Fast Batch Matrix Multiplication in Ciphertexts

Jung Hee Cheon[1,2], Minsik Kang[3], and Junho Lee[1]

[1] Seoul National University.
{jhcheon,abab9579}@snu.ac.kr
[2] CryptoLab Inc.
[3] Korea Institute for Advanced Study
minsik.kang0503@gmail.com

Abstract. Encrypted matrix multiplication (MM) is a fundamental primitive in privacy-preserving machine learning and encrypted data search, but it remains a significant performance bottleneck. Recently, Bae et al. (Crypto'24) and Park (Eurocrypt'25) introduced novel algorithms for ciphertext–plaintext (CPMM) and ciphertext–ciphertext (CCMM) matrix multiplications. These algorithms reduce encrypted MM operations to plaintext matrix multiplications (PPMM), enabling implementation through highly optimized BLAS libraries. While these reduction-based methods offer significant improvements, their benefit is limited to scenarios where the matrix dimension d is comparable to the ring dimension N in RLWE-based CKKS schemes. As a result, they fall short for matrix multiplications involving small or medium-sized matrices.

We extend the reduction-based CPMM/CCMM into small-sized matrix operations by batching instances. We encode a batch of matrices into a single matrix over algebraic integers, where each entry is obtained by applying the inverse Discrete Fourier Transform to the batch matrix entries at the same position. This encoding enables reductions of encrypted batch MM algorithms to a small number of batch PPMMs, which can be efficiently accelerated by BLAS libraries. Our batch encrypted MM flexibly accommodates diverse matrix dimensions and batch sizes independent of the ring dimension N, thereby extending its applicability to practical real-world settings.

For two $d \times d$ matrices with N/d batches, our batch CPMM and CCMM algorithms achieve $O(d^2N)$ cost, improving over Bae et al.'s $O(dN^2)$ and Jiang et al.'s $O(d^2N \log N)$ (CCS'18). We further extend our techniques to rectangular matrices, achieving $O(dN^2)$ for multiplying a $d \times N$ matrix by an $N \times N$ matrix, improving previous $O(N^3)$ methods. A proof-of-concept implementation validates these improvements: multiplying 128 batches of 64×64 matrices takes 0.20s (CPMM) and 1.08s (CCMM), yielding 240× and 52× speedups over previous methods. For a 64×2048 by 2048×2048 multiplication, our CCMM completes in 7.5s, achieving a 29× speedup compared to Park's algorithm.

1 Introduction

Encrypted Matrix Multiplication (MM) takes as input ciphertext(s) for a matrix $\boldsymbol{M} \in \mathbb{R}^{d_1 \times d_2}$ and a matrix $\boldsymbol{U} \in \mathbb{R}^{d_2 \times d_3}$, either in plaintext or ciphertext form,

and outputs ciphertext(s) for the matrix $\boldsymbol{M} \cdot \boldsymbol{U} \in \mathbb{R}^{d_1 \times d_3}$. When $\boldsymbol{U}$ is provided in plaintext, the operation is referred to as ciphertext–plaintext matrix multiplication (CPMM), while when $\boldsymbol{U}$ is provided as ciphertext(s), it is referred to as ciphertext–ciphertext matrix multiplication (CCMM).

Encrypted MM is a central primitive in privacy-preserving machine learning (PPML), where a server trains or evaluates models on encrypted client data. In particular, large language models (LLMs) such as GPT [38] and LLaMA [41,42] require both CPMM and CCMM as essential building blocks: CPMM for linear transformations with model parameters, and CCMM for pairwise interactions among encrypted representations [35,19,43]. Moreover, encrypted MM arises in encrypted data search, including batch-query processing for privacy-preserving retrieval-augmented generation (RAG) [28] and private information retrieval (PIR) [33,20,29,34]. Beyond these, encrypted MM facilitates federated principal component analysis (PCA) [13] and has applications in smart contracts, healthcare, and finance [31].

To address this need, numerous encrypted matrix multiplication (MM) algorithms have been proposed [23,21,32,15,39,45,14,9,44,22,30,6]. Most of these approaches rely heavily on ciphertext rotations and multiplications, each of which requires expensive key-switching operations. Key-switching is costly because it involves multiple Number-Theoretic Transforms (NTT), which incur $O(N \log N)$ time complexity with a large constant factor. As a result, these algorithms are inefficient both asymptotically and in practice.

Recently, reduction-based encrypted MM algorithms have been proposed. Rather than relying on ciphertext operations, encrypted MM is reduced to unencrypted MM over plaintext (PPMM), thereby lowering the computational overhead to within a single-digit factor compared to plaintext analogues [31,3,36,4]. The key advantage is that reducing to PPMMs allows the use of highly optimized linear algebra libraries such as OpenBLAS [1].

These methods provide significant improvements for large encrypted datasets over prior approaches. However, their applicability is limited, as they are tailored to matrix multiplications where the matrix dimension d is comparable to the ring degree N in RLWE-based FHE schemes, with N typically ranging from 2^{12} to 2^{17}. To mitigate this limitation when d is not comparable to N, Bae et al. [3] propose using Module-LWE ciphertexts [26] to reduce the ring degree from N to d, while preserving security by employing N/d independent secrets of degree-d polynomials. However, their algorithm reduces CPMM between two $d \times d$ matrices to PPMM between $N \times d$ and $d \times d$ matrices, resulting in $O(d^2 N)$ complexity. For CCMM, the MLWE-based approach is not applicable, and one must instead rely on the method of [23], which cannot leverage high-performance BLAS libraries.

Furthermore, real-world applications often require matrix multiplications between rectangular rather than square matrices. For instance, inference with LLaMA2-7B [42] involves highly unbalanced rectangular matrices, such as multiplication between a 128×4096 and a 4096×4096 matrix. Federated PCA [13] likewise requires rectangular MMs across diverse dimensions. To handle such

cases, prior approaches [3,36,4] set the RLWE ring degree N according to the largest matrix dimension d among the rectangular matrices. Since the computational cost is dominated by N, these methods become inefficient, especially for highly unbalanced rectangular instances.

1.1 Our Contributions

Our goal is to design reduction-based CPMM and CCMM algorithms whose complexity is independent of the ring degree N and remains efficient for matrices of moderate dimension. To accommodate the large ring degree required for secure encryption, we consider batch matrix multiplication in the encrypted domain.

Given a batch size k, encrypted MM takes as input ciphertext(s) encrypting families of matrices $\{\boldsymbol{M}_i\}_{i\in[k]} \subset \mathbb{R}^{d_1\times d_2}$ and $\{\boldsymbol{U}_i\}_{i\in[k]} \subset \mathbb{R}^{d_2\times d_3}$, where each matrix may be provided in plaintext or ciphertext form. It outputs ciphertext(s) encrypting $\{\boldsymbol{M}_i \cdot \boldsymbol{U}_i\}_{i\in[k]} \subset \mathbb{R}^{d_1\times d_3}$. We first focus on the square case $d_1 = d_2 = d_3 = d$.

Key Ideas. The key challenge is to decouple reduction-based encrypted MM algorithms from the RLWE ring degree N. Existing reduction-based methods interpret the RLWE equation as a matrix equation over the integers by representing the underlying ring elements as coefficient vectors, thereby rigidly tying the row or column dimension d to the ring degree N.

We instead reinterpret RLWE encryption as a matrix equation over algebraic integers. We choose a subring R' of the given RLWE ring R and leverage a module isomorphism $R \cong (R')^d$ of extension degree d, under which the encryption is represented as a d-dimensional matrix equation over R'.

We then encode a batch of $d \times d$ matrices into a single $d \times d$ matrix over R', where each entry is an algebraic integer obtained by applying the inverse Discrete Fourier Transform (DFT) to the entries at the same position across the batch. Under this encoding, matrix multiplication over R' naturally corresponds to batch matrix multiplication. This perspective enables asymptotically optimal reduction-based batch CPMM and CCMM that fully leverage high-performance BLAS libraries, while preserving the underlying RLWE structure and security.

Technical Overview. Let $q \geq 2$, and define $\mathcal{R} = \mathbb{Z}[X]/(X^N + 1)$ and $\mathcal{R}_q = \mathbb{Z}_q[X]/(X^N + 1)$ for a power-of-two integer N. For a matrix dimension $d < N$, we first assume $d \mid N$ and set $k = N/d$. If necessary, we replace d by the smallest d' such that $d \leq d'$ and $d' \mid N$, and embed the matrices into dimension d' by zero-padding. By introducing the indeterminate $Y = X^d$, we obtain

$$\mathcal{R} = \mathbb{Z}[X,Y]/(X^d - Y, Y^k + 1) = \mathcal{R}'[X]/(X^d - Y), \quad \mathcal{R}' = \mathbb{Z}[Y]/(Y^k + 1).$$

This induces a module isomorphism $\mathcal{R} \simeq \mathcal{R}'^d$, given by

$$m = \sum_i m_i(Y)X^i \mapsto (m_0(Y), \ldots, m_{d-1}(Y)) \in \mathcal{R}'^d.$$

Under this isomorphism, a single RLWE ciphertext (b, a) encrypting $m \in \mathcal{R}$ admits the following matrix equation over $\mathcal{R}'$:

$$\boldsymbol{b} + \mathtt{Toep}(\mathsf{sk}) \cdot \boldsymbol{a} \approx \boldsymbol{m} \pmod{q},$$

where $\mathtt{Toep}(\mathsf{sk}) \in {\mathcal{R}'}^{d \times d}$ denotes the matrix whose j-th column corresponds to $X^j \cdot \mathsf{sk}$ under this isomorphism. By concatenating d RLWE ciphertexts column-wise, we obtain a *matrix encryption* over $\mathcal{R}'$:

$$\boldsymbol{B} + \mathtt{Toep}(\mathsf{sk}) \cdot \boldsymbol{A} \approx \boldsymbol{M} \pmod{q}, \tag{1}$$

where $(\boldsymbol{B}, \boldsymbol{A})$ encrypts the matrix $\boldsymbol{M} \in {\mathcal{R}'}^{d \times d}$, each column of which corresponds to an RLWE encryption.

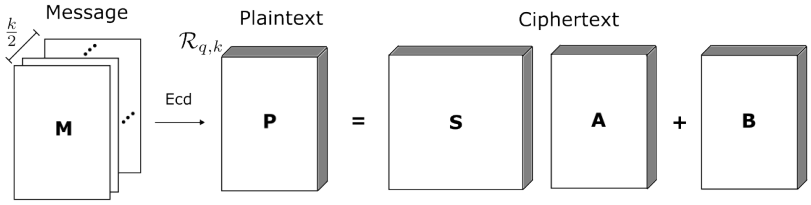


Fig. 1. Overview of the batch matrix encoding and $\mathcal{R}'$-matrix encryption structure

To introduce batch MM, we use the inverse DFT to encode multiple matrices into a single matrix over algebraic integers $\mathcal{R}'$. We define *batch matrix encoding* as follows. Let $\{\boldsymbol{M}_\ell\}_{\ell=1,\ldots,k/2} \subset \mathbb{C}^{d \times d}$ be a collection of matrices, and let $\mathsf{iDFT} : \mathbb{C}^{k/2} \to \mathcal{R}'$ denote the inverse DFT map. We define $\boldsymbol{M} = (m_{i,j}) = \mathsf{MatEcd}(\{\boldsymbol{M}_\ell\}) \in {\mathcal{R}'}^{d \times d}$ by applying iDFT entrywise:

$$m_{i,j} = \mathsf{iDFT}((\boldsymbol{M}_1)_{i,j}, \ldots, (\boldsymbol{M}_{k/2})_{i,j}) \in \mathcal{R}'.$$

Due to the inverse DFT, this encoding satisfies the following homomorphic property. For $\boldsymbol{M} = \mathsf{MatEcd}(\{\boldsymbol{M}_\ell\})$ and $\boldsymbol{U} = \mathsf{MatEcd}(\{\boldsymbol{U}_\ell\})$, we have

$$\boldsymbol{M} \cdot \boldsymbol{U} = \mathsf{MatEcd}(\{\boldsymbol{M}_\ell \cdot \boldsymbol{U}_\ell\}),$$

where $\boldsymbol{M} \cdot \boldsymbol{U}$ and $\boldsymbol{M}_\ell \cdot \boldsymbol{U}_\ell$ are matrix multiplications over $\mathcal{R}'$ and $\mathbb{C}$, respectively. Therefore, batch matrix multiplication reduces to matrix multiplication over $\mathcal{R}'$, which can be efficiently implemented using Fast Fourier Transform (FFT) and high-performance BLAS libraries.

With batch matrix encoding and the matrix encryption structure in place, we obtain batch CPMM and CCMM algorithms. Suppose $(\boldsymbol{B}, \boldsymbol{A})$ is a matrix encryption of $\boldsymbol{M} = \mathsf{MatEcd}(\{\boldsymbol{M}_\ell\}) = (m_{i,j}) \in {\mathcal{R}'}^{d \times d}$ given by (1), where the j-th column of $(\boldsymbol{B}, \boldsymbol{A})$ corresponds to an RLWE encryption of $m_j = \sum_i m_{i,j} X^i \in \mathcal{R}$. Our batch CPMM algorithm proceeds as follows: for a plaintext matrix $\boldsymbol{U} = \mathsf{MatEcd}(\{\boldsymbol{U}_\ell\}) \in {\mathcal{R}'}^{d \times d}$, right multiplication by $\boldsymbol{U}$ yields

$$(\boldsymbol{B} \cdot \boldsymbol{U}) + \mathtt{Toep}(\mathsf{sk}) \cdot (\boldsymbol{A} \cdot \boldsymbol{U}) \approx \boldsymbol{M} \cdot \boldsymbol{U} = \mathsf{MatEcd}(\{\boldsymbol{M}_\ell \cdot \boldsymbol{U}_\ell\}),$$

reducing batch CPMM into two PPMM's.

For batch CCMM, suppose we have two matrix encryptions $(\boldsymbol{B}, \boldsymbol{A})$ and $(\boldsymbol{B}', \boldsymbol{A}')$ encrypting $\boldsymbol{M}$ and $\boldsymbol{M}'$ via batch matrix encoding. We compute a matrix triple $(\boldsymbol{C}_0, \boldsymbol{C}_1, \boldsymbol{C}_2)$ satisfying

$$\begin{aligned}\boldsymbol{M} \cdot \boldsymbol{M}' &\approx (\boldsymbol{B} + \mathtt{Toep}(\mathsf{sk}) \cdot \boldsymbol{A}) \cdot (\boldsymbol{B}' + \mathtt{Toep}(\mathsf{sk}) \cdot \boldsymbol{A}') \\ &\approx \boldsymbol{C}_0 + \mathtt{Toep}(\mathsf{sk}) \cdot \boldsymbol{C}_1 + \mathtt{Toep}(\mathsf{sk})^2 \cdot \boldsymbol{C}_2 \pmod q,\end{aligned}$$

where the quadratic term $\mathtt{Toep}(\mathsf{sk})^2$ is removed by column-wise relinearization. To obtain the required structured form, we introduce a batch ciphertext matrix transpose (CMT) algorithm over $\mathcal{R}'$. Given a column-wise matrix encryption $(\boldsymbol{B}, \boldsymbol{A})$, the algorithm outputs a row-wise matrix encryption $(\boldsymbol{B}', \boldsymbol{A}')$ satisfying

$$\boldsymbol{B}' + \boldsymbol{A}' \cdot \mathtt{Toep}(\mathsf{sk})^T \approx \boldsymbol{B} + \mathtt{Toep}(\mathsf{sk}) \cdot \boldsymbol{A} \pmod q.$$

Thus, CMT rearranges the position of the $\mathtt{Toep}(\mathsf{sk})$ terms in the matrix encryption. Our batch CMT algorithm generalizes the method of [36] to support transposes for matrices of diverse dimensions and, under batch matrix encoding, corresponds to a batch transpose of the underlying matrices. Consequently, the batch CCMM algorithm reduces to four PPMMs and three CMT operations.

We note that our CMT algorithm can be used not only for batch CCMM but also for flexible left or right multiplication by plaintext matrices, as needed. Using this CMT algorithm, we extend our construction to rectangular CPMM and CCMM by partitioning the $d \times N/2$ and $N/2 \times N/2$ matrices into $d \times d$ blocks. Each block-wise product is computed via batch CPMM or CCMM, and the resulting products are summed across blocks by extracting constant terms using the batch CMT algorithm. Our rectangular MM algorithm therefore efficiently handles diverse matrix dimensions beyond the square case.

Main results. We now summarize the performance of our matrix multiplication algorithms. Our algorithms achieve asymptotically optimal complexity while maintaining practical efficiency.

For $d \times d$ matrix multiplications with $k/2 = N/2d$ batches, our batch CPMM and CCMM achieve computational complexities $2k \cdot T_{\mathsf{PPMM}}(d)$ and $4k \cdot T_{\mathsf{PPMM}}(d)$, respectively. For rectangular multiplication of a $d \times N/2$ matrix with an $N/2 \times N/2$ matrix, our rectangular CPMM and CCMM achieve $2k^2 \cdot T_{\mathsf{PPMM}}(d)$ and $4k^2 \cdot T_{\mathsf{PPMM}}(d)$, respectively. Here, $T_{\mathsf{PPMM}}(n)$ denotes the cost of multiplying two $n \times n$ plaintext matrices; in particular, $T_{\mathsf{PPMM}}(d) = d^3$ for the textbook MM algorithm.

These bounds are asymptotically optimal, as the dominant cost consists of PPMMs over $\mathcal{R}'$. The factor k reflects the inherent batching structure and introduces no superfluous dependence on the ring degree N.

Moreover, our construction minimizes the overhead introduced by homomorphic operations. In particular, our algorithms do not rely on rotation operations as in [23] or format conversions between RLWE and MLWE ciphertexts as in [3], which substantially reduces the number of key-switching operations. As a result,

the cost of key-switching becomes negligible compared to the dominant PPMMs. Since these PPMMs can be efficiently implemented using optimized BLAS libraries, the overall constant factors remain small in practice. We summarize our theoretical results and compare them with prior approaches in Table 1.

	Method	Cost	Depth	Red. to PPMM	# Auto	# Keys
Batch CP	[23]	$O(d^2N/2 \cdot \log N)$	3	-	$O(d^2)$	$2\sqrt{d}$
	[3]	$N^2/(2d^2) \cdot T_{\mathsf{PPMM}}(d)$	1	✓	d	$N/2d$
	Ours	$2N/d \cdot T_{\mathsf{PPMM}}(d)$	1	✓	0	0
Batch CC	[23]	$O(d^2N/2 \cdot \log N)$	3	-	$O(d^2)$	$2\sqrt{d}$
	[36]	$4 \cdot T_{\mathsf{PPMM}}(N)$	1	✓	$3N$	3
	Ours	$4N/d \cdot T_{\mathsf{PPMM}}(d)$	1	✓	$3d$	2
Rect. CP	[3]	$2 \cdot T_{\mathsf{PPMM}}(N/2)$	1	✓	$N/2$	N/d
	Ours	$2(N/d)^2 \cdot T_{\mathsf{PPMM}}(d)$	1	✓	$N/2$	1
Rect. CC	[36]	$4 \cdot T_{\mathsf{PPMM}}(N)$	1	✓	$3N$	3
	Ours	$4(N/d)^2 \cdot T_{\mathsf{PPMM}}(d)$	1	✓	$N/2$	2

Table 1. Comparison of our batch and rectangular CPMM/CCMM algorithms with prior works [23,3,36]. The first two rows compare $d \times d$ CPMM and CCMM for $N/2d$ batches, respectively. The last two rows compare rectangular CPMM and CCMM for multiplying a $d \times N/2$ matrix with a $N/2 \times N/2$ matrix.

