Low-Latency Linear Transformations with Small Key Transmission for Private Neural Network on Homomorphic Encryption*

Byeongseo Min[a], Joon-Woo Lee[b,*]

[a]Department of Electrical Engineering, Pohang University of Science and Technology (POSTECH), 77 Cheongam-ro, Nam-gu, Pohang, 37673, Gyeongsangbuk-do, South Korea

[b]Department of Computer Science and Engineering, Chung-Ang University, 84 Heukseok-ro, Dongjak-gu, Seoul, 06974, South Korea

ARTICLE INFO

Keywords:
Privacy-Preserving Machine Learning
Fully Homomorphic Encryption
Cheon-Kim-Kim-Song (CKKS) schemes
Convolution

ABSTRACT

As Machine Learning as a Service expands, protecting client data has become a critical challenge. Homomorphic Encryption (HE) enables computation on encrypted data, but practical HE-based AI still suffers from high latency and large key sizes. Among neural network operations, convolution remains a central target for optimization, as it is fundamental to CNNs and increasingly used in transformers and state-space models. To realize HE convolution, various packing strategies—defining the data layout within a ciphertext—have been introduced, leading to state-of-the-art methods such as multiplexed parallel convolution (MPConv), which is implemented with multiplexed parallel packing. In this paper, we propose *Rotation-Optimized Multiplexed Parallel Convolution* (RO-MPConv), which enhances usability by reducing the number of rotation operations and rotation keys—major contributors to latency and key sizes in MPConv. Additionally, we introduce a *small-level key system* to further decrease key sizes and a *parallel baby-step giant-step matrix-vector multiplication*, which reduces rotations for packing strategies with multiple identical data entries, including multiplexed parallel packing. Experimental results demonstrate that RO-MPConv achieves up to an 81% reduction in convolution latency compared to MPConv. For key transmission, combining RO-MPConv with the hierarchical rotation key system and our small-level key system achieves a 29× reduction in rotation key size compared to MPConv. Furthermore, integrating RO-MPConv into existing state-of-the-art models reduces their total latency by up to 26%, and our proposed matrix-vector multiplication method reduces latency by 69%. Our code is fully available from https://github.com/byeongseomin51/RO-MPConv.git.

1. Introduction

As Machine Learning as a Service (MLaaS) becomes increasingly integral to various industries, the protection of client data has emerged as a critical challenge. This has catalyzed research into Privacy-Preserving Machine Learning (PPML), which aims to reconcile the utility of MLaaS with the fundamental right to data privacy.

A comprehensive survey by [1] identifies Homomorphic Encryption (HE), Differential Privacy (DP), and Secure Multi-Party Computation (MPC) as primary technologies for achieving PPML, each with distinct trade-offs. For instance, DP ensures privacy by adding statistical noise, which may come at the cost of model accuracy. MPC offers robust security for collaborative computation but often incurs high communication costs, posing a significant drawback in client-server scenarios. In contrast, Homomorphic Encryption presents a compelling alternative by enabling direct computation on encrypted data. This approach preserves data utility, thereby maintaining high model performance, and provides strong cryptographic privacy guarantees, establishing it as a robust foundation for secure MLaaS.

In HE-based PPML, deep neural networks are typically implemented using Fully Homomorphic Encryption (FHE), as its bootstrapping functionality is essential for handling the substantial number of sequential operations required by deep architectures. Among the various FHE schemes, the Residue Number System variant of the Cheon-Kim-Kim-Song (RNS-CKKS) scheme [2], [3] is widely adopted for its native support for real-number arithmetic, a critical requirement for neural network computations. In standard FHE-based PPML, the inference process begins with a client encrypting their data and sending it with evaluation keys to a server. The server performs FHE model inference on the encrypted data and returns an encrypted result, which only the client can decrypt. Despite its security strengths, FHE-based PPML faces practical challenges, namely the significant computational latency of FHE inference and the large size of the required keys. Consequently, enhancing the efficiency of FHE-based PPML remains a central focus of current research.

Research on improving HE-based models can be broadly classified into two main avenues: the optimization of linear and non-linear operations. Homomorphic encryption natively supports linear operations such as addition, multiplication, and rotation, allowing the results of these computations to be nearly identical to those performed on plaintext. However, a key challenge arises from the data layout within

*This research was supported by the Institute of Information & Communications Technology Planning & Evaluation (IITP) grant funded by the Korea government (MSIT) (No.RS-2024-00398360, Development of a user-friendly and efficiency-optimized real-time homomorphic statistical analysis processing platform & No.RS-2024-00399401, Development of post-quantum security infrastructure transition and comprehensive quantum security verification technology).

*Corresponding author.

minbyeongseo@postech.ac.kr (B. Min); jwlee2815@cau.ac.kr (J. Lee)

ORCID(s): 0009-0007-3944-8653 (B. Min); 0000-0002-4125-6331 (J. Lee)

a ciphertext, which stores data in a one-dimensional array of slots. The performance of multi-dimensional operations, therefore, critically depends on the packing strategy—the method used to map high-dimensional data onto this one-dimensional structure. The choice of this strategy directly dictates the efficiency of homomorphic linear operations, which is measured by the number of computationally expensive operations required, such as rotations and ciphertext-ciphertext multiplications, and the consumption of the multiplicative level.

Conversely, non-linear operations, such as activation functions, are generally handled by first approximating them with polynomials, which can then be evaluated using HE operations. For polynomial approximations, the packing strategy has a less direct impact on performance. Instead, a fundamental trade-off exists: higher-degree polynomials yield better accuracy but consume more multiplicative levels. The rapid consumption of the multiplicative level, in turn, increases the frequency of bootstrapping—the most computationally intensive operation in FHE—which is required to refresh the ciphertext's capacity for further multiplications.

In this paper, we focus on the optimization of linear operations, with a specific emphasis on the convolution operation. Convolution is a cornerstone of Convolutional Neural Networks (CNNs) and is increasingly being integrated into modern Artificial Intelligence (AI) architectures such as Transformers [4, 5, 6] and State-Space Models (SSMs) [7]. To implement HE-based convolution, various packing strategies have been proposed. Among these, multiplexed parallel packing [8] is recognized as a state-of-the-art approach that enhances ciphertext efficiency by compactly filling the empty slots that arise from strided convolutions and by arranging multiple small input data blocks contiguously. This packing strategy is the foundation for multiplexed parallel convolution (MPConv), a state-of-the-art FHE convolution algorithm introduced in [8] to implement ResNet inference on multiplexed parallel packed data (hereafter MP-CNN) for the CIFAR-10 dataset. The significance of MPConv is underscored by its adoption in subsequent research, such as in AutoFHE [9] or CryptoFace [10].

However, MPConv suffers from two significant drawbacks: a high number of required rotation operations and a correspondingly large set of rotation keys. Lee et al. [8] mitigated this issue by strategically designing the network architecture to execute MPConv operations at the lowest possible multiplicative levels, where rotations are fastest. While this design is effective for MP-CNN, it is not a general solution. As research into non-linear function approximation advances, the multiplicative level consumption can change, and modifications to the CNN architecture may make it difficult or impossible to constrain MPConv operations to these low levels.

To overcome these limitations, this paper makes the following key contributions:

1. We propose *Rotation-Optimized Multiplexed Parallel Convolution* (RO-MPConv), an algorithm that reconstructs MPConv to reduce the number of rotations and the volume of rotation keys without other performance degradation. This approach directly addresses the fundamental challenges of latency and key volume in FHE-based PPML. Our experiments show that RO-MPConv reduces latency by up to **56%** and the number of rotation keys by up to **95%** compared to the original MPConv.
2. To provide greater flexibility for designing privacy-preserving convolutional layers, we introduce a method that enables a trade-off between consuming additional multiplicative levels and reducing the number of rotation operations. By strategically consuming one or two additional levels, our RO-MPConv variants can achieve further latency reductions of up to **79%** and **81%**, respectively. Crucially, we demonstrate that simply replacing MPConv with our RO-MPConv in existing state-of-the-art models improves their **total inference latency by up to 26%**.
3. We further propose a *small-level key system* that leverages the fact that convolution operations are typically confined to a specific range of multiplicative levels. When combined with the existing hierarchical rotation key system [11], this method reduces the rotation key volume by approximately 29-fold compared to MP-CNN.

As an additional contribution, we introduce parallel Baby-Step Giant-Step (BSGS) matrix-vector multiplication to accelerate operations like fully connected layers. By requiring fewer rotations than the standard BSGS algorithm with the diagonal method in [12], our algorithm achieves a latency reduction of up to 69%. It is especially efficient in settings where a ciphertext holds multiple identical data entries sequentially, a common feature in various packing strategies, including multiplexed parallel packing.

2. Related Works

The efficiency of deep neural networks over homomorphic encryption is fundamentally tied to the chosen packing strategy, which dictates how multi-dimensional data is mapped onto one-dimensional ciphertexts. The optimal packing strategy is often problem-dependent, varying with the model architecture and its specific requirements. Among the existing approaches, multiplexed parallel packing [8] has emerged as a state-of-the-art strategy for implementing HE-based CNNs and their variants. Its ability to support generalized strided convolutions and its modular, block-like applicability make it a versatile solution. Consequently, it is widely adopted in research aiming to enhance the practicality of HE-based models.

The broad applicability of MPConv is evident in its use across various advanced research domains. For instance, AutoFHE [9] and LPFHE [13] employed MPConv while exploring mixed-degree CNNs, which utilize polynomials of varying degrees for non-linear approximations across the network instead of a single, fixed degree. CryptoFace [10] implemented a private face recognition system using a

Patch CNN architecture, employing MPConv to perform the convolution operations. Research into HE-specific hardware accelerators or compilers [14, 15, 16, 17] frequently uses MP-CNN as a primary ResNet baseline for performance evaluation, underscoring its status as a benchmark in the field. The method's versatility has also been demonstrated in hybrid-packed CNNs [18] and Graph Convolutional Networks (GCNs) [19]. Furthermore, it is orthogonal to techniques like group convolution [20] and the summed-area table [21], which suggests that these methods can be combined for even greater performance gains.

While multiplexed parallel packing is highly effective, it is important to acknowledge the diverse landscape of alternative packing and encoding schemes developed to address specific challenges in HE-based CNNs. In this context, packing is the strategy for arranging high-dimensional tensor data into a one-dimensional vector, whereas encoding is the process of transforming this vector into a plaintext polynomial that the encryption system can process. The choice of encoding directly influences the packing strategy. The most conventional approach is slot encoding, which is the method utilized by the MP-CNN baseline.

Other notable approaches are as follows: Falcon [22] employs frequency-domain packing with spectral encoding, while Gazelle [23] uses a packed Single-Input Single-Output (SISO) packing method. Some works focus primarily on novel encoding schemes, such as the optimal group encoding in CrossNet [24], coefficient encoding in [25], and the Coefficients-in-Slot (CinS) encoding in Neujeans [26]. Others, like Orion [27] and Chet [28], introduce compiler-level optimizations, such as Toeplitz matrix-based packing with double-hoisting and automatic data layout selection, respectively.

However, each of these specialized methods entails trade-offs that limit their general applicability. For instance, Falcon's frequency-domain approach requires computationally expensive Fourier transforms for non-convolutional layers. The methods in CrossNet and Gazelle are designed for interactive settings that require frequent client communication, unlike the non-interactive scenarios we consider. The techniques in Chet and Orion are often tightly coupled with their specific compilers, reducing their flexibility for custom modifications. For instance, because Orion explicitly utilizes double-hoisted rotations, it is challenging to directly employ major HE libraries that do not natively support this specific optimization (e.g., DESILO [29], SEAL [30]) as backends without non-trivial adaptation. Furthermore, coefficient-based encodings like that of [25] sacrifice SIMD efficiency, and joint optimization strategies like the convolution-bootstrapping fusion in Neujeans [26] are ineffective when convolution and bootstrapping operations are not performed sequentially, such as in mixed-degree models.

Ultimately, no single packing or encoding strategy has proven universally dominant for all HE applications. Each approach presents a unique set of advantages and trade-offs. However, because multiplexed parallel packing is model-agnostic and highly applicable to a diverse range of architectures and practical use cases, it remains one of the most robust and widely used methods. Therefore, optimizing its core algorithm, MPConv, is a critical step toward bridging the gap between theoretical HE-based models and their real-world deployment. By enhancing the efficiency of this foundational algorithm, our work contributes to making complex, privacy-preserving AI systems more practical and accessible.

3. Preliminaries

3.1. RNS-CKKS Fully Homomorphic Encryption

The residue number system variant of the Cheon-Kim-Kim-Song (RNS-CKKS) is a Fully Homomorphic Encryption (FHE) scheme that enables real number operations on encrypted data. Specifically, ciphertexts are in the form of $(b, a) \in R_Q^2$, where Q is a product of some prime numbers and $R_Q = \mathbb{Z}_Q[X]/\langle X^N + 1\rangle$. A single ciphertext contains $N/2$ slots, each slot consisting of a single complex number. In this paper, $N/2$ is represented as n_t. All plaintexts have a one-dimensional shape. For any one-dimensional data A, including plaintexts, $A[i]$ denotes the i-th entry, where indexing starts from 0. Similarly, this applies equally to higher-dimensional data. For instance, in two-dimensional data $\overline{A}$, the value at the i-th row and j-th column is denoted as $\overline{A}[i][j]$.

In RNS-CKKS, there are three main homomorphic operations: addition, multiplication, and rotation. The addition and multiplication operations correspond to addition and multiplication between slots at the same positions within the ciphertexts, respectively. Rotation involves cyclic shifting of the values stored in each slot. In our paper, we will represent these operations as follows: For ciphertexts $\mathrm{Enc}(m_1)$ and $\mathrm{Enc}(m_2)$, which represent the encrypted plaintext messages m_1 and m_2, respectively,

- $\mathrm{Enc}(m_1) \oplus \mathrm{Enc}(m_2) = \mathrm{Enc}(m_1 + m_2)$
- $\mathrm{Enc}(m_1) \odot m_2 = m_1 \odot \mathrm{Enc}(m_2) = \mathrm{Enc}(m_1 \cdot m_2)$
- $\mathsf{Rot}(\mathrm{Enc}(m_1); r) = \mathrm{Enc}(\mathsf{PRot}(m_1; r))$,

where $m_1 \cdot m_2$ denotes component-wise multiplication and $\mathsf{PRot}(m_1; r)$ denotes the cyclically shifted plaintext vector of m_1 by r to the left. RNS-CKKS is known to satisfy indistinguishability under chosen plaintext attack (IND-CPA) security [31], which guarantees that no information leakage occurs even after applying all supported homomorphic operations to ciphertexts. Moreover, it is well known that the RNS-CKKS homomorphic encryption scheme can also satisfy the stronger security notion of IND-CPA^{D}, simply by adding appropriately sized Gaussian noise to the resulting ciphertexts while keeping the original homomorphic operations unchanged [32]. Since all algorithms proposed in this paper rely solely on the three main operations inherently supported by the RNS-CKKS scheme, the security of our algorithms is

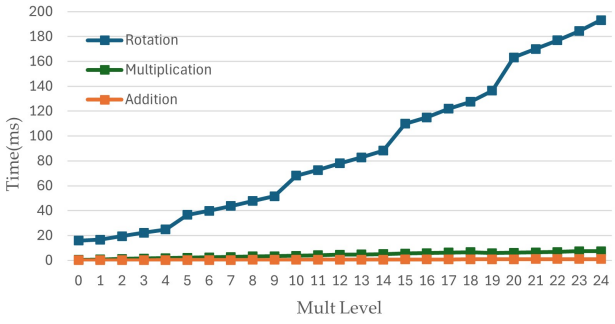

Figure 1: Latency of RNS-CKKS basic operation - addition, plaintext multiplication, rotation.

naturally ensured by the IND-CPA (or IND-CPA^{D}) security of RNS-CKKS. Moreover, data integrity is preserved with only negligible error introduced by the approximate nature of the scheme, ensuring that the ciphertexts undergo minimal distortion during homomorphic computations.

In this paper, the rotation operation of rotating a ciphertext by r is denoted as *rotation with r shift*. Rotation with the same shift requires the same rotation key during the operation. This means that rotation operations with different shifts require different rotation keys, and since the size of these rotation keys can be quite large, reducing them is one of the important challenges for FHE-based PPML. Also, it should be noted that in this paper, we only use plaintext multiplication, meaning that multiplication occurs only between plaintext and ciphertext, not between ciphertexts.

