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

Byeong-Seo Min, and Joon-Woo Lee

Abstract—In the field of Artificial Intelligence (AI), convolution operations have primarily been used in Convolutional Neural Networks (CNNs). However, its utility is increasing with the appearance of convolution integrated transformers or state space models where convolution is a constituent element. In the field of private AI, generalized algorithm, multiplexed parallel convolution was recently proposed to implement CNNs based on the Homomorphic Encryption scheme, residue number system variant Cheon-Kim-Kim-Song. Multiplexed parallel convolution is highly applicable, but its usage has been partly limited due to requiring many rotation operations. In this paper, we propose *rotation optimized convolution*, which reduces the rotation required for multiplexed parallel convolution, thus lowering latency, enhancing usability, and additionally decreasing the required rotation key. We additionally reduce the size of rotation keys by applying the hierarchical rotation key system, and our proposed small level key system. We also propose a new form of matrix-vector multiplication called *Parallel Baby-Step Giant-Step matrix-vector multiplication* which also reduces the number of rotations. In our experiment case, rotation optimized convolution achieved a maximum 70% reduction in execution time and 29× reduction for rotation keys using our method. Also, our proposed matrix-vector multiplication method achieved a reduction of execution time by up to 64%.

Index Terms—Cheon-Kim-Kim-Song (CKKS) schemes, Convolution, Fully homomorphic encryption, Hierarchical rotation key system, Private artificial intelligence

1 Introduction

As artificial intelligence services advance in various industries, but the issue of data privacy persists. Utilizing cloud-based AI systems requires clients to disclose their data to cloud servers, raising concerns about data privacy infringement. To address this, recent research has been consistently focusing on homomorphic encryption-based private AI models. Homomorphic encryption allows computations on encrypted data, enabling cloud servers to process encrypted service results without knowing the underlying data. However, designing efficient and high-performance secure AI models requires consideration of the unique characteristics of homomorphic encryption operations, which differ significantly from conventional computations. Thus, research on designing homomorphic encryption-based secure AI models has been ongoing. Various efforts persist, such as modifying AI models to align with the properties of homomorphic encryption [1], [2], [3], [4], [5] or focuses on devising efficient computation methods using homomorphic encryption without altering AI models [6], [7], [8], [9], [10], [11], [12], [13].

In this paper, our focus will be on convolution operations. Convolution has primarily been utilized in AI model implementations, notably in CNNs. However, recent research has seen efforts to integrate convolution operations into newly developed AI models like transformers [14], [15], [16]. Also, State Space Model (SSM) in [17], which have recently garnered attention, convolution itself is a part of the model. Consequently, the usability of convolution operations in developing AI models is increasing. Therefore, optimization of convolution operations is crucial in the current and future development of AI models. Recently, in [18], CNN was implemented using the well-known Homomorphic Encryption(HE) scheme, residue number system variant Cheon-Kim-Kim-Song (RNS-CKKS). The multiplexed parallel convolution in [18], integrated as part of this CNN structure, aims to enable the implementation of high-performing neural networks like ResNet without modifying the AI model itself. Designed to be applicable in various scenarios such as ResNet's strided convolution, multiplexed parallel convolution can be seen as a HE convolution technique that can be generally applied in neural networks beyond just CNNs like ResNet.

However, currently, multiplexed parallel convolution suffers from the drawback of heavy usage of rotation operations. Rotation operations tend to slow down significantly as they progress in higher levels of ciphertext, making it burdensome for multiplexed parallel convolution to be used at higher levels. Indeed, in paper [18], efforts were made to overcome this limitation by designing the network to perform multiplexed parallel convolution at the lowest level possible. While such measures may be feasible in the context of [18], it cannot be guaranteed that similar measures will be applicable to various present or future AI models. Therefore, rather than focusing solely on optimizing AI models by considering the level at which convolution is computed, enhancing the performance of convolution itself is crucial for its widespread usage. Also, multiplexed parallel convo-

• *Byeong-Seo Min, and Joow-Woo Lee are with the School of Computer Science and Engineering, Chung-Ang University, Seoul 06974, South Korea. E-mail: mbyeongseo@gmail.com, jwlee2815@cau.ac.kr*

Corresponding author: Joon-Woo Lee

lution utilizes various shifts of rotations, requiring a wide range of rotation keys. As a result, there's a significant increase in the amount of key transmission from the client to the server, placing a heavy burden on the client.

Therefore, to evolve the multiplexed parallel convolution technique into a more applicable method, we propose *rotation optimized convolution*. This novel approach aims to reduce the usage of rotation operations in multiplexed parallel convolution, effectively both decreasing execution time and lowering the number of required rotation keys. By doing so, we enhance its usability and make it more suitable for widespread adoption. Rotation optimized convolution technique consists of three main approaches:

1) We suggest reconstructing the multiplexed parallel convolution operation method to reduce the number of rotations. This improvement is achieved by reducing the number of rotations required for combining channels or adjusting the roles of each process slightly. This approach reduces the execution time, enhances their usability, and also reduces the number of rotation keys by maintaining the same depth consumption with multiplexed parallel convolution. As a result, compared to the conventional multiplexed parallel convolution, it was possible to reduce the execution time by up to around 43%.
2) We also propose a technique that reduces the number of rotation operations while consuming additional depth compared to the existing multiplexed parallel convolution. Through this, we can further decrease the execution time. It was observed that a convolution consuming one additional depth compared to the traditional convolution reduced the execution time by up to approximately 66% while consuming two additional depths could reduce the execution time by up to 70%.
3) We propose a small level key system to decrease the level of rotation keys, thereby additionally reducing the size of each rotation key. Furthermore, by introducing a hierarchical rotation key system, we will apply a technique to reduce the number of rotation keys that clients need to generate and transmit by allowing the server to create some of the necessary rotation keys. Ultimately, we can reduce the size of the rotation keys by approximately 29×.

Additionally, associated with rotation optimized convolution, we have reduced the number of rotation operations for downsampling operations, thereby decreasing latency and reducing the number of rotation keys required.

Furthermore, we also propose a new form of matrix-vector multiplication called parallel Baby-Step Giant-Step(BSGS) matrix-vector multiplication, applicable in many scenarios such as the fully connected layer of AI models. It supports multiplication operations between a single plaintext square matrix and a ciphertext vector, which reduces the number of rotation operations compared to conventional methods that employ the Baby-step Giant-Step (BSGS) algorithm with the diagonal method in [19]. Consequently, this further reduces latency and the number of rotation keys, enhancing its usability. [19] is not only applicable to matrix multiplication in a state of multiplexed parallel packing but also can be efficiently applied in scenarios where multiple identical data are contained within a single ciphertext. In our implementation, we were able to achieve a maximum reduction in execution time of around 63%.