We implement our algorithms using the HEaaN library [12] in a single-thread setting. For 128 batches of two 64×64 matrix multiplications, batch CPMM and CCMM run in 0.20 and 1.08 seconds, achieving 240× and 52× speedups over [3] and [36], respectively. Our rectangular CPMM and CCMM take 2.2 and 7.5 seconds, respectively, to multiply a 64×2048 matrix with a 2048×2048 matrix, achieving 8.6× and 29× speedups over [3] and [36].

1.2 Related Works

Gentry–Lee [16]. There has been concurrent work by Gentry–Lee [16], which also proposes reduction-based batch encrypted MM algorithms. While both approaches enable structured batching, the underlying algebraic constructions differ. Our approach operates within the standard power-of-two cyclotomic ring and derives batching from an explicit subring decomposition. In contrast, Gentry–Lee work over the non-standard cyclotomic ring $\mathbb{Z}[X, W]/(X^N + 1, \Phi_p(W))$, where N is a power of two and p is prime, following the framework of [17,11]. In their construction, batching arises from the tensor-product structure induced by the independent indeterminates X and W. The batching factor is determined by $\varphi(p) = p - 1$, and is therefore fixed once p is chosen.

In the work of Gentry–Lee, encrypted batch MM is performed in the slot-encoding domain, allowing computations to remain in slots without homomorphic DFT and inverse DFT operations. This is advantageous for circuits requiring slot-encoded computation without bootstrapping. In contrast, although we use

batch matrix encoding, it can be fused with the DFT in bootstrapping, thereby absorbing the conversion cost without additional overhead when bootstrapping is present. We also show that our batch MM can be executed at the base level before bootstrapping. Details are provided in Section 6.

As a consequence of the multivariate structure, NTT operations along the W-axis are performed over a prime cyclotomic order. Such transforms rely on algorithms such as Rader's FFT [37], rather than the power-of-two radix-2 FFT available in the standard power-of-two ring. Accordingly, these transforms do not benefit from the highly optimized radix-2 FFT implementations standard in power-of-two cyclotomic settings. Furthermore, increasing the batching factor requires enlarging p, which changes the parameter set and increases the cost of the associated NTT operations.

To the best of our knowledge, Gentry–Lee do not provide a dedicated bootstrapping procedure tailored to its multivariate setting, except for the approach of [11]. As a result, bootstrapping in [16] does not benefit from the same level of structural optimization as in the standard power-of-two setting.

CinS Encoding [24]. To accelerate encrypted convolution in convolutional neural networks, Ju et al. [24] introduce the *CinS* encoding, which packs DFT values of multiple small-degree polynomials into slots. They further raise the question of a dual encoding that embeds local slots into coefficients and call for its systematic development and applications. Our matrix encoding realizes this dual perspective by embedding local slots into coefficients in a structured manner.

2 Preliminaries

2.1 Notations

Vectors (resp. matrices) are denoted in bold lower-case (resp. upper-case) letters. For a positive integer n, we write $[n] = \{0, 1, \ldots, n-1\}$, let $\mathsf{rev}_n : [n] \to [n]$ denote the bit-reversal permutation, and let $\boldsymbol{P}_n \in \{0,1\}^{n\times n}$ denote the corresponding bit-reversal permutation matrix. For a set F, we denote the set of n-dimensional (column) vectors and $n \times n$ matrices over F by F^n and $F^{n\times n}$, respectively. For a vector $\boldsymbol{v} \in F^n$, we write $\mathsf{diag}_n(\boldsymbol{v})$ for the $n \times n$ diagonal matrix over F with diagonal vector $\boldsymbol{v}$. For matrices $\boldsymbol{A} \in F^{n\times n}$, $\boldsymbol{B} \in F^{m\times m}$, we write $\boldsymbol{A} \otimes \boldsymbol{B} \in F^{nm\times nm}$ for their Kronecker product. For a matrix $\boldsymbol{A} = (a_{ij}) \in \mathbb{C}^{n\times n}$, we write $\boldsymbol{A}^T = (a_{ji})$ for the transpose of $\boldsymbol{A}$ and define $\|\boldsymbol{A}\|_\infty = \max_{i,j} |A_{ij}|$. We denote by $T_{\mathsf{PPMM}}(n)$ the time complexity of multiplying two $n \times n$ cleartext matrices.

For a power of two N and $q \geq 2$, we define K_N as the number field $\mathbb{Q}[X]/(X^N + 1)$, its ring of integers as $\mathcal{R}_N = \mathbb{Z}[X]/(X^N + 1)$, and $\mathcal{R}_{q,N} = \mathbb{Z}_q[X]/(X^N + 1)$. For a polynomial $m = \sum_{i=0}^{N-1} m_i X^i \in \mathcal{R}$ (resp. $m \in \mathcal{R}_q$), we denote by $\boldsymbol{m}$ (resp. $\boldsymbol{m} \in \mathbb{Z}_q^N$) its coefficient vector $(m_0, m_1, \ldots, m_{N-1})$. We define the Discrete Fourier Transform (DFT) $\mathsf{DFT}_N : \mathcal{R}_N \to \mathbb{C}^{N/2}$ as

$$\mathsf{DFT}_N(m(X)) = (m(\omega_{2N}^{5^j}))_{j\in[N/2]},$$

where $\omega_{2N} = \exp(\pi i/N)$ is a primitive $(2N)$-th root of unity. We identify this map as a matrix $\mathsf{DFT}_N = \left(\omega_{2N}^{5^i \cdot j}\right)_{i,j \in [N/2]}$. The inverse Discrete Fourier Transform $\mathsf{iDFT}_N : \mathbb{C}^{N/2} \to \mathcal{R}$ is defined as the inverse of this map.

2.2 The CKKS scheme

CKKS [10] is an RLWE-based FHE scheme that supports approximate computations on real and complex numbers. Its plaintext space is given by $\mathcal{R}_N$, within which message vectors can be encoded as described below.

Encoding methods. In CKKS, there are two basic methods of encoding a message vector into a plaintext. The slot encoding is a map $\mathsf{Ecd}(\cdot) : \mathbb{C}^{N/2} \to \mathcal{R}$, while the coefficient encoding is given by $\mathsf{Ecd}_{\mathrm{coeff}}(\cdot) : \mathbb{R}^N \to \mathcal{R}_N$.

- Slot Encoding. Given a message $\boldsymbol{z} \in \mathbb{C}^{N/2}$, the slot encoding with a scaling factor Δ outputs $\mathsf{Ecd}(\boldsymbol{z}) = \lfloor \Delta \cdot \mathsf{iDFT}(\boldsymbol{z}) \rceil \in \mathcal{R}_N$.
- Coefficient Encoding. Given a message $\boldsymbol{m} \in \mathbb{R}^N$, the coefficient encoding with scaling factor Δ outputs $\mathsf{Ecd}_{\mathrm{coeff}}(\boldsymbol{m}) = \sum_{i \in [N]} \lfloor \Delta \cdot m_i \rceil X^i$.

We note that in the CKKS scheme, every message $\boldsymbol{m}$ is multiplied by a scaling factor Δ and then rounded. After each multiplication, a Rescale operation is required to control the error. For the sake of simplicity, we often omit the scaling factor on the message side.

We clarify the relation between slot and coefficient encodings more concretely. Let $\boldsymbol{V}_n = \left(\omega_{4n}^{5^i \cdot j}\right)_{i,j \in [n]}$ be a unitary matrix for power-of-two integer n. For a polynomial $m \in \mathbb{R}[X]/(X^N + 1)$ with coefficient vector $\boldsymbol{m} \in \mathbb{R}^N$, we decompose $\boldsymbol{m}$ as $\boldsymbol{m} = (\boldsymbol{m}_0 \mid \boldsymbol{m}_1)$, where $\boldsymbol{m}_0, \boldsymbol{m}_1 \in \mathbb{R}^{N/2}$. Then we have

$$\boldsymbol{z} = \mathsf{DFT}_N(m) = \boldsymbol{V}_{N/2}\big(\boldsymbol{m}_0 + i\,\boldsymbol{m}_1\big) \in \mathbb{C}^{N/2}.$$

Accordingly, we define $[\![m]\!] = \boldsymbol{m}_0 + i\,\boldsymbol{m}_1 \in \mathbb{C}^{N/2}$ and

$$m = \mathsf{Ecd}(\boldsymbol{z}) \in \mathcal{R}_N \iff \boldsymbol{z} = \boldsymbol{V}_{N/2} \cdot [\![m]\!] \in \mathbb{C}^{N/2}.$$

Throughout this paper, we often use the bit-reversed order. We define the bit-reversed coefficient vector by

$$[\![m]\!]_{\mathsf{rev}} = \boldsymbol{P}_{N/2} \cdot [\![m]\!],$$

and denote $\boldsymbol{W}_n = \boldsymbol{V}_n \cdot \boldsymbol{P}_n = \left(\omega_{4n}^{5^i \cdot \mathsf{rev}_n(j)}\right)_{i,j \in [n]}$, whose columns are in bit-reversed order. Since $\boldsymbol{P}_{N/2}^2 = \boldsymbol{I}_{N/2}$, we have

$$m = \mathsf{Ecd}(\boldsymbol{z}) \in \mathcal{R}_N \iff \boldsymbol{z} = \boldsymbol{V}_{N/2} \cdot [\![m]\!] = \boldsymbol{W}_{N/2} \cdot [\![m]\!]_{\mathsf{rev}}.$$

Homomorphic operations. As a homomorphic encryption scheme, the CKKS scheme admits the following basic operations.

- KeyGen. Given the security parameter λ, $\mathsf{KeyGen}(1^\lambda)$ returns a public key $\mathsf{pk} \in \mathcal{R}_{q,N}^2$ and a secret key $\mathsf{sk} \in \mathcal{R}_N$.
- Enc. Given an encoded plaintext $m \in \mathcal{R}_N$, $\mathsf{Enc}_{\mathsf{pk}}$ returns a RLWE-format ciphertext $\mathsf{ct} = (a, b) \in \mathcal{R}_{q,N}^2$ such that $a \cdot \mathsf{sk} + b = m + e$ for a small error e.
- Dec. Given a ciphertext $\mathsf{ct} = (a, b) \in \mathcal{R}_{q,N}^2$, $\mathsf{Dec}_{\mathsf{sk}}$ returns $a \cdot \mathsf{sk} + b$.
- SwkGen. Given two integers p, q and two keys $\mathsf{sk}, \mathsf{sk}' \in \mathcal{R}_N$, SwkGen returns a switching key $\mathsf{swk}_{q,p,\mathsf{sk}\to\mathsf{sk}'} \in \mathcal{R}_{pq}^2$ from key sk to sk' for ciphertexts modulo q and with auxiliary modulus p. The switching key is an encryption of $p \cdot \mathsf{sk}$ under the secret key sk'.
- $\mathsf{KeySwitch}$. Given a ciphertext $\mathsf{ct} \in \mathcal{R}_{q,N}^2$ encrypting $m \in \mathcal{R}_N$ under secret key sk, $\mathsf{KeySwitch}_{\mathsf{sk}\to\mathsf{sk}'}$ returns a ciphertext $\mathsf{ct}' \in \mathcal{R}_{q,N}^2$ encrypting m under a different secret key sk'.
- Add. Given two ciphertexts $\mathsf{ct}, \mathsf{ct}' \in \mathcal{R}_{q,N}^2$ encrypting $m, m' \in \mathcal{R}_N$ respectively, Add returns a ciphertext encrypting $m + m'$.
- $\mathsf{Rescale}$. Given a ciphertext $\mathsf{ct} \in \mathcal{R}_{q,N}^2$ encrypting $m \in \mathcal{R}_N$, $\mathsf{Rescale}$ returns $\mathsf{ct}' \in \mathcal{R}_{q',N}^2$ encrypting $\lfloor (q/q')\, m \rceil$.
- PCMult. Given a ciphertext $\mathsf{ct} \in \mathcal{R}_{q,N}^2$ encrypting $m \in \mathcal{R}_N$ and a plaintext $\mu \in \mathcal{R}_N$, PCMult returns a ciphertext ct' that encrypts $\mu \cdot m \in \mathcal{R}_N$.
- Mult. Given ciphertexts $\mathsf{ct}, \mathsf{ct}' \in \mathcal{R}_{q,N}^2$ encrypting $m, m' \in \mathcal{R}_N$ respectively, Mult returns a ciphertext ct'' that encrypts $m \cdot m' \in \mathcal{R}_N$. It requires a relinearization key $\mathsf{swk}_{q,p,\mathsf{sk}^2\to\mathsf{sk}}$.
- Auto. For a given ciphertext $\mathsf{ct} \in \mathcal{R}_{q,N}^2$ encrypting $m \in \mathcal{R}_{q,N}$ and an automorphism $\sigma \in \mathsf{Gal}(\mathcal{R}/\mathbb{Z})$, Auto_σ returns a ciphertext ct' which encrypts $\sigma(m)$. This procedure requires an automorphism key $\mathsf{swk}_{q,p,\sigma(\mathsf{sk})\to\mathsf{sk}}$. Specifically, we will denote $\mathsf{Auto}(\mathsf{ct}; 2j+1)$ for an automorphism $X \mapsto X^{2j+1}$.

Operations such as Add and PCMult require only $O(N)$ operations over $\mathbb{Z}_q$, whereas operations involving $\mathsf{KeySwitch}$ (e.g., Mult and Auto) require $O(N \log N)$.

Bootstrapping. Bootstrapping is a technique to recover the ciphertext modulus that has been reduced by $\mathsf{Rescale}$ operations. A contemporary CKKS bootstrapping algorithm consists of the following steps:

- $\mathsf{S2C}$ is a homomorphic DFT which changes the message encoding. Namely, $\mathsf{S2C}$ receives a ciphertext encrypting $\mathsf{Ecd}(\boldsymbol{m})$, and returns a ciphertext that encrypts $m = \mathsf{Ecd}_{\mathrm{coeff}}(\boldsymbol{m})$.
- $\mathsf{ModRaise}$ lifts the ciphertext (a, b) from $\mathcal{R}_{q_0,N}$ to $\mathcal{R}_{Q,N}$ for $Q > q_0$, where it satisfies the identity $b + a \cdot \mathsf{sk} = m + q_0 \cdot I \pmod{Q}$ for some $I \in \mathcal{R}_N$.
- $\mathsf{C2S}$ is a homomorphic inverse DFT. It receives a ciphertext encrypting $m + q_0 \cdot I$ and returns a ciphertext encrypting $\mathsf{Ecd}(\boldsymbol{m} + q_0 \boldsymbol{I})$.
- $\mathsf{EvalMod}$ homomorphically removes the I part to recover the ciphertext encrypting $\mathsf{Ecd}(\boldsymbol{m}) \in \mathcal{R}_{q,N}$ for some modulus $q_0 < q < Q$.

2.3 Decomposition of the DFT Matrix

To accelerate homomorphic DFT and inverse DFT during bootstrapping, [7,18] proposed decomposing the bit-reversed DFT matrix $\boldsymbol{W}_{N/2}$ into a product of

sparse diagonal matrices for efficient linear transformation. More concretely, one can decompose the matrix $\boldsymbol{W}_{N/2}$ as

$$\boldsymbol{W}_{N/2} = \boldsymbol{V}_{N/2} \cdot \boldsymbol{P}_{N/2} = \boldsymbol{S}_{N/4} \cdot \boldsymbol{S}_{N/8} \cdots \boldsymbol{S}_1,$$

where

$$\boldsymbol{S}_n = \boldsymbol{I}_{\frac{N}{4n}} \otimes \begin{bmatrix} \boldsymbol{I}_n & \boldsymbol{\Omega}_n \\ \boldsymbol{I}_n & -\boldsymbol{\Omega}_n \end{bmatrix} \in \mathbb{C}^{N/2 \times N/2}$$

with $\boldsymbol{\Omega}_n = \mathsf{diag}\big(\omega_{8n}^{5^i}\big)_{i \in [n]}$ for $n \mid N$. Then we can interpret $\mathsf{S2C}$ and $\mathsf{C2S}$ as homomorphic linear transformations:

$$\mathsf{S2C} : \mathsf{Ecd}(\boldsymbol{z}) \xmapsto{\boldsymbol{W}_{N/2}} \mathsf{Ecd}(\boldsymbol{W}_{N/2} \cdot \boldsymbol{z}) = m, \text{ where } [\![m]\!]_{\mathsf{rev}} = \boldsymbol{z},$$
$$\mathsf{C2S} : m = \mathsf{Ecd}(\boldsymbol{z}) \xmapsto{\boldsymbol{W}_{N/2}^{-1}} \mathsf{Ecd}(\boldsymbol{W}_n^{-1}\boldsymbol{z}) = \mathsf{Ecd}([\![m]\!]_{\mathsf{rev}}).$$

In particular, the linear transformations above can be accelerated by exploiting the decomposition of $\boldsymbol{W}_{N/2}$. We use a shorthand notation $\boldsymbol{S}_{j \leftarrow i} = \boldsymbol{S}_{j/2} \cdot \boldsymbol{S}_{j/4} \cdots \boldsymbol{S}_i$. We decompose $\boldsymbol{W}_{N/2}$ into two matrices as

$$\boldsymbol{W}_{N/2} = \boldsymbol{S}_{N/2 \leftarrow n} \cdot \boldsymbol{S}_{n \leftarrow 1},$$
$$\boldsymbol{W}_{N/2}^{-1} = \boldsymbol{S}_{n \leftarrow 1}^{-1} \cdot \boldsymbol{S}_{N/2 \leftarrow n}^{-1} = (1/n)\, \boldsymbol{S}_{n \leftarrow 1}^{*} \cdot (2n/N)\, \boldsymbol{S}_{N/2 \leftarrow n}^{*}.$$

We note that $\boldsymbol{S}_{n \leftarrow 1} = \boldsymbol{I}_{\frac{N}{2n}} \otimes \boldsymbol{W}_n$ and

$$\boldsymbol{S}_{N/2 \leftarrow n} = \big(\mathsf{diag}_n(\boldsymbol{w}_{i,j})\big), \quad \boldsymbol{S}_{N/2 \leftarrow n}^{-1} = \big(\mathsf{diag}_n((2n/N)\, \overline{\boldsymbol{w}_{j,i}})\big) \in \mathbb{C}^{N/2 \times N/2}$$

for some complex vectors $\boldsymbol{w}_{i,j} \in \mathbb{C}^n$.

2.4 Galois Automorphisms and Trace

The ring $\mathcal{R}_N = \mathbb{Z}[X]/(X^N + 1)$ inherits automorphisms $\mathsf{Gal}(K_N/\mathbb{Q}) = \{\sigma_h : X \mapsto X^h \mid h \in \mathbb{Z}_{2N}^{\times}\}$ from the cyclotomic field $K_N = \mathbb{Q}[X]/(X^N + 1)$. Furthermore, the subgroup $\mathsf{Gal}(K_N/K_k) \subset \mathsf{Gal}(K_N/\mathbb{Q})$ preserves the subfield $K_k = \mathbb{Q}[Y]/(Y^k + 1)$ as well as the subring $\mathcal{R}_k = \mathbb{Z}[Y]/(Y^k + 1)$, since they stabilize $Y = X^d$ for $d = N/k$. We denote this group $\mathsf{Gal}(K_N/K_k)$ by Gal_k^N. Then

$$\mathsf{Gal}_k^N = \{\sigma_h : X \mapsto X^h \mid h \in H\},$$

where $H = \{2kt+1 \mid t \in [d]\} \in \mathbb{Z}_{2N}^{\times}$. We also make use of the field trace adapted to $\mathcal{R}_N$. We define

$$\mathtt{Tr}_k^N(m) = \sum_{h \in H} \sigma_h(m),$$

which coincides with $\mathtt{Tr}_{K_N/K_k}$ and satisfies $\mathtt{Tr}_k^N\big(\sum_i m_i(Y)X^i\big) = d \cdot m_0(Y)$. We note that the above discussion extends naturally to $\mathcal{R}_{q,N}$.