Every ciphertext has a unique positive integer called the multiplicative level (abbreviated as mult level or level when there is no confusion). The level of a ciphertext indicates the number of remaining homomorphic multiplications that can be performed. When multiplying ciphertext with levels l_1 and l_2, the resulting ciphertext will have a level of $\min(l_1, l_2) - 1$. Similarly, for multiplication between an l_1-level ciphertext and a plaintext, the resulting ciphertext has a level of $l_1 - 1$. If the level becomes 0 due to repeated multiplication operations, further multiplication becomes impossible, and a bootstrapping operation is required to increase the level again. This operation is the most computationally intensive operation in RNS-CKKS. The number of levels consumed when performing an HE algorithm is referred to as the depth of the algorithm. Therefore, when designing a homomorphic encryption algorithm, it is crucial to consider not only the number of heavy operations like rotations and ciphertext-ciphertext multiplications, but also the depth consumed.

The latency of all homomorphic operations, including addition, multiplication, and rotation, is influenced by the level of the ciphertexts. Operations performed at higher levels require more time. The precise execution time of each operation varies slightly depending on several initial settings of CKKS, and in this paper, the time required for each operation in our environment is depicted in Figure 1.

Table 1
Summary of convolution-related notation.

Notation	Description
h_i, w_i	Height and width of the input feature map
h_o, w_o	Height and width of the output feature map
c_i	Number of input channels
c_o	Number of output channels
f_h, f_w	Height and width of the convolution kernel
s	Stride of the convolution
k_i, k_o	Gap parameters of the input and output ciphertext layouts, respectively
t_i, t_o	Numbers of input/output channel groups, where $t_i = \lceil c_i/k_i^2 \rceil$ and $t_o = \lceil c_o/k_o^2 \rceil$
p_i, p_o	Numbers of identical parallel data blocks in the input/output ciphertexts, respectively
p_c	Number of parallel data blocks merged in the CrossCombine stage
q	Number of ciphertexts used to cover all output channels, $q = \lceil c_o/p_i \rceil$
n_t	Number of slots in a ciphertext
r_l	Latency of one rotation operation at multiplicative level l

As indicated in Figure 1, the rotation operation takes the longest time compared to addition and plaintext multiplication operations, and it also exhibits the highest rate of increase in execution time with respect to the level. Therefore, to achieve optimization of execution time in programs utilizing RNS-CKKS, it is effective to reduce the number of rotation operations.

3.2. Multiplexed Parallel Convolution

Due to its feature of supporting real number operations in an encrypted state, RNS-CKKS is widely used in Privacy-Preserving Machine Learning (PPML). In [8], CNN, particularly ResNet, was implemented using multiplexed parallel packing. Multiplexed parallel packing was proposed to address situations involving convolutions with stride values greater than 1. It possesses the property of being multiplexed, meaning multiple channels are interleaved, and also exhibits the parallel property where the same data is replicated multiple times within a ciphertext.

The convolution algorithm, performed on ciphertext that is multiplexed parallel packed, is proposed as multiplexed parallel convolution (MPConv) in [8]. MPConv involves three main processes, denoted as SISOConv, RotationSum, and ZeroOutCombine. These correspond to the *SISO convolution*, *rotation and sum*, and *zeroout & rotation, rotation & sum* stages illustrated in Figure 6 of [8], respectively; we adopt the compact names SISOConv, RotationSum, and ZeroOutCombine throughout this paper for readability. To describe these processes, [8] introduced various convolution-related parameters, which we also adopt throughout this paper for consistency and to avoid confusion. For readability, Table 1 summarizes the main notation used in both MPConv and the proposed method, while the precise parameter

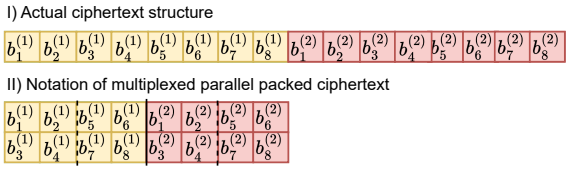

Figure 2: Notation of multiplexed parallel packed ciphertext in this paper.

values and detailed configurations for each convolution are provided in Appendix B.

The SISOConv process is based on single-input single-output convolution proposed in [23, 33]. In terms of plaintext size, assume that the three-dimensional single input to this process is denoted as $A \in \mathbb{R}^{w_i \times h_i \times c_i}$. For a single kernel K, which can be denoted as $K \in \mathbb{R}^{f_h \times f_w \times c_i}$, the result of the SISOConv process is $B \in \mathbb{R}^{w_o \times h_o \times c_i}$. The RotationSum process corresponds to combining the values at each position of the channels in B. In this process, rotation and addition operations are repeated $\log_2 c_i$ times. The result of the RotationSum process can be denoted as $C \in \mathbb{R}^{w_o \times h_o}$. In the ZeroOutCombine process, since there are c_o kernels, by combining c_o copies of $C \in \mathbb{R}^{w_o \times h_o}$, we can get final results as $C' \in \mathbb{R}^{w_o \times h_o \times c_o}$. In particular, the ZeroOutCombine process can be viewed as divided into two subprocesses: the ZeroOut subprocess, which removes invalid values, and the Combine subprocess, which combines the results of each channel. Finally, to maintain the characteristic of the parallel existence of identical data within the ciphertext, the output is copied by a few rotation and addition operations.

Since rotation operations are relatively computationally intensive, the approximate performance of the algorithm can be gauged by the number of rotation operations performed. The number of rotations performed in each of the processes SISOConv, RotationSum, ZeroOutCombine and copying data are denoted as $f_h f_w - 1$, $q\left(2\lceil \log_2 k_i \rceil + \lceil \log_2 t_i \rceil\right)$, c_o, and $\log_2 p_o$, respectively. Considering that the number of rotations varies per level, let the execution time of rotation operations at level l be r_l. When multiplexed parallel convolution is executed on a ciphertext with level l', and since the entire process consumes two multiplicative levels, the execution time can be summarized as follows. $r_{l'}\left(f_h f_w - 1\right) + r_{l'-1}\left(q\left(2\lceil \log_2 k_i \rceil + \lceil \log_2 t_i \rceil\right) + c_o\right) + r_{l'-2}\left(\log_2 p_o\right)$.

Throughout this paper, we use the notation in Figure 2 to illustrate a multiplexed parallel packed ciphertext, specifically for the case where the gap $k = 2$. Figure 3 simplifies the multiplexed parallel convolution where each variable is defined as $w_i = h_i = w_o = h_o = 1, f_w = f_h = 3, c_i = c_o = 8, p_i = p_o = 2, k_i = k_o = 2$. $b_j^{(i_1)}$ corresponds to the data of the j-th channel when the input data of convolution is multiplied by the weight of i_1-th kernel, and c_{i_1} can be defined as $c_{i_1} = \sum_{j=1}^{8} b_j^{(i_1)}$.

In the input of the convolution, each ciphertext contains two identical data blocks existing in parallel, and

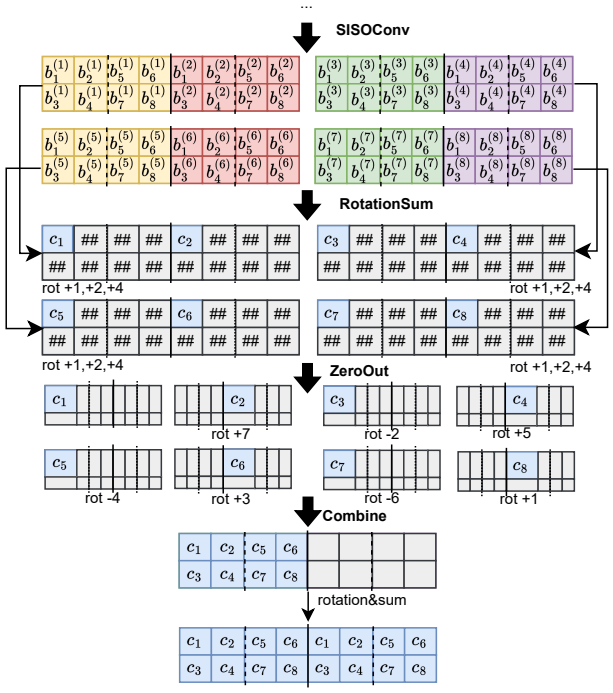

Figure 3: A conceptual diagram example of MPConv [8] where $w_i = h_i = w_o = h_o = 1, f_w = f_h = 3, c_i = c_o = 8, p_i = p_o = 2, k_i = k_o = 2$.

each data block is multiplied by different kernel weights. This corresponds to the SISOConv process. In the RotationSum process, each data block has eight channels, so with $\log_2 8 = 3$ rotations per ciphertext, the values of the channels are summed. During this process, meaningless values are generated which are denoted as ## in Figure 3. In the ZeroOutCombine process, these meaningless values are eliminated through multiplication operations and rearranged to the correct positions through rotation operations. Lastly, one rotation ensures that each ciphertext maintains the structure containing two identical parallel data blocks.

Furthermore, in the conventional method [8], multiplexed parallel convolution was tested in the situation of six convolutions, which are CONV1, CONV2, CONV3s2, CONV3, CONV4s2, and CONV4. To compare with this, we also conducted several tests targeting these convolutions. The parameters for each convolution can also be found in Appendix B. In [8], the function Vec is often used to map a three-dimensional tensor to a vector to describe convolution. To avoid confusion, in our paper, Vec is defined similarly: for a three-dimensional tensor $B \in \mathbb{R}^{h_i \times w_i \times c_i}$, $\mathrm{Vec}(B)$ maps tensor B to a vector in $\mathbb{R}^{n_t}$. $\mathrm{Vec}(B) = (b_0, \ldots, b_{n_t - 1}) \in \mathbb{R}^{n_t}$, where b can be defined as

$$b[i] = \begin{cases} B\left[\lfloor (i \bmod h_i w_i)/w_i \rfloor\right] \\ \left[i \bmod w_i\right]\left[\lfloor i/h_i w_i \rfloor\right], & \text{if } 0 \le i < h_i w_i c_i \\ 0, & \text{otherwise,} \end{cases}$$

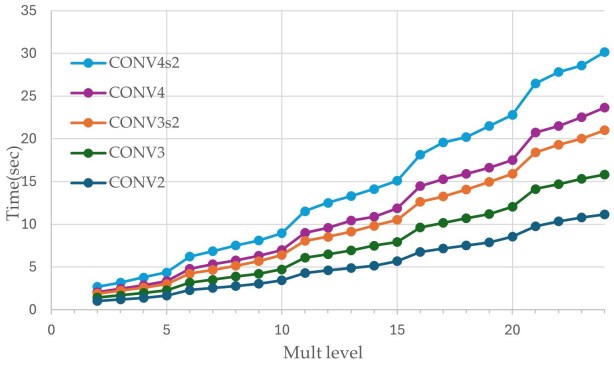

Figure 4: Latency of MPConv in various multiplication levels.

4. Rotation-Optimized Multiplexed Parallel Convolution

As established in Section 2, MPConv provides a broadly applicable and structurally flexible approach to homomorphic convolution. However, its practical deployment is hindered by significant computational bottlenecks, primarily due to its reliance on a large number of rotation operations. As illustrated in Figure 4, the execution time of MPConv grows super-linearly with the multiplicative level. Furthermore, the required rotation keys introduce a prohibitively large volume of data, creating a practical bottleneck for transmission and storage in client-server settings.

To overcome these fundamental limitations, this paper introduces Rotation-Optimized Multiplexed Parallel Convolution (RO-MPConv). The core principle of our method is to restructure the convolution algorithm to minimize the number of rotation operations, thereby reducing both computational latency and rotation key volume without sacrificing other performance metrics. This section details the three key strategies behind RO-MPConv: a fundamental redesign to reduce rotations, the exploitation of the correlation between the RotationSum and CrossCombine processes, and the strategic use of additional multiplicative depth for further performance trade-offs.

4.1. High-Level Idea

The primary strategy of RO-MPConv is to reduce both computational latency and key transmission volume by minimizing the number of rotation operations. As summarized in Table 2, the ZeroOutCombine process in the conventional MPConv is a prime target for optimization. This is because the number of rotation operations and the number of required rotation keys are nearly identical, implying that each rotation uses a unique shift and, therefore, a unique rotation key. Consequently, reducing the rotations in this specific process yields a dual benefit: decreased latency and a smaller key volume.

The key idea of our approach is to modify the order of convolution weights and the direction of rotations during the RotationSum process by considering the final arrangement of the multiplexed parallel packed result from the outset. In the conventional MPConv, illustrated in Figure 3, the data corresponding to each channel is aligned to the same position (the top-left in the diagram) after the RotationSum process. However, to restore the multiplexed parallel packed state in the final output, the Combine subprocess must then perform a number of rotation operations nearly equal to the number of output channels to move each channel to its correct final location.

Table 2
Number of rotation keys and rotation operations during MPConv.

Convolution	Process	# RotationKey	# Rotation operation
CONV1	SISOConv	8	8
	RotationSum	2	4
	ZeroOutCombine	16	17
CONV2	SISOConv	8	8
	RotationSum	4	32
	ZeroOutCombine	16	17
CONV3s2	SISOConv	8	8
	RotationSum	4	64
	ZeroOutCombine	33	34
CONV3	SISOConv	8	8
	RotationSum	5	40
	ZeroOutCombine	33	34
CONV4s2	SISOConv	8	8
	RotationSum	5	80
	ZeroOutCombine	66	67
CONV4	SISOConv	8	8
	RotationSum	6	48
	ZeroOutCombine	66	67

Figure 5 illustrates the high-level idea of RO-MPConv, where the entire process is conducted in consideration of the final channel arrangement. In the final output ciphertext, channels c_1 through c_4 are grouped in the initial block of the ciphertext, while channels c_5 through c_8 are grouped in the subsequent block. Furthermore, within these blocks, channel pairs such as (c_1, c_5), (c_2, c_6), (c_3, c_7), (c_4, c_8) occupy the same relative positions.

To achieve this structure efficiently, we introduce two key modifications. First, the order in which convolution weights are multiplied in the SISOConv process is revised. Instead of a simple sequential multiplication, weights are applied in an order that pre-positions the resulting channels. For example, the results for channels 1-4 are generated from the first parallel data block, while channels 5-8 are generated from the second. Additionally, channel pairs with similar target positions in the final layout (e.g., 1 and 5, 2 and 6) are generated within the same ciphertext. This deliberate placement ensures that they can be processed with the same rotation direction in the subsequent RotationSum stage. Second, during the RotationSum process, ciphertexts undergo rotations in both positive and negative directions, not just

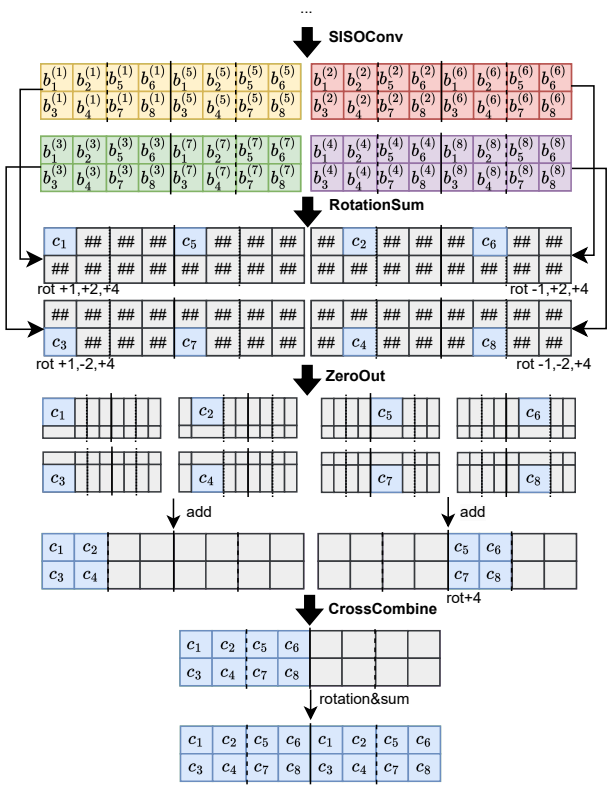

Figure 5: A conceptual diagram of the high-level idea of RO-MPConv, under the same conditions as in Figure 3.

positive ones. This strategic variation in rotation direction allows the resulting valid values (the summed channels) to be placed in locations that closely approximate their final positions in the output ciphertext, rather than being gathered in a single default location.

As a result of these modifications, the majority of channels are already aggregated into near-final positions before the final combining step. This allows us to replace the original Combine process, which meticulously repositioned each channel one by one, with a new, far more efficient CrossCombine subprocess. The CrossCombine operation takes these large, pre-arranged blocks of channels and moves them into their final configuration with only a minimal number of rotations.

