed by FHE's SIMD paradigms complicate the handling of rectangular MM in existing works [17], [30], [31]. Therefore, the demand for $S$ does not scale linearly with the actual encoded data volume. For `CGEMM` in [17], the number of required plaintext slots $S_r$ is defined as

$$S_r(m, n, k) = \max\{m, k\} \cdot \max\{k, n\}. \tag{1}$$

For any rectangular matrix case, $(m + n)k < 2S_r$. This encoding process reconstructs the raw data, potentially altering the original meaning of partitioning based on dimensions of plaintext matrices (e.g., processor grid partitioning for reducing communication costs).

Furthermore, Table II shows that for rectangular MM with the same $S_r$, the required homomorphic operations vary depending on its type. Referring back to Fig. 1, when local `CGEMM` has the same minimum dimension and $S_r$, the larger dimension $k$ in the case where $p = 4 \times 1 \times 4$ necessitates more homomorphic data movements.

### C. Block Intermediate Representation (BIR) of FHE-MM

The above issues (§ III-B) suggest the need to introduce an intermediate representation to eliminate factors influenced by FHE. A critical step is determining the optimal number of plaintext slots $S_r$ (referred to as $S_o$[2]), where the optimal value is precisely determined by noise and security considerations. For convenience, this paper also uses the partitioning configuration $[b_m, b_n, b_k]$ to represent the number of blocks in matrices $A_{m\times k}$ and $B_{k\times n}$ along the $m$, $n$, and $k$ dimensions, respectively. This configuration is called the *block layout* to distinguish it from the processor grid.

Given $S_o$ and a block layout, the *block size* $H$ can be straightforwardly defined [2]. In the block layout $[b_m, b_n, b_k]$, all encryptions share a uniform size $H$. Only mod-switching or rescaling operations can potentially modify (or reduce) $H$ within `CGEMM`. Here, the initial size is taken as an approximate upper bound. A ***BIR*** of $A_{m\times k}B_{k\times n}$, denoted as $\texttt{BIR}(AB)$, can then be defined as two sets of encrypted matrix blocks, $A^*$ and $B^*$. Here, $A^*$ is an abbreviation for $A^*_{[0:b_m;0:b_k]}$ when a block layout is provided in context; similarly for $B^*$.

Algorithm 1 uses BIR to describe the standard FHE-BMM.

**Algorithm 1:** Standard FHE-BMM

```
Input: $A^*_{[b_m,b_k]}, B^*_{[b_k,b_n]}$: $\texttt{BIR}(AB)$
Output: $C^*_{[b_m;b_n]}$ : $C^* = \texttt{Enc}((AB)_{[b_m;b_n]})$
for $i = 0$ to $b_m - 1$ do
    for $j = 0$ to $b_n - 1$ do
        $C^*_{i,j} \leftarrow \texttt{CGEMM}(A^*_{i,0}, B^*_{0,j})$
        for $t = 1$ to $b_k - 1$ do
            $C^*_{i,j} \leftarrow \texttt{ADD}(C^*_{i,j}, \texttt{CGEMM}(A^*_{i,t}, B^*_{t,j}))$
```

**Total computation cost $F^*$:** According to Algorithm 1, the total computation costs can be expressed as

$$F^* = b_m b_n b_k \cdot \texttt{CGEMM}_F(\frac{m}{b_m}, \frac{n}{b_n}, \frac{k}{b_k}) + b_m b_n b_k \cdot F_{\texttt{ADD}}, \tag{2}$$

where $\texttt{CGEMM}_F$ and $F_{\texttt{op}}$ denote the computational complexity of `CGEMM` and the homomorphic operation `op`, respectively.

## IV. The Enhanced FHE3DMM Algorithm

Based on BIR(§III-C), we propose an enhanced FHE3DMM to mitigate the impact of partitioning on computation costs. Furthermore, leveraging this BIR enables us to efficiently schedule computation tasks on encrypted data, akin to scheduling on unencrypted data, thereby minimizing communication costs. Here, we assume that $S_o$ has already been obtained.

### A. Optimal Block Layout (OBL) of FHE-MM

Consider performing $A_{m\times k}B_{k\times n}$. The following analysis is confined to the scenario where $S_r(m, n, k) > S_o$; otherwise, a single `CGEMM` is adequate. Once $[b_m, b_n, b_k]$ is determined, the total computation costs are fixed from Eq. (2). To approach optimal parameters, $S_r(\frac{m}{b_m}, \frac{n}{b_n}, \frac{k}{b_k})$ needs to be close enough to $S_o$. Thus, the optimization problem of finding the OBL for minimizing $F^*$ is equivalent to minimizing the number of homomorphic operations asymptotically with a sub-target $\min b_m b_n b_k$ as follows:

$$\text{minimize:} \quad b_m b_n b_k \cdot \texttt{CGEMM}(\frac{m}{b_m}, \frac{n}{b_n}, \frac{k}{b_k}) \tag{3}$$

$$\text{subject to:} \quad \max\{\frac{m}{b_m}, \frac{k}{b_k}\} \cdot \max\{\frac{k}{b_k}, \frac{n}{b_n}\} \leq S_o \tag{4}$$

$$b_m b_n b_k \neq 1 \quad \text{and} \quad b_m, b_n, b_k \geq 1$$

Algorithm 2 delineates an OBL of matrices $A_{m\times k}$ and $B_{k\times n}$ guaranteed by Theorem 1.

**Algorithm 2:** Optimal block layout generation

```
Procedure: $[\hat{b}_m, \hat{b}_n, \hat{b}_k] \leftarrow \texttt{OBL}(m, n, k)$
Let $x, y, z$ refer to $m, n, k$ s.t. $x \geq y \geq z$.
if $k \geq \sqrt{S_o}$ then                                ▷ Case (1)
    if $m \geq \sqrt{S_o}$ and $n \geq \sqrt{S_o}$ then   ▷ Sub-case(1.1)
        return $[\frac{m}{\sqrt{S_o}}, \frac{n}{\sqrt{S_o}}, \frac{k}{\sqrt{S_o}}]$
    else if $mn \geq S_o$ then                            ▷ Sub-case (1.2)
        $\hat{b}_x \leftarrow \frac{mn}{S_o}, \hat{b}_z \leftarrow 1, \hat{b}_y \leftarrow \frac{y}{z}$
        return $[\hat{b}_m, \hat{b}_n, \hat{b}_k]$
    else if $xk \geq S_o$ then                            ▷ Sub-case (1.3)
        return $[1, 1, \frac{xk}{S_o}]$
    else ▷ Sub-case (1.4): $mn < S_o$ and $xk < S_o$
        return $[1, 1, \frac{k}{\sqrt{S_o}}]$
else                                         ▷ Case (2): $k < \sqrt{S_o}$
    $P_1 \leftarrow \frac{xy}{S_o}, P_2 \leftarrow \frac{y}{z}$
    if $y = k$ then                                       ▷ Sub-case (2.1)
        $\hat{b}_y \leftarrow \min\{P_1, P_2\}, \hat{b}_x \leftarrow P_1/\hat{b}_y$
    else                                                  ▷ Sub-case (2.2)
        Select any $\hat{b}_x, \hat{b}_y$ s.t. $\hat{b}_x\hat{b}_y = P_1$ where
          $1 \leq \hat{b}_x, \hat{b}_y \leq \min\{P_1, P_2\}$.
    $\hat{b}_z \leftarrow 1$
    return $[\hat{b}_m, \hat{b}_n, \hat{b}_k]$
```

[2] More details and more formal definitions are provided in Appendix A.

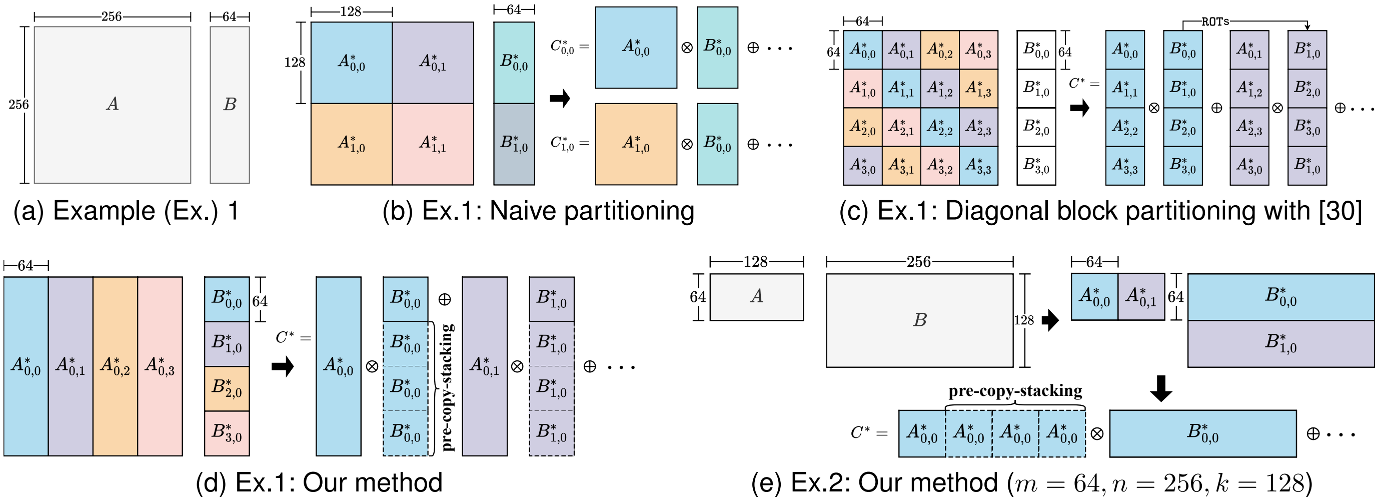


Fig. 4. Partitioning methods for $A_{m\times k}B_{k\times m}$ with FHE. Here, $S_o = 128^2$ and ⊗ denotes CGEMM. (All these partitioning have the same $S_r$. Fig. IV: naive partitioning [2, 1, 2]. Fig. IV: *diagonal block partitioning* proposed in [1]. Fig. IV and IV: the sub-case (1.2) in Algorithm 2, where $\texttt{OBL}(256, 64, 256) = [\frac{mn}{S_o}, 1, \frac{k}{n}] = [1, 1, 4]$ and $\texttt{OBL}(64, 256, 128) = [1, \frac{mn}{S_o}, \frac{k}{m}] = [1, 1, 2]$)