3 Matrix Encryption and Encoding over Subrings

In this section, we introduce a novel encoding and encryption framework for RLWE-based ciphertexts used in our batch matrix multiplication algorithms. We reinterpret a ring element as a vector over a subring of algebraic integers, rather than as a length-N coefficient vector over the integers. This gives a matrix-valued view of standard RLWE encryption in which the matrix dimension is decoupled from the RLWE ring degree.

We formalize this view as *matrix encryption over a subring* and combine it with *batch matrix encoding*, which packs multiple complex matrices into a single matrix over the same subring. Since the encoding preserves entrywise ring operations, matrix multiplication over the subring realizes the corresponding batch of complex matrix multiplications.

3.1 Matrix Encryption over the Subring

For a divisor $k \mid N$, we set $d = N/k$ and introduce an indeterminate $Y = X^d$. This yields a subring $\mathcal{R}_k$ of $\mathcal{R}_N$ with $\mathcal{R}_k = \mathbb{Z}[Y]/(Y^k + 1)$, and we let $\mathcal{R}_{q,k} = \mathcal{R}_k/q\mathcal{R}_k$. Now, we generalize the matrix encryption of [3] to the $\mathcal{R}_{q,k}$. Note that $\mathcal{R}_{q,N}$ can be viewed as a rank-d $\mathcal{R}_{q,k}$-module, which induces the following module isomorphism:

$$\begin{aligned} \mathtt{Vec}_k^d : \mathcal{R}_{q,N} &\longrightarrow \mathcal{R}_{q,k}^d \\ m(X) &\longmapsto (m_i(Y))_{i\in[d]}, \end{aligned}$$

where $m(X) = \sum_{i=0}^{d-1} m_i(Y)X^i$. We regard $\mathtt{Vec}_k^d(m)$ as a d-dimensional column vector over $\mathcal{R}_{q,k}^d$. We now generalize the Toeplitz matrix over $\mathbb{Z}_q$ used in [3,36,4] to the setting over $\mathcal{R}_{q,k}$. For an element $s \in \mathcal{R}_N$, we let $\mathtt{Vec}_k^d(s(X)) = (s_i(Y))_{i\in[d]}$. Then we define

$$\begin{aligned} \mathtt{Toep}_k^d(s) &:= \left[\mathtt{Vec}_k^d(s) \mid \mathtt{Vec}_k^d(Xs) \mid \cdots \mid \mathtt{Vec}_k^d(X^{d-1}s)\right] \\ &= \begin{bmatrix} s_0(Y) & Ys_{d-1}(Y) & \cdots & Ys_1(Y) \\ s_1(Y) & s_0(Y) & \cdots & Ys_2(Y) \\ \vdots & \vdots & \ddots & \vdots \\ s_{d-1}(Y) & s_{d-2}(Y) & \cdots & s_0(Y) \end{bmatrix} \in \mathcal{R}_k^{d\times d}. \end{aligned}$$

Utilizing the Toeplitz matrix defined over $\mathcal{R}_k$, we can reinterpret a single RLWE encryption in a column-wise vectorized form over $\mathcal{R}_k$.

Theorem 1 (Vector encryption over $\mathcal{R}_{q,k}$). *For an RLWE ciphertext $(b = -a \cdot \mathsf{sk} + m + e, a) \in \mathcal{R}_{q,N}^2$, the following matrix equation holds over $\mathcal{R}_{q,k}$:*

$$\mathtt{Toep}_k^d(\mathsf{sk}) \cdot \mathtt{Vec}_k^d(a) + \mathtt{Vec}_k^d(b) = \mathtt{Vec}_k^d(m) + \mathtt{Vec}_k^d(e) \pmod q.$$

In other words, an RLWE ciphertext encrypting $m \in \mathcal{R}_N$ corresponds to a vector encryption *of the vector $\mathtt{Vec}_k^d(m) \in \mathcal{R}_k^d$.*

Proof. We first establish the following identity for $\mathsf{sk} \in \mathcal{R}_N$ and $a \in \mathcal{R}_{q,N}$:

$$\mathtt{Toep}_k^d(\mathsf{sk}) \cdot \mathtt{Vec}_k^d(a) = \mathtt{Vec}_k^d(\mathsf{sk} \cdot a).$$

Indeed, let $\mathtt{Vec}_k^d(a) = (a_i(Y))_{i\in[d]}$. Then

$$\begin{aligned}
\mathtt{Vec}_k^d(\mathsf{sk} \cdot a) &= \mathtt{Vec}_k^d\left(\sum_{i\in[d]} \mathsf{sk} \cdot a_i(Y) X^i\right) = \sum_{i\in[d]} \mathtt{Vec}_k^d(X^i \mathsf{sk}) \cdot a_i(Y) \\
&= \left[\mathtt{Vec}_k^d(\mathsf{sk}) \mid \cdots \mid \mathtt{Vec}_k^d(X^{d-1}\mathsf{sk})\right] \begin{bmatrix} a_0(Y) \\ \vdots \\ a_{d-1}(Y) \end{bmatrix} \\
&= \mathtt{Toep}_k^d(\mathsf{sk}) \cdot \mathtt{Vec}_k^d(a).
\end{aligned}$$

Therefore, we have

$$\mathtt{Vec}_k^d(m) + \mathtt{Vec}_k^d(e) = \mathtt{Vec}_k^d(\mathsf{sk} \cdot a + b) = \mathtt{Toep}_k^d(\mathsf{sk}) \cdot \mathtt{Vec}_k^d(a) + \mathtt{Vec}_k^d(b).$$

□

Now, we consider d' RLWE ciphertexts $\{(b_j = -a_j \cdot \mathsf{sk} + m_j + e_j,\, a_j) \in \mathcal{R}_{q,N}^2\}_{j\in[d']}$, each encrypting a message $m_j \in \mathcal{R}_N$ with error e_j. By applying Theorem 1 d' times in a column-wise manner, we obtain

$$\boldsymbol{B} + \mathtt{Toep}_k^d(\mathsf{sk}) \cdot \boldsymbol{A} = \boldsymbol{M} + \boldsymbol{E} \pmod q,$$

where each matrix is formed column-wise by viewing corresponding RLWE elements as column vectors in $\mathcal{R}_{q,k}^d$ via $\mathtt{Vec}_k^d$. We refer to this as a *matrix encryption* of $\boldsymbol{M} \in \mathcal{R}_k^{d\times d'}$. For brevity, we often omit the error matrix $\boldsymbol{E}$ from the matrix encryption and write

$$\boldsymbol{B} + \mathtt{Toep}_k^d(\mathsf{sk}) \cdot \boldsymbol{A} \approx \boldsymbol{M} \pmod q.$$

3.2 Batch Matrix Encoding

The choice of subring also admits a natural SIMD (slot) encoding to $\mathcal{R}_{q,k}$, enabling parallel operations over $\mathbb{C}^{k/2}$. Building on this structure, we further extend the slot encoding to a *batch matrix encoding*, which packs a batch of matrices into a single matrix over algebraic integers.

Definition 1 (Batch Matrix Encoding). *Let $\{\boldsymbol{M}_\ell\}_{\ell\in[k/2]} \subset \mathbb{C}^{d_1\times d_2}$ be a collection of matrices. The* batch matrix encoding *is the map*

$$\mathsf{MatEcd}_k^{d_1\times d_2} : (\mathbb{C}^{d_1\times d_2})^{k/2} \longrightarrow \mathcal{R}_k^{d_1\times d_2}$$

defined entrywise as follows: each (i,j)-th entry is given by

$$\left\lfloor \Delta \cdot \mathsf{iDFT}_k((\boldsymbol{M}_0)_{i,j}, \cdots, (\boldsymbol{M}_{k/2-1})_{i,j}) \right\rceil \in \mathcal{R}_k.$$

In particular, in the case where $d_1 = d_2 = d$, we abbreviate $\mathsf{MatEcd}_k^{d_1 \times d_2}$ as MatEcd_k^{d}. A key property of the batch matrix encoding is its compatibility with matrix operations, which follows from the ring-homomorphic nature of iDFT_k. In particular, for any collections $\{\boldsymbol{M}_\ell\}_{\ell \in [k/2]} \subset \mathbb{C}^{d_1 \times d_2}$ and $\{\boldsymbol{M}'_\ell\}_{\ell \in [k/2]} \subset \mathbb{C}^{d_2 \times d_3}$, the encoding satisfies

$$\mathsf{MatEcd}_k^{d_1 \times d_3}(\{\boldsymbol{M}_\ell \cdot \boldsymbol{M}'_\ell\}) \approx \mathsf{MatEcd}_k^{d_1 \times d_2}(\{\boldsymbol{M}_\ell\}) \cdot \mathsf{MatEcd}_k^{d_2 \times d_3}(\{\boldsymbol{M}'_\ell\})/\Delta,$$

which enables batch matrix multiplication. We now define the corresponding batch matrix encryption.

Definition 2 (Batch Matrix Encryption). *Let $\{\boldsymbol{M}_\ell\}_{\ell \in [k/2]} \subset \mathbb{C}^{d \times d'}$ be a collection of matrices and denote by*

$$m_j = \sum_{i \in [d]} \mathsf{MatEcd}_k^{d \times d'}(\{\boldsymbol{M}_\ell\}_{\ell \in [k/2]})_{i,j} \cdot X^i \in \mathcal{R}_N.$$

We define a matrix encryption *of $\{\boldsymbol{M}_\ell\}_{\ell \in [k/2]}$ as follows: Using RLWE encryptions $(b_j, a_j) \leftarrow \mathsf{Enc}_{\mathsf{pk}}(m_j)$ with encryption errors e_j, we denote*

$$\boldsymbol{A} = [\mathtt{Vec}_k^d(a_0) \mid \cdots \mid \mathtt{Vec}_k^d(a_{d'-1})], \quad \boldsymbol{B} = [\mathtt{Vec}_k^d(b_0) \mid \cdots \mid \mathtt{Vec}_k^d(b_{d'-1})] \in \mathcal{R}_{q,k}^{d \times d'}.$$

This gives the equation

$$\boldsymbol{B} + \mathtt{Toep}_k^d(\mathsf{sk}) \cdot \boldsymbol{A} = \mathsf{MatEcd}_k^{d \times d'}(\{\boldsymbol{M}_\ell\}_{\ell \in [k/2]}) + \boldsymbol{E} \pmod q,$$

with the error matrix $\boldsymbol{E} = [\mathtt{Vec}_k^d(e_0) \mid \cdots \mid \mathtt{Vec}_k^d(e_{d'-1})] \in \mathcal{R}_k^{d \times d'}$.

We note that decoding and decryption of the matrix encryption are defined analogously by applying DFT_k and then dividing by Δ. The encryption error in Definition 2 is analyzed in Appendix A.

4 Batch Encrypted Matrix Multiplication

Matrices encoded and encrypted using the framework of the previous section support both batch ciphertext-plaintext matrix multiplication (CPMM) and batch ciphertext-ciphertext matrix multiplication (CCMM). By generalizing the techniques of [3,36], we show that both operations can be reduced to collections of batch plaintext-plaintext matrix multiplications (PPMMs).

4.1 Batch Ciphertext-Plaintext Matrix Multiplication

Suppose we are given a matrix encryption $(\boldsymbol{B}, \boldsymbol{A}) \in \mathcal{R}_{q,k}^{d \times d'} \times \mathcal{R}_{q,k}^{d \times d'}$ of encoded batch matrices $\boldsymbol{M} = \mathsf{MatEcd}(\{\boldsymbol{M}_\ell\})_{\ell \in [k/2]} \in \mathcal{R}_k^{d \times d'}$, together with a plaintext matrix $\boldsymbol{U} = \mathsf{MatEcd}(\{\boldsymbol{U}_\ell\})_{\ell \in [k/2]} \in \mathcal{R}_k^{d' \times d''}$. By construction, we have

$$\mathtt{Toep}_k^d(\mathsf{sk}) \cdot \boldsymbol{A} + \boldsymbol{B} \;\approx\; \mathsf{MatEcd}_k^{d \times d'}(\{\boldsymbol{M}_\ell\}_{\ell \in [k/2]}) \in \mathcal{R}_{q,k}^{d \times d'},$$

Using the homomorphic property, it follows that

$$\mathsf{Toep}_k^d(\mathsf{sk}) \cdot (\boldsymbol{A} \cdot \boldsymbol{U}) + \boldsymbol{B} \cdot \boldsymbol{U} \;\approx\; \boldsymbol{M} \cdot \boldsymbol{U} = \mathsf{MatEcd}_k^{d \times d''}(\{\boldsymbol{M}_\ell \cdot \boldsymbol{U}_\ell\}) \in \mathcal{R}_{q,k}^{d \times d''}.$$

In other words, the matrix pair $(\boldsymbol{B} \cdot \boldsymbol{U}, \boldsymbol{A} \cdot \boldsymbol{U})$ constitutes a matrix encryption of $\mathsf{MatEcd}_k^{d \times d''}(\{\boldsymbol{M}_\ell \cdot \boldsymbol{U}_\ell\}_{\ell \in [k/2]})$. Consequently, a batch CPMM reduces to two PPMMs over $\mathcal{R}_{q,k}$, which can be formally described in Algorithm 1.

Algorithm 1: Batch CPMM

Input: Matrix encryption $(\boldsymbol{B}, \boldsymbol{A})$ of
$\boldsymbol{M} = \mathsf{MatEcd}(\{\boldsymbol{M}_\ell\}_{\ell \in [k/2]} \subset \mathbb{C}^{d \times d'}) \in \mathcal{R}_{q,k}^{d \times d'}$, and encoded plaintext matrix $\boldsymbol{U} = \mathsf{MatEcd}(\{\boldsymbol{U}_\ell\}_{\ell \in [k/2]} \subset \mathbb{C}^{d' \times d''}) \in \mathcal{R}_{q,k}^{d' \times d''}$

Output: Matrix encryption $(\boldsymbol{B}', \boldsymbol{A}')$ of
$\boldsymbol{M} \cdot \boldsymbol{U} = \mathsf{MatEcd}(\{\boldsymbol{M}_\ell \cdot \boldsymbol{U}_\ell\}_{\ell \in [k/2]}) \in \mathcal{R}_{q/\Delta,k}^{d \times d''}$.

1 $(\boldsymbol{B}, \boldsymbol{A}) \leftarrow (\boldsymbol{B} \cdot \boldsymbol{U}, \boldsymbol{A} \cdot \boldsymbol{U})$
2 $(\boldsymbol{B}', \boldsymbol{A}') \leftarrow (\lfloor \frac{1}{\Delta} \cdot \boldsymbol{B} \rceil, \lfloor \frac{1}{\Delta} \cdot \boldsymbol{A} \rceil)$ ▷ Rescale by Δ
3 **return** $(\boldsymbol{B}', \boldsymbol{A}') \in \mathcal{R}_{q/\Delta,k}^{d \times d''} \times \mathcal{R}_{q/\Delta,k}^{d \times d''}$

Theorem 2 (Batch CPMM). *Let $(\boldsymbol{B}, \boldsymbol{A})$ be a matrix encryption satisfying*

$$\boldsymbol{B} + \mathsf{Toep}_k^d(\mathsf{sk}) \cdot \boldsymbol{A} = \mathsf{MatEcd}_k^{d \times d'}(\{\boldsymbol{M}_\ell\}_{\ell \in [k/2]}) + \boldsymbol{E} \in \mathcal{R}_{q,k}^{d \times d'}.$$

For any plaintext $\boldsymbol{U} = \mathsf{MatEcd}_k^{d' \times d''}(\{\boldsymbol{U}_\ell\}_{\ell \in [k/2]}) \in \mathcal{R}_k^{d' \times d''}$, Algorithm 1 outputs a matrix encryption

$$\boldsymbol{B}' + \mathsf{Toep}_k^d(\mathsf{sk}) \cdot \boldsymbol{A}' = \mathsf{MatEcd}_k^{d \times d''}(\{\boldsymbol{M}_\ell \cdot \boldsymbol{U}_\ell\}_{\ell \in [k/2]}) + \boldsymbol{E}_{\mathrm{out}} \in \mathcal{R}_{q/\Delta,k}^{d \times d''},$$

where the output error matrix $\boldsymbol{E}_{\mathrm{out}}$ is written as

$$\boldsymbol{E}_{\mathrm{out}} = \frac{1}{\Delta} \cdot \boldsymbol{E}\,\boldsymbol{U} + \boldsymbol{E}_{\mathrm{rs}}, \quad \boldsymbol{E}_{\mathrm{rs}} = \frac{1}{\Delta}\left([\boldsymbol{B}\,\boldsymbol{U} \bmod q]_\Delta + \mathsf{Toep}_k^d(\mathsf{sk}) \cdot [\boldsymbol{A}\,\boldsymbol{U} \bmod q]_\Delta\right).$$

We note that, for sufficiently large Δ, the noise introduced by Algorithm 1 is dominated by the error $\boldsymbol{E}\,\boldsymbol{U}/\Delta \in \mathcal{R}_k^{d \times d''}$, which contributes to the message noise in the MatEcd representation through the local inverse DFT (iDFT_k), rather than the full inverse DFT (iDFT_N) used in the original slot encoding. We refer to Appendix A.1 for the rigorous noise analysis of the algorithm.

Cost analysis. We assume that ciphertexts and plaintexts are NTT transformed over $\mathcal{R}_{q,k}$ in advance. For the case $d = d' = d''$, the algorithm requires $2k \cdot T_{\mathrm{PPMM}}(d)$ $\mathbb{Z}_q$-operations for the PPMMs (Step 1), and $O(dN)$ $\mathbb{Z}_q$-operations for $\mathsf{Rescale}$ (Step 2). Inverse NTT operations are needed before $\mathsf{Rescale}$, and NTT operations after $\mathsf{Rescale}$, which add a conversion cost of $O(d^2 k \log k)$. In total, Algorithm 1 for batch CPMM requires $2k \cdot T_{\mathrm{PPMM}}(d) + O(d^2 k \log k)$ $\mathbb{Z}_q$-operations.

Batch CPMM over real space. Since our batch CPMM algorithm is defined over complex matrices, we describe how to handle real matrices more efficiently than by directly embedding them into the complex space. Suppose $\{\boldsymbol{M}_\ell\}_{\ell\in[k/2]} \subset \mathbb{R}^{2d\times d'}$ in ciphertext form and $\{\boldsymbol{U}_\ell\}_{\ell\in[k/2]} \subset \mathbb{R}^{d'\times d''}$ in plaintext form are given as inputs. For each $\boldsymbol{M}_\ell$, we split it row-wise as

$$\boldsymbol{M}_\ell = \begin{bmatrix} \boldsymbol{M}_{\ell,0} \\ \boldsymbol{M}_{\ell,1} \end{bmatrix}$$

and we pack it as a single complex matrix $\widetilde{\boldsymbol{M}}_\ell = \boldsymbol{M}_{\ell,0} + i \cdot \boldsymbol{M}_{\ell,1} \in \mathbb{C}^{d\times d'}$. Since

$$\widetilde{\boldsymbol{M}}_\ell \cdot \boldsymbol{U}_\ell = (\boldsymbol{M}_{\ell,0} \cdot \boldsymbol{U}_\ell) + i \cdot (\boldsymbol{M}_{\ell,1} \cdot \boldsymbol{U}_\ell),$$

it suffices to obtain a matrix encryption of $\mathsf{MatEcd}_k^{d\times d'}(\{\widetilde{\boldsymbol{M}}_\ell\})$ and a plaintext matrix $\mathsf{MatEcd}_k^{d'\times d''}(\{\boldsymbol{U}_\ell\})$, thereby reducing the problem to PPMMs between them. This allows the multiplication of $2d \times d'$ real matrices with $d' \times d''$ real matrices in $k/2$ batches for $N = dk$, thereby doubling the batch size and yielding a twofold improvement in throughput over the embedding approach.