It is important to note that while the modifications in the SISOConv and RotationSum processes are crucial for enabling this optimization, they are performance-neutral. The actual performance gain arises exclusively from replacing the Combine process with the CrossCombine process. To describe the gain achieved by the newly introduced CrossCombine process, we define p_c as the number of parallel data blocks that must be merged in CrossCombine. Unlike the original Combine process, which individually repositions each output channel, CrossCombine merges pre-arranged channel groups, thereby significantly reducing the required number of rotations. Quantitatively, this change reduces the required number of rotation operations from c_o to $p_c - 1$. In the six convolution scenarios previously mentioned, c_o is 16, 32, or 64, whereas p_c is a much smaller value, around 2, 4, or 8. This leads to a notable observation: RO-MPConv is particularly effective for convolutions with a large number of output channels (c_o) as the reduction in rotations becomes more significant as (c_o) increases.

This improvement is reflected in the revised execution time formulation. Defining the execution time of a rotation at level l as r_l, the total execution time of RO-MPConv operating on a ciphertext at level l' can be summarized as follows: $r_{l'}\left(f_h f_w - 1\right) + r_{l'-1}\left(q\left(2\lceil\log_2 k_i\rceil + \lceil\log_2 t_i\rceil\right)\right) + r_{l'-2}\left(p_c - 1 + \log_2 p_o\right)$.

4.2. Using the Correlation of the RotationSum and CrossCombine Processes

The objectives of the three main processes in multiplexed parallel convolution are distinct. The SISOConv process computes the product of the input data and the kernel weights. The RotationSum process then aggregates the values from each of these resulting channels. Finally, the ZeroOutCombine process filters out invalid values generated during RotationSum and rearranges the valid values into their correct final positions.

As established in the previous section, the replacement of the Combine subprocess with CrossCombine in RO-MPConv reduces the number of rotations required in the final stage. This newfound efficiency opens up an opportunity for a further optimization: a strategic trade-off between the RotationSum and CrossCombine processes. The core idea is to perform fewer merge operations during the RotationSum phase and compensate for the incomplete summation by performing additional rotations during the CrossCombine phase.

In the conventional RotationSum process, each rotation-and-addition operation typically halves the number of distinct channel components within a ciphertext, requiring $\lceil\log_2 c_i\rceil$ rotations per ciphertext to fully merge all channels. If we reduce the number of rotations performed per ciphertext by x, the number of rotations in the RotationSum stage decreases by $q{\cdot}x$. However, this leaves 2^x partial sums within each ciphertext that must be combined later. Consequently, the number of parallel data blocks to be merged in the CrossCombine stage increases by a factor of 2^x. This trade-off is generally infeasible in the original MPConv because the number of rotations in its Combine step is already large (c_o), and doubling it would negate any savings. In contrast, RO-MPConv's CrossCombine process requires only a small number of rotations ($p_c - 1$), making this trade-off viable and often advantageous.

Figure 6 provides a conceptual example of this technique. In this diagram, the notation $b^{(i)}_{j_1,j_2,j_3,j_4}$ represents the sum of four channel values: $b^{(i)}_{j_1} + b^{(i)}_{j_2} + b^{(i)}_{j_3} + b^{(i)}_{j_4}$. After the SISOConv process, each parallel data block contains 8 channels. In the standard RO-MPConv shown in Figure 5, the RotationSum process would perform $3 \times 4 = 12$ rotations, followed by 1 rotation in the CrossCombine process, for a total of 13 rotations. However, in Figure 6, we

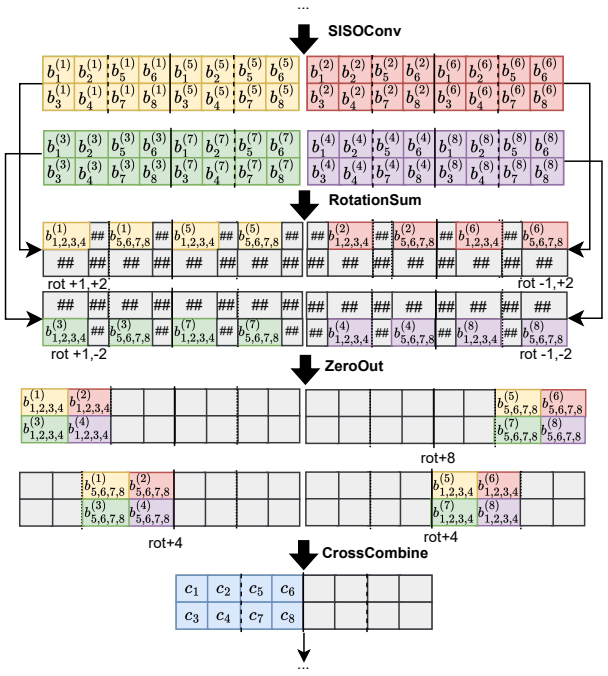

Figure 6: A conceptual diagram showing how RO-MPConv leverages the correlation between the RotationSum and CrossCombine stages to reduce rotation counts.

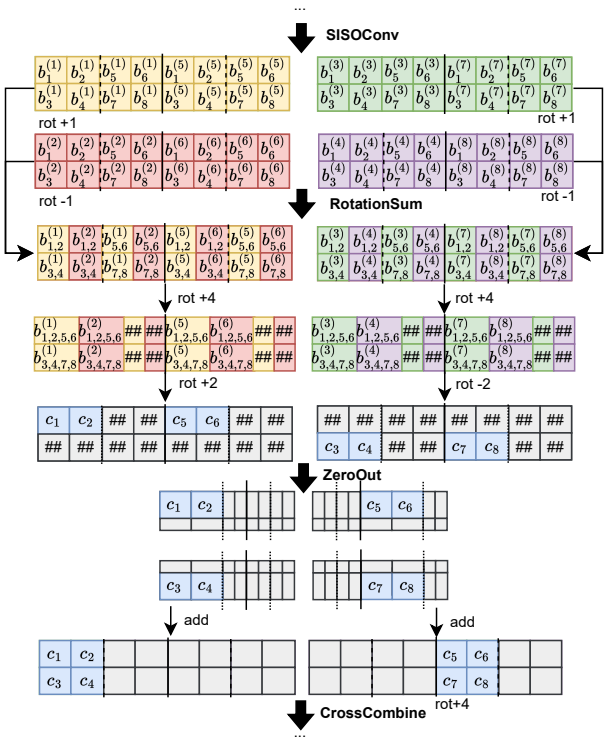

Figure 7: A conceptual diagram of the RO-MPConv variant that utilizes an additional multiplicative depth.

reduce the number of rotations in RotationSum to 2 per ciphertext, for a total of 2×4=8 rotations. This leaves two partial sums per block. The number of items to be merged in CrossCombine, therefore, increases from 2 to 4, requiring 3 rotations. The new total is 8+3=11 rotations, demonstrating a net reduction.

However, the optimal number of rotations to defer is not fixed, as the execution time of a rotation varies with its multiplicative level. Specifically, rotations in the RotationSum process occur at a higher (and thus slower) level than those in the CrossCombine process. Therefore, a careful balance must be struck to achieve the minimum possible latency.

To formalize this trade-off, let x denote the number of rotation stages deferred from the RotationSum process. When using this correlation, the optimal value of x for a 2-depth convolution starting at level l can be found by solving the following minimization problem:

$$\underset{x}{\operatorname{argmin}} \left(r_{l-1} q \left(\lceil \log_2 c_i \rceil - x \right) + r_{l-2} \left(2^x p_c - 1 \right) \right) \quad (1)$$

where x is an integer satisfying $x \in \left[0,\ \log_2 \left(2^{\lceil \log_2 c_i \rceil} p_i / c_o\right)\right]$. The first term in the equation accounts for the total time of the reduced rotations in RotationSum, while the second term reflects the time for the increased rotations in CrossCombine. The upper bound for x is determined by the finite length of the ciphertext, which limits the number of partial sums that can be combined, an issue discussed further in Section 8.

4.3. Convolution with Additional Depth

The latency of FHE algorithms is mainly determined by the number of expensive homomorphic operations, such as rotations and ciphertext multiplications, and by the frequency of bootstrapping when the multiplicative level is exhausted. Thus, efficient FHE algorithm design requires balancing the number of such expensive operations against multiplicative-depth consumption. This type of trade-off is common in the FHE literature. For example, Chen et al. [34] study the trade-off between additional depth and execution time in the SlotToCoeff and CoeffToSlot procedures of bootstrapping, while Lee et al. [35] study the trade-off between additional depth and approximation accuracy for ReLU evaluation.

In this subsection, we introduce a family of RO-MPConv variants that use additional multiplicative depth to further reduce latency. The key idea is to merge ciphertexts that require the same rotation shift during the RotationSum process. Because RO-MPConv allows ciphertexts to follow different rotation directions, their valid values can appear at different positions, enabling such ciphertexts to be merged after filtering invalid values. This reduces the number of subsequent rotations. This flexibility is not available in the original MPConv: in the conventional structure shown in Figure 3, all ciphertexts follow the same rotation pattern in RotationSum, so their valid values appear at identical positions and cannot be effectively merged without additional rearrangement.

Figure 7 illustrates this additional-depth optimization. Starting from the RO-MPConv layout in Figure 5, ciphertexts can have different valid-value positions because they are rotated in different directions during RotationSum. In the example, ciphertexts that require a +2 rotation and those that require a −2 rotation are first grouped separately. Within each group, the ciphertexts are filtered to remove invalid values and then merged before applying the corresponding rotation. This filtering step requires plaintext multiplication, and therefore consumes one additional multiplicative level. However, because the valid values occupy different positions, the merged ciphertexts do not overlap in their valid slots.

Compared with the 2-depth RO-MPConv in Figure 5, the additional-depth variant in Figure 7 reduces the number of rotations in the RotationSum and ZeroOutCombine processes from 13 to 9. This example shows that additional multiplicative depth can be used to reduce rotation operations. From the rotation-key perspective, the optimization mainly reduces the number of times existing rotation shifts are used, rather than introducing new rotation shifts. Therefore, additional-depth RO-MPConv variants have negligible impact on the total number of required rotation keys.

The maximum additional depth that an RO-MPConv variant can consume is a function of q, the number of ciphertexts available for merging in the RotationSum process. The underlying mechanism involves iteratively combining these q ciphertexts in pairs, where each iteration of merging consumes one additional multiplicative level. For $q \geq 2$, this merging process can be repeated at most $\lfloor \log_2 q \rfloor - 1$ times. Given that the standard RO-MPConv already consumes two levels, the maximum total depth for a variant becomes $2 + (\lfloor \log_2 q \rfloor - 1) = \lfloor \log_2 q \rfloor + 1$.

We now formalize the performance of RO-MPConv variants that utilize both the RotationSum-CrossCombine correlation and the additional depth-consuming technique. Consider an RO-MPConv variant that consumes $d + 1$ multiplicative levels, where $d \geq 2$, and starts from level l, where $l \geq d + 1$. The rotation latency of the RotationSum and CrossCombine processes can be approximated as the following equation:

$$r_{l-1}\left(x_1 q\right) + \sum_{m=2}^{d} r_{l-m}\left(\frac{x_m q}{\prod_{j=1}^{m-1} q_j}\right) + r_{l-d-1}\left(2^x p_c - 1\right) \quad (2)$$

where variables are defined as $\prod_{m=1}^{d} q_m = q$, $\lceil \log_2 c_i \rceil - x = \sum_{m=1}^{d} x_m$ and $x_m \geq \lceil \log_2 q_m \rceil$ for all m. Here, x has the same meaning as in Subsection 4.2; equivalently, $\lceil \log_2 c_i \rceil - x$ is the number of rotation stages performed in RotationSum. The variable x_m denotes the number of rotation stages performed before the m-th ciphertext-merging stage, and q_m denotes the number of ciphertexts merged at that stage. Since x_m rotation-and-addition stages can aggregate up to 2^{x_m} components within a ciphertext, at most 2^{x_m} ciphertexts can be merged in the m-th stage. This gives the constraint $x_m \geq \lceil \log_2 q_m \rceil$.

In practice, the use of additional depth is determined offline. First, the target depth parameter d is fixed according to the available multiplicative levels before the next bootstrapping operation. Then, the parameters x, $\{x_m\}_{m=1}^{d}$, and $\{q_m\}_{m=1}^{d}$ are selected to minimize Equation 2 subject to the above constraints. Because the feasible combinations are limited and Equation 2 is simple to evaluate, this optimization can be performed by brute-force search. Moreover, these parameters are fixed during the implementation of RO-MPConv and are not optimized during inference, so this search has no impact on runtime latency.

4.4. Algorithm Description

In presenting the main idea of RO-MPConv, we focused on reducing the number of rotations required in the ZeroOutCombine process to reduce execution time and the size of rotation keys. Also, by leveraging the relationship between RotationSum and CrossCombine, as well as using convolution with additional depth, we were able to achieve further execution time reduction. In this section, the focus is on explaining the algorithm of RO-MPConv, taking into account such ideas.

Through our high-level idea proposed in Subsection 4.1, it is evident that optimization of computations is feasible. However, due to the nature of the idea, which involves considering the arrangement of results, the method of carrying out computations is not unique. For instance, Figure 8 is a modified convolution of Figure 5. Upon observation, it can be seen that although the batch of kernels and types of rotations may vary slightly, the output of the convolution remains the same, and the number of rotations is also identical. These modifications in the RO-MPConv algorithm do not affect the performance, which means the number of rotations or the size of rotation keys does not change, and it only changes the shift of the rotation operation. Indeed, the RO-MPConv algorithm is not unique and possesses a heuristic aspect.

Thus, we will not provide a single fixed generalization of how kernel weights are arranged and which rotation operations are employed in the RotationSum process and CrossCombine process; rather, we will provide **blueprint**s detailing these aspects. Each rotation-optimized convolution requires three blueprints: KernelBP, RotationSumBP, and CrossCombineBP. These blueprints contain information about the kernel placement, the rotation shifts used during the RotationSum process, and the configuration for the CrossCombine process, respectively. The precise values for each blueprint used in our experiments can be found in Appendix A.

To describe the proposed RO-MPConv algorithm, we first define several utility functions and tensors for manipulating data within ciphertexts. These include valid value-selecting tensors (S, S', S'') and a weight-to-vector mapping function (ParWgt).

$S^{(r)}$ filter serves to select a valid value from the ciphertext, which is rotated by r. $S^{(r)} = \left(S^{(r)}\left[i_1\right]\right)_{0 \leq i_1 < n_t} \in \mathbb{R}^{n_t}$.

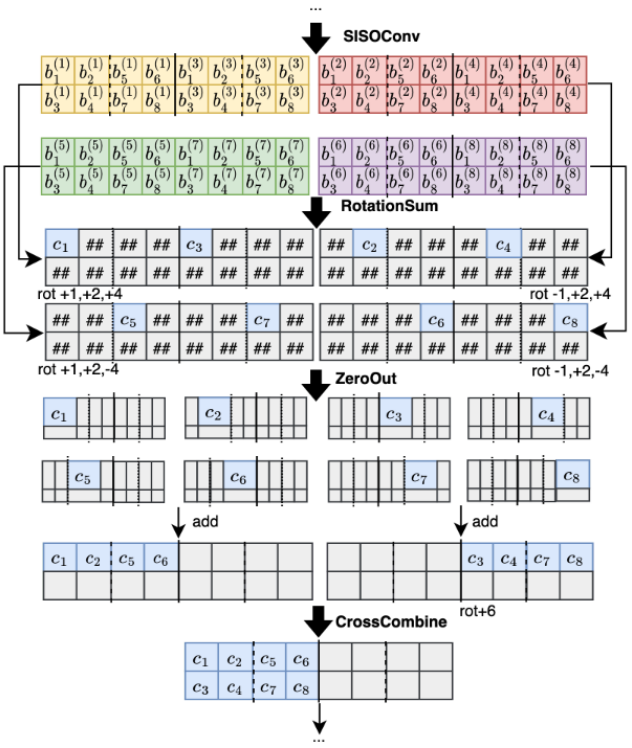

Figure 8: Showing the non-uniqueness property of RO-MPConv. Same condition, and same performance as Figure 5, but the kernel arrangement and shift amount of rotation are different.