**Theorem 1.** Given matrices $A_{m\times k}$ and $B_{k\times n}$, the block layout determined by Algorithm 2 results in equality for (4), and Eq. (3) approaches a minimum asymptotically.

A detailed proof of Theorem 1 is provided in Appendix B. To illustrate the generality of our method, we compare it with existing methods through concrete examples in Fig. 4.

Figs. 4(b), (c), and (d) perform the same MM on the encryption of Fig. (a). They all execute the same number of CGEMMs.[3] For a single CGEMM, an equivalent number of MULTs can be observed in Table II and the analysis presented in [30]. Disregarding the logarithmic terms in Table II and inter-block rotation in [1], Figs. 4(c) and (d) also perform the same number of ROTs (i.e., 1024 times), and both showcase a reduction by half compared to Fig. (b) (i.e., 2048 times). In the scenario in [1], eliminating the logarithmic term is feasible through the pre-copy-stacking of four identical $B_{i,0}$ into a single plaintext before encryption. Therefore, it becomes apparent that our partitioning method demonstrates the same computation costs as their specifically optimized BMM method for slim MM.

However, diagonal partitioning inherently imposes restrictions on matrix shapes, rendering the extension of [1] to general matrices still reliant on *standard block partitioning* (or block layout). For instance, to perform the MM in Fig. IV, one can further decompose $B$ into square blocks to apply diagonal partitioning, leading to $S_r = 128 \times 64 < S_o$. Note that $S_o$ can be only determined by the required security level and the depth of homomorphic circuits. [1] imposes strict limitations on $S_r$, **making it difficult to match** $S_o$ **in many scenarios**, thus resulting in wasted idle plaintext slots.

**Remark.** Our method prioritizes generality, making it applicable to any MM. It ensures optimal computation for all cases, but the trade-off lies in not minimizing $(b_m + b_n)b_k$ for the truly minimum number of encrypted matrices. Instead, it minimizes $b_m b_n b_k$ to simplify the OBL generation. Most importantly, we emphasize the necessity of introducing BIR, which is independent of the partitioning method. Thus, subsequent task scheduling based on BIR and optimizations in FHE4DMM are also applicable to other partitioning methods.

### B. Framework of Enhanced FHE3DMM

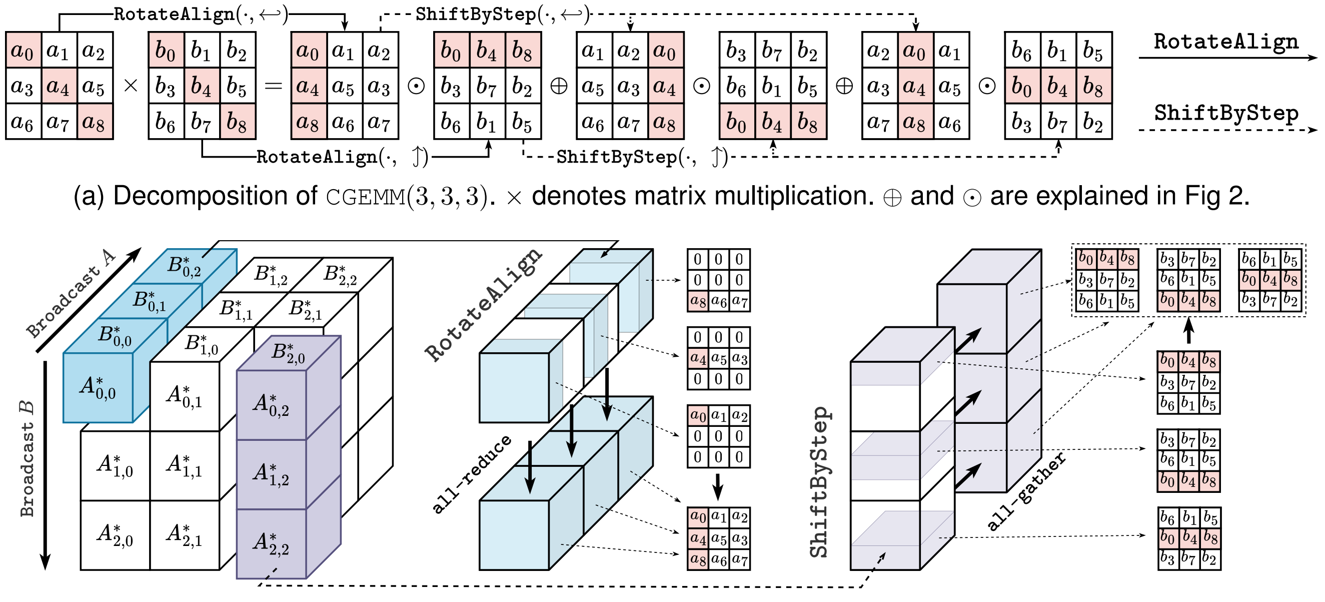
The algorithm comprises the client and server sides. The former chooses how to distribute encrypted data, while the latter is responsible for collaborative computation. The server-side part is identical to trivial FHE3DMM. Therefore, we focus on devising a scheduling strategy based on BIR with OBL to achieve optimal or near-optimal communication costs.

**Algorithm 3:** Client-side setup for plaintext matrices $A_{m\times k}, B_{k\times n}$, and $p$ server-side processors.

```
$[\hat{b}_m, \hat{b}_n, \hat{b}_k] \leftarrow \texttt{OBL}(m, n, k)$
if $p \geq \hat{b}_m \hat{b}_n \hat{b}_k$ then
    $[p_m, p_n, p_k] \leftarrow [\hat{b}_m, \hat{b}_n, \hat{b}_k]$
else
    Find optimal 3D processor grid $p_m \times p_n \times p_k$ by
      minimizing (5) and maximizing its sub-target.
$A^*_{[\hat{b}_m;\hat{b}_k]}, B^*_{[\hat{b}_k;\hat{b}_n]} \leftarrow \texttt{BIR}(AB)$.
According to $[p_m, p_n, p_k]$, $A^*$ is divided into $\mathcal{A}^*_{i,t} = \left\{A^*\left[\frac{i\hat{b}_m}{p_m} : \frac{(i+1)\hat{b}_m}{p_m}; \frac{t\hat{b}_k}{p_k} : \frac{(t+1)\hat{b}_k}{p_k}\right] : i \in [p_m], t \in [p_k]\right\}$.
  Similarly, $\mathcal{B}^*_{t,j}$ is defined by $t \in [p_k], j \in [p_n]$.
Send $\mathcal{A}^*_{i,t}$ to processor $P_{i0t}$, and send $\mathcal{B}^*_{t,j}$ to $P_{0jt}$.
```

**Client-Side Setup.** Algorithm 3 shows how the client distributes data. In contrast to the trivial one, we first obtain the OBL $[\hat{b}_m, \hat{b}_n, \hat{b}_k]$. If $p \geq \hat{b}_m \hat{b}_n \hat{b}_k$, we directly consider $[\hat{b}_m, \hat{b}_n, \hat{b}_k]$ as the processor grid; otherwise, we partition the processor grid on BIR. The encrypted matrices can then be

[3] In Fig. IV, the technique proposed in [30] involves encrypting multiple matrices into a single ciphertext and performing simultaneous inter-CGEMMs.



(a) Decomposition of $\texttt{CGEMM}(3, 3, 3)$. × denotes matrix multiplication. $\oplus$ and $\odot$ are explained in Fig 2.

(b) 4D Optimization on a $3 \times 3 \times 3$ processor grid

Fig. 5. Server-side 4D optimization for FHE3DMM

viewed as new matrices of dimensions $\hat{b}_m \times \hat{b}_k$ and $\hat{b}_k \times \hat{b}_n$, with each element of size $H$.