4.2 Batch Ciphertext Matrix Transpose

As a building block for batch ciphertext-ciphertext matrix multiplication (CCMM), we introduce the batch ciphertext matrix transpose (CMT). The batch CMT transforms a matrix encryption of $\boldsymbol{M} = \mathsf{MatEcd}_k^d(\{\boldsymbol{M}_\ell\}_{\ell\in[k/2]}) \in \mathcal{R}_{q,k}^{d\times d}$ into a matrix encryption of its transpose $\boldsymbol{M}^T = \mathsf{MatEcd}_k^d(\{\boldsymbol{M}_\ell^T\}_{\ell\in[k/2]})$.

More concretely, let CMT_k^d denote the CMT algorithm. Given a (column-wise) matrix encryption $\{(b_j, a_j)\}_{j\in[d]}$ satisfying

$$b_j + a_j \cdot \mathsf{sk} \approx m_j(X) = \sum_{i\in[d]} M_{i,j} \cdot X^i \pmod q, \quad \forall j \in [d],$$

where $\boldsymbol{M} = (M_{i,j})_{i,j\in[d]}$. The goal of CMT_k^d is to output d ciphertexts $(b'_j, a'_j)_{j\in[d]}$ such that

$$b'_j + a'_j \cdot \mathsf{sk} \approx m'_j(X) = \sum_{i\in[d]} M_{j,i} \cdot X^i \pmod q, \quad \forall j \in [d].$$

The resulting ciphertexts thus form a matrix encryption of the transposed matrix $\boldsymbol{M}^T = (M_{j,i})_{i,j\in[d]}$.

The construction of [36] extends naturally to our setting, with minor modifications. Our approach relies on the trace map $\mathtt{Tr}_k^N : \mathcal{R}_{q,N} \to \mathcal{R}_{q,k}$, which satisfies

$$\mathtt{Tr}_k^N(X^{-i} \cdot m_j(X)) = d \cdot M_{i,j}.$$

Using this property, we derive

$$\begin{aligned} d \cdot m'_j(X) &= \sum_{i\in[d]} \mathtt{Tr}_k^N(X^{-j} \cdot m_i(X)) \cdot X^i = \sum_{i\in[d]} \sum_{\sigma\in \mathsf{Gal}_k^N} \sigma(X^{-j} \cdot m_i(X)) \cdot X^i \\ &= \sum_{\sigma\in \mathsf{Gal}_k^N} \sigma(X^{-j}) \sum_{i\in[d]} \sigma(m_i(X) \cdot \sigma^{-1}(X^i)) \\ &= \sum_{h\in H} X^{-jh} \cdot \sigma_h \left(\sum_{i\in[d]} m_i(X) X^{ih^{-1}} \right), \end{aligned}$$

where $\mathsf{Gal}_k^N = \{\sigma_h : X \mapsto X^h \mid h \in H\}$ with $H = \{h = 2kt + 1 \mid t \in [d]\} \subset \mathbb{Z}_{2N}^\times$.

Hence, the algorithm $\mathsf{CMT}_k^d(\{m_i\}_{i\in[d]}) = \{m'_j\}_{j\in[d]}$ proceeds in three steps:

$$\begin{aligned} \mathsf{Adjust}(\{m_i\}_{i\in[d]}) &= \{\tilde{m}_t = d^{-1} \sum_{i\in[d]} m_i(X) \cdot X^{i(2kt+1)}\}_{t\in[d]}, \\ \mathsf{ScrambleAuto}(\{\tilde{m}_t\}_{t\in[d]}) &= \{\tilde{m}_t = \tilde{m}_{t^*}(X^{2kt+1})\}_{t\in[d]}, \\ \mathsf{invAdjust}(\{\tilde{m}_t\}_{t\in[d]}) &= \{m'_j = \sum_{t\in[d]} \tilde{m}_t X^{-j(2kt+1)}\}_{j\in[d]}, \end{aligned}$$

where $t^* \in [d]$ satisfies $2kt^* + 1 \equiv (2kt + 1)^{-1} \pmod{2N}$. The $\mathsf{ScrambleAuto}$ step requires d key switchings at a cost of $O(dN \log N)$ $\mathbb{Z}_q$-operations. For Adjust and $\mathsf{invAdjust}$, we propose the TWEAK_k^N algorithm, adapted from the TWEAK algorithm in [36], to accelerate both steps. Specifically, TWEAK_k^N takes as input d ciphertexts encrypting $\{m_i\}_{i\in[d]}$, and outputs d ciphertexts encrypting

$$\left\{ \sum_{i\in[d]} m_i(X) \cdot X^{2kij} \right\}_{j\in[d]}.$$

We present Cooley–Tukey style pseudocode implementing TWEAK_k^N, summarized in Algorithm 2.

As a result, both the Adjust and $\mathsf{invAdjust}$ steps are accelerated by TWEAK_k^N, reducing their cost to $O(dN \log d)$ $\mathbb{Z}_q$-operations. The procedure of CMT over $\mathcal{R}_{q,k}$ is summarized in Algorithm 3.

For a matrix encryption $(\boldsymbol{B}, \boldsymbol{A})$ of a matrix $\boldsymbol{M}$, CMT_k^d outputs a matrix encryption $(\boldsymbol{B}', \boldsymbol{A}')$ of $\boldsymbol{M}^T$, which can be written as a row-wise matrix encryption $(\boldsymbol{B}'^T, \boldsymbol{A}'^T)$ of $\boldsymbol{M}$ as

$$\boldsymbol{B}'^T + \boldsymbol{A}'^T \cdot \mathtt{Toep}_k^d(\mathsf{sk})^T \approx \boldsymbol{M} \pmod{q}$$

by taking the transpose. Thus, CMT_k^d can be viewed as a conversion from column-wise to row-wise matrix encryption.

Algorithm 2: TWEAK_k^N

Input: RLWE ciphertexts $(\mathsf{ct}_j)_{j\in[d]}$ encrypting $m_j \in \mathcal{R}_{q,N}$.
Input: $\mathsf{sgn} \in \{\pm 1\}$.
Output: RLWE ciphertexts $(\mathsf{ct}'_j)_{j\in[d]}$ encrypting $m'_j \in \mathcal{R}_{q,N}$ such that $m'_i = \sum_{j\in[d]} X^{2kij\cdot\mathsf{sgn}} \cdot m_j$.

```
if d = 1 then
    return {ct_0}
else
    {aux_j}_{j∈[d/2]} ← TWEAK^N_{2k}({ct_{2j}}_{j∈[d/2]}, sgn)
    {aux_{j+d/2}}_{j∈[d/2]} ← TWEAK^N_{2k}({ct_{2j+1}}_{j∈[d/2]}, sgn)
    for j ∈ [d/2] do
        ct'_j ← aux_j + X^{2kj·sgn} · aux_{j+d/2}
        ct'_{j+d/2} ← aux_j − X^{2kj·sgn} · aux_{j+d/2}
    return {ct'_j}_{j∈[d]}
```

Algorithm 3: CMT_k^d

Input: Switching keys $\mathsf{swk}_{\sigma_{2kt+1}(\mathsf{sk})\to\mathsf{sk}}$ for each $t \in [d]$
Input: RLWE ciphertexts $(\mathrm{ct}_i)_{i\in[d]}$ in $\mathcal{R}_{q,N}$ encrypting columns of $\boldsymbol{M} \in \mathcal{R}_{q,k}^{d\times d}$
Output: RLWE ciphertexts $(\mathrm{ct}'_i)_{i\in[d]}$ in $\mathcal{R}_{q,N}$ encrypting columns of $\boldsymbol{M}^T \in \mathcal{R}_{q,k}^{d\times d}$.

```
{ct_i}_{i∈[d]} ← {X^i · ct_i}_{i∈[d]}
{aux_i}_{i∈[d]} ← TWEAK^N_k({ct_j}_{j∈[d]}, 1)
for t ← [d] do
    t* ← (((2kt + 1)^{-1} mod 2N) − 1)/(2k)
    aux_{t*} ← (d^{-1} mod Q) · aux_{t*}
    aux'_t ← Auto(aux_{t*}; 2kt + 1)
ct' ← TWEAK^N_k({aux'_j}_{j∈[d]}, −1)
{ct'_i}_{i∈[d]} ← {X^{-i} · ct'_i}_{i∈[d]}
return {ct'_i}_{i∈[d]}
```

4.3 Batch Ciphertext-Ciphertext Matrix Multiplication

Batch CCMM takes as input $\{\boldsymbol{M}_\ell\}_{\ell\in[k/2]} \subset \mathbb{C}^{d\times d}$ and $\{\boldsymbol{M}'_\ell\}_{\ell\in[k/2]} \subset \mathbb{C}^{d\times d}$, both encrypted, and outputs ciphertexts encrypting $\{\boldsymbol{M}_\ell \cdot \boldsymbol{M}'_\ell\}_{\ell\in[k/2]} \subset \mathbb{C}^{d\times d}$. Here, we leverage MatEcd to obtain two matrix encryptions $(\boldsymbol{B}, \boldsymbol{A}), (\boldsymbol{B}', \boldsymbol{A}') \in \mathcal{R}_{q,k}^{d\times d}$ as:

$$\boldsymbol{B} + \mathsf{Toep}_k^d(\mathsf{sk}) \cdot \boldsymbol{A} \approx \boldsymbol{M} = \mathsf{MatEcd}_k^d(\{\boldsymbol{M}_\ell\}) \pmod q,$$
$$\boldsymbol{B}' + \mathsf{Toep}_k^d(\mathsf{sk}) \cdot \boldsymbol{A}' \approx \boldsymbol{M}' = \mathsf{MatEcd}_k^d(\{\boldsymbol{M}'_\ell\}) \pmod q.$$

We transpose the column-wise matrix encryption $(\boldsymbol{B}', \boldsymbol{A}')$ to obtain a row-wise matrix encryption $(\underline{\boldsymbol{B}}', \underline{\boldsymbol{A}}')$ of $\boldsymbol{M}'$:

$$\underline{\boldsymbol{B}}' + \underline{\boldsymbol{A}}' \cdot \mathsf{Toep}_k^d(\mathsf{sk})^T \approx \boldsymbol{M}' \pmod q.$$

Then, we multiply $(\boldsymbol{B}, \boldsymbol{A})$ and $(\underline{\boldsymbol{B}}', \underline{\boldsymbol{A}}')$ in the block matrix representation:

$$\begin{aligned}\boldsymbol{M} \cdot \boldsymbol{M}' &\approx [\boldsymbol{I}_d \mid \mathtt{Toep}_k^d(\mathsf{sk})] \begin{bmatrix}\boldsymbol{B} \\ \boldsymbol{A}\end{bmatrix} [\underline{\boldsymbol{B}}' \mid \underline{\boldsymbol{A}}'] \begin{bmatrix}\boldsymbol{I}_d \\ \mathtt{Toep}_k^d(\mathsf{sk})^T\end{bmatrix} \\ &= [\boldsymbol{I}_d \mid \mathtt{Toep}_k^d(\mathsf{sk})] \begin{bmatrix}\boldsymbol{C}_{0,0} & \boldsymbol{C}_{0,1} \\ \boldsymbol{C}_{1,0} & \boldsymbol{C}_{1,1}\end{bmatrix} \begin{bmatrix}\boldsymbol{I}_d \\ \mathtt{Toep}_k^d(\mathsf{sk})^T\end{bmatrix}\end{aligned}$$

We transpose two row-wise matrix encryptions $(\boldsymbol{C}_{0,0}, \boldsymbol{C}_{0,1})$ and $(\boldsymbol{C}_{1,0}, \boldsymbol{C}_{1,1})$ to obtain two column-wise matrix encryptions $(\boldsymbol{D}_0, \boldsymbol{D}_1)$ and $(\boldsymbol{D}_2, \boldsymbol{D}_3)$ as:

$$\begin{aligned}\boldsymbol{C}_{0,0} + \boldsymbol{C}_{0,1} \cdot \mathtt{Toep}_k^d(\mathsf{sk})^T &\approx \boldsymbol{D}_0 + \mathtt{Toep}_k^d(\mathsf{sk}) \cdot \boldsymbol{D}_1 \pmod q, \\ \boldsymbol{C}_{1,0} + \boldsymbol{C}_{1,1} \cdot \mathtt{Toep}_k^d(\mathsf{sk})^T &\approx \boldsymbol{D}_2 + \mathtt{Toep}_k^d(\mathsf{sk}) \cdot \boldsymbol{D}_3 \pmod q,\end{aligned}$$

so one has

$$\begin{aligned}\boldsymbol{M} \cdot \boldsymbol{M}' &\approx [\boldsymbol{I}_d \mid \mathtt{Toep}_k^d(\mathsf{sk})] \begin{bmatrix}\boldsymbol{D}_0 + \mathtt{Toep}_k^d(\mathsf{sk}) \cdot \boldsymbol{D}_1 \\ \boldsymbol{D}_2 + \mathtt{Toep}_k^d(\mathsf{sk}) \cdot \boldsymbol{D}_3\end{bmatrix} \\ &= \boldsymbol{D}_0 + \mathtt{Toep}_k^d(\mathsf{sk}) \cdot (\boldsymbol{D}_1 + \boldsymbol{D}_2) + \mathtt{Toep}_k^d(\mathsf{sk}^2) \cdot \boldsymbol{D}_3.\end{aligned}$$

From the matrix encryption $(\boldsymbol{0}, \boldsymbol{D}_3)$ with respect to the secret key sk^2, we apply Relin to obtain a matrix encryption $(\boldsymbol{E}_0, \boldsymbol{E}_1)$:

$$\mathtt{Toep}_k^d(\mathsf{sk}^2) \cdot \boldsymbol{D}_3 \approx \boldsymbol{E}_0 + \mathtt{Toep}_k^d(\mathsf{sk}) \cdot \boldsymbol{E}_1 \pmod q.$$

As the last step, we rescale each column of $(\boldsymbol{D}_0 + \boldsymbol{E}_0, \boldsymbol{D}_1 + \boldsymbol{D}_2 + \boldsymbol{E}_1)$ to obtain the column-wise matrix encryption of $\boldsymbol{M} \cdot \boldsymbol{M}' = \mathsf{MatEcd}_k^d(\{\boldsymbol{M}_\ell \cdot \boldsymbol{M}'_\ell\})$. We provide the entire batch CCMM algorithm in Algorithm 4.

Algorithm 4: Batch CCMM

Input: Matrix encryption $(\boldsymbol{B}, \boldsymbol{A})$ of $\boldsymbol{M} = \mathsf{MatEcd}_k^d(\{\boldsymbol{M}_\ell\}_{\ell \in [k/2]} \subset \mathbb{C}^{d \times d}) \in \mathcal{R}_{q,k}^{d \times d}$ and matrix encryption $(\boldsymbol{B}', \boldsymbol{A}')$ of $\boldsymbol{M}' = \mathsf{MatEcd}_k^d(\{\boldsymbol{M}'_\ell\}_{\ell \in [k/2]} \subset \mathbb{C}^{d \times d}) \in \mathcal{R}_{q_1,k}^{d \times d}$.

Input: Relinearization key $\mathsf{swk}_{q,p,\mathsf{sk}^2 \to \mathsf{sk}}$.

Output: Matrix encryption $(\boldsymbol{B}_{\mathrm{res}}, \boldsymbol{A}_{\mathrm{res}})$ of $\boldsymbol{M} \cdot \boldsymbol{M}' = \mathsf{MatEcd}_k^d(\{\boldsymbol{M}_\ell \cdot \boldsymbol{M}'_\ell\}_{\ell \in [k/2]}) \in \mathcal{R}_{q/\Delta,k}^{d \times d}$

1 $(\underline{\boldsymbol{B}}, \underline{\boldsymbol{A}}) \leftarrow \mathsf{CMT}_k^d((\boldsymbol{B}', \boldsymbol{A}'))^T$

2 $\begin{bmatrix}\boldsymbol{C}_{0,0} & \boldsymbol{C}_{0,1} \\ \boldsymbol{C}_{1,0} & \boldsymbol{C}_{1,1}\end{bmatrix} \leftarrow \begin{bmatrix}\boldsymbol{B} \\ \boldsymbol{A}\end{bmatrix} \cdot [\underline{\boldsymbol{B}} \mid \underline{\boldsymbol{A}}]$

3 $(\boldsymbol{D}_0, \boldsymbol{D}_1) \leftarrow \mathsf{CMT}_k^d((\boldsymbol{C}_{0,0}, \boldsymbol{C}_{0,1}))^T$

4 $(\boldsymbol{D}_2, \boldsymbol{D}_3) \leftarrow \mathsf{CMT}_k^d((\boldsymbol{C}_{1,0}, \boldsymbol{C}_{1,1}))^T$

5 $(\boldsymbol{E}_0, \boldsymbol{E}_1) \leftarrow \mathsf{KeySwitch}_{\mathsf{sk}^2 \to \mathsf{sk}}((\boldsymbol{0}, \boldsymbol{D}_3))$

6 $(\boldsymbol{B}_{\mathrm{res}}, \boldsymbol{A}_{\mathrm{res}}) \leftarrow (\boldsymbol{E}_0, \boldsymbol{E}_1) + (\boldsymbol{D}_0, \boldsymbol{D}_1 + \boldsymbol{D}_2)$

7 $(\boldsymbol{B}_{\mathrm{res}}, \boldsymbol{A}_{\mathrm{res}}) \leftarrow (\lfloor \frac{1}{\Delta} \cdot \boldsymbol{B}_{\mathrm{res}} \rceil, \lfloor \frac{1}{\Delta} \cdot \boldsymbol{A}_{\mathrm{res}} \rceil)$ ▷ Rescale by Δ

8 **return** $(\boldsymbol{B}_{res}, \boldsymbol{A}_{res})$

Cost analysis. In summary, our batch CCMM reduces CCMM over $\mathcal{R}_{q,k}$ to four PPMMs over $\mathcal{R}_{q,k}$ (Step 2), three CMTs over $\mathcal{R}_{q,k}$ (Steps 1, 3, and 4), d

key-switchings (Step 5), and d rescalings as described in Algorithm 4, resulting in a $k/2$-batch CCMM over $\mathbb{C}$. Assuming that ciphertexts are NTT transformed over $\mathcal{R}_{q,k}$ in advance, each rescaling requires NTT conversion and inverse NTT conversion before and after, incurring $O(d^2 k \log k)$ $\mathbb{Z}_q$-operations. The total computational cost is $4k \cdot T_{\mathrm{PPMM}}(d) + O(d^2 k \log dk)$ $\mathbb{Z}_q$-operations.