2 RELATED WORKS

There have been several recent papers that have conducted research related to convolution operations. [20] achieved the first implementation of ResNet targeting CIFAR-10 images. Although convolution operations were implemented in [20], efficient computation of operations like strided convolution was not achieved. [18] proposed multiplexed parallel convolution, which is generally applicable in various scenarios including strided convolution. In this paper, we mainly focus on optimizing multiplexed parallel convolution.

[21] proposes a new CNN structure that using hybrid packing, that combining various packing techniques including multiplexed parallel packing. In this process, [21] use modified multiplexed parallel convolution, which operates at a higher level compared to [18]. [22] proposed a new Graph Convolutional Network (GCN) model LinGCN, and incorporates multiplexed parallel convolution as one of its components. As seen in both [22] and [21], we can observe the utilization of multiplexed parallel convolution across various AI models, often at higher levels of ciphertext as well. Since our propose algorithm reduces the number of rotations compared to multiplexed parallel convolution, it efficiently alleviates the burden of using convolution at a higher level. In other words, AI models that incorporate multiplexed parallel convolutions at higher levels would likely see a more pronounced reduction in execution time by applying our rotation optimized convolution technique.

[23] focused on group convolution, and the group convolution method suggested in [23] is orthogonal to multiplexed parallel convolution. This implies that by combining these two techniques, we can achieve even higher performance improvements. Using our rotation optimized convolution, which builds upon the advancements of multiplexed parallel convolution, can lead to even greater enhancements in performance.

[13] proposed a technique to expedite the convolution process by incorporating convolution midway through the bootstrapping process. While this approach is applicable in networks where convolution and activation functions are simply repeated, such as ResNet, it becomes structurally challenging to perform convolution operations freely within complex networks due to the necessity of incorporating them mid-bootstrapping. Therefore, to execute a convolution block widely used across various networks in a homomorphic encryption setting, it's more appropriate to utilize the multiplexed parallel convolution technique, a conventional method for performing convolution in its normal packed state. Hence, we chose to optimize multiplexed parallel convolution.

3 PRELIMINARIES

3.1 RNS-CKKS Fully Homomorphic Encryption

The residue number system variant Cheon-Kim-Kim-Song (RNS-CKKS) is a Fully Homomorphic Encryption

(FHE) scheme that enables real number operations on encrypted data. In RNS-CKKS, all ciphertexts consist of one-dimensional complex number data. Specifically, they 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 plaintext also has a one-dimensional shape, and for all one-dimensional data including plaintext in this paper, assuming there is a one-dimensional data A, $A[i]$ denotes the i-th data of A(where i 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 $\text{Enc}(m_1)$ and $\text{Enc}(m_2)$, which represent the encrypted plaintext messages m_1 and m_2, respectively,

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

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.

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 can be a part of the optimization process. Also, it should be noted that in this paper, we only use plaintext multiplication, means that multiplication occurs only between plaintext and ciphertext, not between ciphertexts.

Every ciphertext has a unique positive integer, multiplicative level (abbreviated as mult level or level when there is no confusion). The level of ciphertext represents the capacity of further multiplication. 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, multiplication between l_1 level-ciphertext with plaintext, the resulting ciphertext will have 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.

All homomorphic operations, including addition, multiplication, and rotation, are influenced by the level of the ciphertexts. This means that 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 Fig. 1.

As indicated in Fig.1, 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.

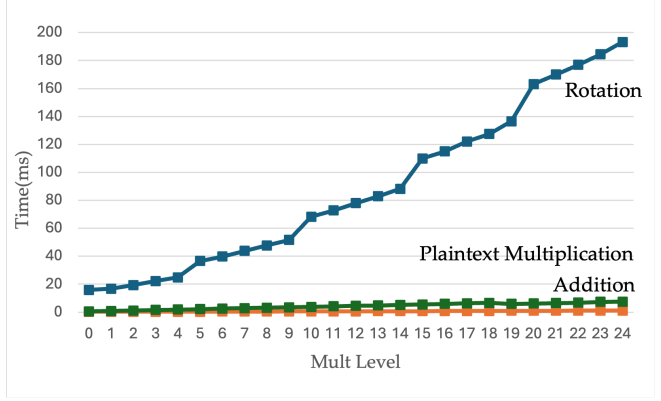

Fig. 1: Execution time of RNS-CKKS basic operation - addition, plaintext multiplication, rotation.

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 [18], CNN, particularly ResNet, was implemented using RNS-CKKS, and during this process, computations were performed in a state where ciphertexts were multiplexed parallel packed. Multiplexed parallel packing is a packing method 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 in [18]. Multiplexed parallel convolution involves three main processes, denoted as $\mathsf{SISOConv}$, $\mathsf{RotationSum}$, and $\mathsf{ZeroOutCombine}$. To express each process of multiplexed parallel convolution, various parameters will be utilized. The precise definitions of each parameter are documented in Appendix B. $\mathsf{SISOConv}$ process is based on single-input single-output convolution proposed in [4], [20]. From the perspective of the size of plaintext, 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 single kernel K, which can be denoted as $K \in \mathbb{R}^{f_h \times f_w \times c_i}$, the result of $\mathsf{SISOConv}$ process is $B \in \mathbb{R}^{w_o \times h_o \times c_i}$. The process of combining the values at each position of the channels in B corresponds to the $\mathsf{RotationSum}$ process. In this process, rotation and addition operations are repeated $\log_2 c_i$ times to combine the values. The result of the process can be denoted as $C \in \mathbb{R}^{w_o \times h_o}$. In $\mathsf{ZeroOutCombine}$ process, since there are c_o numbers of kernels, combining c_o numbers 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 $\mathsf{ZeroOutCombine}$ process can be viewed as divided into two subprocesses: the $\mathsf{ZeroOut}$ subprocess, which removes invalid values, and the $\mathsf{Combine}$ 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 $f_h f_w - 1$, $q(2\lceil\log_2 k_i\rceil + \lceil\log_2 t_i\rceil)$, c_o, and $\log_2 p_o$, respectively. Considering that the number of rotations varies per level, let's define the execution time of rotation operations at level l as r_l. When multiplexed parallel convolution is executed on ciphertext with level l', the execution time can be summarized as follows. $r_{l'}(f_h f_w - 1) + r_{l'-1}(q(2\lceil\log_2 k_i\rceil + \lceil\log_2 t_i\rceil + c_o) + r_{l'-2}(\log_2 p_o)$.

Fig. 3 simplifies the multiplexed parallel convolution where each variable is defined as $w_i = h_i = w_o = h_o = 1, f_w = f_h = 3, c_i = c_o = 8, p_i = p_o = 2, k_i = k_o = 2$ and multiplexed parallel packed ciphertext was drawn using the expression of Fig. 2. The multiplexed parallel packed ciphertext in Fig. 2 corresponds to the case where gap k is 2. $b_j^{(i_1)}$ corresponds to the data of the jth channel when input data of convolution is multiplied by the weight of i_1th 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 parallelly existing data, and each data is multiplied by different kernel weights. This corresponds to the SISOConv process. In RotationSum process, each data 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 Fig. 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 of containing two identical parallelly existing data.