For any processor grid $[p_m, p_n, p_k]$, the surface area of the sub-cuboid, which corresponds to the data sent and received within local GEMMs on individual processors, serves as a metric for the communication costs on each processor [18]. We can directly employ this idea on BIR to enhance FHE3DMM. The optimal or near-optimal communication costs can thus be achieved by minimizing Eq. (5),

$$W^*_{\text{3D}} = (p_m \hat{b}_k \hat{b}_n + p_n \hat{b}_m \hat{b}_k + p_k \hat{b}_m \hat{b}_n) \cdot H, \tag{5}$$

with constraint $p_m p_n p_k \le p$ and a sub-target $\max p_m p_n p_k$.

In practice, the optimal solution can be obtained by enumerating all possible $[p_m, p_n, p_k]$. While this scheduling is premised on the use of OBL, the minimality of $\hat{b}_m \hat{b}_n \hat{b}_k$ indicates that no other block layout is superior.

### C. Issues of Enhanced FHE3DMM

Using BIR can directly integrate traditional PGEMM into FHE, but it overlooks potential problems associated with performing homomorphic operations on large-scale data, particularly data dependencies across multiple ciphertexts (e.g., FHE-BMM). These issues can be categorized into two types of redundant data movement: (1) **redundant homomorphic data movement** and (2) **redundant transfer** of its corresponding *rotation keys* (i.e., a type of evaluation key).

When executing server-side FHE3DMM, we simplify by considering only the case where sub-MMs involve square matrices (i.e., $\texttt{CGEMM}(d, d, d)$). Additional pre- or post-operations required in the rectangular matrix case face similar issues.

Fig. 5(a) illustrates the computational details of `CGEMM`, where internal homomorphic data movements are divided into two stages: `RotateAlign` and `ShiftByStep` [4]. In enhanced FHE3DMM, after broadcasting the initial data, each processor must perform these internal homomorphic data movements before completing the remaining computations indicated on the right side of the equation in Fig. 5(a). There are considerable redundant data movements in the broadcast group related to each sub-matrices of $A$ and $B$. These redundant operations are also the root cause of redundant transmissions during client-side public key broadcasting.

## V. The FHE4DMM Algorithm

Improper SIMD homomorphic computations (§ IV-C) can waste network bandwidth, storage, and computational resources. Hence, FHE4DMM builds on enhanced FHE3DMM to eliminate the redundancy caused by additional homomorphic data movement compared to traditional 3DMM.

### A. Server-Side 4D Optimization

For simplicity, we consider cases where sub-MMs involve square matrices. Redundant computations can be eliminated through a *MapReduce*-like process (see Fig. 5(b)). Similar processes can be easily extended to additional pre- or post-operations required in the rectangular matrix case. Specifically, FHE4DMM refines Step 2 of FHE3DMM (§ III-A) into the following sub-steps:

**Step 2.1. Mapping Phase:** Processor group $P_{i:t}$ evenly distributes the computations in `RotateAlign` on $\mathcal{A}^*_{i,t}$ (cf. Algorithm 3). **Aggregation Phase:** Each processor in the group reduces local results, followed by a global reduction by `all-reduce`. Similarly, processor group $P_{:jt}$ undertakes analogous procedures for $\mathcal{B}^*_{t,j}$.

**Step 2.2.** The homomorphic data movements in `ShiftByStep` are computed collaboratively like Step 2.1. All intermediate results are shared by `all-gather` to conclude the remaining computations.

[4] The semantic of `ROT` here is in the context of hypercube encoding, which indicates the data movement along specific dimensions of plaintext slots.

**Algorithm 4:** FHE-BMM with 4D optimization

```
Input: $A^*_{[b_m,b_k]}, B^*_{[b_k,b_n]}$: BIR($AB$)
Output: $C^*_{[b_m;b_n]}$ : $C^* = $ Enc($(AB)_{[b_m;b_n]}$)
$A' \leftarrow$ RotateAlign($A^*, \hookleftarrow$)
$B' \leftarrow$ RotateAlign($B^*, \Uparrow$)
for $r = 0$ to $R - 1$ do    ▷ $R = \min\{\frac{m}{b_m}, \frac{n}{b_n}, \frac{k}{b_k}\}$
    for $t = 0$ to $b_k - 1$ do
        for $i = 0$ to $b_m - 1$ do ▷ kept in memory
            $A'_{i,t,r} \leftarrow$ ROT($A'_{i,t}, \hookleftarrow, r$)    ▷ $r$: offset
        for $j = 0$ to $b_n - 1$ do ▷ kept in memory
            $B'_{t,j,r} \leftarrow$ ROT($B'_{t,j}, \Uparrow, r$)
        for $i = 0$ to $b_m - 1$ do
            for $j = 0$ to $b_n - 1$ do
                $C^*_{i,j} \leftarrow$ ADD($C^*_{i,j}$, MULT($A'_{i,t,r}, B'_{t,j,r}$))
```

When each processor runs standard FHE-BMM locally in Step 2.1, performing `all-gather` on all encrypted sub-matrices at once incurs a substantial memory burden. Processing sub-matrices individually fails to eradicate local redundancy. Therefore, FHE4DMM adopts loop interchanging to optimize the workflow of Algorithm 1, as shown in Algorithm 4[5]. `RotateAlign` does not involve `all-gather`, only Step 2.2 requires fine-tuning. For `ShiftByStep`, FHE4DMM conducts $b_k R$ rounds to batch these redundant homomorphic data movements (cf. the highlighted blue part in Algorithm 4) collaboratively before `all-gather`. We later (§ VI-A3) elaborate on the practical parallel strategies, where some loops can be swapped or merged.

### B. Client-Side FHE4DMM

The FHE4DMM alters the computation and communication costs of enhanced FHE3DMM, necessitating adjustments to the client-side setup for greater efficiency.

*1) Rotation Key Pruning:* Existing FHE schemes handle `ROT`s with different offsets using different rotation keys. In contrast to enhanced FHE3DMM, which broadcasts a complete set of rotation keys, server-side 4D optimization (§ V-A) eliminates redundant `ROT`s. This allows the client to scatter only the necessary rotation keys to the servers, thereby reducing client-side data transfers and server-side memory requirements of FHE4DMM.

*2) Adjustments on Data Partitioning:* Under BIR, data partitioning includes (1) OBL generation and (2) task scheduling.

**Optimal Block Layout.** FHE4DMM merely fine-tunes Algorithm 2 to achieve a more favorable block layout for near-optimal computations, rather than solving a completely precise optimization problem. Specifically, for Case (2) in Algorithm 2 (i.e., at least one among $b_m$ and $b_n$ can be dynamically adjusted), FHE4DMM employs a selection that minimizes $b_m + b_n$ to reduce computations.

TABLE III
INTUITIVE COMPARISON OF FHE-PGEMM

| Algorithm† | T3D (§ III-A) | [1] | E3D (§ IV) | 4D (§ V) |
|---|---|---|---|---|
| Using Packing | ✓ | ✓ | ✓ | ✓ |
| FHE Parameter-adaptive | ✗ | ✓ | ✓ | ✓ |
| Partitioning | 3D | Diagonal | Standard | |
| Matrix Class | All | 3 classes | All | |
| Redundancy‡ | G | L,G | L,G | None |
| Scheduling⋆ | - | - | 3D | 4D |

X → Y: Y is an extension of X    X ⇢ Y: X is a cause of Y

† T3D, E3D, and 4D stand for **T**rivial FHE**3D**MM, **E**nhanced FHE**3D**MM, and FHE**4D**MM, respectively.

‡ "G" indicates redundancy within a processor **G**roup, while "L" indicates redundancy on a **L**ocal processor.

⋆ Here, 3D and 4D refer to finding the processor grid that minimizes $W^*_{\text{3D}}$ in Eq. (5) and $W^*_{\text{4D}}$ in Eq. (6), respectively.

**Task Scheduling.** Apart from broadcasting and reduction in the original 3DMM phase, each processor in the group $P_{:j:}$ engages in `all-reduce` and `all-gather`, with a total data volume of $b_m b_n (R+1) \cdot H$, respectively. Similar to the analysis of $P_{i::}$, FHE4DMM replaces the optimization objective of the processor grid partitioning in Algorithm 3 with the minimization of the following:

$$W^*_{\text{4D}} = \{p_n b_m b_k \cdot [1 + (0.5 \cdot R + 1) \cdot \sigma(p_n)] + p_m b_k b_n \cdot [1 + (0.5 \cdot R + 1) \cdot \sigma(p_m)] + p_k b_m b_n\} \cdot H, \tag{6}$$

where $\sigma(x) = \min\{x - 1, 1\}$. The $\sigma(\cdot)$ term is used for zero-one normalization, which cancels additional communication costs term when `RotateAlign` and `ShiftByStep` phases are completely degenerated to local computation.

### C. Complexity Analysis of FHE4DMM

Before presenting the formal analysis, we summarize our observations of FHE-PGEMM in Table III. The OBL(§ IV-A) is identified under standard partitioning, achieving the same computations as [1], but capable of handling arbitrary matrix dimensions. The diagonal partitioning renders scheduling based on data transfer areas of MM meaningless. Therefore, we consider [1] a weaker version of enhanced FHE3DMM.