The components of filter $S^{(r)}$ are defined as follows:

$$S^{(r)}\left[i_1\right] = \begin{cases} 1, & \text{if } \left(r > 0 \text{ and } \left(i_1 \bmod 2r\right) < r\right) \text{ or} \\ & \quad \left(r < 0 \text{ and } \left(i_1 \bmod |2r|\right) \geq |r|\right) \\ 0, & \text{otherwise,} \end{cases} \tag{3}$$

$S'^{(k,l)}$ filters out parallel data in the ciphertext. For a ciphertext which has l identical data in parallel, filter $S'^{(k,l)}$ selects the k-th data. $S'^{(k,l)} = (S'^{(k,l)}[i_1])_{0 \leq i_1 < n_t} \in \mathbb{R}^{n_t}$. The components of filter $S'^{(k,l)}$ are defined as follows:

$$S'^{(k,l)}\left[i_1\right] = \begin{cases} 1, & \text{if } n_t k/l \leq i_1 < n_t(k+1)/l \\ 0, & \text{otherwise,} \end{cases} \tag{4}$$

S'' filters out data considering when the stride value of convolution is greater than 1. $S'' = (S''[i_1])_{0 \leq i_1 < n_t} \in \mathbb{R}^{n_t}$. The components of the filter S'' are defined as follows:

$$S''\left[i_1\right] = \begin{cases} 1, & \text{if } i_1 \bmod k_o^2 w_o < k_i k_o w_o \text{ and } i_1 \bmod k_o < k_i \\ 0, & \text{otherwise,} \end{cases} \tag{5}$$

The function $\mathsf{ParWgt}\left(U; i_1, i_2, i_3\right)$ maps the weight tensor $U \in \mathbb{R}^{h_i \times w_i \times c_i \times c_o}$ to a plaintext vector in $\mathbb{R}^{n_t}$ via $\mathsf{ParWgt}\left(U; i_1, i_2, i_3\right) = \mathsf{Vec}\left(\overline{U}''^{(i_1,i_2,i_3)}\right)$ where $i_1 \in [0, f_h)$, $i_2 \in [0, f_w)$, $i_3 \in [0, q)$. Parallelly multiplexed shifted weight tensor $\overline{U}''^{(i_1,i_2,i_3)}$ is defined as follows. $\overline{U}''^{(i_1,i_2,i_3)} = \left(\overline{U}''^{(i_1,i_2,i_3)}\left[i_5\right]\left[i_6\right]\left[i_7\right]\right) \in \mathbb{R}^{k_i h_i \times k_i w_i \times t_i p_i}$ where $i_5 \in [0, k_i h_i)$, $i_6 \in [0, k_i w_i)$, and $i_7 \in [0, t_i p_i)$.

$$\overline{U}''^{(i_1,i_2,i_3)}[i_5][i_6][i_7] = \begin{cases} 0, & \text{if } k_i^2(i_7 \bmod t_i) + k_i(i_5 \bmod k_i) \\ & \quad + i_6 \bmod k_i \geq c_i \\ & \text{or } \lfloor i_7/t_i \rfloor + p_i i_3 \geq c_o \\ & \text{or } \lfloor i_5/k_i \rfloor - (f_h - 1)/2 \\ & \quad + i_1 \notin [0, h_i - 1] \\ & \text{or } \lfloor i_6/k_i \rfloor - (f_w - 1)/2 \\ & \quad + i_2 \notin [0, w_i - 1], \\ U[i_1][i_2][i_8][i_9] & \text{otherwise,} \end{cases} \tag{6}$$

i_8 and i_9 are defined as $i_8 = k_i^2\left(i_7 \bmod t_i\right) + k_i\left(i_5 \bmod k_i\right) + \left(i_6 \bmod k_i\right)$, and $i_9 = \mathsf{KernelBP}\left[p_i i_3\right]\left[\lfloor i_7/t_i \rfloor\right]$.

Since the RotationSum process used in RO-MPConv follows different rules, we redefine the SumSlots algorithm in [8] as AdaptSumSlots (Algorithm 1), which incorporates both the RotationSum process and SISOConv. Its execution flow is determined by RotationSumBP, whose entries specify the operation type through opType. When opType is 0, the algorithm performs a standard rotation-and-add operation in the RotationSum process. For example, the entry $[0, 2048]$ means that the ciphertext is rotated by 2048 and added to the original ciphertext. When opType is 1, the algorithm performs an intermediate ciphertext-merging step inside RotationSum, thereby consuming additional multiplicative depth. For example, the entry $[1, 4, 1, 32]$ means that four ciphertexts are aligned using sign-dependent rotations with shifts 1 and 32, filtered to remove invalid values, and then merged. When opType is 2, the algorithm performs the final preprocessing step before CrossCombine. This operation separates the partial outputs into different blocks by applying sign-dependent rotations and filtering invalid values, so that each block can be handled separately in the subsequent CrossCombine stage.

The three blueprints used in RO-MPConv specify different parts of the computation. KernelBP determines the placement of kernel outputs within ciphertexts and is used in the mapping function ParWgt. RotationSumBP determines the recursive RotationSum procedure in Algorithm 1. Specifically, RotationSumBP[0][0] stores the last index of RotationSumBP, and RotationSumBP[i][0] stores the opType of the i-th operation. For opType 1 or 2, RotationSumBP[i][1] stores the number of ciphertext blocks associated with that operation. Finally, CrossCombineBP specifies the final CrossCombine stage in Algorithm 2: CrossCombineBP[0] stores the number of intermediate blocks to be combined, denoted by CrossNum, and CrossCombineBP[$i+1$] gives the rotation shift applied to the i-th intermediate block.

As an example, consider the operation illustrated in Figure 8. Here, KernelBP $= [[1, 3], [2, 4], [5, 7], [6, 8]]$,

Algorithm 1 AdaptSumSlots($i', d, \mathsf{ct}'_\alpha, U$)

Input: Indicating the order of ciphertext i', Rotated parallely multiplexed tensor ciphertexts ct'_α, Current location within RotationSumBP d, and weight tensor U
Output: Tensor ciphertext ct_b and filter of ct_b that apply zero out S_b.

$\mathsf{ct}_\mathsf{b} \leftarrow \mathsf{ct}_\mathsf{zero}$
$\mathsf{S}_\mathsf{b} \leftarrow \mathsf{S}_\mathsf{one}$
$\mathsf{opType} \leftarrow \mathsf{RotationSumBP}[d][0]$
if $d = 0$ **then**
  for $i_1 \leftarrow 0$ to $f_h - 1$ **do**
    for $i_2 \leftarrow 0$ to $f_w - 1$ **do**
      $\mathsf{ct}_\mathsf{b} \leftarrow \mathsf{ct}_\mathsf{b} \oplus \mathsf{ct}'^{(i_1,i_2)}_\alpha \odot \mathsf{ParWgt}(U; i_1, i_2, i')$
    end for
  end for
end if
if $d > 0$ **then**
  if $\mathsf{opType} = 0$ **then**
    $\mathsf{ct}_\mathsf{a}, S_a \leftarrow \mathrm{AdaptSumSlots}(i', d-1, \mathsf{ct}'_\alpha, U)$
    $r' \leftarrow \mathsf{RotationSumBP}[d][1]$
    $\mathsf{ct}_\mathsf{b} \leftarrow \mathsf{ct}_\mathsf{a} \oplus \mathsf{Rot}(\mathsf{ct}_\mathsf{a}; r')$
    $S_b \leftarrow S_a \odot S^{(r')}$
  end if
  if $\mathsf{opType} = 1$ **then**
    $\mathsf{SumN} \leftarrow \mathsf{RotationSumBP}[d][1]$
    for $i'' \leftarrow \mathsf{SumN} \times i'$ to $(\mathsf{SumN}+1) \times i' - 1$ **do**
      $\mathsf{ct}_\mathsf{a}, S_a \leftarrow \mathrm{AdaptSumSlots}(i'', d-1, \mathsf{ct}'_\alpha, U)$
      for $j \leftarrow 2$ to $\log_2(\mathsf{SumN}) + 1$ **do**
        $r' \leftarrow \mathsf{RotationSumBP}[d][j]$
        if $\left(\left(i'' >> (j-2)\right) \& 1\right) = 1$ **then**
          $\mathsf{ct}_\mathsf{a} \leftarrow \mathsf{ct}_\mathsf{a} \oplus \mathsf{Rot}(\mathsf{ct}_\mathsf{a}; -r')$
          $S_a \leftarrow S_a \odot S^{(-r')}$
        else
          $\mathsf{ct}_\mathsf{a} \leftarrow \mathsf{ct}_\mathsf{a} \oplus \mathsf{Rot}(\mathsf{ct}_\mathsf{a}; r')$
          $S_a \leftarrow S_a \odot S^{(r')}$
        end if
      end for
      $\mathsf{ct}_\mathsf{b} \leftarrow \mathsf{ct}_\mathsf{b} \oplus (\mathsf{ct}_\mathsf{a} \odot S_a)$
    end for
  end if
  if $\mathsf{opType} = 2$ **then**
    $\mathsf{ct}_\mathsf{b}, S_b \leftarrow \mathrm{AdaptSumSlots}(i', d-1, \mathsf{ct}'_\alpha, U)$
    for $j \leftarrow 2$ to $\log_2(\mathsf{RotationSumBP}[d][1]) + 1$ **do**
      $r' \leftarrow \mathsf{RotationSumBP}[d][j]$
      if $\left(\left(i' >> (j-2)\right) \& 1\right) = 1$ **then**
        $\mathsf{ct}_\mathsf{b} \leftarrow \mathsf{ct}_\mathsf{b} \oplus \mathsf{Rot}(\mathsf{ct}_\mathsf{b}; -r')$
        $S_b \leftarrow S_b \odot S^{(-r')}$
      else
        $\mathsf{ct}_\mathsf{b} \leftarrow \mathsf{ct}_\mathsf{b} \oplus \mathsf{Rot}(\mathsf{ct}_\mathsf{b}; r')$
        $S_b \leftarrow S_b \odot S^{(r')}$
      end if
    end for
    $S_b \leftarrow S_b \odot S''$
  end if
end if
return $\mathsf{ct}_\mathsf{b}, S_b$

Algorithm 2 RO-MPConv($\mathsf{ct}'_\mathsf{a}, U$)

Input: Parallelly multiplexed tensor ciphertext ct'_a and weight tensor U.
Output: Parallelly multiplexed tensor ciphertext ct'_c
$\mathsf{ct}'_\mathsf{c} \leftarrow \mathsf{ct}_\mathsf{zero}$
for $i_1 \leftarrow 0$ to $f_h - 1$ **do**
  for $i_2 \leftarrow 0$ to $f_w - 1$ **do**
    $\mathsf{ct}'^{(i_1,i_2)}_\alpha \leftarrow \mathsf{Rot}(\mathsf{ct}'_\mathsf{a}; k_i^2 w_i(i_1 - (f_h-1)/2) + k_i(i_2 - (f_w-1)/2))$
  end for
end for
$\mathsf{CrossNum} \leftarrow \mathsf{CrossCombineBP}[0]$
for $i_3 \leftarrow 0$ to $\mathsf{CrossNum} - 1$ **do**
  $\mathsf{ct}'^{(i_3)}_\beta \leftarrow \mathsf{ct}_\mathsf{zero}$
end for
$\mathsf{BPlen} \leftarrow \mathsf{RotationSumBP}[0][0]$
for $i_4 \leftarrow 0$ to $\mathsf{RotationSumBP}[\mathsf{BPlen}][1] - 1$ **do**
  $\mathsf{ct}'_\mathsf{b}, S_a \leftarrow \mathrm{AdaptSumSlots}\left(i_4, \mathsf{BPlen}, \mathsf{ct}'_\alpha, U\right)$
  for $i_3 \leftarrow 0$ to $\mathsf{CrossNum} - 1$ **do**
    $\mathsf{ct}'^{(i_3)}_\beta \leftarrow \mathsf{ct}'^{(i_3)}_\beta \oplus \mathsf{ct}'_\mathsf{b} \odot \left(S_a \odot S'^{(i_3, \mathsf{CrossNum})}\right)$
  end for
end for
for $i_1 \leftarrow 0$ to $\mathsf{CrossNum} - 1$ **do**
  $\mathsf{ct}'_\mathsf{c} \leftarrow \mathsf{ct}'_\mathsf{c} \oplus \mathsf{Rot}\left(\mathsf{ct}'^{(i_1)}_\beta; \mathsf{CrossCombineBP}[i_1 + 1]\right)$
end for
for $j \leftarrow 0$ to $\log_2 p_o - 1$ **do**
  $\mathsf{ct}'_\mathsf{c} \leftarrow \mathsf{ct}'_\mathsf{c} \oplus \mathsf{Rot}\left(\mathsf{ct}'_\mathsf{c}; -2^j \left(n_t / p_o\right)\right)$
end for
return ct'_c

which specifies the pairing of kernel outputs within each ciphertext. The RotationSum procedure is described by $\mathsf{RotationSumBP} = [[2], [0,4], [2,4,1,4]]$. The first entry indicates that the last blueprint index is 2. The entry $[0,4]$ performs a rotation-and-add operation with shift 4, while $[2,4,1,4]$ performs the pre-CrossCombine separation using sign-dependent rotations with shifts 1 and 4, followed by filtering. The final combination is specified by $\mathsf{CrossCombineBP} = [2,0,6]$, meaning that two intermediate blocks are combined: the first block is accumulated without rotation, and the second block is rotated by 6 before accumulation.

A comparison of the performance of RO-MPConv and MPConv centered on the number of rotation operations can be summarized as follows. Still, the time required for rotation operations at mult level l' of the ciphertext is denoted as $r_{l'}$. For conventional MPConv, assuming that operations start at level l, its performance can be represented as follows: $r_l(f_h f_w - 1) + r_{l-1}(q(2\lceil \log_2 k_i \rceil + \lceil \log_2 t_i \rceil) + c_o) + r_{l-2}(\log_2 p_o)$.

For 2-depth consuming RO-MPConv which prioritizes minimizing execution time, assuming that operations start at level l, its performance can be represented as follows:

$\min_x(r_l(f_h f_w - 1) + r_{l-1}q(\lceil \log_2 c_i \rceil - x) + r_{l-2}(2^x p_c - 1 + \log_2 p_o))$ where $x \in [0, \log_2 \left(2^{\lceil \log_2 c_i \rceil} p_i / c_o\right)]$.

For 3 or more depth consuming RO-MPConv variants which prioritize minimizing execution time, assuming that operations start at level l, its performance can be represented as follows: $\min_{x,x_1,\ldots,x_d,q_1,\ldots,q_d}(r_l(f_h f_w - 1) + r_{l-1}(x_1 q) + \sum_{m=2}^{d} r_{l-m}\left(x_m \frac{q}{\prod_{j=1}^{m-1} q_j}\right) + r_{l-d-1}(2^x p_c - 1 + \log_2 p_o))$ where $x \in [0, \log_2 \left(2^{\lceil \log_2 c_i \rceil} p_i / c_o\right)]$, $\prod_{m=1}^{d} q_m = q$, $\lceil \log_2 c_i \rceil - x = \sum_{m=1}^{d} x_m$ and $x_m \geq \lceil \log_2 q_m \rceil$ for all m.

Once again, due to the heuristic nature of our approach, this paper does not provide a single, fixed algorithm for RO-MPConv. Instead, we have presented a set of performance equations that define the expected optimal performance for any given convolution configuration. These equations, therefore, serve as a benchmark for validation; an implementation of RO-MPConv can be considered correct if its performance aligns with the predictions derived from these equations.

5. Parallel BSGS Matrix-Vector Multiplication

Matrix-vector multiplication, including inference of fully connected layers of neural networks, is a common operation in recent AI models. Several methods have been devised for FHE matrix-vector multiplication [12, 36, 37, 38, 39]. One of the well-known methods to multiply a plaintext matrix and a ciphertext vector is the diagonal method from [12], and the Baby-step Giant-step (BSGS) algorithm can be applied to further reduce the number of rotations and thus the latency. Lee et al. [8] also implemented the fully connected layer by applying the BSGS algorithm to the diagonal method (hereinafter referred to as the BSGS diagonal method).

The characteristic of the BSGS diagonal method is that it determines the number of slots used only based on the lengths of the matrix and vector, regardless of the length of the ciphertext. For instance, when implementing ResNet targeting the CIFAR-10 ([40]) images in [8], the length of the ciphertext used is 32768. The fully connected layer of this ResNet involves a matrix-vector multiplication with a matrix of size 10×64 and a ciphertext containing 64 valid values. Due to the characteristics of multiplexed parallel packing, this operation starts with eight identical data copies within a single ciphertext. However, the BSGS diagonal method does not take this into account and operates on only a single data copy, using around 72 slots for the matrix-vector multiplication. In other words, it does not efficiently utilize the ciphertext.

Since underutilizing the ciphertext usually necessitates additional computations, many FHE algorithms, including the conventional method [8], frequently employ structures with multiple identical data copies within a single ciphertext. This pattern is particularly evident in bootstrapping, the most computationally intensive operation in FHE. Sparse slot bootstrapping [41], widely adopted across many FHE systems, operates under the assumption that multiple identical data copies exist within a single ciphertext.