Furthermore, in conventional [18], multiplexed parallel convolution was tested in the situation of six convolutions, which are CONV1, CONV2, CONV3s2, CONV3, CONV4s2, and CONV4. To compare with this, our paper also conducted several tests targeting these convolutions. The parameters for each convolution can also be found in Appendix B. In [18], function Vec often used to map some three-dimension tensor to a vector to describe convolution. To avoid confusion, in our paper, Vec similarly defined as, for some three-dimension tensor $B \in \mathbb{R}^{h_i \times w_i \times c_i}$, $\mathsf{Vec}(B)$ maps tensor B to a vector in $\mathbb{R}^{n_t}$. $\mathsf{Vec}(B) = (b_0, \ldots, b_{n_t-1}) \in \mathbb{R}^{n_t}$, where b can be defined as

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

4 Rotation Optimized Convolution

Multiplexed parallel convolution involves multiple rotation operations, so the execution time at lower levels may be relatively low, but usage at higher levels can be burden-

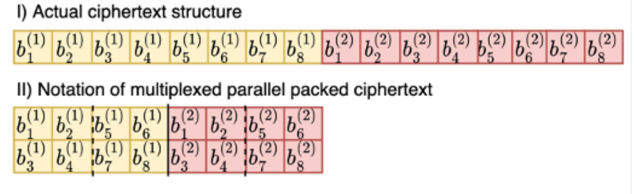

Fig. 2: Notation of multiplexed parallel packed ciphertext in this paper.

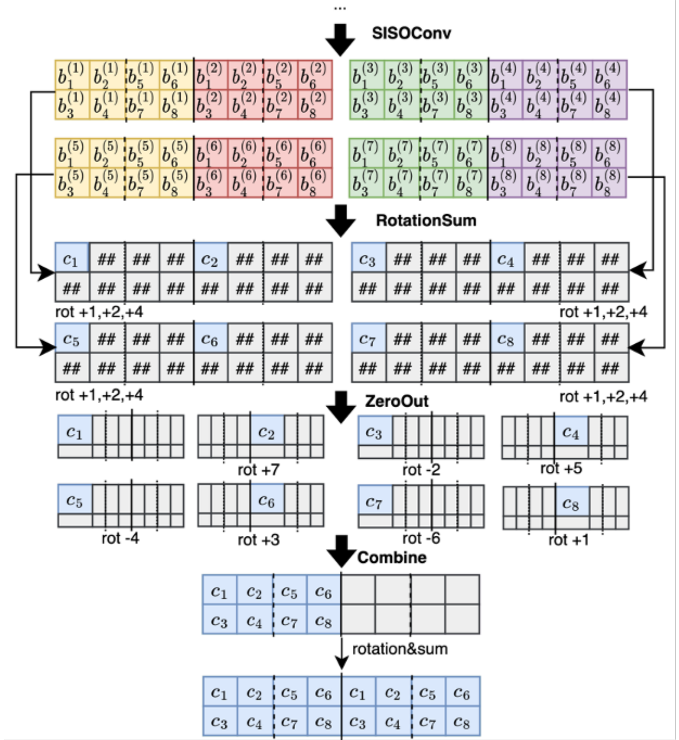

Fig. 3: Simple abstract example of multiplexed parallel convolution in [18] where $w_i = h_i = w_o = h_o = 1, f_w = f_h = 3, c_i = c_o = 8, p_i = p_o = 2, k_i = k_o = 2$.

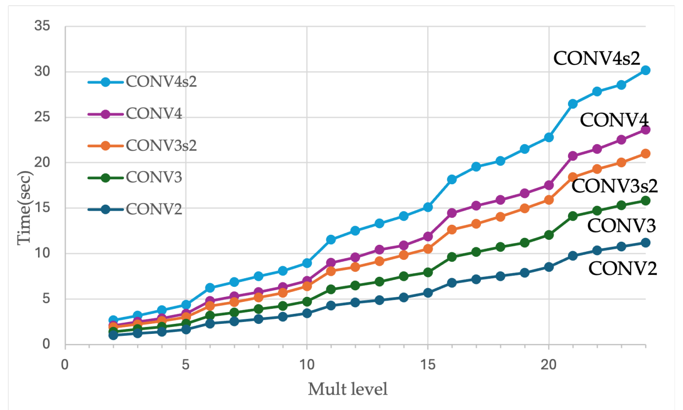

Fig. 4: Execution time of multiplexed parallel convolution in various multiplication levels.

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

TABLE 1: Number of rotation keys and rotation operations during multiplexed parallel convolution.

some. As shown in Fig. 4, the execution time of various multiplexed parallel convolutions varies significantly depending on the level. Since multiplexed parallel convolution consumes 2 depth, the minimum level at which operations can be performed is 2. If the level is raised to 6, the execution time of all multiplexed parallel convolutions approximately doubles, and if the level is raised to 9, the execution time triples. If operations are conducted at the maximum level, the execution time becomes 11 times longer. As evident from this, if convolution involves numerous rotation operations, it can pose a burden when used across various levels. Furthermore, in the scenario where the client sends the key to the server, there is a notable burden on the total size of rotation keys. For all six scenarios of multiplexed parallel convolution (CONV1, CONV2, CONV3s2, CONV3, CONV4s2, CONV4), the size of all rotation keys is currently 29100 MB. This is quite heavy in the actual client-server relationship, indicating the need for optimization of rotation keys.

In this section, we propose rotation optimized convolution, aiming to reduce the number of rotations used during convolution to decrease execution time and rotation key size. Focusing on this main technique, we will propose techniques for further reducing the number of rotations by exploring the rotation correlation between the RotationSum process and the ZeroOutCombine process, therefore, reducing additional execution time. Lastly, we propose rotation optimized convolution with consuming additional depth to reduce further execution time.