We assume that butterfly network collectives for communication costs analysis in [32] are optimal or near-optimal in the $\alpha$-$\beta$ model [29] ($\alpha$ signifies network latency and $\beta$ represents the time for moving a word between processors). The costs of these collective operations used are listed here, where $w$ is the message size, and $p$ is the number of processors involved:

$$T_{\texttt{broadcast}}(w,p) = \alpha(\log(p) + p - 1) + 2\beta \cdot w \cdot \frac{p-1}{p},$$

$$T_{\texttt{all-gather}}(w,p) = \alpha \log(p) + \beta \cdot w \cdot \frac{p-1}{p},$$

$$T_{\texttt{reduce/all-reduce}}(w,p) = 2\alpha \log(p) + 2\beta \cdot w \cdot \frac{p-1}{p}.$$

[5] For simplicity, Algorithm 4 breaks down `CGEMM` but hides some `PMULT`s, which are also included in homomorphic data movements.

TABLE IV
MEMORY REQUIREMENTS (MR) OF E3D AND 4D IN TABLE III

| Algorithm | MR of ciphertexts ($M$) | # of rotation keys |
|---|---|---|
| E3D | $(\frac{b_m b_k}{p_m p_k} + \frac{b_n b_k}{p_n p_k} + \frac{b_m b_n}{p_m p_n}) \cdot H$ | $O(\frac{k}{b_k})$ |
| 4D | $M_{3\text{D}} + (\frac{b_m}{p_m} + \frac{b_n}{p_n}) \cdot H$ | $O(\frac{1}{p} \cdot \frac{k}{b_k})$ |

Assuming the OBL $[b_m, b_n, b_k]$ has been attained, the optimal processor grid $p = p_m \times p_n \times p_k$ can be achieved by minimizing Eq. (6). We can thus derive the communication latency $L$ of FHE4DMM as follows:

$$L = \alpha \cdot [2 \log (p_m p_n) + (p_m - 1) + (p_n - 1) + 2 \log p] + 2\beta \cdot \frac{W^*_{4\text{D}}}{p}. \tag{7}$$

Note that $[p_m, p_n, p_k]$ is determined by minimizing $W^*_{4\text{D}}$ in Eq. (6), which tends to minimize $p_m$ and $p_n$.

FHE4DMM primarily reduces the number of ROTs and PMULTs. In existing approaches, these operations on $A^*$ or $B^*$ are performed several times equivalent to the number of those within a single CGEMM multiplied by $\frac{b_m b_n b_k}{p}$. **Presently, this factor has been reduced to $\frac{b_m b_k}{p_m p_k}$ for $A^*$ and $\frac{b_n b_k}{p_n p_k}$ for** $B^*$. This optimization alters the memory demands of FHE-PGEMM, as summarized in Table IV.

## VI. IMPLEMENTATIONS OF FHE4DMM

The implementation relies on hybrid parallelism integrating Message Passing Interface (MPI) and Open Multi-Processing (OpenMP). This section focuses on optimizing CGEMM and provides several workarounds to facilitate MPI in FHE.

### A. Efficient Single-Node CGEMM

In response to performance bottlenecks observed during experiments, we systematically optimized local computation from both algorithmic and hardware resource utilization levels.

*1) Encoding Reuse:* In the evaluation of the CGEMM with large FHE parameters (i.e., $N \approx 2^{16}, \log q = 614$), we observed that encoding accounts for **59.48%** of the total execution time. In our scenarios where encoded constants only consist of 0s or 1s, the encoding process can be regarded as polynomial additions (i.e., a special case of the Chinese Remainder Theorem (CRT) for polynomials) in BFV/BGV. Specifically, each non-zero plaintext slot contributes to one polynomial addition, implying $N$ modular additions.

Especially for SELECT operation [10], two complementary constant vectors (i.e., $S$ non-zero plaintext slots in total) are required, leading to $O(NS)$ modular additions. The encoding on the fly before each PMULT would result in significant overhead when parameters are large. Therefore, we decouple the encoding process from PMULTs, treating it as a pre-computation phase independent of CGEMM.

The encoding phase for any given CGEMM is executed only once. To enhance parallelism, the encoding phase is divided into encoding and combination processes (see Fig. 6) to optimize the reuse of polynomial constants. For instance, our application initially performs row- or column-wise pre-encoding based on matrix dimensions. These pre-encoded constants are then reassembled according to the specific constants required. This optimization reduces the number of modular additions in CGEMM$(d, d, d)$ from $O(d^3 \cdot N)$ to $O(d^2 \cdot N)$.

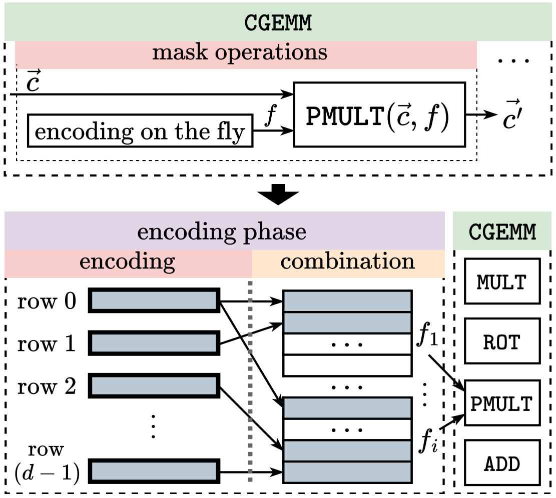


Fig. 6. Encoding reuse optimization. The encoded constants are denoted by polynomials $f$ and $f_i$. A ciphertext is a polynomial pair, denoted by $\vec{c}$.

*2) Parallel BSGS:* We employ BSGS optimization for CGEMM and utilize OpenMP's task dependency feature to fully overlap the independent computations.

*3) 4D Optimization for Shared Memory:* FHE4DMM, through data reuse, diminishes redundant computations and thus imposes a minimum requirement on memory size. This minimum demand (§ V-C) is sufficient for multi-machine parallelism and is typically much less than the available memory of modern machines. Current high-performance applications, including FHE4DMM, extensively employ multi-machine and multi-core hybrid parallel mechanisms.

The available shared memory directly influences the algorithm's parallel strategy. FHE4DMM employs two distinct strategies based on the size of extra memory held by each processor: (1) **Limited Memory:** no loop parallelization is conducted. Instead, only the results of either the loop at Line 5 or Line 7 in Algorithm 4 are preserved in memory. While the other loop is merged into Lines 9∼10 as an outer loop and computed on the fly. (2) **Large Memory:** building upon the "limited memory" parallel strategy, one or both of the loops at Lines 3 and 4 in Algorithm 4 is parallelized based on the hardware resources available.

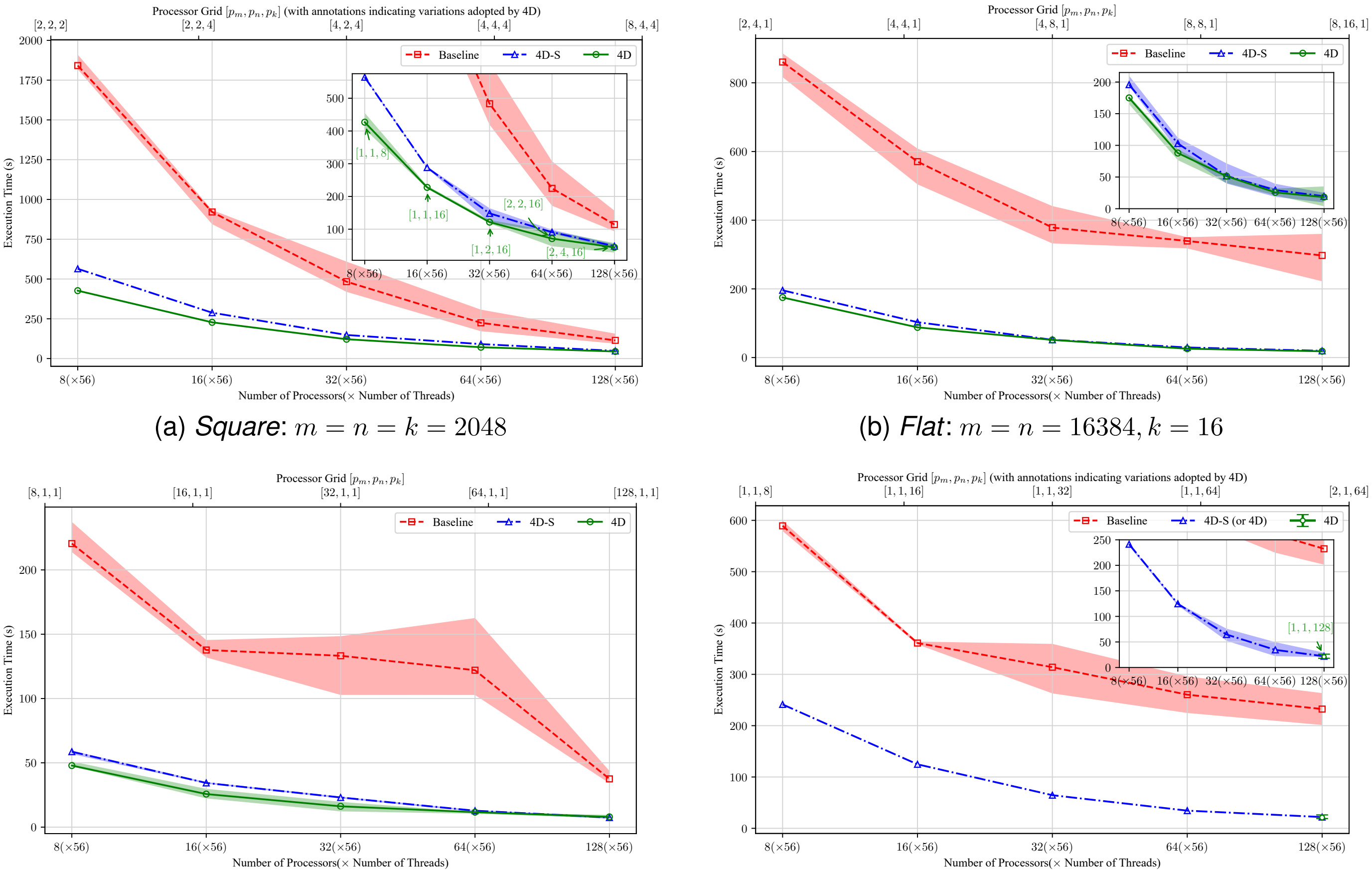
### B. MPI-enabled FHE

We implement FHE4DMM based on serializable ciphertexts without involving distributed polynomial operations.