Lightweight CMT and CCMM algorithm. As in [36], a potential concern for the batch CMT and CCMM algorithms is the size of the evaluation keys. While the batch algorithm requires reduced key size of $|H| = d$ compared to N evaluation keys in [36], this may still be unaffordable for small devices. To address this issue, we adopt the lightweight CMT algorithm of [36], building on the method proposed in [27], and extend it to our batch CMT setting. Since $H = \{2ki + 1 \mid i \in [d]\} = \langle 5^{k/2} \rangle \leq \mathbb{Z}_{2N}^{\times}$, the Galois group Gal_k^N is cyclic with a single generator $X \mapsto X^{5^{k/2}}$. Assuming $\mathsf{dnum} = 1$ for brevity, the required automorphism keys for batch CMT are of the form:

$$\left\{\mathsf{Enc}(p \cdot \mathsf{sk}(X^{5^{i \cdot (k/2)}}))\right\}_{i \in [d]}$$

Consequently, one switching key can be derived from another by evaluating the automorphism $X \mapsto X^{5^{k/2}}$. Given a single master rotation key $\mathsf{Enc}(p' p \cdot \mathsf{sk}(X))$ and a trivial initial switching key $\mathsf{Enc}(p \cdot \mathsf{sk}(X)) = (0, p)$, all required automorphism keys can be generated. Hence, only a single master automorphism key for $X \mapsto X^{5^{k/2}}$ is needed for CMT, and one additional relinearization key suffices for batch CCMM.

5 Rectangular CPMM and CCMM

Rectangular matrix multiplication refers to the product of a $d_1 \times d_2$ matrix encryption and a $d_2 \times d_3$ matrix, where the dimensions d_1, d_2, and d_3 are not necessarily equal. In particular, the case $d_1 < N$ has not been fully addressed for either CPMM or CCMM, and previous approaches [3,36] still have complexity depending asymptotically on N rather than d_1.

In this section, we address this gap by partitioning the input into $d_1 \times d_1$ blocks and applying our batch CPMM or CCMM algorithms. The algorithms take batch matrix encryptions as input and output coefficient-encoded encryptions, which are suitable for bootstrapping. For clarity, we focus on multiplying a $d \times N/2$ matrix by an $N/2 \times N/2$ matrix, where $d \mid N$. This setting naturally generalizes to multiplying a $d_1 \times d_2$ matrix by a $d_2 \times d_2$ matrix whenever $d_1 \mid d_2$.

5.1 Rectangular Matrix Multiplication via Batching

Let $\boldsymbol{M} \in \mathbb{C}^{d \times N/2}$ and $\boldsymbol{U} \in \mathbb{C}^{N/2 \times N/2}$, where $N = dk$. We partition $\boldsymbol{M}$ and $\boldsymbol{U}$ into block matrices with blocks $\boldsymbol{M}_i \in \mathbb{C}^{d \times d}$ and $\boldsymbol{U}_i \in \mathbb{C}^{d \times N/2}$ as

$$\boldsymbol{M} = [\, \boldsymbol{M}_0 \mid \cdots \mid \boldsymbol{M}_{k/2-1} \,], \quad \boldsymbol{U}^T = [\, \boldsymbol{U}_0^T \mid \cdots \mid \boldsymbol{U}_{k/2-1}^T \,].$$

Then we have $\boldsymbol{M} \cdot \boldsymbol{U} = \sum_{\ell \in [k/2]} \boldsymbol{M}_\ell \cdot \boldsymbol{U}_\ell$, which implies that rectangular MM is reduced to a batch MM between the collections $\{\boldsymbol{M}_\ell\}_{\ell \in [k/2]}$ and $\{\boldsymbol{U}_\ell\}_{\ell \in [k/2]}$, followed by an aggregation across batches. More specifically, we encode the matrices

$$\boldsymbol{M} = \mathsf{MatEcd}_k^d(\{\boldsymbol{M}_\ell\}) \in \mathcal{R}_k^{d \times d}, \quad \boldsymbol{U} = \mathsf{MatEcd}_k^d(\{\boldsymbol{U}_\ell\}) \in \mathcal{R}_k^{d \times N/2},$$

and our batch CPMM or CCMM will output the matrix encryption of

$$\boldsymbol{V} := \boldsymbol{M} \cdot \boldsymbol{U} = \mathsf{MatEcd}_k^d(\{\boldsymbol{V}_\ell = \boldsymbol{M}_\ell \cdot \boldsymbol{U}_\ell\}_{\ell \in [k/2]}) \in \mathcal{R}_k^{d \times N/2}.$$

Thus, what remains to be addressed is to obtain the encryptions of $\sum_\ell \boldsymbol{V}_\ell$. We next address how to efficiently perform this summation process in the encrypted setting.

5.2 Summation with Encoding Conversion

Suppose we are given a matrix encryption of $\boldsymbol{V} = \mathsf{MatEcd}(\{\boldsymbol{V}_\ell\}_{\ell \in [k/2]}) \in \mathcal{R}_k^{d \times N/2}$. Our goal is to aggregate these batches into a matrix encryption of $\sum_{\ell \in [k/2]} \boldsymbol{V}_\ell \in \mathbb{C}^{d \times N/2}$. By definition, the (i,j)-entry of $\boldsymbol{V}$ is given as

$$\boldsymbol{V}_{i,j} = \mathsf{iDFT}_k((\boldsymbol{V}_0)_{i,j}, \ldots, (\boldsymbol{V}_{k/2-1})_{i,j}) \in \mathcal{R}_k.$$

The key observation is that, when examining the coefficients of $\boldsymbol{V}_{i,j}$, the quantity $\sum_\ell (\boldsymbol{V}_\ell)_{i,j}$ appears as the constant term of $\boldsymbol{V}_{i,j}$ for each i, j. Therefore, the task reduces to extracting the constant terms, which is precisely the content of the following lemma.

Lemma 1. *Consider a complex vector $\boldsymbol{z} \in \mathbb{C}^{k/2}$. For $m = \mathsf{iDFT}_k(\boldsymbol{z}) \in \mathcal{R}_k$ with $[\![m]\!] = (m_0, \ldots, m_{k/2-1}) \in \mathbb{C}^{k/2}$, we have*

$$\sum_{j \in [k/2]} \boldsymbol{z}_j = (k/2) \cdot m_0.$$

Proof. Since $\boldsymbol{z} = \mathsf{DFT}_k(m) = \boldsymbol{V}_k \cdot [\![m]\!]$, we have $z_i = \sum_{j \in [k/2]} m_j \cdot \omega_{2k}^{5^i \cdot j}$ for $i \in [k/2]$. Since $\{5^i \pmod{2k} \mid i \in [k/2]\} = \{4i+1 \mid i \in [k/2]\}$, we have

$$\begin{aligned}
\sum_{i \in [k/2]} z_i &= \sum_{i \in [k/2]} \sum_{j \in [k/2]} m_j \omega_{2k}^{5^i \cdot j} = \sum_{j \in [k/2]} m_j \cdot \left(\sum_{i \in [k/2]} \omega_{2k}^{5^i \cdot j} \right) \\
&= \sum_{j \in [k/2]} m_j \cdot \left(\sum_{i \in [k/2]} \omega_{2k}^{(4i+1) \cdot j} \right) = \sum_{j \in [k/2]} m_j \omega_{2k}^{j} \cdot \left(\sum_{i \in [k/2]} \omega_{k/2}^{i \cdot j} \right) \\
&= (k/2) \cdot m_0.
\end{aligned}$$

Here, we note that m_0 is the constant term of $m \in \mathcal{R}_k = \mathbb{Z}[i][Y]/(Y^{k/2} - i)$. □

By leveraging Lemma 1, the summation process in our rectangular matrix multiplication can be carried out by extracting the constant terms. To facilitate the extractions, we use CMT algorithms for format conversions between column-wise and row-wise matrix encryptions. We provide the corresponding theorem, and present the rectangular CPMM and CCMM algorithms in Algorithms 5 and 6, respectively.

Theorem 3 (Summation of encrypted matrices). *Suppose we are given a collection of matrices* $\{\boldsymbol{M}_\ell\}_{\ell\in[k/2]} \subset \mathbb{C}^{d\times N/2}$. *Let*

$$\widetilde{\boldsymbol{M}} = \frac{2}{k}\sum_{\ell\in[k/2]} \boldsymbol{M}_\ell = [\widetilde{\boldsymbol{M}}_0 \mid \cdots \mid \widetilde{\boldsymbol{M}}_{k/2-1}]$$

for $\widetilde{\boldsymbol{M}}_j \in \mathbb{C}^{d\times d}$. *Given a matrix encryption of the batch matrices*

$$\boldsymbol{B} + \mathtt{Toep}_k^d(\mathsf{sk})\cdot\boldsymbol{A} \approx \boldsymbol{M} = \mathsf{MatEcd}(\{\boldsymbol{M}_\ell\}) \in \mathcal{R}_{q,k}^{d\times N/2},$$

we can obtain a matrix encryption

$$\boldsymbol{B}' + \mathtt{Toep}_k^d(\mathsf{sk})\cdot\boldsymbol{A}' \approx \boldsymbol{M}' = \sum_{j\in[k/2]} \widetilde{\boldsymbol{M}}_j \cdot Y^j \in \mathcal{R}_{q,k}^{d\times d}$$

through an application of $\mathsf{CMT}_2^{N/2}$, *a truncation, and* CMT_k^d.

Proof. Write $\boldsymbol{M} = [\mathtt{Vec}_k^d(m_0) \mid \cdots \mid \mathtt{Vec}_k^d(m_{N/2-1})]$. By Theorem 1, the matrix encryption is equivalent to N RLWE ciphertexts, each encrypting m_j. Invoking Theorem 1 again with $k = 2$, we obtain an equivalent matrix encryption

$$\widehat{\boldsymbol{B}} + \mathtt{Toep}_2^{N/2}(\mathsf{sk})\cdot\widehat{\boldsymbol{A}} \approx \widehat{\boldsymbol{M}} = [\mathtt{Vec}_2^{N/2}(m_0) \mid \cdots \mid \mathtt{Vec}_2^{N/2}(m_{N/2-1})].$$

Observe that the Y^ℓ-coefficient in $\boldsymbol{M}_{i,j} = (\mathtt{Vec}_k^d(m_j))_i \in \mathcal{R}_k$ corresponds to $(\mathtt{Vec}_2^{N/2}(m_j))_{d\ell+i}$. For $i \in [d]$, the constant term of $\boldsymbol{M}_{i,j}$ is $(\mathtt{Vec}_2^{N/2}(m_j))_i$. Hence, by Lemma 1, it suffices to extract the top d rows of $\widehat{\boldsymbol{M}}$.

To this end, we first apply $\mathsf{CMT}_2^{N/2}$ to obtain a row-wise encryption

$$\underline{\boldsymbol{B}} + \underline{\boldsymbol{A}}\cdot\mathtt{Toep}_2^{N/2}(\mathsf{sk}) \approx \widehat{\boldsymbol{M}} \pmod q.$$

For each $i \in [d]$, there exists a triple $(\underline{b}_i, \underline{a}_i, \mu_i) \in \mathcal{R}_{q,N}^2$ such that $\mathtt{Vec}_2^{N/2}(\underline{b}_i)$, $\mathtt{Vec}_2^{N/2}(\underline{a}_i)$, and $\mathtt{Vec}_2^{N/2}(\mu_i)$ correspond to the i-th row vectors of $\underline{\boldsymbol{B}}$, $\underline{\boldsymbol{A}}$, and $\underline{\boldsymbol{M}}$, respectively. Then, we have

$$\underline{b}_i + \underline{a}_i\cdot\mathsf{sk} \approx \mu_i \pmod q, \quad i \in [d].$$

We then apply Theorem 1 with $k = N/d$ in a row-wise manner to obtain a row-wise matrix encryption:

$$\boldsymbol{B}' + \boldsymbol{A}'\cdot\mathtt{Toep}_k^d(\mathsf{sk})^T \approx \begin{bmatrix} \mathtt{Vec}_k^d(\mu_0) \\ \vdots \\ \mathtt{Vec}_k^d(\mu_{d-1}) \end{bmatrix} = \boldsymbol{M}' \in \mathcal{R}_{q,k}^{d\times d}.$$

Note that $\boldsymbol{M}_{\mathrm{trunc}}$, obtained by taking the top-d rows of $\widehat{\boldsymbol{M}}$, satisfies

$$\boldsymbol{M}_{\mathrm{trunc}} = \begin{bmatrix} \mathtt{Vec}_2^{N/2}(\mu_0) \\ \vdots \\ \mathtt{Vec}_2^{N/2}(\mu_{d-1}) \end{bmatrix},$$

whose data is reflected in $\boldsymbol{M}'$. Finally, applying CMT_k^d yields a column-wise encryption

$$\bar{\boldsymbol{B}}' + \mathtt{Toep}_k^d(\mathsf{sk}) \cdot \bar{\boldsymbol{A}}' \approx \boldsymbol{M}' \pmod q.$$

Then $(\bar{\boldsymbol{B}}', \bar{\boldsymbol{A}}')$ is a matrix encryption of $\boldsymbol{M}'$, as desired. □

Algorithm 5: Rectangular CPMM algorithm

Input: Matrix encryption $\mathbf{ct}_M$ of $\mathsf{MatEcd}(\{\boldsymbol{M}_l\}_{l\in[k/2]} \subset \mathbb{C}^{d\times N/2})$, where $\boldsymbol{M} = [\boldsymbol{M}_0 \mid \cdots \mid \boldsymbol{M}_{k/2-1}]$.

Input: Encoded matrix $\widetilde{\boldsymbol{U}} = \mathsf{MatEcd}(\{\boldsymbol{U}_l\}_{l\in[k/2]} \subset \mathbb{C}^{N/2\times N/2})$, where $\boldsymbol{U}^T = [\boldsymbol{U}_0^T \mid \cdots \mid \boldsymbol{U}_{k/2-1}^T]$.

Output: Matrix encryption $\mathbf{ct}_W$ of coefficient-encoded $\{\boldsymbol{W}_l\}_{l\in[k/2]} \subset \mathbb{C}^{d\times d}$, where $\boldsymbol{W} = \boldsymbol{M}\cdot\boldsymbol{U} = [\boldsymbol{W}_0 \mid \cdots \mid \boldsymbol{W}_{k/2-1}]$.

$\mathbf{ct}_V \leftarrow \frac{k}{2} \cdot \mathsf{CPMM}(\mathbf{ct}_M, \widetilde{\boldsymbol{U}})$
$\{\mathsf{ct}'_j\}_{j\in[N/2]} \leftarrow \mathsf{CMT}_2^{N/2}(\boldsymbol{ct}_V)$
$\mathbf{ct}_W \leftarrow \mathsf{CMT}_k^d(\{\mathsf{ct}'_j\}_{j\in[d]})$
return $\mathbf{ct}_W$

Algorithm 6: Rectangular CCMM algorithm

Input: Matrix encryption $\mathbf{ct}_M$ of $\mathsf{MatEcd}(\{\boldsymbol{M}_l\}_{l\in[k/2]} \subset \mathbb{C}^{d\times d})$, where $\boldsymbol{M} = [\boldsymbol{M}_0 \mid \cdots \mid \boldsymbol{M}_{k/2-1}]$,

Input: Matrix encryption $\boldsymbol{ct}_U$ of $\mathsf{MatEcd}(\{\boldsymbol{U}_l\}_{l\in[k/2]} \subset \mathbb{C}^{N/2\times N/2})$, where $\boldsymbol{U}^T = [\boldsymbol{U}_0^T \mid \cdots \mid \boldsymbol{U}_{k/2-1}^T]$.

Output: Matrix encryption $\boldsymbol{ct}_W$ of coefficient-encoded $\{\boldsymbol{W}_l\}_{l\in[k/2]} \subset \mathbb{C}^{d\times d}$, where $\boldsymbol{W} = \boldsymbol{M}\cdot\boldsymbol{U} = [\boldsymbol{W}_0 \mid \cdots \mid \boldsymbol{W}_{k/2-1}]$.

$[\boldsymbol{ct}_{U,0} \mid \cdots \mid \boldsymbol{ct}_{U,k-1}] \leftarrow \boldsymbol{ct}_U$ ▷ Split into square $d \times d$ batch matrices for Batch CCMM
for $i \leftarrow 0$ **to** $k-1$ **do**
  $\boldsymbol{ct}_{V,i} \leftarrow \frac{k}{2} \cdot \mathsf{CCMM}(\boldsymbol{ct}_M, \boldsymbol{ct}_{U,i})$
$\boldsymbol{ct}_V \leftarrow \left[\boldsymbol{ct}_{V,0} \mid \cdots \mid \boldsymbol{ct}_{V,k-1}\right]$ ▷ Combine back $d \times d$ batch matrices
$\{\mathsf{ct}'_j\}_{j\in[N/2]} \leftarrow \mathsf{CMT}_2^{N/2}(\boldsymbol{ct}_V)$
$\boldsymbol{ct}_W \leftarrow \mathsf{CMT}_k^d(\{\mathsf{ct}'_j\}_{j\in[d]})$
return $\boldsymbol{ct}_W$

Cost analysis. The rectangular CPMM/CCMM is computed in two steps. We perform a $k/2$-batch CPMM/CCMM of size $d \times d \times N$ over $\mathbb{C}$ (Step 1) and the results are aggregated using $\mathsf{CMT}_2^{N/2}$ and CMT_k^d (Step 2,3). Hence, rectangular CPMM and CCMM require $2k^2 \cdot T_{\mathsf{PPMM}}(d) + O(N^2 \log N)$ and $4k^2 \cdot T_{\mathsf{PPMM}}(d) + O(N^2 \log N)$ $\mathbb{Z}_q$-operations, respectively.

6 Batch Matrix Multiplication in Homomorphic Computation

Although batch matrix encoding accelerates CPMM and CCMM, many subsequent homomorphic operations are more naturally performed in slot encoding. In this section, we show how to combine these two representations in encrypted computation.

We first give a conversion algorithm from batch slot encodings to batch matrix encoding. The conversion follows from the linear relation between the two representations and is implemented using one batch CPMM and two applications of batch CMT. We then show that the conversion back to slot encoding can be integrated into CKKS bootstrapping: after batch matrix computations, the bootstrapping routine refreshes the ciphertexts, restores the modulus level, and returns slot-encoded outputs using the standard optimized subroutines.

6.1 Encoding Conversion for Slot-Encoded Computations

Recall that $\mathsf{rev}_n : [n] \to [n]$ denotes the bit-reversal permutation. We observe that the batch matrix encoding can be viewed as the mapping

$$\mathsf{MatEcd}_k^d : (\mathbb{C}^{k/2})^{d\times d} \longrightarrow \mathcal{R}_{q,k}^{d\times d}, \qquad \mathsf{MatEcd}_k^d((\boldsymbol{z}_{i,j})) = (m_{i,j})_{i,j\in[d]},$$

where $[\![m_{i,j}]\!]_{\mathsf{rev}} = \boldsymbol{W}_{k/2}^{-1} \cdot \boldsymbol{z}_{i,j}$. We recall the decomposition of the DFT matrix $\boldsymbol{W}_{N/2} = \boldsymbol{S}_{N/2\leftarrow k/2} \cdot \boldsymbol{S}_{k/2\leftarrow 1}$, where $\boldsymbol{S}_{k/2\leftarrow 1} = \boldsymbol{I}_d \otimes \boldsymbol{W}_{k/2}$ and

$$\boldsymbol{S}_{N/2\leftarrow k/2} = \big(\mathsf{diag}_{k/2}(\boldsymbol{w}_{i,j})\big)_{i,j\in[d]}, \quad \boldsymbol{S}_{N/2\leftarrow k/2}^{-1} = \big(\mathsf{diag}_{k/2}(\boldsymbol{w}'_{i,j})\big)_{i,j\in[d]},$$

for some complex vectors $\boldsymbol{w}_{i,j}, \boldsymbol{w}'_{i,j} \in \mathbb{C}^{k/2}$. We denote the matrices $\boldsymbol{T}_{N/2\leftarrow k/2} = \mathsf{MatEcd}_k^d((\boldsymbol{w}_{i,j}))$ and $\boldsymbol{T}'_{N/2\leftarrow k/2} = \mathsf{MatEcd}_k^d((\boldsymbol{w}'_{i,j}))$ over $\mathcal{R}_k$. We now formalize the relationship between batch matrix encoding and slot encoding.