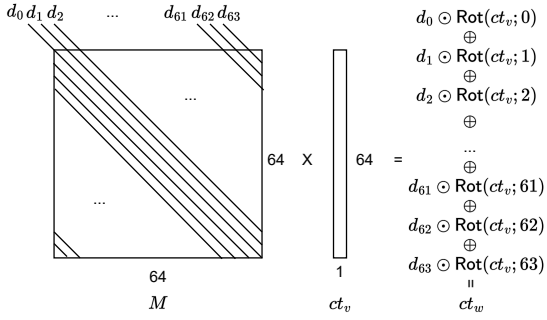

Figure 9: A simple illustration of diagonal method([12]) for matrix-vector multiplication.

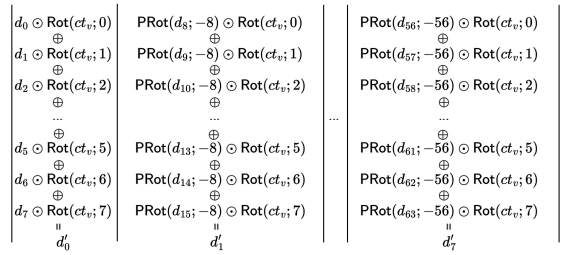

Figure 10: A simple illustration of BSGS diagonal method.

Therefore, in this section, we propose parallel BSGS matrix-vector multiplication, which supports the multiplication of a single plaintext square matrix and ciphertext vector containing multiple identical data copies. Parallel BSGS matrix-vector multiplication optimizes the number of rotations required compared to the BSGS diagonal method, especially when ciphertext can contain numerous identical parallel data copies, including multiplexed parallel packed ciphertext. To explain parallel BSGS matrix-vector multiplication, an understanding of both the conventional diagonal method and the BSGS diagonal method is required. For an intuitive explanation, we provide illustrations of an example for the multiplication of an $n \times n$ matrix and an $n \times 1$ ciphertext where $n = 64$. Here, the matrix is represented as M, the $n \times 1$ vector as v, and the result as w. In other words, $w = Mv$. Since w and v are ciphertexts in practice, we denote the ciphertext containing the vector v at its head as ct_v, and the ciphertext for w as ct_w. Both ct_v and ct_w have the same length, n_t.

Figure 9 illustrates the diagonal method introduced in [12]. For diagonals of M, d is defined as $d_i[j] = M[j][j+i]$ where $i, j \in [0, n)$. And each d_i 's length is n_t, which is the same as that of ct_v. As shown in Figure 9, we can obtain ct_w by accumulating the products of each d_i and $\mathsf{Rot}(ct_v; i)$. More precisely, it can be expressed as follows. $ct_w =$

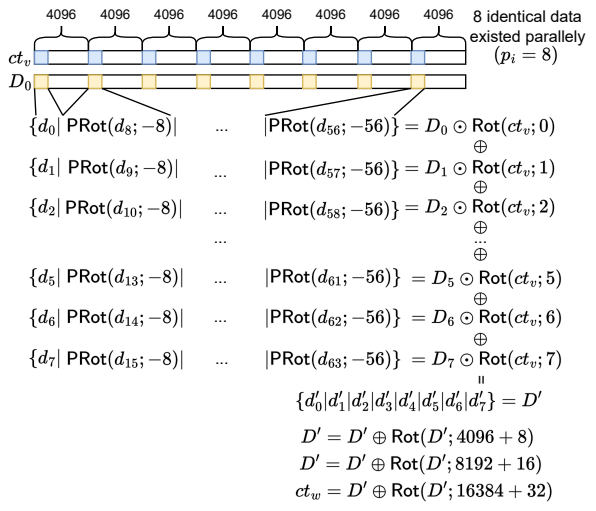

Figure 11: A simple illustration of parallel BSGS diagonal method.

$\bigoplus_{i=0}^{n-1}\left(d_i \odot \mathsf{Rot}(ct_v;i)\right)$. Since the number of rotations and plaintexts needed during the diagonal method is determined by the number of d_i, it can be observed that $n-1=63$ rotations and $n=64$ plaintexts are required.

By applying the BSGS algorithm to Figure 9, we can examine examples of BSGS diagonal method as in Figure 10. The idea leverages the fact that d_is are plaintexts, so precomputing them incurs no overhead. Therefore, optimization is achieved by pre-rotating d_is. To represent the BSGS diagonal method, several additional parameters are defined. n_1 denotes the small number of pre-rotated ciphertexts, where $n = n_1 n_2$. d' denotes the intermediate sums of products between pre-rotated plaintexts and ciphertexts. $d'_i = \bigoplus_{j=n_1 i}^{n_1 i+n_1-1}\left(\mathsf{PRot}\left(d_j;-n_1 i\right)\odot \mathsf{Rot}\left(ct_v;j \bmod n_1\right)\right)$ for $i \in \left[0,n_2\right)$. ct_w denotes the ciphertext result of matrix-vector multiplication. $ct_w = \bigoplus_{i=0}^{n_2-1}\mathsf{Rot}(d'_i;n_1 i)$. The number of rotations needed in the BSGS diagonal method is determined by $n_1=8$ and $n_2=8$. It can be observed that $n_1-1+n_2-1=14$ rotations are required. The number of plaintexts needed is still the same as the diagonal method. It requires $n=64$ plaintexts.

An example of our proposed algorithm, parallel BSGS matrix-vector multiplication, can be seen in Figure 11. In Figure 11, we showcase the scenario where parallel BSGS matrix-vector multiplication is applied to the fully connected layer of the ResNet implementation of [8] (for CIFAR-10 images). In the input ciphertext ct_v, 8 identical data copies are located in parallel, and the length of the ciphertext is $n_t = 32768$. Unlike the conventional BSGS diagonal method, which only utilizes the leftmost copy, we utilize all 8 copies. To further clarify the concept, additional definitions are necessary to explain Figure 11. Here, p_i signifies that there are p_i identical data copies which are positioned at equal intervals within a single ciphertext. Specifically, $p_i = 8$ in the case of ct_v. $\left(|v_0|v_1|v_2|v_3|v_4|v_5|v_6|v_7|\right)$ indicates that within one ciphertext, the value of v_i is located at positions $in_t/8$. $\boldsymbol{D}$ represents the arrangement of multiple d's within one ciphertext. It can be precisely defined as follows:

$$\boldsymbol{D}_i = \Big(|d_i|\mathsf{PRot}(d_{i+n_1};-n_1)|\mathsf{PRot}(d_{i+2n_1};-2n_1)|\ \ldots \\ \ldots|\mathsf{PRot}(d_{i+(n_2-1)n_1};-(n_2-1)n_1)|\Big) \tag{7}$$

for $0 \le i < n_1$. In other words, $\boldsymbol{D}$ is obtained by arranging n_2 number of d's within one ciphertext at equal intervals. $\boldsymbol{D}'$ represents a similar arrangement where multiple d's exist within the ciphertext: $\boldsymbol{D}' = \left(|d'_0|d'_1|\ \ldots\ |d'_{n_2-1}|\right)$. As observed in Figure 11, $\boldsymbol{D}'$ can be obtained as the sum of the multiplications of D and the rotated ct_v. Ultimately, $\boldsymbol{D}'$ is generated through three rotations and additions to combine d', thus reducing the number of rotations during the process of combining d'.

The key idea is to modify the plaintext such that, unlike the conventional structure where there was only one d within one ciphertext, multiple d's exist, as illustrated by $\boldsymbol{D}$. This idea increases the number of valid values within the plaintext and reduces the total number of plaintexts required. Additionally, by repeatedly applying rotation and addition operations to the ciphertext after computation, d' can be effectively combined. In practice, the advantage is evident in the reduced number of rotations required, as observed in Figure 11, where the rotation count is reduced to $2\log_2 n_2 + n_1 - \log_2 p_i = 11$ with $n = 64 = n_1 n_2 = 8\times 8$. Furthermore, the required number of plaintexts is reduced to $n_1 = 8$.

In Figure 11, the number of identical data copies within the ciphertext is fixed at 8. However, increasing this number could further reduce the number of rotation operations. For instance, consider increasing the number of identical data copies within the ciphertext to 16 by setting $n_1 = 4$ and $n_2 = 16$ and applying $ct_v = ct_v \oplus \mathsf{Rot}(ct_v;2048)$. In this scenario, when applying parallel BSGS matrix-vector multiplication, the required rotation count reduces to $2\log_2 n_2 + n_1 - \log_2 p_i = 9$. The required number of plaintexts also decreases to $n_1 = 4$. Thus, depending on how n_1 and n_2 are set, performance can vary. Therefore, we provide an equation to determine n_2 and $n_1 = n/n_2$ for a given n, which minimizes the rotation when using parallel BSGS matrix-vector multiplication.

$$\underset{n_2}{\operatorname{argmin}}\left(2\log_2\left(n_2\right) + \left(n/n_2\right) - \log_2 p_i\right) \tag{8}$$

At first glance, it may seem that increasing the value of n_2, which determines the number of d in a single ciphertext, indefinitely could lead to more optimization. However, since the length of a single ciphertext is limited, the number of identical data copies that can exist within the ciphertext is also limited. In other words, in Equation 8, n_2 is bounded as $p_i \le n_2 \le n_t/(2n)$. With the determined values of n_1 and n_2 obtained in this manner, we can employ our parallel BSGS matrix-vector multiplication (ParBSGSMatVecMul)

Algorithm 3 ParBSGSMatVecMul($D, \mathsf{ct_v}, n_1, n_2, p_i$)

Input: Plaintexts of parallelly located diagonals of $n \times n$ matrix D, Parallelly multiplexed tensor ciphertext that contains n valid value $\mathsf{ct_v}$, number of identical data in $\mathsf{ct_v}$ p_i, n_1 and n_2 that satisfy $n = n_1 n_2$.
Output: Parallelly multiplexed tensor ciphertext that contains n valid value $\mathsf{ct_w}$.

$\mathsf{ct_w} \leftarrow \mathsf{ct_{zero}}$
$\mathsf{ct_v} \leftarrow \mathsf{ct_v} \oplus \mathsf{Rot}(\mathsf{ct_v}; -n_1 n_2)$
for $i_1 \leftarrow 1$ **to** $\log_2(n_2/p_i)$ **do**
    $\mathsf{ct_v} \leftarrow \mathsf{ct_v} \oplus \mathsf{Rot}(\mathsf{ct_v}; -(n_t/p_i)/2^{i_1})$
end for
for $i_1 \leftarrow 0$ **to** $n_1 - 1$ **do**
    $\mathsf{ct_w} \leftarrow \mathsf{ct_w} \oplus D[i_1] \odot \mathsf{Rot}(ct_v; i_1)$
end for
for $i_1 \leftarrow 0$ **to** $\log_2(n_2) - 1$ **do**
    $\mathsf{ct_w} \leftarrow \mathsf{ct_w} \oplus \mathsf{Rot}(\mathsf{ct_w}; 2^{i_1}(n_1 + n_t/n_2))$
end for
return $\mathsf{ct_w}$

algorithm, as detailed in Algorithm 3. ParBSGSMatVecMul's input, the parallel diagonal plaintext D, can be further generalized as follows: $D_i = \sum_{j=0}^{n_2-1} \mathsf{PRot}\Big(d[jn_1+i]; -jn_1 - (n_t/n_2)j\Big)$ where $i \in [0, n_1)$

Finally, in terms of guidance for practical usage, to effectively utilize ParBSGSMatVecMul, it is necessary for there to be multiple identical data copies within a single ciphertext. Therefore, to use ParBSGSMatVecMul, 1) the number of slots for ciphertexts and plaintexts should be large relative to the size of the matrix and vector, and 2) multiple instances of the same data should be allowed within a ciphertext. Additionally, if there are already multiple identical data copies within a ciphertext (denoted as p_i), the effectiveness of ParBSGSMatVecMul can be enhanced, although the value of p_i is not critical to performance.

6. Rotation Key Reduction

Rotation keys required by a given FHE algorithm are typically generated and transmitted from the client to the server prior to execution, and are then stored for use during computation. Therefore, they do not directly affect the latency of the algorithm itself. However, if the number of such required rotation keys is excessively large, it may result in substantial transmission latency, which can become a practical bottleneck in real-world deployments.

RO-MPConv significantly reduced the number of rotations and effectively reduced the total size of the required rotation keys to about one-third (Table 3). In this section, we aim to further minimize the size of the rotation keys that need to be transmitted from the client to the server using additional methods. We employ the hierarchical rotation key system introduced by [11]. The hierarchical rotation key system organizes computation keys into hierarchical levels, reducing the size of the keys that need to be transmitted from the client to the server. If we define the keys required to perform rotations on the server as 0 key-level rotation keys, the client only sends 1 key-level rotation keys. Then, the server generates 0 key-level rotation keys from the 1 key-level rotation keys. Generating these 0 key-level rotation keys requires an additional server-side key generation step. Lee et al. [11] report that this step takes about 10–20 seconds for ResNet-scale models. This system, known as the two-level hierarchical rotation key system, allowed us to reduce the size of rotation keys.

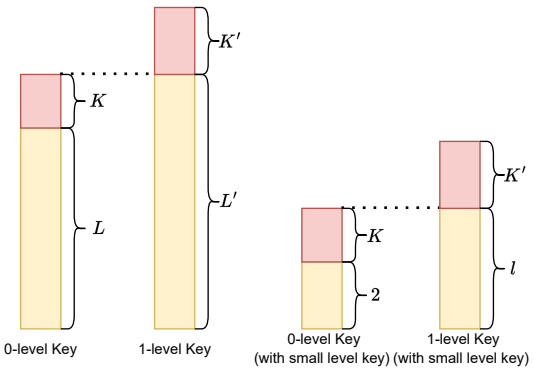

Figure 12: A simple and abstract illustration of applying the hierarchical rotation key system and small-level key system to rotation key.

Furthermore, additional optimization is possible by leveraging the structure of rotation keys. In CKKS, rotation keys have a combined structure supporting rotation operations from level 0 up to the maximum mult level in the CKKS parameters to ensure support for rotation operations at all mult levels. However, if we know precisely at which mult level rotation operations occur, we can transmit only the keys relevant to that mult level, reducing the transmission volume. When applied to RO-MPConv technology, by default, 2 levels are consumed during RO-MPConv operations. Therefore, transmitting only the rotation keys corresponding to these levels can reduce the transmission volume of rotation keys. We define the technique that reduces the size of the rotation key by limiting the rotation key to support rotation operations at a limited level as the *small-level key system*. We ultimately extended this to a two-level hierarchical rotation key system.

A simple illustration of the key structure used in this method is shown in Figure 12. The 0-level key in Figure 12 represents the conventional rotation key. L denotes the maximum level that the ciphertext can have, and K corresponds to the special modulus. In our implementation environment, L and K are 24 and 5, respectively. By applying the two-level hierarchical rotation key system, we can derive the 1-level key, whose maximum level becomes $L' = L + K = 29$. The key labeled as "1-level Key" in the figure corresponds to this. Furthermore, by applying our small-level key system, the supported level of the 0-level key can be reduced to 2. As a result, the maximum level of the 1-level key is also

Table 3
Comparison of rotation key size and transmission latency, assuming 100Mbps for network connection. The evaluation is conducted on all the convolution operations utilizing 2 depth used in ResNet that take CIFAR-10 images as input.

Method	MP-Conv [8]	RO-MPConv	RO-MPConv +Hierarchical rotation key system	RO-MPConv +Hierarchical rotation key system +Small-level key system
Rotation key size(MB)	29100	8250	2100	1100
Transmission latency(min)	38.8	11.0	2.8	1.5

reduced to $l = K+2 = 7$, allowing us to achieve a significant reduction in the size of each rotation key.

Table 3 presents the experimental results of applying a two-level hierarchical rotation key system and a small-level key system to reduce the size of rotation keys required for RO-MPConv. As mentioned in Subsection 4.3, utilizing additional depth is unrelated to the number of rotation keys. Therefore, experiments were conducted using the convolutions utilizing 2 depth. By applying the hierarchical rotation key system and small-level key system to the rotation keys transmitted in RO-MPConv, we were able to reduce the size of the final transmitted rotation keys by approximately 29× compared to [8]. Transmission latency showed similar results, decreasing from 38.8 minutes to 1.5 minutes, a result that suggests a meaningful improvement in the method's practical applicability.

7. Experimental Results

In this section, we will compare the execution time and size of rotation keys of the various operations proposed by us with those of the conventional method. The homomorphic encryption experiments are conducted with Lattigo v5 (a lattice-based homomorphic encryption library [42]) on an AMD Ryzen 9 7900X at 3.2 GHz (24 cores) with 124 GB of RAM, running Ubuntu 20.04. We set the main Lattigo CKKS parameter as follows: $N = 2^{16}, \log Q = [51, 46, 46, 46, 46, 46, 46, 46, 46, 46, 46, 46, 46, 46, 46, 46, 46, 46, 46, 46, 46, 46, 46, 46, 46, 46, 46], \log P = [60, 60, 60, 60, 60]$. This CKKS parameter set provides 128-bit security. P denotes the special modulus, and the length of special modulus corresponds to the length of $\log P$. Since our algorithm relies solely on standard CKKS operations, there are no particular parameter settings under which it performs exceptionally better or worse. To ensure the reproducibility of our results, the complete source code and scripts for all convolution and matrix-vector multiplication experiments presented in this paper have been made available at our GitHub repository.