**Adaptation of MPI Collectives.** It is difficult to reduce ciphertexts with dynamically varying lengths through existing MPI APIs. Therefore, FHE4DMM employs binary-tree all-to-all reduction based on basic MPI_Send/Recvs. For all-gather, FHE4DMM introduces a proposed MPI_Allreduce to ascertain the maximum message length,

TABLE V
CONFIGURATIONS AND FHE PARAMETERS. HERE, $S_o = 2^{14}$ AND NOISE CAPACITY IS SUFFICIENT FOR AT LEAST ONE CGEMM.

| Type | $(m,$ | $n,$ | $k)$ | Case for Algor. 2 | $\texttt{OBL}(m,n,k)$ | plaintext slot shape | $\log t$ | $\log q$ | $N$ | Security ($\lambda$) |
|---|---|---|---|---|---|---|---|---|---|---|
| *Square* | (2048, | 2048, | 2048) | Case (1.1) | $[16,16,16]$ | $128 \times 128$ | 31 | 503 | 19072 | 128.358 |
| *Flat* | (16384, | 16384, | 16) | Case (2.2) | $[128,128,1]$ | | | | | |
| *LargeM* | (1048576, | 64, | 16) | Case (2.2) | $[1024,4,1]$ | $1024 \times 16$ | 31 | 503 | 19456 | 128.012 |
| *Slim* | (16384, | 16, | 16384) | Case (1.2) | $[16,1,1024]$ | | | | | |



(a) *Square*: $m = n = k = 2048$

(b) *Flat*: $m = n = 16384, k = 16$

(c) *LargeM*: $m = 1048576, n = 64, k = 16$

(d) *Slim*: $m = k = 16384, n = 16$

Fig. 7. Strong scaling tests of *Baseline*, *4D-S* and *4D*. The minimal, mean (marked line), and the maximal execution time at least in five runs are shown. The top right subfigure is a zoomed-in view of *4D-S* and *4D*.

subsequently padding the message buffer to the same length before actually invoking `MPI_Allgather`.

**Packing MPI Operations.** Considering the additional overhead introduced by each MPI operation, particularly evident when each processor holds multiple encrypted blocks, FHE4DMM utilizes message-packing techniques to alleviate the impact of excessive ciphertexts. Specifically, we pack all bitstreams representing these encrypted blocks into a unified message, supplemented with necessary auxiliary data for unpacking. This approach significantly reduces the actual number of MPI operations within FHE4DMM.

### C. Variants for Different FHE Schemes

Given that only HElib [10] supports BGV with hypercube encoding, our implementation of FHE4DMM is based on it. Specifically, we implement both BGV and CKKS variants. For CKKS, due to the necessity of consuming moduli after processing non-scalar `PMULT`, there are slight differences in implementing homomorphic data movement.

## VII. EXPERIMENT

The experiments (§ VII-A and § VII-B) were conducted on an AMD EPYC 7452 cluster equipped with an InfiniBand network boasting 56Gbps bandwidth. Each processor features $2 \times 16$ cores (at 2.35GHz), coupled with 256GB DDR4 memory in total. Consequently, every machine within the cluster can concurrently execute up to 64 threads. All performance tests utilized the hypercube-encoding version (§ VI-C) with the appropriate number of threads.

**Infrastructure Details.** All implementations were compiled using GCC 12.2.0. Intel MPI 2017.5.239 was chosen for the MPI backend. Multi-core programming was facilitated using GNU OpenMP 4.5. HElib 2.3.0 was utilized for the implementation of FHE schemes.

### A. Performance of FHE4DMM

***Baseline.*** Some intuitive comparisons between [1] (*TDSC-Baseline*) and FHE4DMM are listed in Table III. We reproduce *TDSC-Baseline* in the same environment as FHE4DMM due to several reasons: (1) *TDSC-Baseline* is implemented based on CKKS in SEAL. It is difficult to achieve fair comparisons across different libraries and schemes. (2) FHE4DMM enhances individual CGEMMs through encoding reuse and thread-task overlap. These optimizations are also applied to *Baseline* to ensure that the focus remains on the core enhancements introduced by FHE4DMM. (3) *TDSC-Baseline* supports only three matrix shapes, which does not encompass all problem dimensions tested here. We employ enhanced FHE3DMM to address the cases not supported by *TDSC-Baseline*.

**Comparison Targets and Test Cases.** We compare two versions of FHE4DMM (*FHE**4DMM-S**hared* and *FHE**4DMM***) with *Baseline*: (1) ***4D-S*** uses the same processor grid as *Baseline* and eliminates “L” redundancy (cf. Table III). (2) ***4D*** further eliminates “G” redundancy of *4D-S* and uses 4D scheduling (cf. Table III).

We test four classes of matrix dimensions, covering various cases of Algorithm 2: (1) *Square*: $m = n = k$. (2) *Flat*: $m = n \gg k$. (3) *LargeM*: $m \gg n \approx k$. (4) *Slim*: $m = k \gg n$. Table V lists the specific test cases and configurations.

**Scalability Tests.** We conducted tests employing 8 to 128 nodes, with each processor initiating one MPI process. Fig. 7 illustrates the server-side execution time of three methods. Time measurements ensure synchronization before and after execution through explicit barriers. All tests ensure full parallelism: *Baseline* uses nested parallelism, and FHE4DMM selects the appropriate parallel strategy.

The two versions of FHE4DMM exhibit good parallel scalability on all problem classes in Table V. *Baseline* performs poorly for *LargeM* and *Slim*, especially when utilizing more processors. Moreover, the elimination of redundant computations through loop interchanging (§ V-A) yields substantial performance enhancements for FHE4DMM. Compared to *Baseline*, FHE4DMM achieves a maximum speedup of 4.31x (averaging 3.61x) for *Square*, 16.62x (averaging 9.72x) for *Flat*, 10.59x (averaging 5.88x) for *LargeM*, and 14.44x (averaging 13.09x) for *Slim*.

To demonstrate the scalability of FHE4DMM more clearly, we conducted experiments on a slightly smaller scale *Square* and *Slim*, employing parameter configurations identical to the same matrix classes in Table V. The experimental results depicted in Fig. 8 show that FHE4DMM achieves a **nearly linear speedup**. Furthermore, for *Square*, the workload of a single machine in Fig. 8 coincides precisely with the workload distributed to each machine in the 8-node case of Fig. 7, resulting in similar execution time. This demonstrates that FHE4DMM has **good weak scaling performance**.

Given the economical cost associated with testing larger matrices, we limited our experiments with 256 machines to *Square* ($m = n = k = 4096$) and *Slim* ($m = k = 32768, n = 16$). Similarly, we employ parameter configurations identical to the corresponding matrix classes in Table V. The outcomes of these experiments are presented in Table VI.

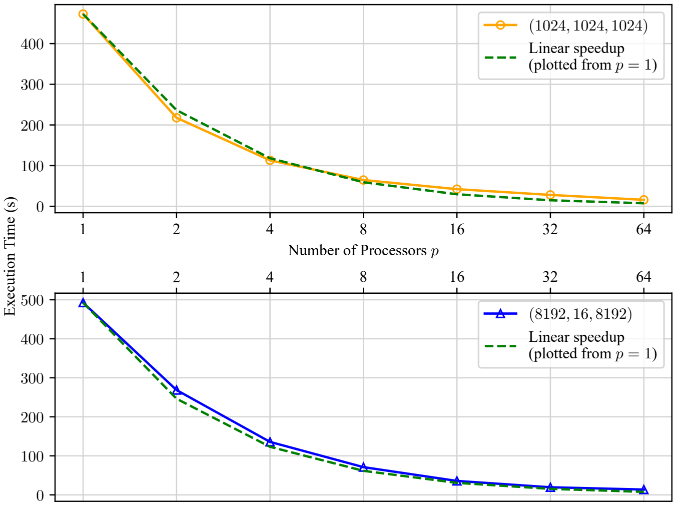


Fig. 8. Strong scaling tests of FHE4DMM.