4.1 High-level Idea

The number of rotation keys used and the number of rotation operations in each process of multiplexed parallel convolution are summarized in TABLE 1. Particularly, ZeroOutCombine process stands out for its characteristic of having almost the same number of rotation keys and rotation operations. Since rotation operations with different shifts require different types of rotation keys, we observe that in ZeroOutCombine process, rotation operations with different shifts are performed. Therefore, by reducing the

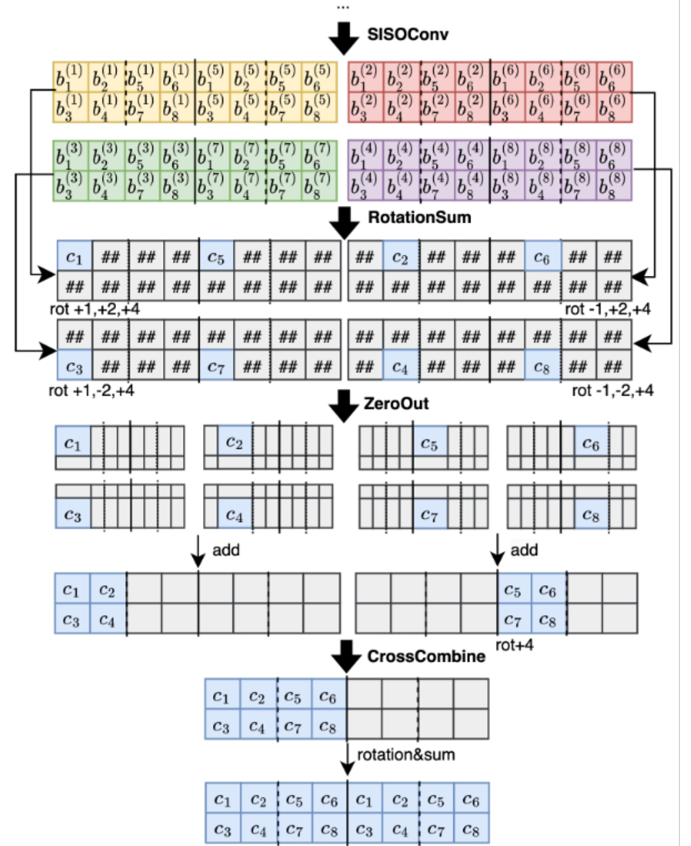

Fig. 5: Simple abstract example of rotation optimized convolution. Same condition as Fig. 3.

number of rotation operations in ZeroOutCombine process, we can achieve a decrease in execution time and a reduction in rotation key size simultaneously. We reduce the number of rotation operations in ZeroOutCombine process by leveraging the characteristics of multiplexed parallel packing.

Multiplexed parallel-packed ciphertext inherently possesses two characteristics. Firstly, multiple channels exist in a multiplexed state within a single ciphertext. Secondly, identical data is arranged in a parallel manner within a single ciphertext. An important key idea to reduce the number of rotation operations is to consider these multiplexed parallel packed results arrangement when performing convolution. Fig. 3 illustrates the conventional multiplexed parallel convolution method.

In Fig. 3, during the SISOConv process, different weights are multiplied to the parallel existing identical data within a ciphertext to produce multiple channels simultaneously. However, in pairing these, the final positions were not considered, and the pairs were simply arranged in order. Also, it can be seen that after the RotationSum process, the positions of each channel are the same. As a result, rotation operations with different shifts must be performed on multiple channels derived from a single ciphertext during ZeroOutCombine process. More precisely, the ZeroOutCombine process is divided into two subprocesses, ZeroOut and Combine. Subprocess ZeroOut involves multiplication operations that eliminate invalid values generated during the RotationSum process, while in subprocess Combine, rotation operations

are performed. Therefore, in the Combine subprocess of the ZeroOutCombine process, rotation operations with different shifts are conducted.

To reduce the number of rotation operations, this method can be modified as shown in Fig. 5. First, we adjust the positions of channels derived from the RotationSum process to match the final positions. In the original method, as shown in Fig. 3, the positions of each channel derived from the RotationSum process were all unified to the position of the first channel. However, considering the characteristic of multiplexed packing, this approach leads to unnecessary rotation, such as in the case of a ciphertext containing c_2. Indeed, c_2 is positioned in the second channel of the result, but after the RotationSum process, its position is saved at the first channel, necessitating a rotation operation. To optimize this process, in Fig. 5, when deriving a ciphertext containing c_2 and c_6 during the RotationSum process, the result channels are pre-arranged at the final positions. This can be achieved by occasionally performing rotation operations in the opposite direction during the RotationSum process.

The second modification involves arranging pairs of channels produced in a single operation differently from the original method which considers the parallel nature of multiplexed parallel packing. In the SISOConv process, pairs of channels operated on from the same ciphertext are constructed such that the relative positional differences between channels match those of the target positions. For instance, Fig. 3 and Fig. 5 show different types of channels within a ciphertext after the SISOConv process; c_1 and c_2 coexist in one ciphertext in Fig. 3, while c_1 and c_5 coexist in one ciphertext in Fig. 5. From the perspective of parallel data, in Fig. 3, the left data generates c_1, c_3, c_5, c_7 while the right data generates c_2, c_4, c_6, c_8, whereas in Fig. 5, the left data generates c_1, c_2, c_3, c_4 while the right data generates c_5, c_6, c_7, c_8. This modification is made to maintain the same relative positional differences between result channels after the SISOConv process and after all convolution processes have concluded.

With these two modifications, it is possible to generate ciphertexts corresponding to specific parts of the outcome after the RotationSum process. As a result, unlike in Fig. 3 where rotation operations were performed for each channel separately in the ZeroOutCombine process, in Fig. 5, only a few rotation operations, which combines the parallel data, is performed. We denote these rotations as the CrossCombine operation. That is, in contrast to multiplexed parallel convolution, the subprocess Combine of ZeroOutCombine is modified as CrossCombine subprocess in rotation optimized convolution. Unlike the multiplexed parallel convolution where each channel had to be combined one by one in the Combine subprocess, the CrossCombine process rotates some parts of the results generated after the RotationSum process all at once, reducing the number of rotation operations and, consequently, the number of rotation keys.

The key difference so far is, that while the objectives of other processes remain unchanged, only the Combine subprocess of the ZeroOutCombine process has been modified to the CrossCombine process, resulting in a change in the number of rotations. To express it precisely in terms of equations, in the ZeroOutCombine process of multiplexed parallel convolution, the number of rotations is c_o, whereas in rotation optimized convolution, the number of rotations is $p_c - 1$. Where c_o was approximately 16, 32, 64 in the previously mentioned six convolution scenarios, and p_c was around 2, 4, 8, it is evident that the number of rotations decreases significantly. Especially, as c_o, the number of channels in the output, increases, the performance difference becomes more prominent. The comparison of exact execution time has been summarized in Section 7. To summarize again, defining the execution time of rotation operations at level l as r_l, executing the rotation optimized convolution on ciphertext with level l', the execution time can be summarized as follows: $r_{l'}(f_h f_w - 1) + r_{l'-1}(q(2\lceil \log_2 k_i \rceil + \lceil \log_2 t_i \rceil) + r_{l'-2}(p_c - 1 + \log_2 p_o)$.

4.2 Using correlation of RotationSum process and CrossCombine process

The objectives of the three processes in multiplexed parallel convolution are clear. In SISOConv, the operation involves computing the product of data and kernel weights. The RotationSum process aggregates values from each channel, while the ZeroOutCombine process filters out meaningless values generated during RotationSum, rotates valid values, and rearranges them into the correct positions in the final output. Among these processes, it's possible to optimize additional rotation operations by merging channels less during the RotationSum stage and combining them in the CrossCombine process.