To ensure a comprehensive evaluation of RO-MPConv, we benchmarked our algorithms on nine distinct convolution configurations. Six of these (CONV1, CONV2, CONV3s2, CONV3, CONV4s2, CONV4) are based on the ResNet architecture for CIFAR-10, following the precedent set in the original MPConv study. To demonstrate broader applicability, we also implemented three convolution operations from

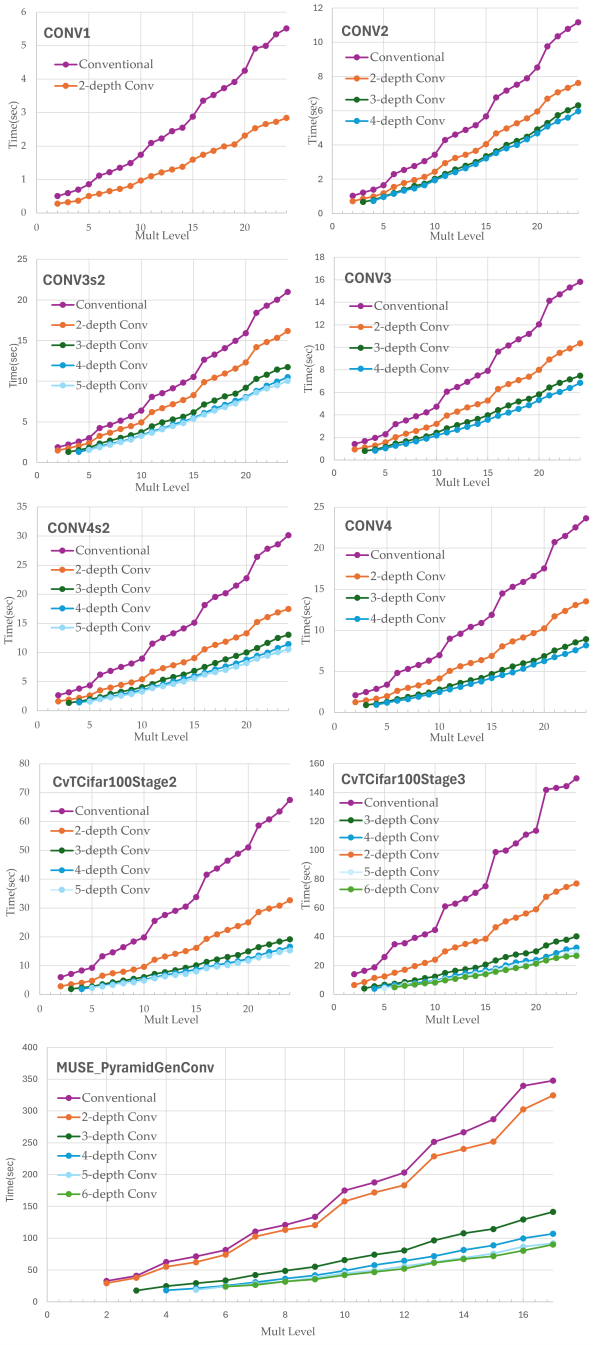

Figure 13: Latency comparison of proposed (RO-MPConv) and conventional (multiplexed parallel convolution in [8]) convolution. For MuseConv, we used a lighter CKKS parameter setting with $\log Q = [51, 41, 40, 40, 41, 40, 41, 40, 40, 40, 40, 41, 41, 40, 41, 40, 40, 40]$ and $\log P = [50, 51, 50]$ due to memory constraints.

Table 4
Comparison of the number of rotations and rotation keys used in the RotationSum and ZeroOutCombine processes of convolution. The SISOConv process, which consistently requires 8 rotations and 8 rotation keys and remains unchanged between the conventional and proposed methods, is omitted to reduce redundancy. Total = SISOConv (8) + RotationSum + ZeroOutCombine.

		No. of rotations						No. of rotation keys	
Convolution	Process	MPConv [8]	2-depth Conv	3-depth Conv	4-depth Conv	5-depth Conv	6-depth Conv	MPConv [8]	2-depth Conv
CONV1	RotationSum	4	4	-	-	-	-	2	3
	ZeroOutCombine	17	2	-	-	-	-	16	2
	Total	**29**	**14**	-	-	-	-	**26**	**13**
CONV2	RotationSum	32	24	16	14	-	-	4	6
	ZeroOutCombine	17	4	4	4	-	-	16	4
	Total	**57**	**36**	**28**	**26**	-	-	**28**	**18**
CONV3s2	RotationSum	64	64	40	32	30	-	4	8
	ZeroOutCombine	34	5	5	5	5	-	33	5
	Total	**106**	**77**	**53**	**45**	**43**	-	**45**	**21**
CONV3	RotationSum	40	32	22	18	-	-	5	7
	ZeroOutCombine	34	9	5	5	-	-	33	9
	Total	**82**	**49**	**35**	**31**	-	-	**46**	**24**
CONV4s2	RotationSum	80	64	44	36	32	-	5	8
	ZeroOutCombine	67	10	6	6	6	-	66	10
	Total	**155**	**82**	**58**	**50**	**46**	-	**79**	**26**
CONV4	RotationSum	48	48	24	20	-	-	6	9
	ZeroOutCombine	67	10	10	10	-	-	66	10
	Total	**123**	**66**	**42**	**38**	-	-	**80**	**27**
CvTConv1	RotationSum	144	120	72	57	51	-	6	10
	ZeroOutCombine	195	18	10	10	10	-	194	18
	Total	**347**	**146**	**90**	**75**	**69**	-	**208**	**36**
CvTConv2	RotationSum	384	288	156	126	108	102	8	12
	ZeroOutCombine	388	35	19	11	11	11	387	35
	Total	**780**	**331**	**183**	**145**	**127**	**121**	**403**	**55**
MuseConv	RotationSum	4608	4608	1920	1408	1184	1088	9	18
	ZeroOutCombine	512	0	0	0	0	0	511	0
	Total	**5128**	**4616**	**1928**	**1416**	**1192**	**1096**	**528**	**26**

modern, state-of-the-art AI models: two from the Convolutional Vision Transformer (CvT) [4] for CIFAR-100 inference (CvTConv1, CvTConv2) and one from the MUSE [7] model's multi-scale feature pyramid generator (MuseConv). The detailed parameters for these convolutions, such as their tensor shapes and stride values, are available in Appendix B. In the following discussion and figures, an i-depth Conv refers to an RO-MPConv variant specifically designed to consume i multiplicative levels during its execution.

It should be noted that our experiments on RO-MPConv focus on operator-level convolution evaluation rather than complete-model task accuracy. Complete-model task accuracy for MPConv-based encrypted inference has already been demonstrated in prior studies [8, 9, 10]. Since RO-MPConv is a drop-in replacement for MPConv under the same multiplexed parallel packing, the relevant numerical validation in this paper is whether RO-MPConv preserves the convolution-output approximation behavior of MPConv. Appendix C confirms that RO-MPConv and its variants produce errors on the same order as MPConv.

Figure 13 presents the latency comparison of our RO-MPConv against the conventional MPConv. It should be noted that for MuseConv, due to memory constraints caused by the massive rotation key requirements of the conventional MPConv, we conducted experiments using a lighter CKKS parameter setting with a maximum level of 18. The results of Figure 13 clearly show that RO-MPConv significantly reduces execution time across all tested configurations, with performance gains becoming more pronounced at higher multiplicative levels. For the ResNet-based convolutions, our 2-depth RO-MPConv achieves latency reductions of 20% to 45%, with further improvements up to 66% and 70% for the 3-depth and 4-depth variants, respectively. The performance gains were even more substantial for modern architectures. For the CvT operations, the 2-depth variant reduced latency by approximately 56%, while the 3-depth and 4-depth variants achieved reductions of up to 79% and

Table 5
Performance improvements from replacing MPConv with RO-MPConv in state-of-the-art HE-based models. Consistent with the experimental settings in AutoFHE and CryptoFace, these evaluations were conducted using a maximum multiplicative level of 30, with bootstrapping consuming 14 levels. The table shows the fraction of total latency attributed to convolution operations, the latency improvement of the convolution module itself, and the resulting improvement in total end-to-end inference latency. The results demonstrate that RO-MPConv provides substantial system-level performance gains, particularly for models with a high convolution latency fraction.

	Model	Convolution latency fraction	MPConv (sec)	RO-MPConv (sec)	Convolution latency improvement	Total latency improvement
MP-CNN [8]	ResNet20	15.2%	31.89	20.17	36.8%	**5.59%**
	ResNet32	14.7%	49.95	31.69	36.6%	**5.37%**
	ResNet44	14.3%	68.02	43.21	36.5%	**5.22%**
AESPA [43]	ResNet20	52.0%	120.64	75.75	37.2%	**19.36%**
	ResNet32	52.0%	190.81	119.67	37.3%	**19.37%**
	ResNet44	51.0%	260.98	163.59	37.3%	**19.02%**
AutoFHE [9]	ResNet20 (boot5)	47.7%	96.93	51.37	47.0%	**22.43%**
	ResNet20 (boot11)	30.6%	102.16	44.7	56.2%	**17.20%**
	ResNet32 (boot8)	53.0%	155.27	95.24	38.7%	**20.47%**
	ResNet32 (boot19)	25.1%	125.84	55.46	55.9%	**14.05%**
	ResNet44 (boot8)	58.1%	185.6	102.48	44.8%	**26.04%**
	ResNet44 (boot22)	28.7%	156.92	67.21	57.2%	**16.39%**
CryptoFace [10]	CryptoFace4	62.9%	305.39	199.18	34.8%	**21.89%**

81%, respectively. In the case of MuseConv, the 2-depth implementation offered a modest 10% improvement, but the 3-depth and 4-depth versions provided 62% and 72% latency reductions.

The source of these improvements is detailed in Table 4, which quantifies the reduction in the number of rotation operations and rotation keys. The most significant reduction occurs in the ZeroOutCombine process, where our novel CrossCombine subprocess replaces a number of rotations proportional to the output channels (c_o) with a much smaller number related to the parallel data count ($p_c - 1$). This reduction is particularly impactful for convolutions with a large number of channels, such as MuseConv. Although its 2-depth latency improvement was moderate, Table 4 shows that RO-MPConv reduced the number of required rotation keys for MuseConv by approximately 95%.

To demonstrate the practical impact of these algorithmic improvements on overall FHE inference latency, we integrated RO-MPConv into state-of-the-art models that originally utilized MPConv. We selected representative models from three distinct categories to showcase the broad applicability and effectiveness of our approach. These categories include: 1) **structurally modified CNNs** such as the Patch CNN (PCNN) in CryptoFace [10], which alters the standard sequential architecture to manage depth consumption in face recognition; 2) **mixed-degree CNNs** from AutoFHE [9], which employ varying degrees of polynomial approximations for ReLU across different layers; and 3) **low-degree approximations**, represented by a baseline configuration utilized in both the AutoFHE and CryptoFace studies, where the MP-CNN activation function is replaced with a low-degree polynomial from AESPA [43] to prioritize speed over accuracy.

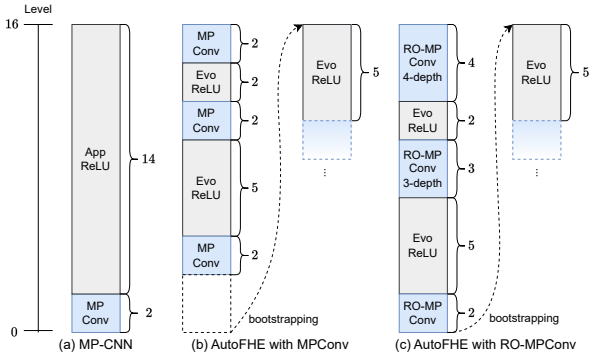

Figure 14: Comparison of operational layouts across multiplicative levels in different model architectures. (a) The baseline MP-CNN, which utilizes a deep polynomial activation (AppReLU), requiring MPConv operations to be scheduled at the lowest (fastest) levels immediately before bootstrapping. (b) AutoFHE with standard MPConv, where the use of mixed-degree activations (EvoReLU) results in convolutions being scheduled at higher levels and creates intermittent unused level gaps. (c) Our proposed application, where MPConv is replaced by RO-MPConv. By employing RO-MPConv variants with additional depth (e.g., 3-depth, 4-depth), we can efficiently schedule operations to fill these previously wasted levels, creating an optimized computational graph and reducing overall latency.

MPConv's model-agnostic nature makes it a straightforward choice for these diverse applications. However, these

advanced models often incur higher convolution-related overhead compared to the original MP-CNN, a challenge explicitly reported in studies like AutoFHE and CryptoFace. This trend is corroborated by our findings in Table 5, where the "Convolution latency fraction" for these models is notably higher than that of the baseline MP-CNN.

The primary reason for this increased overhead is illustrated in Figure 14. The original MP-CNN architecture was specifically designed to execute all MPConv operations at the lowest possible multiplicative levels, where they are fastest (Figure 14a). In contrast, models like AutoFHE introduce structural variations and mixed-degree activation functions, which force convolution operations to be executed at higher, and thus slower, levels (Figure 14b). This often leaves small, unused gaps in the level consumption between layers.

Our solution directly targets this inefficiency. As shown in Figure 14c, we not only replace MPConv with the fundamentally faster RO-MPConv but also strategically utilize RO-MPConv variants that consume additional depth. This allows us to perfectly fill the previously wasted multiplicative levels, creating a more compact and efficient computational graph. Priority for using these depth-consuming variants is given to convolutions operating at higher levels, where the potential latency reduction is greatest. This strategy is applicable when unused multiplicative levels are available before the next bootstrapping operation. Such situations commonly occur because activation functions are typically implemented through polynomial approximation, and bootstrapping is usually placed outside the polynomial evaluation rather than within it. During polynomial evaluation, the number of ciphertexts involved in the computation can increase, making intermediate bootstrapping inefficient; for instance, AutoFHE also explicitly avoids placing bootstrapping within ReLU evaluation. Thus, additional-depth RO-MPConv variants can reduce convolution latency by utilizing these otherwise idle levels without introducing extra burden.

By applying this substitution strategy, we achieved significant performance gains. As detailed in Table 5, the latency of the convolution modules themselves was reduced by up to **57.2%**. More importantly, this translated into a substantial reduction in the total end-to-end model latency by up to **26.04%**. While RO-MPConv also reduces convolution latency in the baseline MP-CNN, the overall improvement is modest because the convolution fraction was already small. However, in advanced models where architectural trade-offs pushed convolution to higher levels and increased its latency contribution, RO-MPConv proved to be highly effective at reducing the entire model's inference time.

As an additional contribution, this paper introduces a parallel Baby-Step Giant-Step (BSGS) matrix-vector multiplication algorithm designed to accelerate other critical linear operations, such as fully connected layers. The effectiveness of this method is demonstrated in Figure 15, which compares its latency against the conventional BSGS diagonal method [12] under various scenarios.

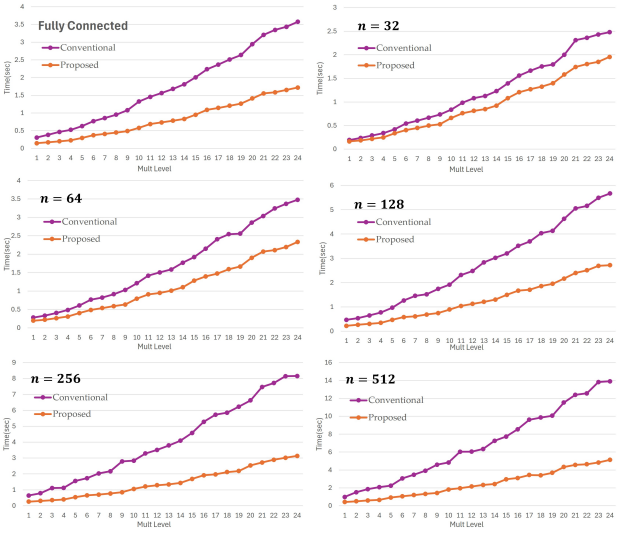

Figure 15: Latency comparison of parallel BSGS matrix-vector multiplication and BSGS diagonal method.