TABLE VI
LARGE-SCALE TESTS OF FHE4DMM ON 256 MACHINES

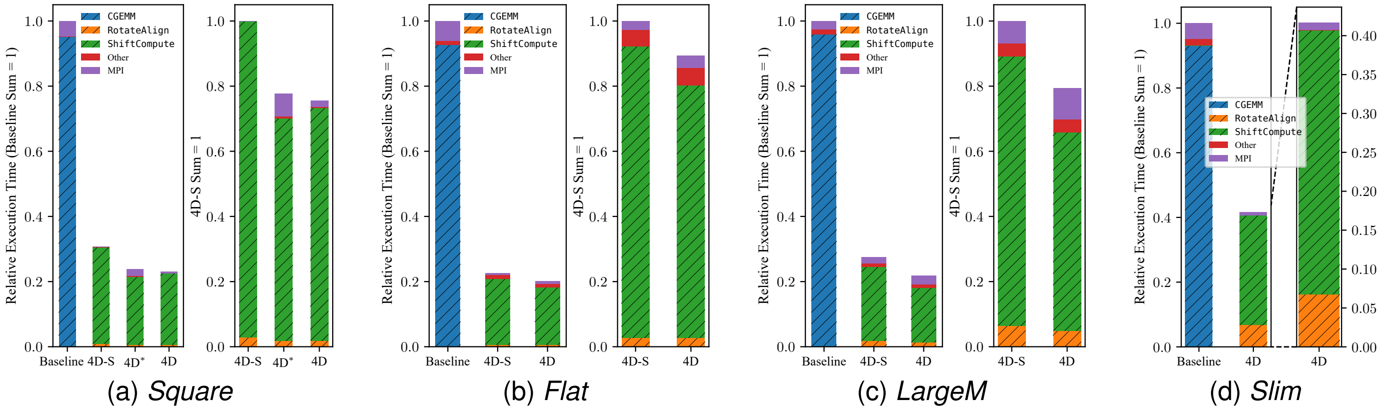
| $(m, n, k)$ | Local Computation (s) | MPI (s) | Total (s) |
|---|---|---|---|
| (4096, 4096, 4096) | 171.979 | 0.255 | 172.234 |
| (32768, 16, 32768) | 65.1457 | 10.9836 | 76.1293 |

**Performance Enhancement Analysis.** Fig. 9 presents normalized runtime breakdowns of Fig. 7. The hatched entries in other versions notably diminish compared to those in *Baseline*, signifying that FHE4DMM eliminates considerable redundant homomorphic data movement. Comparisons between *4D-S* and *4D* in other test cases reveal that the introduction of minimal communication relative to the overall runtime reduces numerous redundant computations within the broadcast group.

In Fig.9(a), an additional comparative experiment has been incorporated to elucidate the impacts of 4D scheduling. The block layout per processor is $[8, 8, 8]$ (i.e., $[\frac{b_m}{p_m}, \frac{b_n}{p_n}, \frac{b_k}{p_k}]$) for both *4D-S* and *4D*$^*$. For the former, the total number of ROTs and PMULTs is equivalent to the necessary number of these operations for $A$ (or $B$) within CGEMM multiplied by $(8 + 8) \cdot 8 = 128$. In contrast, the latter eliminates “G” redundancy, reducing this factor to $(8 + 8) \cdot 8/2 = 64$ (as $p_m = p_n = 2$) through additional communication. For *4D*, the block layout per processor becomes $[16, 16, 2]$, with the aforementioned factor related to homomorphic data movement amounting to $(16 + 16) \cdot 2 = 64$ and fewer additional communication costs. The increased somewhat computation cost observed in *4D* compared to *4D*$^*$, arises from adjustments on parallel strategies (§ VI-A3) at the implementation level.

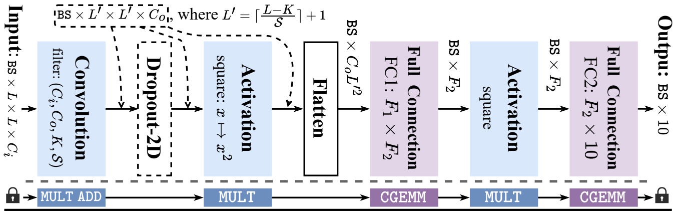
### B. Real-World Applications of FHE4DMM

We employ a basic convolutional neural network (CNN) to evaluate the practical performance of FHE4DMM.



(a) *Square* (b) *Flat* (c) *LargeM* (d) *Slim*

Fig. 9. Normalized execution time breakdowns of *Baseline*, *4D-S* and *4D* with 8 processors in Fig.7. The stacked entries with hatch markers represent local computation (where CGEMM = RotateAlign + ShiftCompute and ShiftCompute = ShiftByStep + MULTs). The MPI entry includes time spent in communication and synchronization. *4D** keeps the same processor grid as *4D-S*. For *Slim*, *4D* mirrors *4D-S*, so only a zoomed-in view is provided.



| Dataset | $L$ | $C_i$ | $C_o$ | $K$ | $\mathcal{S}$ | $L'$ | $F_1(= C_o L'^2)$ | $F_2$ |
|---|---|---|---|---|---|---|---|---|
| EMNIST | 28 | 1 | 4 | 7 | 3 | 8 | 256 | 64 |
| CIFAR-10 | 32 | 3 | 64 | 5 | 3 | 10 | 6400 | 128 |

Fig. 10. Five-layer neural network (BS: batch size. $C_i, C_o$: input and output channels. $K, \mathcal{S}$: filter size and filter stride. $F_1, F_2$: feature dimensions)

*1) CNN Architecture and Dataset:* A simple 5-layer network is used for the Extended MNIST (EMNIST [33]) and CIFAR-10 [34], both 10-class datasets. Fig. 10 illustrates the network architecture, with the above showing the model we used for training. Layers are labeled with input and output dimensions between them. The highlighted boxes below indicate the homomorphic operations required for secure outsourced inference. The table beneath Fig. 10 lists the network's detailed parameters. These two datasets represent different types of large-scale applications: (1) Predicting EMNIST involves smaller matrix dimensions, which we scale up by increasing the batch size BS, thus transforming it into large-scale MM and significantly enhancing outsourcing throughput. (2) For CIFAR-10, this simple network contains only one convolutional layer, leading to large matrix dimensions in the fully connected layers, which we consider as the application itself requiring large-scale MM.

**Remark** (Practicality)**.** We emphasize that a key innovation of FHE4DMM lies in its reuse optimization methods in SIMD homomorphic computation, which, if neglected, will significantly impact the practical latency. These optimizations include reusing homomorphic data movements (§ V-A) and encoded constants (§ VI-A2), providing valuable insights for broader FHE-based applications. Using a 5-layer network allows for parameter settings that avoid *bootstrapping*, enabling us to focus on FHE-PGEMM primarily. Indeed, the capability of this network is limited, achieving 98.7% accuracy on EMNIST but only 61% on CIFAR-10. It is noteworthy that FHE4DMM benefits from the implementation of more complex networks. For example, we can batch StC and CtS in CKKS bootstrapping [35], treating them as plaintext-ciphertext MM. This approach enables us to perform bootstrapping collaboratively while avoiding redundant rotation keys.

*2) Applying FHE4DMM to Practical Applications:* The OBL fixes the data layout for entire homomorphic computations. Therefore, the other layers in Fig. 10 must adhere to the input matrix format of the fully connected layers, which dictates how the client packs the raw data before encrypting.

FHE4DMM supports consecutive MMs. However, deriving the OBL from any two pairs of matrices in the matrix chain leads to varying homomorphic data movements. Fortunately, this optimization problem resembles the *matrix chain multiplication* (MCM) algorithm. Based on the traditional MCM, we can efficiently calculate the OBL for consecutive MMs in a greedy way, rather than enumerating exponentially many cases. The complete algorithm is provided in Appendix C.

*3) Secure Outsourced Inference:* We implement outsourced inference with encrypted data and model parameters. We uniformly use the hypercube-encoding version of FHE4DMM for testing, requiring conversion from floating-point numbers to fixed-point numbers to fit the underlying BGV scheme.

**Client Setup and Finalization.** To prevent overflow of $t$ due to excessively high fixed-point precision, we employ CRT on raw data. In the 5-layer network for EMNIST and CIFAR-10 (see Fig. 10), the MULT depth of 5 consistently dominates the total noise (specifically $\log q$ is about 620 to 650). Thus, under the condition of ensuring a 128-bit security level, $S_o$ remains around $2^{14}$. Although $L, C_i, C_o, K$ and $\mathcal{S}$ have a slight impact on noise (i.e., only increasing the ADD depth), they considerably affect the accumulation of plaintext values. By setting $\log t = 31$ and scaling by $2^{10}$, a 4-layer CRT suffices for prediction on both datasets without loss of accuracy.

Guided by § VII-B2, we must implement convolution using element-wise operations in the BS × $F_1$ data format. The client thus must pack and encrypt its input in this format before distributing ciphertexts. Packing (i.e., data partitioning) should be determined by the matrix dimensions involved in the fully connected layers. Our application involves only two MMs, allowing us to enumerate the optimal solution. For all tested batch sizes BS, the OBLs for the EMNIST and CIFAR-10 datasets are $[1, \frac{\mathtt{BS}}{1024}, 4]$ and $[1, \frac{\mathtt{BS}}{128}, 50]$ respectively, with plaintext slot shapes $16 \times 1024$ and $128 \times 128$.