In this subsection, we will introduce a technique for reducing the execution time of rotation optimized convolution by leveraging the relationship between the number of rotations in RotationSum and CrossCombine. The key idea of this technique is to exploit the characteristics of the CrossCombine process. In the conventional RotationSum process, each rotation and addition operation combines values corresponding to two channels, resulting in a total of $\log_2 c_o$ rotations per ciphertext. If we reduce the number of rotations conducted per ciphertext by one, we can decrease the total number of rotations by the number of ciphertexts, q. However, the required number of rotations in the ZeroOutCombine process roughly doubles. It's challenging to achieve optimization through this process in conventional multiplexed parallel convolution because the number of rotations in ZeroOutCombine is approximately c_o, which is significantly larger than q.

However, in rotation optimized convolution, the rotations in ZeroOutCombine are for CrossCombine. The number of rotation operations here is approximately p_c, which is relatively small compared to q. Therefore, the technique we propose for optimization is leveraging the relationship between RotationSum and CrossCombine to perform fewer rotations in the RotationSum stage, merging fewer channels, and combining them in CrossCombine.

Figure 6 provides a simple example of using the correlation of RotationSum and CrossCombine. Here, $b^{(i)}_{j_1,j_2,j_3,j_4}$ is defined as follows: $b^{(i)}_{j_1,j_2,j_3,j_4} = b^{(i)}_{j_1} + b^{(i)}_{j_2} + b^{(i)}_{j_3} + b^{(i)}_{j_4}$. After the SISOConv, data existing in parallel has 8 channels each. Hence, in the RotationSum process, it previously required $\log_2 8 = 3$ rotations to merge these channels. However, in this process, one rotation is saved, and instead, the number of data to be merged through CrossCombine increases from

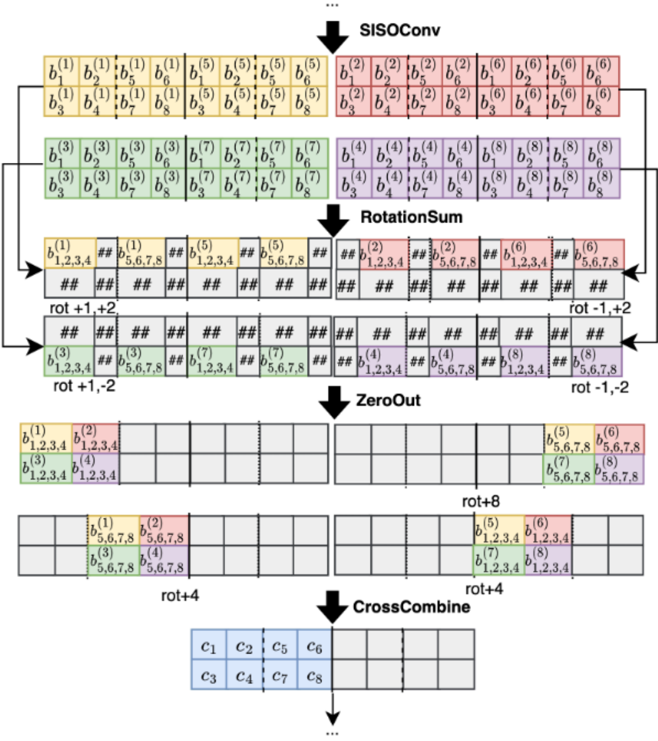

Fig. 6: Simple abstract example of using correlation of RotationSum and CrossCombine during rotation optimized convolution.

2 to 4. As a result, the number of rotations in CrossCombine changes from 1 to 3.

Comparing the overall number of rotations in Fig. 5 and 6, in Fig. 5, $3 \times 4 = 12$ rotations were performed in the RotationSum process, and 1 rotation was performed in the CrossCombine process, totaling $12+1 = 13$ rotations. In Fig. 6, $2 \times 4 = 8$ rotations were performed in the RotationSum process, and 3 rotations were performed in the CrossCombine process, totaling $8 + 3 = 11$ rotations. It's evident that by utilizing the correlation of RotationSum and CrossCombine, the number of rotations can be reduced. However, it is not fixed how many times rotation should be performed for the RotationSum process in rotation optimized convolution to achieve the optimal result. This is because the rotation operation occurs at different ciphertext levels between the RotationSum process and the zero-out and combine process. Generally, performing rotation operations at a higher ciphertext level results in longer computation time, but the difference in computation time may vary depending on the implementation environment.

Therefore, we provide an equation that can help find the optimal situation when using correlation. In the RotationSum process, let's denote the number of rotations each ciphertext needs to undergo as x. And also denotes the time required for rotation operations at mult level l' of the ciphertext as $r_{l'}$. When the 2-depth consuming rotation optimized convolution begins at level l, we can calculate x as follows:

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

4.3 Convolution with additional depth

One of the key features of the CKKS scheme is the consideration of the level of ciphertexts during operations. Each ciphertext has mult levels, if all levels are consumed, heavy operations- bootstrapping need to be used to increase the level of the ciphertext. Simply setting a maximum level highly can't be a solution either, as rotation, addition, and multiplication conducted on a higher level consume more execution time. Therefore, it's important to set the maximum level appropriately and efficiently use the levels to minimize the number of bootstrappings required.

Due to these characteristics, when various operations are proposed in the CKKS scheme, there are often suggestions for minimizing level consumption while also providing additional trade-offs between the consumption of levels and other benefits such as execution time and accuracy, depending on the situation. For instance, [24] organizes the trade-off of consuming additional depth and execution time in SlotToCoeff and CoeffToSlot operations in bootstrapping., and [25] organizes the trade-off of consuming additional depth and accuracy in relu operation.

In this section, we propose a novel technique that introduces additional depth in rotation optimized convolution to further shorten the execution time, which is not present in multiplexed parallel convolution, thus offering a new trade-off. The key idea of this technique is to combine ciphertexts that require the same rotation during the RotationSum process. When looking at the conventional multiplexed parallel convolution in Fig. 3, this kind of optimization is challenging to employ. Because all ciphertexts undergo the same rotation operation during the RotationSum process, the positions where valid values exist are the same for all ciphertexts. Therefore, additional operations are required to combine them.

In contrast, in Fig. 5, rotation-optimized convolution rotates ciphertexts in different directions during the RotationSum processes, resulting in varied positions where valid values exist. Therefore, leveraging this aspect, optimization as Fig. 6 can be achieved. In the RotationSum process, ciphertexts requiring a rotation of +2 and those needing -2 are combined separately and then subjected to rotation operations. Since the valid positions of each ciphertext are different, no additional operations are necessary. However, to combine each ciphertext, invalid values must be filtered out, necessitating the use of multiplication operations and an additional level.