Table 6
Comparison of the number of rotations and plaintexts required for matrix-vector multiplication between the BSGS diagonal method and the proposed ParBSGSMatVecMul algorithm. The scenario is same as Figure 15. Assume matrix size as $n \times n$, and vector size as $n \times 1$.

Scenario	Criteria	BSGS diagonal method	Proposed
Fully connected layer in [8]	# Rotation	15	9
	# Plaintext	64	2
$n = 32$	# Rotation	11	10
	# Plaintext	32	2
$n = 64$	# Rotation	15	12
	# Plaintext	64	2
$n = 128$	# Rotation	23	14
	# Plaintext	128	2
$n = 256$	# Rotation	31	16
	# Plaintext	256	4
$n = 512$	# Rotation	47	26
	# Plaintext	512	16

First, when applied to the fully connected layer configuration from [8] (where $n = 64$, $p_i = 8$, and $n_t = 32768$), our proposed method achieves a nearly 2× speedup across most multiplicative levels. Second, to assess its general applicability, we evaluated its performance on $n \times n$ matrix-vector multiplications for various n, setting $p_i = 1$ to represent cases without pre-existing parallel data. Even in this more challenging setting, our algorithm achieved a substantial latency reduction of up to 69%.

This performance gain stems from a fundamental reduction in the required number of rotation operations. For a

matrix of size $n = n_1 n_2$, the conventional BSGS diagonal method requires approximately $n_1 + n_2 - 1$ rotations. In contrast, our parallel approach requires only $2\log_2(n_2) + n_1 - \log_2 p_i$ rotations. Due to the logarithmic term in our formula, the efficiency advantage of our method becomes more pronounced as the matrix size n increases. This theoretical property is clearly reflected in the performance trends shown in Figure 15. Table 6 provides a detailed breakdown of the number of rotations and plaintexts required for each scenario, further quantifying the improvements.

We note that, in the CNN-based models evaluated in this paper, matrix-vector multiplication appears only in the final fully connected layer. Therefore, even a substantial reduction in matrix-vector multiplication latency has only a limited impact on the total end-to-end inference latency. Nevertheless, the proposed parallel BSGS method can be more beneficial for future HE-based models in which matrix-vector multiplication accounts for a larger fraction of the overall computation.

8. Implementation Considerations

Our experimental results have demonstrated that RO-MPConv is a versatile optimization, successfully enhancing not only the ResNet but also a range of modern SOTA architectures like CvT and MUSE. However, similar to all algorithms built upon FHE, its implementation operates within the fundamental constraint of a finite ciphertext length. This length is a critical system parameter, as arbitrarily increasing it to accommodate larger inputs would introduce prohibitive computational latency.

Consequently, for any FHE-based algorithm, a standard approach is required when input data dimensions exceed the capacity of a single ciphertext: partitioning the data across multiple ciphertexts. To provide clear guidance for implementation, we explicitly define the operational boundaries of our algorithm, preempting potential integration challenges.

The following equations formalize the precise conditions under which the proposed RO-MPConv can be applied directly without requiring such data partitioning.

$$2^{\lceil \log_2 h_i \rceil + \lceil \log_2 w_i \rceil} c_i \leq n_t \tag{9}$$

$$2^x c_o \leq 2^{\lceil \log_2 c_i \rceil} p_i \tag{10}$$

Equation 9 expresses the constraint that the input size of the convolution must not exceed the length of the ciphertext. Equation 10 addresses whether the ciphertext length is sufficient to generate the convolution output. Specifically, p_i represents the number of identical data elements in the ciphertext, and each data element can hold $2^{\lceil \log_2 c_i \rceil}$ channels of information. If the required number of output channels exceeds this capacity, the ciphertext length would be insufficient. Thus, the relationship $c_o \leq 2^{\lceil \log_2 c_i \rceil} p_i$ is defined. Moreover, when considering the correlation between the RotationSum process and the CrossCombine process in Section 4.2, the number of data blocks combined during CrossCombine increases by a factor of 2^x. This leads to the final equation $2^x c_o \leq 2^{\lceil \log_2 c_i \rceil} p_i$. In addition, by rearranging Equation 10 with respect to x, the upper bound for x can be directly obtained as $x \leq \log_2 \left(2^{\lceil \log_2 c_i \rceil} p_i / c_o\right)$.

If a target convolution does not satisfy these conditions, the input or output tensors must be partitioned before applying our algorithm to each segment. This delineation provides a practical guide for integrating RO-MPConv into future HE-based systems.

9. Conclusion and Future Works

In this paper, we addressed the critical performance bottlenecks that hinder the practical deployment of homomorphic encryption-based deep learning: high latency and large key sizes, particularly in convolution operations. We introduced Rotation-Optimized Multiplexed Parallel Convolution (RO-MPConv), a novel algorithm that fundamentally redesigns the state-of-the-art MPConv to minimize the number of costly rotation operations. We further enhanced its flexibility by introducing variants that consume additional multiplicative depth, offering a strategic trade-off for even greater latency reductions.

Our experimental results validate the effectiveness of these contributions. RO-MPConv not only significantly reduces the latency of standalone convolution operations but also effectively reduces the overall inference latency of existing state-of-the-art models by up to 26% through direct substitution. This demonstrates that optimizing core HE primitives translates into substantial system-level gains. Furthermore, by combining our proposed small-level key system with a hierarchical structure, we reduced the rotation key transmission size by a factor of 29, directly tackling the communication overhead in client-server environments. As a complementary contribution, we also presented a parallel BSGS matrix-vector multiplication method, which accelerates other common linear operations by leveraging the data packing characteristics inherent in many HE schemes.

Collectively, our work provides a suite of optimizations that significantly lowers the computational and communication barriers for privacy-preserving machine learning. By making fundamental operations faster and lighter, we enhance the practical feasibility of deploying complex, deep neural networks over encrypted data. Building on this foundation, our future work will focus on utilizing these optimized building blocks to construct fully homomorphic implementations of modern, convolution-integrated architectures, such as advanced CNN variants, convolution-integrated Vision Transformers, and State-Space Models, further expanding the horizons of secure and private AI applications.

A. RO-MPConv Blueprints

This section provides examples of the **blueprint**s and further details regarding their implementation. Specifically,

Algorithm 4 KernelBP Initialization

Require: Convolution parameters $(C_o, k_i, k_o, p_i, p_o, q)$
Ensure: KernelBP

KernelBP $\leftarrow \emptyset$
for $r = 0, \dots, q-1$ **do**
  if $k_i = k_o$ **then**
    for $j = 0, \dots, p_i - 1$ **do**
      $u \leftarrow r + \frac{C_o}{p_o} j$
      if $u \geq C_o$ **then**
        $u \leftarrow 2\lfloor u/C_o \rfloor + (u \bmod C_o)$
      end if
      KernelBP$[r, j] \leftarrow u$
    end for
  else
    $s \leftarrow (r \bmod k_i) + k_o \left\lfloor \frac{r \bmod k_i^2}{k_i} \right\rfloor + k_o^2 \left\lfloor \frac{r}{k_i^2} \right\rfloor$
    if $s \geq C_o$ **then**
      $s \leftarrow 2\lfloor s/C_o \rfloor + (s \bmod C_o)$
    end if
    for $j = 0, \dots, p_i - 1$ **do**
      KernelBP$[r, j] \leftarrow s + k_i(j \bmod 2) + k_o k_i \lfloor j/2 \rfloor$
    end for
  end if
end for
return KernelBP

Algorithm 5 CrossCombineBP Initialization

Require: Convolution parameters (k_o, p_i, W_o, H_o, N) and KernelBP
Ensure: CrossCombineBP

$B \leftarrow N/p_i$
CrossCombineBP $\leftarrow [0]$
for $j = 0, \dots, p_i - 1$ **do**
  $c_j \leftarrow$ KernelBP$[0, j]$
  $\ell_{\text{in}}(j) \leftarrow jB$
  $\ell_{\text{out}}(c_j) \leftarrow (c_j \bmod k_o) + k_o W_o \left\lfloor \frac{c_j \bmod k_o^2}{k_o} \right\rfloor + k_o^2 W_o H_o \left\lfloor \frac{c_j}{k_o^2} \right\rfloor$
  CrossCombineBP $\leftarrow$ CrossCombineBP $\| [\ell_{\text{in}}(j) - \ell_{\text{out}}(c_j)]$
end for
CrossCombineBP$[0] \leftarrow |$CrossCombineBP$| - 1$
return CrossCombineBP

Algorithm 6 RotationSumBP Initialization

Require: Convolution parameters (C_i, k_i, q, W_i, H_i)
Ensure: RotationSumBP

$\mathcal{R} \leftarrow \emptyset$
RotationSumBP $\leftarrow \emptyset$
if $k_i > 1$ **then**
  for $a = 0, \dots, \log_2 k_i - 1$ **do**
    $\mathcal{R} \leftarrow \mathcal{R} \cup \{2^a\}$
  end for
  for $a = 0, \dots, \log_2 k_i - 1$ **do**
    $\mathcal{R} \leftarrow \mathcal{R} \cup \{2^a W_i k_i\}$
  end for
end if
$D \leftarrow W_i H_i k_i^2$
$L_c \leftarrow \lceil \log_2 \left(\lfloor C_i/k_i^2 \rfloor \right) \rceil$
for $a = 0, \dots, L_c - 1$ **do**
  $\mathcal{R} \leftarrow \mathcal{R} \cup \{D2^a\}$
end for
$\tau \leftarrow \lceil \log_2 q \rceil$
Mode$_2 \leftarrow [2, q]$
for $a = 0, \dots, \tau - 1$ **do**
  Mode$_2 \leftarrow$ Mode$_2 \| [\mathcal{R}_a]$
end for
Mode$_0 \leftarrow \emptyset$
for $a = \tau, \dots, |\mathcal{R}| - 1$ **do**
  Mode$_0 \leftarrow$ Mode$_0 \| [[0, \mathcal{R}_a]]$
end for
RotationSumBP $\leftarrow [[|\text{Mode}_0| + 1]] \| \text{Mode}_0 \| [\text{Mode}_2]$
return RotationSumBP

Tables 7–11 present a representative subset of the **blueprint**s used for the RO-MPConv configurations evaluated in our ResNet architectures. Because exhaustively listing all cases is impractical, the complete set of **blueprint**s for all evaluated scenarios is available in our public GitHub repository.

These **blueprint**s are explicitly optimized to minimize computational latency by leveraging the structural correlation between the RotationSum and CrossCombine processes. Due to the heuristic nature of the RO-MPConv algorithm, the proposed **blueprint**s are not strictly unique for a given convolution scenario. While alternative valid configurations exist, they are empirically unlikely to yield superior latency reductions. Alternative valid **blueprint**s can also be tested in our repository.

We also provide a practical guide for constructing and refining these **blueprint**s. First, initialize the **blueprint**s using Algorithms 4–6. While this initialization corresponds to the baseline RO-MPConv layout described in Section 4.1, it does not necessarily produce the optimal configuration. Following this initialization, the **blueprint**s can be further refined by exploiting the correlation between the RotationSum and CrossCombine processes, as discussed in Section 4.2. Specifically, the overall execution time can be optimized by trading a reduction of rotations in the RotationSum stage for a slight increase in the CrossCombine stage, ultimately decreasing the total number of rotations. Furthermore, if additional multiplicative depth is permissible in a given setting, the **blueprint**s can be further optimized following Section 4.3 by merging ciphertexts that share identical rotation shifts during the RotationSum process, thereby reducing the total execution time. It is worth noting that the CONV4 blueprint shown in Table 10 and Table 11 can be directly generated by these initialization algorithms, as it does not exploit the RotationSum–CrossCombine correlation.

Regarding the notation used in the **blueprint** algorithms, the x-th element of a 1-dimensional **blueprint** A is denoted

as $A[x]$, and the y-th element of a 2-dimensional **blueprint** $B[x]$ is denoted as $B[x][y]$. All **blueprint** indices are zero-based (e.g., the first element of A is $A[0]$). Additionally, $\emptyset$ denotes an empty list, $|\cdot|$ denotes the length of a list, and $\|$ denotes list concatenation. For example, if $A = [a_0, a_1]$ and $B = [b_0, b_1]$, then $A\|B = [a_0, a_1, b_0, b_1]$. For simplicity, KernelBP remains consistent regardless of the multiplicative depth consumed by the convolution.

B. Parameters

Various parameters such as $h_i, h_o, w_i, w_o, c_i, c_o, f_h, f_w, s, k_i, k_o, t_i, t_o, p_i, p_o, p_c$ and q are defined differently depending on the shape of each convolution. Table 12 shows the values of these parameters used in each convolution shape. These parameters are also defined to represent the MPConv described in [8]. To avoid confusion, a notation similar to that in [8] is used.

Parameters w_i, h_i, and c_i respectively represent the width, height, and channel numbers of the three-dimensional input data of convolution. Similarly, w_o, h_o, and c_o respectively represent the width, height, and channel numbers of the three-dimensional output data of convolution. f_h, f_w are height and width numbers of kernel data, and s represents the stride value of convolution. k_i, k_o are gaps of the input and output ciphertexts of convolution, respectively. Other variables can be defined as $t_i = \lceil \frac{c_i}{k_i^2} \rceil$, $t_o = \lceil \frac{c_o}{k_o^2} \rceil$, $p_i = 2^{\lfloor \log_2\left(\frac{n_t}{k_i^2 h_i w_i t_i}\right) \rfloor}$, $p_o = 2^{\lfloor \log_2\left(\frac{n_t}{k_o^2 h_o w_o t_o}\right) \rfloor}$, and $q = \lceil \frac{c_o}{p_i} \rceil$. Specifically, p_i and p_o represent the number of identical parallel data blocks within the input and output ciphertexts. Finally, $p_c = \max\left(s^2, p_i/\lceil p_i/p_o \rceil\right)$ is a new parameter proposed to denote the essential number of parallel data blocks to be merged in CrossCombine.

Table 7
5-depth Conv **blueprint**.

Convolution	RotationSumBP	CrossCombineBP
CONV3s2	[[4], [1, 2, 1024], [1, 2, 2048], [1, 2, 4096], [2, 2, 8192]]	[4, 0, 8191, 16352, 24543]
CONV4s2	[[5], [1, 2, 1], [1, 2, 32], [1, 2, 1024], [0, 4096], [2, 2, 2048]]	[4, 0, 8190, 16320, 24510]

Table 8
4-depth Conv **blueprint**.

Convolution	RotationSumBP	CrossCombineBP
CONV2	[[3], [1, 2, 1024], [1, 2, 2048], [2, 2, 4096]]	[4, 0, 8192, 8192, 16384]
CONV3s2	[[3], [1, 2, 1024], [1, 2, 2048], [2, 4, 4096, 8192]]	[4, 0, 8191, 16352, 24543]
CONV3	[[5], [1, 2, 1], [1, 2, 32], [0, 2048], [0, 4096], [2, 2, 1024]]	[4, 0, 6144, 12288, 18432]
CONV4s2	[[4], [1, 2, 1], [1, 2, 32], [0, 4096], [2, 4, 1024, 2048]]	[4, 0, 8190, 16320, 24510]
CONV4	[[6], [1, 2, 1], [1, 2, 2], [0, 64], [0, 1024], [0, 2048], [2, 2, 32]]	[8, 0, 4032, 7168, 11200, 14336, 18368, 21504, 25536]

Table 9
3-depth Conv **blueprint**.

Convolution	RotationSumBP	CrossCombineBP
CONV2	[[2], [1, 2, 1024], [2, 4, 2048, 4096]]	[4, 0, 8192, 8192, 16384]
CONV3s2	[[2], [1, 4, 1024, 2048], [2, 4, 4096, 8192]]	[4, 0, 8191, 16352, 24543]
CONV3	[[4], [1, 4, 1, 32], [0, 2048], [0, 4096], [2, 2, 1024]]	[4, 0, 6144, 12288, 18432]
CONV4s2	[[3], [1, 4, 1, 32], [0, 4096], [2, 4, 1024, 2048]]	[4, 0, 8190, 16320, 24510]
CONV4	[[5], [1, 4, 1, 2], [0, 64], [0, 1024], [0, 2048], [2, 2, 32]]	[8, 0, 4032, 7168, 11200, 14336, 18368, 21504, 25536]

Table 10
2-depth Conv **blueprint**.