Upon receiving secure outsourced computation results (i.e., the encrypted output in CRT format), the client decrypts them with the private key. Subsequently, the client retrieves the predicted labels from the clear data.

**Performance Evaluation of Outsourcing.** We utilize 64 processors with 64 threads to test the performance under various BS settings. The matrices involved in our applications exceed the scope addressed by [1] (which also do not account for further optimization opportunities from consecutive MMs). Therefore, we selected its enhanced version, E3D (cf. Table III), as our baseline. Compared to our method, *Baseline* includes only two additional redundancies: homomorphic data movements and encoded constants. We use the same parallelism strategies as in § VII-A. Fig. 11 presents a runtime analysis comparing two methods during outsourced inference.

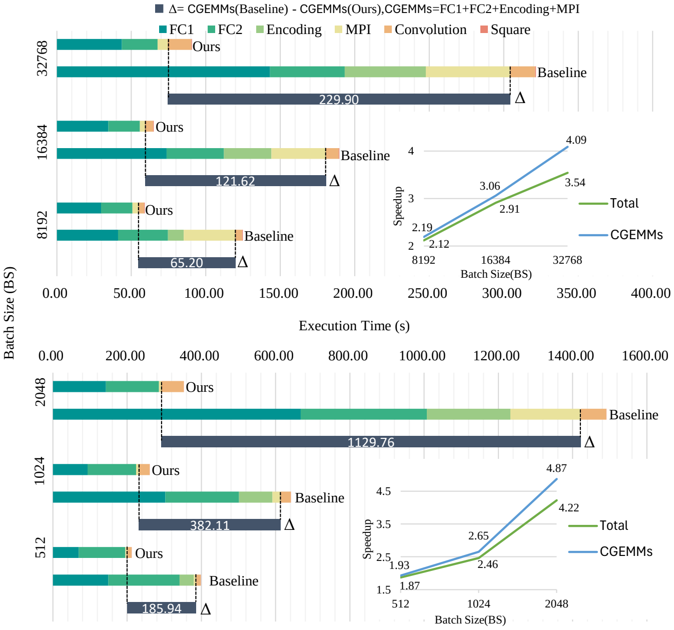


Fig. 11. Execution time breakdowns and acceleration of secure outsourced inference on EMNIST (top) and CIFAR-10 (bottom) with 64 processors.

Note that the encoding time in our method is less than 2 seconds, which is almost negligible The high latency of MPI in *Baseline* confirms its poor scalability (§VII-A). Comparisons of different BS show that our method achieves greater speedup for larger data scales. Specifically, ours achieved a maximum speedup of 3.54x on EMNIST and 4.22x on CIFAR-10.

## VIII. Related Work

FHE has spurred extensive research in secure outsourced computation, with various approaches to secure matrix multiplication. We observed this diversity stems from the underlying mathematical frameworks of FHE schemes (e.g., RLWE), which permit a certain degree of data encoding. Some recent studies [36], [37] employ coefficient encoding to avoid costly homomorphic data movement. However, this method has a low encoding capacity and only supports specific calculations, necessitating the integration of other cryptographic techniques (e.g., secret sharing), which requires additional communication. This paper focuses on FHE-based matrix multiplication for non-interactive secure outsourcing, opting for SIMD encoding. Similar works [1], [30], [38] have two main characteristics: (1) the difficulty of avoiding homomorphic data movement and (2) the significant impact of specific applications on data layout (e.g., privacy-preserving CNN inference requires operations involving flexible data layouts, such as convolution and flattening).

Numerous studies [1], [17], [30], [31], [39]–[41] have focused on optimizing matrix multiplication which packs the entire matrix (referred to as CGEMM in this paper). However, partitioning the raw data remains unavoidable in practical applications as noted in § III-B. Several implementations of distributed matrix multiplication have been proposed in [1], [42], all of which are constrained to parallelizing specific applications. In other words, None of these works delve into the combination of distributed algorithms and FHE. While our method offers maximum generality, it may not be directly suitable for all applications. Nevertheless, it addresses common, often overlooked issues that significantly impact the practical latency of computations on large-scale encrypted data.

## IX. Conclusion

This paper presents FHE4DMM, a low-latency large-scale secure matrix multiplication based on fully homomorphic encryption with two encrypted matrices. Fundamentally, FHE4DMM introduces an HE-friendly data partitioning scheme, which comprehensively considers FHE parameters (i.e., FHE parameter-adaptive). This partitioning, guided by mathematical methods, aims to minimize the objective function relevant to computation while facilitating scheduling for achieving optimal or near-optimal communication costs.

The key insight of FHE4DMM highlights new challenges faced in parallel computing within FHE-based secure outsourcing scenarios, particularly for parallelized SIMD homomorphic computation. First, and most importantly, it is crucial to consider the reuse of costly homomorphic data movements. If data parallelism is confined solely to the raw data layer, it can lead to substantial redundant computation on encrypted data both locally and across processors, resulting in significant overhead. Second, the reuse of encoded constants required for SIMD homomorphic operations (e.g., PMULT) is essential. An evident observation is that the encoded constants required for each algorithm are typically fixed. The encoding process should be separated from homomorphic operations, and its results should be reused. A deeper optimization is recognizing the repetitive computations in the encoding process, such as within CGEMM, where the encoded constants are reusable across matrix rows or columns. Combined with the techniques inspired by these insights, FHE4DMM demonstrates practical efficiency for any matrix dimension and processor number in real-world applications.

## ACKNOWLEDGMENTS

This work was supported in part by National Key Research and Development Program of China (Grant No. 2022YFB4501502) and National Natural Science Foundation of China (Grant No. 62372202).

## REFERENCES

[1] Z. Huang, C. Hong, C. Weng, W. Lu, and H. Qu, "More efficient secure matrix multiplication for unbalanced recommender systems," *IEEE Trans. Dependable Secur. Comput.*, vol. 20, no. 1, pp. 551–562, 2023.

[2] C. Gentry, "Fully homomorphic encryption using ideal lattices," in *Proceedings of Symposium on the Theory of Computing (STOC 2009), Bethesda, MD, USA, May 31 - June 2, 2009*, pp. 169–178.

[3] J. Fan and F. Vercauteren, "Somewhat practical fully homomorphic encryption," *IACR Cryptol. ePrint Arch.*, pp. 1–19, 2012.

[4] Z. Brakerski, C. Gentry, and V. Vaikuntanathan, "(Leveled) fully homomorphic encryption without bootstrapping," in *Proceedings of Innovations in Theoretical Computer Science (ITCS 2012), Cambridge, MA, USA, January 8-10, 2012*, pp. 309–325.

[5] J. H. Cheon, K. Han, A. Kim, M. Kim, and Y. Song, "A full RNS variant of approximate homomorphic encryption," in *Proceedings of Selected Areas in Cryptography (SAC 2018), Calgary, AB, Canada, August 15-17, 2018*, pp. 347–368.

[6] L. Ducas and D. Micciancio, "FHEW: bootstrapping homomorphic encryption in less than a second," in *Proceedings of 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT 2015), Sofia, Bulgaria, April 26-30, 2015*, pp. 617–640.

[7] I. Chillotti, N. Gama, M. Georgieva, and M. Izabachène, "Faster fully homomorphic encryption: Bootstrapping in less than 0.1 seconds," in *Proceedings of 22nd International Conference on the Theory and Application of Cryptology and Information Security (ASIACRYPT 2016), Hanoi, Vietnam, December 4-8, 2016*, pp. 3–33.

[8] 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 Workshop on Encrypted Computing & Applied Homomorphic Cryptography (WAHC@CCS 2022), Los Angeles, CA, USA, November 7, 2022*, pp. 53–63.

[9] H. Chen, K. Laine, and R. Player, "Simple encrypted arithmetic library - SEAL v2.1," in *Proceedings of 21st International Conference on Financial Cryptography and Data Security (FC 2017), Sliema, Malta, April 7, 2017*, pp. 3–18.

[10] S. Halevi and V. Shoup, "Algorithms in helib," in *Proceedings of 34th Annual Cryptology Conference, Santa Barbara (CRYPTO 2014), CA, USA, August 17-21, 2014*, pp. 554–571.

[11] S. Fan, Z. Wang, W. Xu, R. Hou, D. Meng, and M. Zhang, "TensorFHE: Achieving practical computation on encrypted data using GPGPU," in *Proceedings of IEEE International Symposium on High-Performance Computer Architecture (HPCA 2023), Montreal, QC, Canada, February 25 - March 1, 2023*, pp. 922–934.

[12] Z. Wang, P. Li, R. Hou, Z. Li, J. Cao, X.F. Wang, and D. Meng, "HE-Booster: An efficient polynomial arithmetic acceleration on gpus for fully homomorphic encryption," *IEEE Trans. Parallel Distributed Syst.*, vol. 34, no. 4, pp. 1067–1081, 2023.

[13] H. Yang, S. Shen, W. Dai, L. Zhou, Z. Liu, and Y. Zhao, "Phantom: A cuda-accelerated word-wise homomorphic encryption library," *IEEE Trans. Dependable Secur. Comput.*, vol. 21, no. 5, pp. 4895–4906, 2024.