For an exact comparison, compared to the 2-depth consuming rotation optimized convolution in Fig. 5, the number of rotations used during RotationSum process and ZeroOut-Combine process reduced by 13 to 9. It can be observed that using additional depth can reduce the number of rotation operations. Moreover, from the perspective of rotation keys, when combining ciphertexts that require the same shift and then performing rotation operations, the number of rotation operations decreases without altering the shifts of rotation operations. Therefore, using convolution with additional depth is unrelated to the number of rotation keys.

Furthermore, the key idea for reducing rotation counts lies in combining ciphertexts requiring the same rotation during the RotationSum process. Therefore, the additional

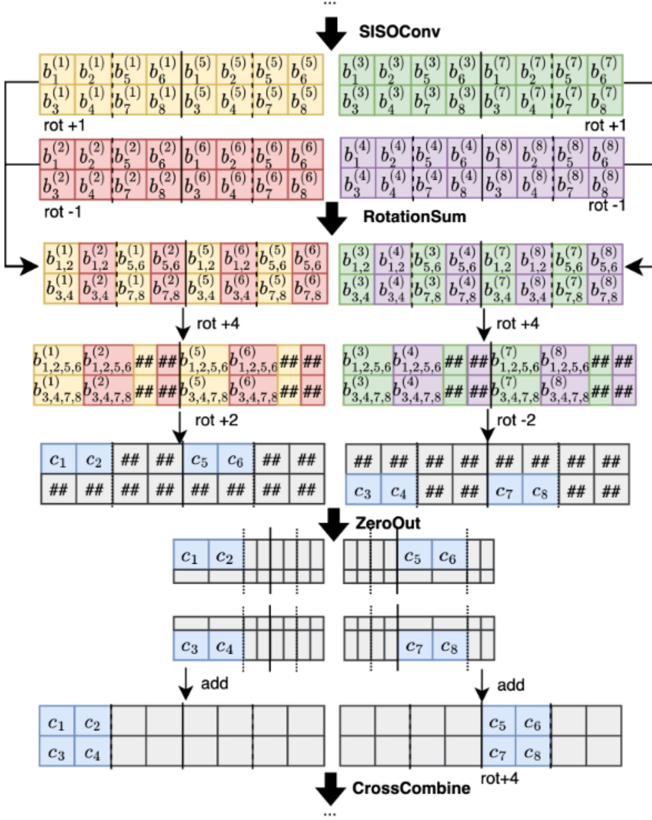

Fig. 7: Simple abstract example of using additional depth for rotation optimized convolution.

depth available for rotation optimized convolution is related to the number of ciphertexts passing through the RotationSum processes, denoted as q in this paper. More precisely, when combining two ciphertexts into one during the RotationSum process, it can be repeated a maximum of $\log_2 q - 1$ times. Since the original rotation optimized convolution consumes 2 depths, if convolution with additional depth is used, it allows for the use of convolution with a maximum depth of $\log_2 q + 1$.

Similar to the correlation of RotationSum and CrossCombine technique in Subsection 4.2, rotations occur at various mult levels within the RotationSum process. Hence, the optimal scenario may vary depending on the implementation environment. Therefore, we provide the guide equation to find the optimal scenario considering both the correlation of RotationSum and CrossCombine and additional depth-consuming convolution techniques. If a convolution consuming $d+1 (d \geq 2)$ depth is utilized, starts at multilevel $l (l \geq d+1)$ then the approximate performance of this convolution in the RotationSum, and CrossCombine processes can be expressed as the following equation:

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

where variables defined as $\prod_{m=1}^{d} q_m = q$, $\log_2 c_i - x = \sum_{m=1}^{d} x_m$ and $x_m \geq \log_2 q_m$ for all m. x represents the same as in Subsection 4.2. In other words, $\log_2 c_i - x$ denotes the number of rotation and addition operations needed for RotationSum process. In the RotationSum process, the variable determining how many times rotation and addition should be performed before the m-th combining of the ciphertexts is denoted as x_m, and the number of ciphertexts being combined is denoted as q_m. Since x_m rotation and addition combines 2^{x_m} data in ciphertext, maximum 2^{x_m} ciphertexts can be combined during m-th combining. Therefore, value of q_m is bounded by x_m as $x_m \geq \log_2 q_m$.

Providing guidelines to use additional depth in rotation optimized convolution, first, the value of d, which specifies how many depths will be consumed during convolution, has to be determined. Then, while satisfying the following conditions $\prod_{m=1}^{d} q_m = q$, $\log_2 c_i - x = \sum_{m=1}^{d} x_m$ and $x_m \geq \log_2 q_m$ for all m, x, x_m and q_m within the range $1 \leq m \leq d$ to minimize Equation 2 has to be selected. Since the number of combinations satisfying all conditions and the simplicity of Equation 2 allow for brute force, finding its minimum value is not a burdensome task. Moreover, x, x_m and q_m are values pre-determined during the implementation process of rotation optimized convolution, not during the operation itself. Therefore, determining the minimum value of Equation 2 doesn't impact performance, whereas identifying the accurate value does.

4.4 Algorithm Description

In presenting the main idea of rotation optimized convolution, 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 rotation optimized convolution, 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, Fig. 8 is a modified convolution of Fig. 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 rotation optimized convolution algorithm do not affect the performance, which means the number of rotations or size of rotation keys doesn't change and it only changes the shift of rotation operation. Indeed, the rotation optimized convolution algorithm is not unique and possesses a heuristic aspect. Thus, we will not statically generalize how kernel weights are arranged and what rotation operations are employed in the RotationSum process and CrossCombine process; rather, we will provide **blueprint**s in detailing these aspects.

These **blueprint**s are written about the kernel placement, and shift of rotation to use during RotationSum, and CrossCombine processes. There are four types of **blueprint**s as follows: '2-depth Conv', '3-depth Conv', '4-depth Conv', and '5-depth Conv'. All these **blueprint**s are created considering the correlation of RotationSum and CrossCombine. '2-depth Conv' **blueprint**s are basic rotation optimized convolutions that only consume 2-depth. The **blueprint**s