Lemma 2. *Let $\boldsymbol{Z} = (\boldsymbol{z}_{i,j}) \in (\mathbb{C}^{k/2})^{d\times d}$. For each $j \in [d]$, we set*

$$\boldsymbol{z}_j = (\boldsymbol{z}_{0,j} \mid \cdots \mid \boldsymbol{z}_{d-1,j}) \in \mathbb{C}^{N/2}.$$

Then the batch matrix encoding admits the column-wise decomposition

$$\mathsf{MatEcd}_k^d(\boldsymbol{Z}) = [\mathtt{Vec}_k^d(\mu_0) \mid \cdots \mid \mathtt{Vec}_k^d(\mu_{d-1})], \quad \mu_j = \mathsf{Ecd}((\boldsymbol{S}_{N/2\leftarrow k/2}\, \boldsymbol{P}_{out}) \cdot \boldsymbol{z}_j),$$

where $\boldsymbol{P}_{out} = \boldsymbol{P}_d \otimes \boldsymbol{I}_{k/2}$ permutes the blocks by bit-reversed order, i.e.,

$$\boldsymbol{P}_{out} \cdot (\boldsymbol{z}_{0,j} \mid \cdots \mid \boldsymbol{z}_{d-1,j}) = (\boldsymbol{z}_{\mathsf{rev}_d(0),j} \mid \cdots \mid \boldsymbol{z}_{\mathsf{rev}_d(d-1),j}).$$

Proof. Let $\mathsf{MatEcd}(\boldsymbol{Z}) = \boldsymbol{M} = (m_{i,j})_{i,j\in[d]}$, so that $[\![m_{i,j}]\!]_{\mathsf{rev}} = \boldsymbol{W}_{k/2}^{-1} \cdot \boldsymbol{z}_{i,j}$. Once we denote $\mu_j = \sum_{i\in[d]} m_{i,j} \cdot X^i$, we have $\boldsymbol{M} = [\mathtt{Vec}_k^d(\mu_0) \mid \cdots \mid \mathtt{Vec}_k^d(\mu_{d-1})]$ by

definition. Noting that $\mathsf{rev}_{N/2}(i + d\ell) = (k/2) \cdot \mathsf{rev}_d(i) + \mathsf{rev}_{k/2}(\ell)$ for $i \in [d]$ and $\ell \in [k/2]$, one can verify that

$$[\![\mu_j]\!]_{\mathsf{rev}} = \big([\![m_{\mathsf{rev}_d(0),j}]\!]_{\mathsf{rev}} \mid \cdots \mid [\![m_{\mathsf{rev}_d(d-1),j}]\!]_{\mathsf{rev}}\big).$$

Hence, it follows that

$$\begin{aligned}
[\![\mu_j]\!]_{\mathsf{rev}} &= \big([\![m_{\mathsf{rev}_d(0),j}]\!]_{\mathsf{rev}} \mid \cdots \mid [\![m_{\mathsf{rev}_d(d-1),j}]\!]_{\mathsf{rev}}\big) \\
&= \big(\boldsymbol{W}_{k/2}^{-1} \cdot \boldsymbol{z}_{\mathsf{rev}_d(0),j} \mid \cdots \mid \boldsymbol{W}_{k/2}^{-1} \cdot \boldsymbol{z}_{\mathsf{rev}_d(d-1),j}\big) \\
&= \big(\boldsymbol{I}_d \otimes \boldsymbol{W}_{k/2}^{-1}\big) \cdot (\boldsymbol{z}_{\mathsf{rev}_d(0),j} \mid \cdots \mid \boldsymbol{z}_{\mathsf{rev}_d(d-1),j}) \\
&= (\boldsymbol{S}_{k/2\leftarrow 1}^{-1}\, \boldsymbol{P}_{\mathrm{out}}) \cdot (\boldsymbol{z}_{0,j} \mid \cdots \mid \boldsymbol{z}_{d-1,j}).
\end{aligned}$$

Therefore, by $[\![\mathsf{Ecd}(\boldsymbol{v})]\!]_{\mathsf{rev}} = \boldsymbol{W}_{N/2}^{-1} \cdot \boldsymbol{v}$ for $\boldsymbol{v} \in \mathbb{C}^{N/2}$, we obtain

$$\begin{aligned}
[\![\mu_j]\!]_{\mathsf{rev}} &= (\boldsymbol{S}_{k/2\leftarrow 1}^{-1}\, \boldsymbol{P}_{\mathrm{out}}) \cdot \boldsymbol{z}_j = \boldsymbol{W}_{N/2}^{-1} \cdot (\boldsymbol{S}_{N/2\leftarrow k/2}\, \boldsymbol{P}_{\mathrm{out}} \cdot \boldsymbol{z}_j) \\
&= [\![\mathsf{Ecd}((\boldsymbol{S}_{N/2\leftarrow k/2}\, \boldsymbol{P}_{\mathrm{out}}) \cdot \boldsymbol{z}_j)]\!]_{\mathsf{rev}},
\end{aligned}$$

and hence $\mu_j = \mathsf{Ecd}((\boldsymbol{S}_{N/2\leftarrow k/2}\, \boldsymbol{P}_{\mathrm{out}}) \cdot \boldsymbol{z}_j)$, as claimed. □

Lemma 3. *For* $\mathsf{Ecd}(\boldsymbol{z}) \in \mathcal{R}_N$ *and* $\boldsymbol{S}_{N/2\leftarrow k/2} = (\mathit{diag}_{k/2}(\boldsymbol{w}_{i,j}))$*, we have*

$$\big(\boldsymbol{T}_{N/2\leftarrow k/2}\, \boldsymbol{P}_d\big) \cdot \mathtt{Vec}_k^d(\mathsf{Ecd}(\boldsymbol{z})) = \mathtt{Vec}_k^d\big(\mathsf{Ecd}((\boldsymbol{S}_{N/2\leftarrow k/2}\, \boldsymbol{P}_{out}) \cdot \boldsymbol{z})\big).$$

Proof. We write $\boldsymbol{z} = (\boldsymbol{z}_0 \mid \cdots \mid \boldsymbol{z}_{d-1})$, $m = \mathsf{Ecd}(\boldsymbol{z}) = \sum_j m_j(Y) \cdot X^j$, $\boldsymbol{u}_j = \mathsf{DFT}_k(\boldsymbol{m}_j)$, and $\boldsymbol{u} = (\boldsymbol{u}_0 \mid \cdots \mid \boldsymbol{u}_{d-1})$. We note that

$$\begin{aligned}
\boldsymbol{T}_{N/2\leftarrow k/2}\, \boldsymbol{P}_d &= \mathsf{MatEcd}_k^d((\boldsymbol{w}_{i,\mathsf{rev}_d(j)})), \\
\boldsymbol{S}_{N/2\leftarrow k/2}\, \boldsymbol{P}_{\mathrm{out}} &= \big(\mathsf{diag}_{k/2}(\boldsymbol{w}_{i,\mathsf{rev}_d(j)})\big).
\end{aligned}$$

By an argument similar to that of Lemma 2, we obtain

$$\boldsymbol{z} = \boldsymbol{W}_{N/2} \cdot [\![m]\!]_{\mathsf{rev}} = \boldsymbol{S}_{N/2\leftarrow k/2}\, \boldsymbol{P}_{\mathrm{out}} \cdot \boldsymbol{u}.$$

Therefore,

$$\begin{aligned}
\big(\mathsf{MatEcd}((\boldsymbol{w}_{i,j}))\, \boldsymbol{P}_d\big) \cdot \mathtt{Vec}_k^d(\mathsf{Ecd}(\boldsymbol{z})) &= \begin{bmatrix} \mathsf{iDFT}_{k/2}\Big(\sum_j \boldsymbol{w}_{0,\mathsf{rev}_d(j)} \odot \boldsymbol{u}_j\Big) \\ \vdots \\ \mathsf{iDFT}_{k/2}\Big(\sum_j \boldsymbol{w}_{d-1,\mathsf{rev}_d(j)} \odot \boldsymbol{u}_j\Big) \end{bmatrix} \\
&= [\,\mathsf{iDFT}_{k/2}(\boldsymbol{z}_0) \mid \cdots \mid \mathsf{iDFT}_{k/2}(\boldsymbol{z}_{d-1})\,]^T,
\end{aligned}$$

which equals $\mathtt{Vec}_k^d\left(\mathsf{Ecd}((\boldsymbol{S}_{N/2\leftarrow k/2}\, \boldsymbol{P}_{\mathrm{out}}) \cdot \boldsymbol{z})\right)$ by a similar argument. □

Now we show that the encoding conversion can be reduced to one batch CPMM and two applications of CMT. Suppose we have d RLWE encryptions of $\mathsf{Ecd}(\boldsymbol{z}_j)$, which can be written as

$$\boldsymbol{B} + \mathtt{Toep}_k^d(\mathsf{sk}) \cdot \boldsymbol{A} \approx [\mathtt{Vec}_k^d(\mathsf{Ecd}(\boldsymbol{z}_0)) \mid \cdots \mid \mathtt{Vec}_k^d(\mathsf{Ecd}(\boldsymbol{z}_{d-1}))] \pmod q.$$

Our goal is to obtain a matrix encryption of $\mathsf{MatEcd}_k^d(\boldsymbol{Z})$. By leveraging Lemma 3 followed by Lemma 2, it suffices to multiply the right-hand side of the above matrix equation by $\boldsymbol{T}_{N/2\leftarrow k/2}\,\boldsymbol{P}_d$ on the left. However, the position of $\mathtt{Toep}_k^d(\mathsf{sk})$ prevents us from directly performing this left multiplication.

We use our batch CMT algorithm CMT_k^d to resolve this issue. Concretely, we first apply CMT_k^d to obtain

$$\underline{\boldsymbol{B}} + \underline{\boldsymbol{A}} \cdot \mathtt{Toep}_k^d(\mathsf{sk})^T \approx [\mathtt{Vec}_k^d(\mathsf{Ecd}(\boldsymbol{z}_0)) \mid \cdots \mid \mathtt{Vec}_k^d(\mathsf{Ecd}(\boldsymbol{z}_{d-1}))] \pmod q,$$

and then left-multiply $\boldsymbol{T}_{N/2\leftarrow k/2}\,\boldsymbol{P}_d$ on both sides to obtain

$$\underline{\boldsymbol{B}'} + \underline{\boldsymbol{A}'} \cdot \mathtt{Toep}_k^d(\mathsf{sk})^T \approx \mathsf{MatEcd}_k^d(\boldsymbol{Z}) \pmod q.$$

Finally, applying CMT_k^d once more converts the result back to column-wise matrix encryption:

$$\boldsymbol{B}' + \mathtt{Toep}_k^d(\mathsf{sk}) \cdot \boldsymbol{A}' \approx \mathsf{MatEcd}_k^d(\boldsymbol{Z}) \pmod q.$$

In practice, when operating on batch matrix encoding, left-multiplication by a batch matrix can be performed before the second CMT step, whereas right-multiplication can be carried out after the second CMT step. This flexibility allows us to handle both cases seamlessly within the same framework.

We also note that if batch CPMM is applied via left-multiplication immediately after the encoding conversion, the conversion step can be fused with the subsequent batch CPMM, since both operations are instances of batch CPMM.

6.2 Bootstrapping with Batch Matrix Encoding

We propose a simple bootstrapping procedure that takes a ciphertext encrypting $\mathsf{MatEcd}_k^d(\boldsymbol{Z})$ and outputs an encryption of $\{\mathsf{Ecd}(\boldsymbol{P}_{\text{out}} \cdot \boldsymbol{z}_j)\}_{j\in[d]}$, as summarized by the following sequence:

$$\mathsf{ModRaise} \;\rightarrow\; \mathsf{C2S} \;\rightarrow\; \mathsf{EvalMod} \;\rightarrow\; \mathsf{Eval}\boldsymbol{S}_{k/2\leftarrow 1}.$$

The steps are as follows:

(1) $\mathsf{ModRaise}$ lifts ciphertexts encrypting $\mathsf{MatEcd}_k^d(\boldsymbol{Z}) = (\mathtt{Vec}_k^d(m_j))_{j\in[d]}$ at modulus q_0 to ciphertexts encrypting $m_j + q_0 \cdot I_j$ for each $j \in [d]$ at a larger modulus $Q > q_0$, where I_j is a small unknown integer polynomial. Before applying $\mathsf{ModRaise}$, we note that by Lemma 2,

$$\mathsf{Ecd}(\boldsymbol{W}_{N/2} \cdot [\![m_j]\!]_{\mathsf{rev}}) = m_j = \mathsf{Ecd}(\boldsymbol{S}_{N/2\leftarrow k/2}\boldsymbol{P}_{\text{out}} \cdot \boldsymbol{z}_j),$$

so that $\boldsymbol{S}_{k/2\leftarrow 1} \cdot [\![m_j]\!]_{\mathsf{rev}} = \boldsymbol{P}_{\text{out}}\,\boldsymbol{z}_j$.

(2) $\mathsf{C2S}$ homomorphically evaluates the matrix $\boldsymbol{W}_N^{-1}$, producing ciphertexts encrypting $\mathsf{Ecd}([\![m_j]\!]_{\mathsf{rev}} + q_0 \cdot [\![I_j]\!]_{\mathsf{rev}})$.

(3) $\mathsf{EvalMod}$ homomorphically applies a mod-q_0 function to remove the $q_0 \cdot [\![I_j]\!]_{\mathsf{rev}}$ term, yielding ciphertexts encrypting $\mathsf{Ecd}([\![m_j]\!]_{\mathsf{rev}})$.

(4) $\mathsf{Eval}\boldsymbol{S}_{k/2\leftarrow 1}$ homomorphically evaluates the matrix $\boldsymbol{S}_{k/2\leftarrow 1}$ to obtain

$$\mathsf{Ecd}(\boldsymbol{S}_{k/2\leftarrow 1} \cdot [\![m_j]\!]_{\mathsf{rev}}) = \mathsf{Ecd}(\boldsymbol{P}_{\mathrm{out}}\, \boldsymbol{z}_j).$$

This bootstrapping procedure enables batch matrix multiplication to be performed at the lowest ciphertext modulus, thereby maximizing computational efficiency. Importantly, our modified bootstrapping coincides with the corresponding subroutine of the standard CKKS bootstrapping algorithm, allowing us to fully leverage existing optimization techniques.

Once the batch multiplication is completed, the refreshed ciphertexts can be directly reused to evaluate SIMD-style operations under slot encoding, including non-linear functions.

Remark. We note that the output of the above bootstrapping has its slots permuted by $\boldsymbol{P}_{\mathrm{out}}$. Depending on the subsequent operations under slot encoding, this permutation may be undesirable.

This issue can be resolved in a simple manner. Before invoking bootstrapping, we perform all intended left matrix multiplications and then additionally multiply by $\boldsymbol{P}_d$ at the final stage. In this case, the input to bootstrapping becomes $\mathsf{MatEcd}_k^d(\boldsymbol{P}_d \cdot \boldsymbol{Z})$, and the output is $\{\mathsf{Ecd}(\boldsymbol{z}_j)\}_{j\in[d]}$, thereby eliminating the permutation effect.

7 Experimental Results

In this section, we present implementation results for the proposed MM algorithms. Our implementations are based on the HEaaN library [12] for CKKS and the FLINT library [40] for modular matrix multiplication. All experiments were conducted on an Intel Xeon Gold 6542Y CPU at 2.90 GHz using a single thread.

The parameter sets are summarized in Table 2. All parameters achieve 128-bit security according to [2]. Here, N denotes the RLWE ring degree, QP the modulus used for key switching, and Q the ciphertext modulus, including both the base modulus and the multiplicative levels. The parameter dnum is the gadget decomposition rank for key switching, and Δ is the scaling factor. For the S13H parameter set, the first QP and dnum values are used for homomorphic computations, while the second values are used for key updates in the lightweight batch CMT algorithm. The F16 parameter set supports CKKS bootstrapping. In this set, Mult denotes the modulus used for both multiplication and $\mathsf{S2C}$, whereas BTS denotes the modulus used for $\mathsf{C2S}$ and $\mathsf{EvalMod}$.

7.1 Encoding, Conversion, and Bootstrapping

We first provide a performance analysis of our batch matrix encoding under various ciphertext operations. Specifically, we evaluate the precision after matrix encryption, key-switching, and ciphertext transpose under slot encoding, coefficient encoding, and our batch matrix encoding with the S13b parameter. The detailed results are reported in Table 3.

Parameter	$\log_2 N$	$\log_2 \Delta$	$\log_2 Q$			$\log_2(QP)$	dnum	hwt
			Base	Mult	BTS			
S12	12	28	36	28×1	-	104	2	256
S13b	13	28	36	28×3	-	160	4	256
S13H	13	28	38	28×1	-	(117, 178)	(2, 3)	256
F16	16	42	58	42×12	58×12	1555	5	256

Table 2. Parameter sets for our implementations. For S13H, the first QP and dnum values are used for homomorphic computations, while the second values are used for key updates in the lightweight batch CMT algorithm.

Slot encoding and batch matrix encoding show lower precision than coefficient encoding after encryption and key-switching, due to the transformations by DFT_N and DFT_k, respectively, in the encoded domains. For CMT, batch matrix encoding achieves precision comparable to coefficient encoding, since CMT_k^d uses only d key-switchings, compared with N key-switchings for CMT_1^N. We refer to Appendix A.2 for a detailed noise analysis.

Method	d	Batch	Encryption (bit)	Key-switch (bit)	Transpose (bit)
Slot			18.1	16.1	-
Coeff			24.2	22.8	16.0
Ours	16	256	20.0	18.6	16.9
	32	128	20.5	19.1	16.9
	64	64	20.9	19.7	16.8
	128	32	21.3	20.1	16.8
	256	16	21.9	20.5	16.6
	512	8	22.2	21.0	16.5

Table 3. Precision of encryption, key-switching, and transpose on S13b. Minimum precision is taken over 20 runs; accuracy is reported in bits.

Table 4 reports the cost of converting d slot-encoded ciphertexts to one batch matrix-encoded ciphertext under S13b. The conversion consumes one level, from 2 to 1, and maintains roughly 16 bits of precision.

d	Batch	Time (s)	Accuracy (bit)
16	256	0.055	16.3
32	128	0.127	16.1
64	64	0.306	16.2
128	32	0.843	16.1
256	16	2.41	16.2
512	8	8.57	16.1

Table 4. Slot-to-MatEcd conversion on S13b. Timings are averaged over 20 runs and reported in seconds; accuracy is reported in bits.

For bootstrapping, we use the F16 parameter set and evaluate one ciphertext encrypting a column vector; the total cost scales linearly with the number of

ciphertexts. Table 5 compares bootstrapping procedures that start from coefficient, slot, and batch matrix encodings and output ciphertexts in slot encoding. Our batch matrix-encoded inputs use the same optimized CKKS bootstrapping subroutines and achieve comparable precision without a separate conversion step.