[14] Y. Zhu, X. Wang, L. Ju, and S. Guo, "FxHENN: FPGA-based acceleration framework for homomorphic encrypted CNN inference," in *Proceedings of IEEE International Symposium on High-Performance Computer Architecture (HPCA 2023), Montreal, QC, Canada, February 25 - March 1, 2023*, pp. 896–907.

[15] N. Samardzic and D. Sánchez, "BitPacker: Enabling high arithmetic efficiency in fully homomorphic encryption accelerators," in *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS 2024), La Jolla, CA, USA, April 27 - May 1, 2024*, pp. 137–150.

[16] J. Kim, S. Kim, J. Choi, J. Park, D. Kim, and J. H. Ahn, "SHARP: A short-word hierarchical accelerator for robust and practical fully homomorphic encryption," in *Proceedings of the 50th Annual International Symposium on Computer Architecture (ISCA 2023), Orlando, FL, USA, June 17-21, 2023*, pp. 18:1–18:15.

[17] L. Zhu, Q.-S. Hua, Y. Chen, and H. Jin, "Secure outsourced matrix multiplication with fully homomorphic encryption," in *Proceedings of 28th European Symposium on Research in Computer Security (ESORICS 2023), The Hague, The Netherlands, September 25-29, 2023*, pp. 249–269.

[18] H. Huang and E. Chow, "CA3DMM: A new algorithm based on a unified view of parallel matrix multiplication," in *Proceedings of International Conference for High Performance Computing, Networking, Storage and Analysis (SC 2022), Dallas, TX, USA, November 13 - 18, 2022*, pp. 28:1–28:15.

[19] R. C. Agarwal, S. M. Balle, F. G. Gustavson, M. V. Joshi, and P. V. Palkar, "A three-dimensional approach to parallel matrix multiplication," *IBM J. Res. Dev.*, vol. 39, no. 5, pp. 575–582, 1995.

[20] R. Dathathri, O. Saarikivi, H. Chen, K. Laine, K. E. Lauter, S. Maleki, M. Musuvathi, and T. Mytkowicz, "CHET: An optimizing compiler for fully-homomorphic neural-network inferencing," in *Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI 2019), Phoenix, AZ, USA, June 22-26, 2019*, pp. 142–156.

[21] S. Halevi and V. Shoup, "Design and implementation of HElib: a homomorphic encryption library," *IACR Cryptol. ePrint Arch.*, pp. 1–42, 2020.

[22] X. Zhang, Q. Wang, and Y. Zhang, "OpenBLAS: An optimized BLAS library," 2011, http://www.openmathlib.org/OpenBLAS/ (Accessed on 2025-01-21).

[23] Intel Corporation, "Intel OneAPI math kernel library," 2017, https://www.intel.com/content/www/us/en/developer/tools/oneapi/onemkl.html (Accessed on 2025-01-21).

[24] NVIDIA Corporation, "CUDA library documentation," 2010, https://docs.nvidia.com/cuda/cublas/index.html (Accessed on 2025-01-21).

[25] V. Strassen, "Gaussian elimination is not optimal," *Numerische mathematik*, vol. 13, no. 4, pp. 354–356, 1969.

[26] G. Kwasniewski, M. Kabic, M. Besta, J. VandeVondele, R. Solcà, and T. Hoefler, "Red-blue pebbling revisited: near optimal parallel matrix-matrix multiplication," in *Proceedings of International Conference for High Performance Computing, Networking, Storage and Analysis (SC 2019), Denver, Colorado, USA, November 17-19, 2019*, pp. 24:1–24:22.

[27] L. Jia, Q.-S. Hua, H. Fan, Q. Wang, and H. Jin, "Efficient distributed algorithms for holistic aggregation function on random regular graphs," *Sci. China Inf. Sci.*, vol. 65, no. 5, pp. 1–19, 2022.

[28] O. Goldreich, *The Foundations of Cryptography - Volume 2: Basic Applications*. Cambridge University Press, 2004.

[29] E. Solomonik, E. C. Carson, N. Knight, and J. Demmel, "Tradeoffs between synchronization, communication, and computation in parallel linear algebra computations," in *Proceedings of 26th ACM Symposium on Parallelism in Algorithms and Architectures (SPAA 2014), Prague, Czech Republic, June 23 - 25, 2014*, pp. 307–318.

[30] X. Jiang, M. Kim, K. E. Lauter, and Y. Song, "Secure outsourced matrix computation and application to neural networks," in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security (CCS 2018), Toronto, ON, Canada, October 15-19, 2018*, pp. 1209–1222.

[31] 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.

[32] R. Thakur, R. Rabenseifner, and W. Gropp, "Optimization of collective communication operations in MPICH," *Int. J. High Perform. Comput. Appl.*, vol. 19, no. 1, pp. 49–66, 2005.

[33] G. Cohen, S. Afshar, J. Tapson, and A. van Schaik, "EMNIST: extending MNIST to handwritten letters," in *Proceedings of International Joint Conference on Neural Networks (IJCNN 2017), Anchorage, AK, USA, May 14-19, 2017*, pp. 2921–2926.

[34] A. Krizhevsky, "Learning multiple layers of features from tiny images," *Technical Report TR-2009, University of Toronto, Toronto, 2009*, pp. 1–60.

[35] J. H. Cheon, K. Han, A. Kim, M. Kim, and Y. Song, "Bootstrapping for approximate homomorphic encryption," in *Proceedings of 37th Annual International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT 2018), Tel Aviv, Israel, April 29 - May 3, 2018*, pp. 360–384.

[36] Z. Huang, W. Lu, C. Hong, and J. Ding, "Cheetah: Lean and fast secure two-party deep neural network inference," in *Proceedings of 31st*

*USENIX Security Symposium (USENIX Security 2022), Boston, MA, USA, August 10-12, 2022*, pp. 809–826.

[37] X. Liu, Z. Liu, Q. Li, K. Xu, and M. Xu, "Pencil: Private and extensible collaborative learning without the non-colluding assumption," in *Proceedings of 31st Annual Network and Distributed System Security Symposium (NDSS 2024), San Diego, California, USA, February 26 - March 1, 2024*, pp. 1–19.

[38] S. Lee, G. Lee, J. W. Kim, J. Shin, and M.-K. Lee, "HETAL: Efficient privacy-preserving transfer learning with homomorphic encryption," in *Proceedings of International Conference on Machine Learning (ICML 2023), 23-29 July 2023, Honolulu, Hawaii, USA*, pp. 19 010–19 035.

[39] S. Halevi and V. Shoup, "Faster homomorphic linear transformations in HElib," in *Proceedings of 38th Annual International Cryptology Conference (CRYPTO 2018), Santa Barbara, CA, USA, August 19-23, 2018*, pp. 93–120.

[40] D. Rathee, P. K. Mishra, and M. Yasuda, "Faster PCA and linear regression through hypercubes in HElib," in *Proceedings of the 2018 Workshop on Privacy in the Electronic Society (WPES@CCS 2018), Toronto, ON, Canada, October 15-19, 2018*, pp. 42–53.

[41] H. Huang and H. Zong, "Secure matrix multiplication based on fully homomorphic encryption," *J. Supercomput.*, vol. 79, no. 5, pp. 5064–5085, 2023.

[42] Q. Li, Z. Huang, W. Lu, C. Hong, H. Qu, H. He, and W. Zhang, "HomoPAI: A secure collaborative machine learning platform based on homomorphic encryption," in *Proceedings of 36th IEEE International Conference on Data Engineering (ICDE 2020), Dallas, TX, USA, April 20-24, 2020*, pp. 1713–1717.

**Yi Chen** received the B.S. degree in computer science from Huazhong University of Science and Technology, Wuhan, in 2022. He is currently pursuing his M.S. degree at the same institution. His current research interests include homomorphic encryption and parallel and distributed computing.

**Qiang-Sheng Hua** (Member, IEEE) received his B.S. and M.S. degrees in computer science from Central South University, Changsha, in 2001 and 2004, respectively, and the Ph.D. degree in computer science from The University of Hong Kong, Hong Kong, in 2009. He is currently a professor with the Huazhong University of Science and Technology, Wuhan. He is interested in the algorithmic aspects of parallel and distributed computing

**Zixiao Hong** received the BS degree in computer science from Wuhan University of Technology, China, in 2022. He is currently working toward the ME degree with the College of Computer, Huazhong University of Science and Technology. His current interest includes distributed acceleration of Fully Homomorphic Encryption.

**Lin Zhu** received the Ph.D. degree in computer science from Huazhong University of Science and Technology, Wuhan, in 2024. His research interests include parallel algorithms and distributed computing.

**Hai Jin** (Fellow, IEEE) is a Chair Professor of computer science and engineering at Huazhong University of Science and Technology (HUST) in China. Jin received his PhD in computer engineering from HUST in 1994. In 1996, he was awarded a German Academic Exchange Service fellowship to visit the Technical University of Chemnitz in Germany. Jin worked at The University of Hong Kong between 1998 and 2000, and as a visiting scholar at the University of Southern California between 1999 and 2000. He was awarded Excellent Youth Award from the National Science Foundation of China in 2001. Jin is a Fellow of IEEE, Fellow of CCF, and a life member of the ACM. He has co-authored more than 20 books and published over 1000 research papers. His research interests include computer architecture, parallel and distributed computing, big data processing, data storage, and system security.