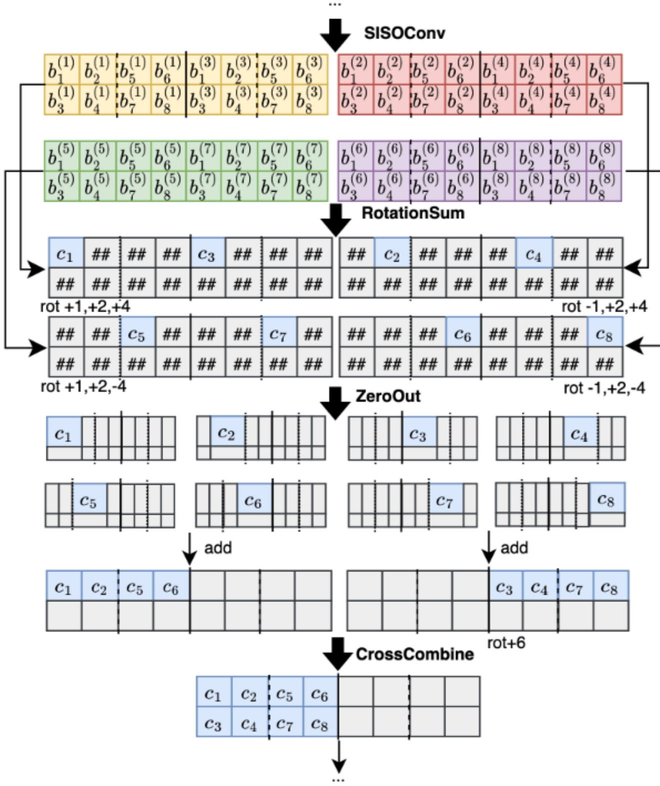

Fig. 8: Showing the non-uniqueness property of rotation optimized convolution. Same condition, and same performance as Fig. 5, but the kernel arrangement and shift amount of rotation are different.

consuming 3, 4, and 5 depths correspond to '3-depth Conv', '4-depth Conv', and '5-depth Conv' respectively. For single **blueprint**, there are three components, KernelBP, RotationSumBP, and CrossCombineBP. KernelBP represents the kernel weight arrangement in input data, and shift of rotation during RotationSum process are saved in RotationSumBP. Lastly, information on shift number of rotation used during CrossCombine saved in CrossCombineBP. The precise values for each blueprint used in our experiments can be found in Appendix A.

To describe the proposed algorithms in this paper, it is required to define a valid value-selecting tensor, S, S', S''. $S^{(r)}$ filter serves to select a valid value from ciphertext which is rotated by r. $S^{(r)} = (S^{(r)}[i_1])_{0 \leq i_1 < n_t} \in \mathbb{R}^{n_t}$

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

for $0 \leq i_1 < n_t$. $S'^{(k,l)}$ filters out parallel data in the ciphertext. For ciphertext which has l identical data parallelly, filter $S'^{(k,l)}$ select k-th data. $S'^{(k,l)} = (S'^{(k,l)}[i_1])_{0 \leq i_1 < n_t} \in \mathbb{R}^{n_t}$

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

for $0 \leq i_1 < n_t$. S'' filters out data considering when the stride value of convolution is bigger than 1. $S'' = (S''[i_1])_{0 \leq i_1 < n_t} \in \mathbb{R}^{n_t}$

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

for $0 \leq i_1 < n_t$.

Before description of RotOptConv in detail, it is also required to define $\mathsf{ParMultWgt}(U; i_1, i_2, i_3)$. $\mathsf{ParMultWgt}(U; i_1, i_2, i_3)$ maps weight tensor $U \in \mathbb{R}^{h_i \times w_i \times c_i \times c_o}$ to an element of $\mathbb{R}^{n_t}$. To define ParMultWgt, parallelly multiplexed shifted weight tensor $\overline{U}''^{(i_1,i_2,i_3)}$ also has to be defined. $\overline{U}''^{(i_1,i_2,i_3)} = (\overline{U}''^{(i_1,i_2,i_3)}[i_5][i_6][i_7]) \in \mathbb{R}^{k_i h_i \times k_i w_i \times t_i p_i}$ where $0 \leq i_5 < k_i h_i, 0 \leq i_6 < k_i w_i, 0 \leq i_7 < t_i p_i$ for $0 \leq i_1 < f_h, 0 \leq i_2 < f_w$, and $0 \leq i_3 < q$ can be defined as follows:

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

for $0 \leq i_5 < k_i h_i$, $0 \leq i_6 < k_i w_i$, $0 \leq i_7 < t_i p_i$. Other variables are defined as $i_8 = k_i^2(i_7 \bmod t_i) + k_i(i_5 \bmod k_i) + i_6 \bmod k_i$ and $i_9 = \mathsf{KernelBP}[p_i i_3][\lfloor i_7/t_i \rfloor]$. Then, ParMultWgt is defined as $\mathsf{ParMultWgt}(U; i_1, i_2, i_3) = \mathsf{Vec}(\overline{U}''^{(i_1,i_2,i_3)})$.

Since RotationSum process used in RotOptConv, have different rules, we redefine the SumSlots algorithm in [18] to AdaptSumSlots, which contains RotationSum process and also SISOConv. The AdaptSumSlots algorithm incorporates many algorithms related to the structure of **blueprint**s because it utilizes **blueprint**s. The opType refers to the type of operation according to the **blueprint**. If opType is 0, for example, the **blueprint** might be [0, 2048], indicating rotating the ciphertext by 2048 and then performing an addition operation. When opType is 1, it signifies performing a Combine operation in the middle of RotationSum, meaning using additional depth. An example **blueprint** for this would be [1, 4, 1, 32], indicating rotating each of the four ciphertexts by +1+32, -1+32, +1-32, and -1-32, respectively, then filtering out invalid values and combining them. When opType is 2, it represents operations before the CrossCombine stage after the RotationSum process. Similar to opType 1, in this case, multiplication is carried out as many times as the number of data involved in the CrossCombine during the filtering process to classify each data separately in parallel. Furthermore, as a detailed rule, RotationSumBP[0][0] stores the maximum index of RotationSumBP, and RotationSumBP[i][0] contains information about the i-th opType. If opType is 1 or 2, then RotationSumBP[i][1] contains information about how many ciphertexts will be merged.

For RotOptConv algorithm, the CrossCombineBP is used, indicating a simple form where each ciphertext at the i-th position is rotated by CrossCombineBP[i] during