Encoding	Level	Time (sec)	Accuracy (bit)
Coeff	$0 \to 12$	7.15	21.41
Slot	$3 \to 12$	9.14	21.74
MatEcd, $k = 32$	$0 \to 11$	8.23	21.92
MatEcd, $k = 512$	$0 \to 10$	8.39	22.36

Table 5. Bootstrapping performance on F16 for a single ciphertext. The output is slot-encoded in all cases. Timings are averaged over 10 runs and reported in seconds; accuracy is reported in bits.

7.2 Experimental Results on MM Algorithms

In this section, we evaluate our encrypted MM algorithms, including batch and rectangular CPMM/CCMM, and compare them with [23,3,36]. For each experiment, we sample each matrix entry uniformly from $[-1, 1]$. Given the ideal output matrix $\boldsymbol{W}$ and the computed result $\boldsymbol{W}'$, we report the relative precision in bits $-\log_2\big(\|\boldsymbol{W} - \boldsymbol{W}'\|_\infty / \|\boldsymbol{W}\|_\infty\big)$, where $\|\boldsymbol{M}\|_\infty = \max_{i,j} |\boldsymbol{M}_{i,j}|$.

For batch CPMM/CCMM, we use the S13b parameter set with ring degree $N = 2^{13}$, and set the batch size to $k = N/d$ for CPMM, using the optimization in Section 4.1, and to $k = N/(2d)$ for CCMM. For the baselines, we compactly pack the available $N/2$ slots for [23], evaluate k iterations of the MLWE-based CPMM of [3] including format-conversion overhead and the associated keys, and use naïve diagonal batching for the CCMM of [36]. We compute the key size as $N \cdot \mathsf{dnum} \cdot \log_2(PQ)$ bits. Following [8,5], we omit the a-parts of the keys, since they can be generated from public seeds using an extendable-output function (XOF). The performance results are summarized in Table 6. In rows labeled [3,36], the CP columns refer to [3] and the CC columns refer to [36].

For batch CPMM, our method achieves latencies between 0.027 s and 4.73 s for matrices ranging from 16×16 (batch size 512) to 512×512 (batch size 16), corresponding to speedups of 11.9× to 4522× over [3]. For batch CCMM, the latency ranges from 0.13 s to 9.98 s for matrices from 16×16 (batch size 256) to 512×512 (batch size 8), yielding speedups of 19.7× to 48.9× over [23].

For precision, batch CPMM loses approximately $\log \sqrt{k/2}$ bits compared with [3], due to the transformation by DFT_k in batch matrix encoding. For batch CCMM, our algorithm achieves precision comparable to [36] and higher than [23]. Since the noise after batch CCMM is dominated by key-switching errors incurred during CMT, this behavior is consistent with the CMT precision results in Table 3.

We present the experimental results of our rectangular CPMM and CCMM for a $d \times N/2$ matrix and an $N/2 \times N/2$ matrix and compare them with the methods of [3,36]. We use the S12 parameter with $N = 2^{12}$ and report the performance

d	Batch		Method	Time (s)		Acc. (bit)		Key size (MB)	
	CP	CC		CP	CC	CP	CC	CP	CC
16	512	256	[23]	3.40	2.56	16.0	15.3	5.24	5.24
			[3,36]	123	1865	24.8	18.4	336	5369 (1.33)
			Ours	0.027	0.13 (0.21)	20.6	17.8 (18.6)	0	10.5 (0.79)
32	256	128	[23]	14.4	9.92	15.7	15.3	7.86	7.86
			[3,36]	71.1	1865	24.9	18.4	168	5369 (1.33)
			Ours	0.074	0.31 (0.45)	21.0	18.2 (18.5)	0	21.0 (0.79)
64	128	64	[23]	56.6	26.4	14.9	14.5	10.5	10.5
			[3,36]	48.1	1865	25.2	18.4	84	5369 (1.33)
			Ours	0.20	0.54 (0.81)	21.5	18.0 (18.7)	0	41.9 (0.79)
128	64	32	[3,36]	36.7	1865	25.0	18.4	42	5369 (1.33)
			Ours	0.41	1.20 (1.77)	22.2	18.2 (18.7)	0	83.9 (0.79)
256	32	16	[3,36]	41.2	1865	25.2	18.4	21	5369 (1.33)
			Ours	1.26	3.20 (4.20)	22.7	18.2 (18.7)	0	168 (0.79)
512	16	8	[3,36]	56.2	1865	25.2	18.4	10	5369 (1.33)
			Ours	4.73	9.98 (13.8)	23.3	18.1 (18.8)	0	336 (0.79)

Table 6. CPMM/CCMM performance on S13b parameter. Here, the batch size k is N/d for batch CPMM, whereas $k = N/(2d)$ for batch CCMM. Timings are averaged over 20 runs and reported in seconds; accuracy is reported in bits; key size is reported in MB. Timings, accuracy and key size for the lightweight version are listed in parentheses.

in Table 7. We note that rectangular CPMM of [3] uses an MLWE-based approach of rank k and degree d for $N/2$ ciphertexts, including format conversion overhead and the corresponding key sizes in their performance evaluation; and for CCMM, we use naïve embedding to square matrices.

Dim.	Method	Time (s)		Accuracy (bit)	
		CPMM	CCMM	CPMM	CCMM
16×2048	[3,36]	19.4	216	25.4	18.3
	Ours	1.67	5.82	20.9	19.0
32×2048	[3,36]	18.7	215	25.0	18.0
	Ours	2.00	6.41	21.6	19.0
64×2048	[3,36]	18.7	216	25.3	18.1
	Ours	2.16	7.45	22.1	19.0
128×2048	[3,36]	18.7	216	25.4	18.2
	Ours	2.74	8.97	22.6	19.0
256×2048	[3,36]	19.2	216	25.3	18.0
	Ours	4.14	12.3	22.9	19.0

Table 7. Rectangular CPMM/CCMM performance on S12 parameter. Timings are averaged over 10 runs and reported in seconds. Accuracy is reported in bits.

For rectangular CPMM, our method achieves latencies between 1.67 s and 4.14 s for matrices from 16×2048 to 256×2048, yielding a 4.6× to 11.6× speedup over [3]. For rectangular CCMM, the latency ranges from 5.82 s to

12.3 s over the same matrix sizes, giving a 17.6× to 37.1× speedup over [36]. The precision follows a trend similar to that of encrypted batch MM. We note that the observed speedups for rectangular CPMM and CCMM are smaller than anticipated, primarily because CMT accounts for a larger fraction of the total runtime than in the previous experiments.

Acknowledgments. This work was conducted while Minsik Kang was with Seoul National University. This work was financially supported by the Institute of Information & Communications Technology Planning & Evaluation (IITP) grant funded by the Korea government (MSIT) (RS-2025-02217656).

References

1. OpenBLAS: An optimized BLAS library (2025), version 0.3.30, available at `https://www.openblas.net/`
2. Albrecht, M.R., Player, R., Scott, S.: On the concrete hardness of learning with errors. Journal of Mathematical Cryptology **9**(3), 169–203 (2015)
3. Bae, Y., Cheon, J.H., Hanrot, G., Park, J.H., Stehlé, D.: Plaintext-ciphertext matrix multiplication and fhe bootstrapping: fast and fused. In: Annual International Cryptology Conference. pp. 387–421. Springer (2024)
4. Bae, Y., Cheon, J.H., Hanrot, G., Park, J.H., Stehlé, D.: Fast homomorphic linear algebra with blas. arXiv preprint arXiv:2503.16080 (2025)
5. Bae, Y., Cheon, J.H., Kim, J., Park, J.H., Stehlé, D.: Hermes: efficient ring packing using mlwe ciphertexts and application to transciphering. In: Annual International Cryptology Conference. pp. 37–69. Springer (2023)
6. Cha, H., Hwang, I., Min, S., Seo, J., Song, Y.: Matrigear: Accelerating authenticated matrix triple generation with scalable prime fields via optimized he packing. In: 2025 IEEE Symposium on Security and Privacy (SP). pp. 2453–2471. IEEE (2025)
7. Chen, H., Chillotti, I., Song, Y.: Improved bootstrapping for approximate homomorphic encryption. In: Annual International Conference on the Theory and Applications of Cryptographic Techniques. pp. 34–54. Springer (2019)
8. Chen, H., Dai, W., Kim, M., Song, Y.: Efficient homomorphic conversion between (ring) lwe ciphertexts. In: International conference on applied cryptography and network security. pp. 460–479. Springer (2021)
9. Chen, J., Yang, L., Wu, W., Liu, Y., Feng, Y.: Homomorphic matrix operations under bicyclic encoding. IEEE Transactions on Information Forensics and Security (2024)
10. Cheon, J.H., Kim, A., Kim, M., Song, Y.: Homomorphic encryption for arithmetic of approximate numbers. In: International conference on the theory and application of cryptology and information security. pp. 409–437. Springer (2017)
11. Cheon, J.H., Kim, A., Yhee, D.: Multi-dimensional packing for heaan for approximate matrix arithmetics. Cryptology ePrint Archive (2018)
12. CryptoLab: HEaaN library (2022), Is available at `https://heaan.it/`
13. Froelicher, D., Cho, H., Edupalli, M., Sousa, J.S., Bossuat, J.P., Pyrgelis, A., Troncoso-Pastoriza, J.R., Berger, B., Hubaux, J.P.: Scalable and privacy-preserving federated principal component analysis. In: 2023 IEEE Symposium on Security and Privacy (SP). pp. 1908–1925. IEEE (2023)

14. Gao, J., Gao, Y.: Gms: an efficient fully homomorphic encryption scheme for secure outsourced matrix multiplication. Journal of Supercomputing **80**(18) (2024)
15. Gao, Y., Quan, G., Homsi, S., Wen, W., Wang, L.: Secure and efficient general matrix multiplication on cloud using homomorphic encryption. The Journal of Supercomputing **80**(18) (2024)
16. Gentry, C., Lee, Y.: Fully homomorphic encryption for matrix arithmetic. IACR eprint (2025), available at `http://eprint.iacr.org/2012/144`
17. Halevi, S., Shoup, V.: Design and implementation of HElib: a homomorphic encryption library. IACR eprint (2020), available at `https://eprint.iacr.org/2020/1481`
18. Han, K., Hhan, M., Cheon, J.H.: Improved homomorphic discrete fourier transforms and fhe bootstrapping. IEEE Access **7**, 57361–57370 (2019)
19. Hao, M., Li, H., Chen, H., Xing, P., Xu, G., Zhang, T.: Iron: Private inference on transformers. Advances in neural information processing systems **35**, 15718–15731 (2022)
20. Henzinger, A., Hong, M.M., Corrigan-Gibbs, H., Meiklejohn, S., Vaikuntanathan, V.: One server for the price of two: Simple and fast {Single-Server} private information retrieval. In: 32nd USENIX Security Symposium (USENIX Security 23). pp. 3889–3905 (2023)
21. Huang, Z., Hong, C., Weng, C., Lu, W.j., Qu, H.: More efficient secure matrix multiplication for unbalanced recommender systems. IEEE Transactions on Dependable and Secure Computing **20**(1), 551–562 (2021)
22. Jang, J., Lee, Y., Kim, A., Na, B., Yhee, D., Lee, B., Cheon, J.H., Yoon, S.: Privacy-preserving deep sequential model with matrix homomorphic encryption. In: Proceedings of the 2022 ACM on Asia Conference on Computer and Communications Security. pp. 377–391 (2022)
23. Jiang, X., Kim, M., Lauter, K., Song, Y.: Secure outsourced matrix computation and application to neural networks. In: Proceedings of the 2018 ACM SIGSAC conference on computer and communications security. pp. 1209–1222 (2018)
24. Ju, J.H., Park, J., Kim, J., Kang, M., Kim, D., Cheon, J.H., Ahn, J.H.: Neujeans: Private neural network inference with joint optimization of convolution and fhe bootstrapping. In: Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security. pp. 4361–4375 (2024)
25. Kim, A., Papadimitriou, A., Polyakov, Y.: Approximate homomorphic encryption with reduced approximation error. In: Cryptographers' Track at the RSA Conference. pp. 120–144. Springer (2022)
26. Langlois, A., Stehlé, D.: Worst-case to average-case reductions for module lattices. Designs, Codes and Cryptography **75**(3), 565–599 (2015)
27. Lee, J.W., Lee, E., Kim, Y.S., No, J.S.: Rotation key reduction for client-server systems of deep neural network on fully homomorphic encryption. In: International Conference on the Theory and Application of Cryptology and Information Security. pp. 36–68. Springer (2023)
28. Lewis, P., Perez, E., Piktus, A., Petroni, F., Karpukhin, V., Goyal, N., Küttler, H., Lewis, M., Yih, W.t., Rocktäschel, T., et al.: Retrieval-augmented generation for knowledge-intensive nlp tasks. Advances in neural information processing systems **33**, 9459–9474 (2020)
29. Li, B., Micciancio, D., Raykova, M., Schultz-Wu, M.: Hintless single-server private information retrieval. In: Annual International Cryptology Conference. pp. 183–217. Springer (2024)
30. Lim, L., Kalagi, V., Agrawal, D., El Abbadi, A.: Tricycle: Private transformer inference with tricyclic encodings. Cryptology ePrint Archive (2025)

31. Liu, J., Zhang, L.F.: Privacy-preserving and publicly verifiable matrix multiplication. IEEE Transactions on Services Computing **16**(3), 2059–2071 (2022)
32. Ma, X., Ma, C., Jiang, Y., Ge, C.: Improved privacy-preserving pca using optimized homomorphic matrix multiplication. Computers & Security **138**, 103658 (2024)
33. Menon, S.J., Wu, D.J.: Spiral: Fast, high-rate single-server pir via fhe composition. In: 2022 IEEE symposium on security and privacy (SP). pp. 930–947. IEEE (2022)
34. Menon, S.J., Wu, D.J.: YPIR: High-throughput single-server pir with silent preprocessing. In: 33rd USENIX Security Symposium (USENIX Security 24). pp. 5985–6002 (2024)
35. Pang, Q., Zhu, J., Möllering, H., Zheng, W., Schneider, T.: Bolt: Privacy-preserving, accurate and efficient inference for transformers. In: 2024 IEEE Symposium on Security and Privacy (SP). pp. 4753–4771. IEEE (2024)
36. Park, J.H.: Ciphertext-ciphertext matrix multiplication: Fast for large matrices. In: Annual International Conference on the Theory and Applications of Cryptographic Techniques. pp. 153–180. Springer (2025)
37. Rader, C.M.: Discrete fourier transforms when the number of data samples is prime. Proceedings of the IEEE **56**(6), 1107–1108 (1968)
38. Radford, A., Narasimhan, K., Salimans, T., Sutskever, I., et al.: Improving language understanding by generative pre-training (2018)
39. Rizomiliotis, P., Triakosia, A.: On matrix multiplication with homomorphic encryption. In: Proceedings of the 2022 on Cloud Computing Security Workshop. pp. 53–61 (2022)
40. The FLINT team: FLINT: Fast Library for Number Theory (2025), version 3.3.1, `https://flintlib.org`
41. Touvron, H., Lavril, T., Izacard, G., Martinet, X., Lachaux, M.A., Lacroix, T., Rozière, B., Goyal, N., Hambro, E., Azhar, F., et al.: Llama: Open and efficient foundation language models. arXiv preprint arXiv:2302.13971 (2023)
42. Touvron, H., Martin, L., Stone, K., Albert, P., Almahairi, A., Babaei, Y., Bashlykov, N., Batra, S., Bhargava, P., Bhosale, S., et al.: Llama 2: Open foundation and fine-tuned chat models. arXiv preprint arXiv:2307.09288 (2023)
43. Zhang, J., Yang, X., He, L., Chen, K., Lu, W.j., Wang, Y., Hou, X., Liu, J., Ren, K., Yang, X.: Secure transformer inference made non-interactive. Cryptology ePrint Archive (2024)
44. Zheng, X., Li, H., Wang, D.: A new framework for fast homomorphic matrix multiplication. Cryptology ePrint Archive (2023)
45. Zhu, L., Hua, Q.s., Chen, Y., Jin, H.: Secure outsourced matrix multiplication with fully homomorphic encryption. In: European Symposium on Research in Computer Security. pp. 249–269. Springer (2023)

A Noise Analysis

We follow the heuristic approach in [10,25]. For a polynomial $a \in \mathcal{R}_k$, let $\omega_{2k} = \exp(\pi i/k)$, and suppose that each coefficient of a has variance σ^2. Then the random variable $a(\omega_{2k})$ follows a complex normal distribution with mean 0 and variance $k\sigma^2$. The canonical embedding norm is defined as

$$\|a\|^{\mathsf{can}} = \max_{j \in [k/2]} |a(\omega_{2k}^{5^j})|,$$

and for a matrix $\boldsymbol{A} = (A_{ij}) \in \mathcal{R}_k^{d \times d'}$, we take the maximum over its entries:

$$\|\boldsymbol{A}\|^{\mathsf{can}} = \max_{i,j} \|A_{ij}\|^{\mathsf{can}}.$$

When all coefficients of A_{ij} have the same variance σ^2, we use the heuristic high-probability bound

$$\|\boldsymbol{A}\|^{\mathsf{can}} \leq 8\sigma\sqrt{k}.$$

We note that $\|\cdot\|^{\mathsf{can}}$ takes the maximum over both matrix entries and local slots. Therefore, we use the more conservative constant 8 instead of the constant 6 commonly used for the canonical embedding norm in the CKKS scheme [10]. We use the standard CKKS noise estimates for fresh encryption, rescaling, and key-switching, and denote the variance of each coefficient after these operations by σ_{enc}^2, σ_{rs}^2, and σ_{ks}^2, respectively; for details, we refer to [25].

Lemma 4 (Error for matrix encryption). *Suppose that the error polynomials e_j are obtained from the same CKKS operation for all $j \in [d']$, and that each coefficient of e_j has variance σ^2. Let*

$$\boldsymbol{E} = \left(\mathtt{Vec}_k^d(e_0), \ldots, \mathtt{Vec}_k^d(e_{d'-1})\right) \in \mathcal{R}_k^{d \times d'}$$

be the error matrix of the resulting matrix encryption. Then $\boldsymbol{E}$ satisfies the heuristic high-probability bound

$$\|\boldsymbol{E}\|^{\mathsf{can}} \leq 8\,\sigma\sqrt{k}.$$

Proof. By construction, the j-th column of $\boldsymbol{E}$ is $\mathtt{Vec}_k^d(e_j)$. Equivalently,

$$e_j = \sum_{i \in [d]} \boldsymbol{E}_{i,j}(Y) X^i,$$

so each entry $\boldsymbol{E}_{i,j} \in \mathcal{R}_k$ consists of k coefficients of e_j, each having variance σ^2. Applying the heuristic canonical-embedding bound above to each entry gives the claimed bound on $\|\boldsymbol{E}\|^{\mathsf{can}}$. □

We note that encryption and rescaling in the matrix encryption are obtained by applying the corresponding CKKS operations in a batch manner. Therefore, the same noise estimates apply. If $\boldsymbol{E}_{\mathrm{enc}}$ and $\boldsymbol{E}_{\mathrm{rs}}$ denote the error matrices after matrix encryption and rescaling, respectively, then Lemma 4 gives

$$\|\boldsymbol{E}_{\mathrm{enc}}\|^{\mathsf{can}} \leq 8\,\sigma_{\mathrm{enc}}\sqrt{k},$$
$$\|\boldsymbol{E}_{\mathrm{rs}}\|^{\mathsf{can}} \leq 8\,\sigma_{\mathrm{rs}}\sqrt{k}.$$

Lemma 5 (Error for multiplication by a plaintext matrix). *Let $\{\boldsymbol{U}_\ell\}_{\ell \in [k/2]} \subset \mathbb{C}^{d' \times d''}$ be matrices satisfying $\max_\ell \|\boldsymbol{U}_\ell\|_\infty \leq \nu$. Let $\boldsymbol{E} \in \mathcal{R}_k^{d \times d'}$ be a matrix whose entries have coefficient variance σ^2, and let $\boldsymbol{U} = \mathsf{MatEcd}_k^{d' \times d''}(\{\boldsymbol{U}_\ell\}_{\ell \in [k/2]})$. Then we have the heuristic high-probability bound*

$$\left\| \frac{\boldsymbol{E} \cdot \boldsymbol{U}}{\Delta} \right\|^{\mathsf{can}} \leq 8\sigma\nu\sqrt{d'k}.$$

Proof. For each (i, j) and each local slot $\ell \in [k/2]$, we have

$$(\boldsymbol{EU})_{i,j}\big(\omega_{2k}^{5^\ell}\big)/\Delta = \sum_{t\in[d']} \boldsymbol{E}_{i,t}\big(\omega_{2k}^{5^\ell}\big)\,(\boldsymbol{U}_\ell)_{t,j},$$

omitting the rounding error. The right-hand side is a sum of d' independent random variables, where each $\boldsymbol{E}_{i,t}(\omega_{2k}^{5^\ell})$ has variance $k\sigma^2$, and each coefficient $(\boldsymbol{U}_\ell)_{t,j}$ is bounded in absolute value by ν. Thus the variance of the sum is at most $d'\nu^2 k\sigma^2$. For sufficiently large d', we apply the Central Limit Theorem to obtain the desired bound. □

A.1 Batch CPMM in Theorem 2

We use the output error expression from Theorem 2 and bound it in the canonical embedding norm.