Convolution	RotationSumBP	CrossCombineBP
CONV1	[[2], [0, 2048], [2, 2, 1024]]	[2, 0, 14336]
CONV2	[[1], [2, 8, 1024, 2048, 4096]]	[4, 0, 8192, 8192, 16384]
CONV3s2	[[1], [2, 16, 1024, 2048, 4096, 8192]]	[4, 0, 8191, 16352, 24543]
CONV3	[[2], [0, 2048], [2, 8, 1, 32, 1024]]	[8, 0, 4096, 6144, 10240, 12288, 16384, 18432, 22528]
CONV4s2	[[1], [2, 16, 1, 32, 1024, 2048]]	[8, 0, 4096, 8190, 12286, 16320, 20416, 24510, 28606]
CONV4	[[4], [0, 64], [0, 1024], [0, 2048], [2, 8, 1, 2, 32]]	[8, 0, 4032, 7168, 11200, 14336, 18368, 21504, 25536]

Table 11
KernelBP **blueprint**.

Convolution	KernelBP
CONV1	[[0, 4, 8, 12, 2, 6, 10, 14], [1, 5, 9, 13, 3, 7, 11, 15]]
CONV2	[[0, 8], [1, 9], [2, 10], [3, 11], [4, 12], [5, 13], [6, 14], [7, 15]]
CONV3s2	[[0, 2], [4, 6], [8, 10], [12, 14], [16, 18], [20, 22], [24, 26], [28, 30], [1, 3], [5, 7], [9, 11], [13, 15], [17, 19], [21, 23], [25, 27], [29, 31]]
CONV3	[[0, 8, 16, 24], [1, 9, 17, 25], [2, 10, 18, 26], [3, 11, 19, 27], [4, 12, 20, 28], [5, 13, 21, 29], [6, 14, 22, 30], [7, 15, 23, 31]]
CONV4s2	[[0, 2, 8, 10], [1, 3, 9, 11], [4, 6, 12, 14], [5, 7, 13, 15], [16, 18, 24, 26], [17, 19, 25, 27], [20, 22, 28, 30], [21, 23, 29, 31], [32, 34, 40, 42], [33, 35, 41, 43], [36, 38, 44, 46], [37, 39, 45, 47], [48, 50, 56, 58], [49, 51, 57, 59], [52, 54, 60, 62], [53, 55, 61, 63]]
CONV4	[[0, 8, 16, 24, 32, 40, 48, 56], [1, 9, 17, 25, 33, 41, 49, 57], [2, 10, 18, 26, 34, 42, 50, 58], [3, 11, 19, 27, 35, 43, 51, 59], [4, 12, 20, 28, 36, 44, 52, 60], [5, 13, 21, 29, 37, 45, 53, 61], [6, 14, 22, 30, 38, 46, 54, 62], [7, 15, 23, 31, 39, 47, 55, 63]]

Table 12
Parameters that are used in MPConv and RO-MPConv.

Component	CONV1	CONV2	CONV3s2	CONV3	CONV4s2	CONV4	CvTConv1	CvTConv2	MuseConv
f_h	3	3	3	3	3	3	3	3	3
f_w	3	3	3	3	3	3	3	3	3
s	1	1	2	1	2	1	2	2	1
h_i	32	32	32	16	16	8	8	4	8
h_o	32	32	16	16	8	8	4	2	8
w_i	32	32	32	16	16	8	8	4	8
w_o	32	32	16	16	8	8	4	2	8
c_i	3	16	16	32	32	64	64	192	512
c_o	16	16	32	32	64	64	192	384	512
k_i	1	1	1	2	2	4	1	2	1
k_o	1	1	2	2	4	4	2	4	1
t_i	3	16	16	8	8	4	64	48	512
t_o	16	16	8	8	4	4	48	24	512
p_i	8	2	2	4	4	8	8	8	1
p_o	2	2	4	4	8	8	8	16	1
p_c	2	2	4	4	4	8	8	8	1
q	2	8	16	8	16	8	24	48	512

Table 13
Maximum absolute error $\|Y_c - Y_p\|_\infty$ between plaintext convolution outputs Y_p and FHE convolution outputs Y_c for MPConv and RO-MPConv variants. For MuseConv, we used a lighter CKKS parameter setting with $\log Q = [51, 41, 40, 40, 41, 40, 41, 40, 40, 40, 40, 41, 41, 40, 41, 40, 40, 40]$ and $\log P = [50, 51, 50]$ due to memory constraints.

Convolution	MPConv	RO-MPConv					
		2-depth Conv	3-depth Conv	4-depth Conv	5-depth Conv	6-depth Conv	7-depth Conv
CONV1	1.790e-08	2.195e-08	-	-	-	-	-
CONV2	3.864e-08	4.310e-08	3.749e-08	3.655e-08	-	-	-
CONV3s2	3.896e-08	4.067e-08	3.347e-08	3.145e-08	3.224e-08	-	-
CONV3	5.275e-08	5.639e-08	5.839e-08	4.702e-08	-	-	-
CONV4s2	5.152e-08	5.095e-08	4.907e-08	4.906e-08	4.651e-08	-	-
CONV4	6.954e-08	6.505e-08	7.402e-08	6.136e-08	-	-	-
CvTConv1	6.628e-08	6.470e-08	6.710e-08	6.924e-08	6.902e-08	-	-
CvTConv2	1.047e-07	9.520e-08	8.922e-08	9.679e-08	1.082e-07	8.500e-08	-
MuseConv	1.569e-05	1.665e-05	1.461e-05	1.547e-05	1.483e-05	1.379e-05	1.232e-05

C. Convolution-Output Consistency

This section reports the operator-level numerical consistency of RO-MPConv. Let Y_p denote the output of the corresponding plaintext convolution and Y_c denote the decrypted output of the FHE convolution. We measure the maximum absolute error $\|Y_c - Y_p\|_\infty$ for MPConv and RO-MPConv. Table 13 shows that the errors of RO-MPConv remain on the same order as those of MPConv across all tested convolution settings. This trend also holds for RO-MPConv variants that consume additional multiplicative depth, indicating that the proposed convolution does not introduce additional numerical degradation beyond the expected CKKS approximation error at the convolution-output level.

CRediT authorship contribution statement

Byeongseo Min: Writing – review & editing, Writing – original draft, Visualization, Software, Methodology, Investigation, Formal analysis. **Joon-Woo Lee:** Writing – review & editing, Validation, Supervision, Resources, Funding acquisition, Conceptualization.

Data Availability Statement

The source code developed for this study is openly available in a GitHub repository at https://github.com/byeongseomin51/RO-MPConv.git.

References

[1] R. Xu, N. Baracaldo, J. Joshi, Privacy-preserving machine learning: Methods, challenges and directions, arXiv preprint arXiv:2108.04417 (2021).

[2] J. H. Cheon, K. Han, A. Kim, M. Kim, Y. Song, A full RNS variant of approximate homomorphic encryption, in: Selected Areas in Cryptography–SAC 2018, Springer, 2019, pp. 347–368.

[3] J. H. Cheon, A. Kim, M. Kim, Y. Song, Homomorphic encryption for arithmetic of approximate numbers, in: Advances in Cryptology–ASIACRYPT 2017, Springer, 2017, pp. 409–437.

[4] H. Wu, B. Xiao, N. Codella, M. Liu, X. Dai, L. Yuan, L. Zhang, CvT: Introducing convolutions to vision transformers, in: Proceedings of the IEEE/CVF International Conference on Computer Vision, 2021, pp. 22–31.

[5] A. Gulati, J. Qin, C.-C. Chiu, N. Parmar, Y. Zhang, J. Yu, W. Han, S. Wang, Z. Zhang, Y. Wu, et al., Conformer: Convolution-augmented transformer for speech recognition, arXiv preprint arXiv:2005.08100 (2020).

[6] H. Yan, Z. Li, W. Li, C. Wang, M. Wu, C. Zhang, ConTNet: Why not use convolution and transformer at the same time?, arXiv preprint arXiv:2104.13497 (2021).

[7] H. Tang, M. Cao, J. Huang, R. Liu, P. Jin, G. Li, X. Liang, MUSE: Mamba is efficient multi-scale learner for text-video retrieval, in: Proceedings of the AAAI Conference on Artificial Intelligence, volume 39, 2025, pp. 7238–7246.

[8] E. Lee, J.-W. Lee, J. Lee, Y.-S. Kim, Y. Kim, J.-S. No, W. Choi, Low-complexity deep convolutional neural networks on fully homomorphic encryption using multiplexed parallel convolutions, in: Proceedings of the International Conference on Machine Learning, PMLR, 2022, pp. 12403–12422.

[9] W. Ao, V. N. Boddeti, AutoFHE: Automated adaption of CNNs for efficient evaluation over FHE, in: 33rd USENIX Security Symposium, 2024, pp. 2173–2190.

[10] W. Ao, V. N. Boddeti, CryptoFace: End-to-end encrypted face recognition, in: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2025, pp. 19197–19206.

[11] J.-W. Lee, E. Lee, Y.-S. Kim, J.-S. No, Rotation key reduction for client-server systems of deep neural network on fully homomorphic encryption, in: Proceedings of the International Conference on the Theory and Application of Cryptology and Information Security, Springer, 2023, pp. 36–68.

[12] S. Halevi, V. Shoup, Algorithms in HElib, in: Advances in Cryptology–CRYPTO 2014, Springer, 2014, pp. 554–571.

[13] J. Wan, D. Li, J. Fang, Z. L. Jiang, LPFHE: Low-complexity polynomial CNNs for secure inference over FHE, in: Proceedings of the European Symposium on Research in Computer Security, Springer, 2024, pp. 403–423.

[14] N. Samardzic, D. Sanchez, BitPacker: Enabling high arithmetic efficiency in fully homomorphic encryption accelerators, in: Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, volume 2, 2024, pp. 137–150.

[15] R. Agrawal, A. Chandrakasan, A. Joshi, HEAP: A fully homomorphic encryption accelerator with parallelized bootstrapping, in: Proceedings of the 51st ACM/IEEE Annual International Symposium on Computer Architecture, 2024, pp. 756–769.

[16] X. Deng, S. Fan, Z. Hu, Z. Tian, Z. Yang, J. Yu, D. Cao, D. Meng, R. Hou, M. Li, et al., Trinity: A general-purpose FHE accelerator,

in: Proceedings of the 57th IEEE/ACM International Symposium on Microarchitecture, 2024, pp. 338–351.
[17] S. Cheon, Y. Lee, D. Kim, J. M. Lee, S. Jung, T. Kim, D. Lee, H. Kim, DaCapo: Automatic bootstrapping management for efficient fully homomorphic encryption, in: Proceedings of the 33rd USENIX Security Symposium, 2024, pp. 6993–7010.
[18] D. Kim, J. Park, J. Kim, S. Kim, J. H. Ahn, HyPHEN: A hybrid packing method and its optimizations for homomorphic encryption-based neural networks, IEEE Access 12 (2024) 3024–3038.
[19] H. Peng, R. Ran, Y. Luo, J. Zhao, S. Huang, K. Thorat, T. Geng, C. Wang, X. Xu, W. Wen, C. Ding, LinGCN: Structural linearized graph convolutional network for homomorphically encrypted inference, in: Advances in Neural Information Processing Systems, volume 36, 2023, pp. 20679–20694.
[20] R. Ran, X. Luo, W. Wang, T. Liu, G. Quan, X. Xu, C. Ding, W. Wen, SpENCNN: Orchestrating encoding and sparsity for fast homomorphically encrypted neural network inference, in: Proceedings of the International Conference on Machine Learning, PMLR, 2023, pp. 28718–28728.
[21] B. Yalavarthi, C. Jutla, N. Ratha, Efficient convolution operator in FHE using summed-area table, in: Proceedings of the International Conference on Pattern Recognition, Springer, 2024, pp. 65–79.
[22] Q. Lou, W.-j. Lu, C. Hong, L. Jiang, Falcon: Fast spectral inference on encrypted data, Advances in Neural Information Processing Systems 33 (2020) 2364–2374.
[23] C. Juvekar, V. Vaikuntanathan, A. Chandrakasan, {GAZELLE}: A low latency framework for secure neural network inference, in: 27th USENIX security symposium (USENIX security 18), 2018, pp. 1651–1669.
[24] Y. Lin, T. Zhang, Y. Mao, S. Zhong, CrossNet: A low-latency MLaaS framework for privacy-preserving neural network inference on resource-limited devices, IEEE Transactions on Dependable and Secure Computing 22 (2025) 1265–1280.
[25] D. Kim, C. Guyot, Optimized privacy-preserving CNN inference with fully homomorphic encryption, IEEE Transactions on Information Forensics and Security 18 (2023) 2175–2187.
[26] J. H. Ju, J. Park, J. Kim, M. Kang, D. Kim, J. H. Cheon, J. H. Ahn, 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, 2024, pp. 4361–4375.
[27] A. Ebel, K. Garimella, B. Reagen, Orion: A fully homomorphic encryption framework for deep learning, in: Proceedings of the 30th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2, 2025, pp. 734–749.
[28] R. Dathathri, O. Saarikivi, H. Chen, K. Laine, K. Lauter, S. Maleki, M. Musuvathi, T. Mytkowicz, Chet: an optimizing compiler for fully-homomorphic neural-network inferencing, in: Proceedings of the 40th ACM SIGPLAN conference on programming language design and implementation, 2019, pp. 142–156.
[29] DESILO Inc., DESILO FHE library, 2025. URL: https://desilo.ai.
[30] SEAL, Microsoft SEAL (release 4.3), https://github.com/Microsoft/SEAL, 2026. Microsoft Research, Redmond, WA.
[31] B. Li, D. Micciancio, On the security of homomorphic encryption on approximate numbers, in: Proceedings of the Annual International Conference on the Theory and Applications of Cryptographic Techniques, Springer, 2021, pp. 648–677.
[32] B. Li, D. Micciancio, M. Schultz-Wu, J. Sorrell, Securing approximate homomorphic encryption using differential privacy, in: Proceedings of the Annual International Cryptology Conference, Springer, 2022, pp. 560–589.
[33] J.-W. Lee, H. Kang, Y. Lee, W. Choi, J. Eom, M. Deryabin, E. Lee, J. Lee, D. Yoo, Y.-S. Kim, et al., Privacy-preserving machine learning with fully homomorphic encryption for deep neural network, IEEE Access 10 (2022) 30039–30054.
[34] H. Chen, I. Chillotti, Y. Song, Improved bootstrapping for approximate homomorphic encryption, in: Proceedings of the Annual International Conference on the Theory and Applications of Cryptographic Techniques, Springer, 2019, pp. 34–54.
[35] E. Lee, J.-W. Lee, J.-S. No, Y.-S. Kim, Minimax approximation of sign function by composite polynomial for homomorphic comparison, IEEE Transactions on Dependable and Secure Computing 19 (2021) 3711–3727.
[36] X. Jiang, M. Kim, K. Lauter, Y. Song, Secure outsourced matrix computation and application to neural networks, in: Proceedings of the ACM SIGSAC Conference on Computer and Communications Security, 2018, pp. 1209–1222.
[37] S. Wang, H. Huang, Secure outsourced computation of multiple matrix multiplication based on fully homomorphic encryption, KSII Transactions on Internet and Information Systems 13 (2019) 5616–5630.
[38] W.-j. Lu, S. Kawasaki, J. Sakuma, Using fully homomorphic encryption for statistical analysis of categorical, ordinal and numerical data, Cryptology ePrint Archive (2016).
[39] H. Huang, H. Zong, Secure matrix multiplication based on fully homomorphic encryption, The Journal of Supercomputing 79 (2023) 5064–5085.
[40] A. Krizhevsky, G. Hinton, Learning Multiple Layers of Features from Tiny Images, Technical Report UTML TR 2009, University of Toronto, Toronto, ON, Canada, 2009.
[41] J. H. Cheon, K. Han, A. Kim, M. Kim, Y. Song, Bootstrapping for approximate homomorphic encryption, in: Advances in Cryptology–EUROCRYPT 2018, Springer, 2018, pp. 360–384.
[42] EPFL-LDS, Tune Insight SA, Lattigo v5, https://github.com/tuneinsight/lattigo, 2023.
[43] J. Park, M. J. Kim, W. Jung, J. H. Ahn, AESPA: Accuracy-preserving low-degree polynomial activation for fast private inference, arXiv preprint arXiv:2201.06699 (2022).



Byeongseo Min Byeongseo Min received the B.S. degree from the School of Computer Science and Engineering, Chung-Ang University, Seoul, South Korea, in 2025. He is currently pursuing the M.S. degree in the Department of Electrical Engineering at Pohang University of Science and Technology (POSTECH), Pohang, South Korea. His research interests include privacy-preserving machine learning and homomorphic encryption.



Joon-Woo Lee Joon-Woo Lee received the B.S. and Ph.D. degrees in Electrical and Computer Engineering from Seoul National University, South Korea, in 2016 and 2022, respectively. He is currently an assistant professor in the Department of Computer Science and Engineering at Chung-Ang University, South Korea. His research interests include privacy- preserving machine learning, homomorphic encryption, post-quantum cryptography, and multi-party computation.