Theorem 4 (Error for Batch CPMM). *In the setting of Theorem 2, suppose that the input error $\boldsymbol{E}$ is a fresh encryption error and $\max_\ell \|\boldsymbol{U}_\ell\|_\infty \le \nu$. Then the output error satisfies the high-probability bound*

$$\|\boldsymbol{E}_{\mathrm{out}}\|^{\mathsf{can}} \le 8\,\sigma_{\mathrm{enc}}\nu\sqrt{d'k} + 8\,\sigma_{\mathrm{rs}}\sqrt{k}.$$

Proof. By Theorem 2, the output error is

$$\boldsymbol{E}_{\mathrm{out}} = \frac{1}{\Delta}\boldsymbol{EU} + \boldsymbol{E}_{\mathrm{rs}}.$$

Lemma 5 gives

$$\|\boldsymbol{EU}/\Delta\|^{\mathsf{can}} \le 8\,\sigma_{\mathrm{enc}}\nu\sqrt{d'k}.$$

Since each coefficient of the rescaling error has variance σ_{rs}^2, Lemma 4 gives

$$\|\boldsymbol{E}_{\mathrm{rs}}\|^{\mathsf{can}} \le 8\,\sigma_{\mathrm{rs}}\sqrt{k}.$$

The claim follows by the triangle inequality.

A.2 Batch CMT

We bound the error of CMT_k^d in the canonical embedding norm.

Theorem 5 (Error for Batch CMT). *Let $(\boldsymbol{B}, \boldsymbol{A})$ be a matrix encryption satisfying*

$$\boldsymbol{B} + \mathsf{Toep}_k^d(\mathsf{sk})\cdot\boldsymbol{A} = \mathsf{MatEcd}_k^d(\{\boldsymbol{M}_\ell\}_{\ell\in[k/2]}) + \boldsymbol{E} \in \mathcal{R}_{q,k}^{d\times d}$$

whose error $\boldsymbol{E}$ is a fresh encryption error. Then Algorithm 3 computes a matrix encryption of the transpose,

$$\boldsymbol{B}' + \mathsf{Toep}_k^d(\mathsf{sk})\cdot\boldsymbol{A}' = \mathsf{MatEcd}_k^d(\{\boldsymbol{M}_\ell^T\}_{\ell\in[k/2]}) + \boldsymbol{E}_{\mathrm{cmt}} \in \mathcal{R}_{q,k}^{d\times d},$$

whose error $\boldsymbol{E}_{\mathrm{cmt}}$ has coefficient variance $\sigma_{\mathrm{enc}}^2 + d\,\sigma_{\mathrm{ks}}^2$, and therefore satisfies the high-probability bound

$$\|\boldsymbol{E}_{\mathrm{cmt}}\|^{\mathsf{can}} \le 8\sqrt{(\sigma_{\mathrm{enc}}^2 + d\,\sigma_{\mathrm{ks}}^2)\,k}.$$

Proof. The output error splits as $\boldsymbol{E}_{\mathrm{cmt}} = \boldsymbol{E}^T + \boldsymbol{E}_{\mathrm{ks}}$ into independent contributions from the input error and the key switchings. By the correctness of CMT_k^d applied to $\boldsymbol{M} + \boldsymbol{E}$, the output without key-switching errors equals $\boldsymbol{M}^T + \boldsymbol{E}^T$. Note that the coefficients of $\boldsymbol{E}^T$ retain the input variance σ_{enc}^2.

Meanwhile, the key switchings occur in the $\mathsf{ScrambleAuto}$ step, where each of the d automorphisms $\mathsf{Auto}(\mathsf{ct}_{j^*};\, 2kj+1)$ is realized by a key switching and contributes an independent error e_j^{ks} of coefficient variance σ_{ks}^2. The subsequent $\mathsf{invAdjust}$ produces in the i-th output ciphertext the key-switching error

$$\varepsilon_i = \sum_{j\in[d]} X^{-i(2kj+1)}\, e_j^{\mathrm{ks}},$$

whose coefficients form the i-th column of $\boldsymbol{E}_{\mathrm{ks}}$ through

$$\varepsilon_i = \sum_{j\in[d]} (\boldsymbol{E}_{\mathrm{ks}})_{j,i} \cdot X^j.$$

Since multiplication by a monomial only permutes and negates coefficients, each summand $X^{-i(2kj+1)}\, e_j^{\mathrm{ks}}$ retains coefficient variance σ_{ks}^2, so their sum ε_i over the d independent e_j^{ks} has coefficient variance $d\,\sigma_{\mathrm{ks}}^2$, as does every entry $(\boldsymbol{E}_{\mathrm{ks}})_{j,i}$. Therefore each coefficient of $\boldsymbol{E}_{\mathrm{cmt}}$ has variance $\sigma_{\mathrm{enc}}^2 + d\,\sigma_{\mathrm{ks}}^2$, and Lemma 4 gives the claimed bound.

A.3 Batch CCMM

We bound the error of batch CCMM in the canonical embedding norm, with the input errors $\boldsymbol{E}, \boldsymbol{E}'$ of coefficient variance σ_{enc}^2 and σ_{ks}^2, σ_{rs}^2 as before.

Theorem 6 (Error for Batch CCMM). *Let $(\boldsymbol{B}, \boldsymbol{A})$ and $(\boldsymbol{B}', \boldsymbol{A}')$ be matrix encryptions satisfying*

$$\boldsymbol{B} + \mathsf{Toep}_k^d(\mathsf{sk}) \cdot \boldsymbol{A} = \mathsf{MatEcd}_k^d(\{\boldsymbol{M}_\ell\}_{\ell\in[k/2]}) + \boldsymbol{E} \in \mathcal{R}_{q,k}^{d\times d},$$
$$\boldsymbol{B}' + \mathsf{Toep}_k^d(\mathsf{sk}) \cdot \boldsymbol{A}' = \mathsf{MatEcd}_k^d(\{\boldsymbol{M}'_\ell\}_{\ell\in[k/2]}) + \boldsymbol{E}' \in \mathcal{R}_{q,k}^{d\times d}$$

whose error coefficients have variance σ_{enc}^2, and suppose $\max_\ell \|\boldsymbol{M}_\ell\|_\infty \le \nu$ and $\max_\ell \|\boldsymbol{M}'_\ell\|_\infty \le \nu'$. Then Algorithm 4 computes a matrix encryption of the product,

$$\boldsymbol{B}_{\mathrm{res}} + \mathsf{Toep}_k^d(\mathsf{sk}) \cdot \boldsymbol{A}_{\mathrm{res}} = \mathsf{MatEcd}_k^d(\{\boldsymbol{M}_\ell \cdot \boldsymbol{M}'_\ell\}_{\ell\in[k/2]}) + \boldsymbol{E}_{\mathrm{out}} \in \mathcal{R}_{q/\Delta,k}^{d\times d},$$

whose error $\boldsymbol{E}_{\mathrm{out}}$ satisfies the high-probability bound

$$\|\boldsymbol{E}_{\mathrm{out}}\|^{\mathsf{can}} \le 8\nu\sqrt{dk(\sigma_{\mathrm{enc}}^2 + d\,\sigma_{\mathrm{ks}}^2)} + 8\,\sigma_{\mathrm{enc}}\nu'\sqrt{dk} + 8\,\sigma_{\mathrm{rs}}\sqrt{k}.$$

Proof. Write $\boldsymbol{M} = \mathsf{MatEcd}_k^d(\{\boldsymbol{M}_\ell\}_{\ell\in[k/2]})$ and $\boldsymbol{M}' = \mathsf{MatEcd}_k^d(\{\boldsymbol{M}'_\ell\}_{\ell\in[k/2]})$ for the encoded messages.

By Theorem 5, applying CMT_k^d to the second input $(\boldsymbol{B}', \boldsymbol{A}')$ produces a row-wise matrix encryption

$$\boldsymbol{B}' + \boldsymbol{A}' \cdot \mathtt{Toep}_k^d(\mathsf{sk})^T = \boldsymbol{M}' + \boldsymbol{E}'_{\mathrm{cmt}} \in \mathcal{R}_{q,k}^{d\times d},$$

whose error $\boldsymbol{E}'_{\mathrm{cmt}}$ has coefficient variance $\sigma_{\mathrm{enc}}^2 + d\,\sigma_{\mathrm{ks}}^2$.

The multiplication of this row-wise encryption with the first input computes an encryption of

$$(\boldsymbol{M} + \boldsymbol{E})(\boldsymbol{M}' + \boldsymbol{E}'_{\mathrm{cmt}}) = \boldsymbol{M}\boldsymbol{M}' + \boldsymbol{M}\boldsymbol{E}'_{\mathrm{cmt}} + \boldsymbol{E}\boldsymbol{M}' + \boldsymbol{E}\boldsymbol{E}'_{\mathrm{cmt}},$$

where, by the homomorphic property of the encoding,

$$\boldsymbol{M}\boldsymbol{M}' = \Delta \cdot \mathsf{MatEcd}_k^d(\{\boldsymbol{M}_\ell \cdot \boldsymbol{M}'_\ell\}_{\ell\in[k/2]}).$$

The algorithm then applies two further CMT_k^d calls and one relinearization, and rescales by Δ. The leading term $\boldsymbol{M}\boldsymbol{M}'/\Delta$ becomes the output message $\mathsf{MatEcd}_k^d(\{\boldsymbol{M}_\ell \cdot \boldsymbol{M}'_\ell\}_{\ell\in[k/2]})$, and the output error is

$$\boldsymbol{E}_{\mathrm{out}} = \frac{1}{\Delta}\,\boldsymbol{M}\boldsymbol{E}'_{\mathrm{cmt}} + \frac{1}{\Delta}\,\boldsymbol{E}\boldsymbol{M}' + \frac{1}{\Delta}\,\boldsymbol{E}\boldsymbol{E}'_{\mathrm{cmt}} + \frac{1}{\Delta}\,\boldsymbol{E}'' + \boldsymbol{E}_{\mathrm{rs}},$$

where $\boldsymbol{E}''$ collects the key-switching errors of the latter two CMT_k^d calls and the relinearization, and $\boldsymbol{E}_{\mathrm{rs}}$ is the rescaling error.

We bound the dominant terms of $\boldsymbol{E}_{\mathrm{out}}$ in the canonical embedding norm, beginning with the two message-by-error terms. For $\boldsymbol{M}\boldsymbol{E}'_{\mathrm{cmt}}/\Delta$, the same argument as in Lemma 5 gives

$$\|\boldsymbol{M}\boldsymbol{E}'_{\mathrm{cmt}}/\Delta\|^{\mathsf{can}} \le 8\nu\sqrt{dk(\sigma_{\mathrm{enc}}^2 + d\,\sigma_{\mathrm{ks}}^2)}.$$

Applying Lemma 5 to $\boldsymbol{E}\boldsymbol{M}'/\Delta$ gives

$$\|\boldsymbol{E}\boldsymbol{M}'/\Delta\|^{\mathsf{can}} \le 8\,\sigma_{\mathrm{enc}}\nu'\sqrt{dk}.$$

The second-order term $\boldsymbol{E}\boldsymbol{E}'_{\mathrm{cmt}}/\Delta$ and the additional key-switching term $\boldsymbol{E}''/\Delta$ are suppressed by the rescaling factor Δ so that they become negligible compared with the dominant terms. Finally, Lemma 4 gives

$$\|\boldsymbol{E}_{\mathrm{rs}}\|^{\mathsf{can}} \le 8\,\sigma_{\mathrm{rs}}\sqrt{k}.$$

Combining the three dominant terms by subadditivity yields

$$\|\boldsymbol{E}_{\mathrm{out}}\|^{\mathsf{can}} \le 8\nu\sqrt{dk(\sigma_{\mathrm{enc}}^2 + d\,\sigma_{\mathrm{ks}}^2)} + 8\,\sigma_{\mathrm{enc}}\nu'\sqrt{dk} + 8\,\sigma_{\mathrm{rs}}\sqrt{k}.$$

□

A.4 Generated Galois keys for lightweight CMT

For an automorphism τ, an original automorphism key $(\beta, \alpha) \in \mathcal{R}_{pq}^2$ has the form

$$\beta + \alpha \cdot \mathsf{sk} = p \cdot \tau(\mathsf{sk}) + \varepsilon \pmod{pq},$$

where ε is sampled from the discrete Gaussian distribution. The Galois key for the automorphism $\tau' \circ \tau$ can be generated by key switching:

$$\beta' + \alpha' \cdot \mathsf{sk} = p \cdot (\tau' \circ \tau)(\mathsf{sk}) + \tau'(\varepsilon) + e_{\mathrm{ks}} \pmod{pq}.$$

When the resulting key is used to apply an automorphism to a ciphertext with error e, the output error is

$$\tau(e) + \tau(a) \cdot e_{\mathrm{key}}/p + e_{\mathrm{round}},$$

where e_{round} is the rounding error from modulus down-conversion and $e_{\mathrm{key}} = \tau'(\varepsilon) + e_{\mathrm{ks}}$ for a generated Galois key. Choosing the auxiliary modulus p sufficiently large makes $\tau(a) \cdot e_{\mathrm{key}}/p$ comparable to the ordinary automorphism key error, so generated Galois keys do not introduce a meaningful additional precision loss in CMT_k^d or batch CCMM. We refer to [27] for a more detailed analysis of generated Galois keys in the context of CKKS.

A.5 Encoding conversion

The encoding conversion used in the rectangular algorithms composes a CMT_k^d, a batch CPMM by the plaintext conversion matrix $\boldsymbol{M'} = \boldsymbol{T}_{N/2 \leftarrow k/2} \boldsymbol{P}_d$, and a second CMT_k^d,

$$\mathsf{CMT}_k^d \;\rightarrow\; \text{batch CPMM} \;\rightarrow\; \mathsf{CMT}_k^d.$$

We bound the error of this composition in the canonical embedding norm, with the input error $\boldsymbol{E}$ of coefficient variance σ_{enc}^2 and σ_{ks}^2, σ_{rs}^2 as before.

Theorem 7 (Encoding conversion error). *Let $(\boldsymbol{B}, \boldsymbol{A})$ be a matrix encryption satisfying*

$$\boldsymbol{B} + \mathsf{Toep}_k^d(\mathsf{sk}) \cdot \boldsymbol{A} = \mathsf{MatEcd}_k^d(\{\boldsymbol{M}_\ell\}_{\ell \in [k/2]}) + \boldsymbol{E} \in \mathcal{R}_{q,k}^{d \times d}$$

whose error coefficients have variance σ_{enc}^2, and let $\boldsymbol{M'} = \boldsymbol{T}_{N/2 \leftarrow k/2} \boldsymbol{P}_d$ be the plaintext conversion matrix. Then the composition above computes a matrix encryption of the converted message,

$$\boldsymbol{B}_3 + \mathsf{Toep}_k^d(\mathsf{sk}) \cdot \boldsymbol{A}_3 = \mathsf{MatEcd}_k^d(\{\boldsymbol{M}'_\ell \, \boldsymbol{M}_\ell\}_{\ell \in [k/2]}) + \boldsymbol{E}_{\mathrm{out}} \in \mathcal{R}_{q/\Delta,k}^{d \times d},$$

whose error $\boldsymbol{E}_{\mathrm{out}}$ satisfies the high-probability bound

$$\|\boldsymbol{E}_{\mathrm{out}}\|^{\mathsf{can}} \leq 8\sqrt{dk(\sigma_{\mathrm{enc}}^2 + d\,\sigma_{\mathrm{ks}}^2)} + 8\,\sigma_{\mathrm{ks}}\sqrt{dk} + 8\,\sigma_{\mathrm{rs}}\sqrt{k}.$$

Proof. Write

$$\mathsf{MatEcd}_k^d(\{\boldsymbol{M}_\ell\}_{\ell\in[k/2]}) = \Delta\boldsymbol{M}.$$

The first CMT_k^d gives

$$\boldsymbol{B}_1 + \boldsymbol{A}_1 \cdot \mathtt{Toep}_k^d(\mathsf{sk})^T = \Delta\boldsymbol{M} + \boldsymbol{E}_{\mathrm{cmt},1},$$

where, by Theorem 5, the coefficients of $\boldsymbol{E}_{\mathrm{cmt},1}$ have variance $\sigma_{\mathrm{enc}}^2 + d\sigma_{\mathrm{ks}}^2$. The local matrices represented by $\boldsymbol{M'} = \boldsymbol{T}_{N/2\leftarrow k/2}\boldsymbol{P}_d$ have entries bounded by 1 in absolute value. Hence, by the same argument as Lemma 5, batch CPMM by $\boldsymbol{M'}$ gives

$$\boldsymbol{B}_2 + \boldsymbol{A}_2 \cdot \mathtt{Toep}_k^d(\mathsf{sk})^T = \Delta(\boldsymbol{M'M}) + \frac{\boldsymbol{M'E}_{\mathrm{cmt},1}}{\Delta} + \boldsymbol{E}_{\mathrm{rs}} \in \mathcal{R}_{q/\Delta,k}^{d\times d},$$

with

$$\|\boldsymbol{M'E}_{\mathrm{cmt},1}/\Delta\|^{\mathsf{can}} \le 8\sqrt{dk(\sigma_{\mathrm{enc}}^2 + d\sigma_{\mathrm{ks}}^2)}.$$

Finally, the second CMT_k^d converts the result back to a column-wise matrix encryption. This transposes the accumulated error and adds d independent key-switching errors, whose contribution is bounded by $8\,\sigma_{\mathrm{ks}}\sqrt{dk}$ by Lemma 4. The rescaling error satisfies $\|\boldsymbol{E}_{\mathrm{rs}}\|^{\mathsf{can}} \le 8\,\sigma_{\mathrm{rs}}\sqrt{k}$. Combining these terms by subadditivity gives the claimed bound. □