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

1: **Input:** Indicating the order of ciphertext i', Rotated parallely multiplexed tensor ciphertexts ct'_α, Current location within RotationSumBP d, and weight tensor U
2: **Output:** Tensor ciphertext $\mathsf{ct_b}$ and filter of $\mathsf{ct_b}$ that apply zero out S_b.
3: $\mathsf{ct_b} \leftarrow \mathsf{ct_{zero}}$
4: $\mathsf{S_b} \leftarrow \mathsf{S_{one}}$
5: opType $\leftarrow$ RotationSumBP$[d][0]$
6: **if** $d = 0$ **then**
7: **for** $i_1 \leftarrow 0$ **to** $f_h - 1$ **do**
8: **for** $i_2 \leftarrow 0$ **to** $f_w - 1$ **do**
9: $\mathsf{ct_b} \leftarrow \mathsf{ct_b} \oplus \mathsf{ct}'^{(i_1,i_2)}_\alpha \odot \mathsf{ParMultWgt}(U; i_1, i_2, i')$
10: **end for**
11: **end for**
12: **end if**
13: **if** $d > 0$ **then**
14: **if** opType $= 0$ **then**
15: $\mathsf{ct_a}, S_a \leftarrow$ ADAPTSUMSLOTS$(i', d-1, \mathsf{ct}'_\alpha, U)$
16: $r' \leftarrow$ RotationSumBP$[d][1]$
17: $\mathsf{ct_b} \leftarrow \mathsf{ct}_a \oplus \mathsf{Rot}(\mathsf{ct_a}; r')$
18: $S_b \leftarrow S_a \odot S^{(r')}$
19: **end if**
20: **if** opType $= 1$ **then**
21: SumNum $\leftarrow$ RotationSumBP$[d][1]$
22: **for** $i'' \leftarrow \mathsf{SumNum} \times i'$ **to** $(\mathsf{SumNum}+1) \times i' - 1$ **do**
23: $\mathsf{ct_a}, S_a \leftarrow$ ADAPTSUMSLOTS$(i'', d-1, \mathsf{ct}'_\alpha, U)$
24: **for** $j \leftarrow 2$ **to** $\log_2(\mathsf{SumNum}) + 1$ **do**
25: $r' \leftarrow$ RotationSumBP$[d][j]$
26: **if** $((i'' >> (j-2))\&1) = 1$ **then**
27: $\mathsf{ct_a} \leftarrow \mathsf{ct}_a \oplus \mathsf{Rot}(\mathsf{ct_a}; -r')$
28: $S_a \leftarrow S_a \odot S^{(-r')}$
29: **else**
30: $\mathsf{ct_a} \leftarrow \mathsf{ct}_a \oplus \mathsf{Rot}(\mathsf{ct_a}; r')$
31: $S_a \leftarrow S_a \odot S^{(r')}$
32: **end if**
33: **end for**
34: $\mathsf{ct_b} \leftarrow \mathsf{ct_b} \oplus (\mathsf{ct_a} \odot S_a)$
35: **end for**
36: **end if**
37: **if** opType $= 2$ **then**
38: $\mathsf{ct_b}, S_b \leftarrow$ ADAPTSUMSLOTS$(i', d-1, \mathsf{ct}'_\alpha, U)$
39: **for** $j \leftarrow 2$ **to** $\log_2(\mathsf{RotationSumBP}[d][1]) + 1$ **do**
40: $r' \leftarrow$ RotationSumBP$[d][j]$
41: **if** $((i' >> (j-2))\&1) = 1$ **then**
42: $\mathsf{ct_b} \leftarrow \mathsf{ct}_c \oplus \mathsf{Rot}(\mathsf{ct_b}; -r')$
43: $S_b \leftarrow S_b \odot S^{(-r')}$
44: **else**
45: $\mathsf{ct_b} \leftarrow \mathsf{ct}_c \oplus \mathsf{Rot}(\mathsf{ct_b}; r')$
46: $S_b \leftarrow S_b \odot S^{(r')}$
47: **end if**
48: **end for**
49: $S_b \leftarrow S_b \odot S''$
50: **end if**
51: **end if**
52: **Return** $\mathsf{ct_b}, S_b$

CrossCombine process. As a detail, CrossCombineBP[0] stores the number of data to be combined in CrossCombine

Algorithm 2 ROTAOPTCONV$(\mathsf{ct}'_\mathsf{a}, U)$

1: **Input:** Parallelly multiplexed tensor ciphertext ct'_a and weight tensor U.
2: **Output:** Parallelly multiplexed tensor ciphertext ct'_c
3: $\mathsf{ct}'_\mathsf{c} \leftarrow \mathsf{ct_{zero}}$
4: **for** $i_1 \leftarrow 0$ **to** $f_h - 1$ **do**
5: **for** $i_2 \leftarrow 0$ **to** $f_w - 1$ **do**
6: $\mathsf{ct}'^{(i_1,i_2)}_\alpha \leftarrow \mathsf{Rot}(\mathsf{ct}'_\mathsf{a}; k_i^2 w_i(i_1 - (f_h-1)/2) + k_i(i_2 - (f_w-1)/2))$
7: **end for**
8: **end for**
9: CrossNum $\leftarrow$ CrossCombineBP$[0]$
10: **for** $i_3 \leftarrow 0$ **to** CrossNum $- 1$ **do**
11: $\mathsf{ct}'^{(i_3)}_\beta \leftarrow \mathsf{ct_{zero}}$
12: **end for**
13: BPlen $\leftarrow$ RotationSumBP$[0][0]$
14: **for** $i_4 \leftarrow 0$ **to** RotationSumBP$[\mathsf{BPlen}][1] - 1$ **do**
15: $\mathsf{ct}'_\mathsf{b}, S_a \leftarrow$ ADAPTSUMSLOTS$(i_4, \mathsf{BPlen}, \mathsf{ct}'_\alpha, U)$
16: **for** $i_3 \leftarrow 0$ **to** CrossNum $- 1$ **do**
17: $\mathsf{ct}'^{(i_3)}_\beta \leftarrow \mathsf{ct}'^{(i_3)}_\beta \oplus \mathsf{ct}'_\mathsf{b} \odot (S_a \odot S'^{(i_3,\mathsf{CrossNum})})$
18: **end for**
19: **end for**
20: **for** $i_1 \leftarrow 0$ **to** CrossNum $- 1$ **do**
21: $\mathsf{ct}'_\mathsf{c} \leftarrow \mathsf{ct}'_\mathsf{c} \oplus \mathsf{Rot}(\mathsf{ct}'^{(i_1)}_\beta; \mathsf{CrossCombineBP}[i_1+1])$
22: **end for**
23: **for** $j \leftarrow 0$ **to** $\log_2 p_o - 1$ **do**
24: $\mathsf{ct}'_\mathsf{c} \leftarrow \mathsf{ct}'_\mathsf{c} \oplus \mathsf{Rot}(\mathsf{ct}'_\mathsf{c}; -2^j(n_t/p_o))$
25: **end for**
26: **Return** ct'_c

process. For intuitive understanding, in the case of Fig. 8 as an example, based on the order of ciphertexts and the order of kernels within each ciphertext, KernelBP = [[1,3],[2,4],[5,7],[6,8]]. Additionally, during the RotationSum process, a rotation operation with a shift of +2 is commonly applied to each ciphertext, while rotations with shifts of ± 1 or ± 4 vary across ciphertexts. Moreover, after rotations with shifts of ± 1 or ± 4, ZeroOut considering CrossCombine should be applied, hence opType becomes 2. Thus, RotationSumBP = [[2],[0,4],[2,4,1,4]]. In the CrossCombine process, since the second data is combined with a rotation operation of shift +6, CrossCombineBP = [2,0,6].

A comparison of the performance of rotation optimized convolution and multiplexed parallel convolution centered around 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 multiplexed parallel convolution, 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 rotation optimized convolution which prioritizing minimizing execution time, assuming that operations start at level l, its performance can be represented as follows: $\min(r_l(f_h f_w - 1) + r_{l-1}q(\log_2 c_i - x) + r_{l-2}(2^x p_c - 1 + \log_2 p_o))$ for $0 \le x \le \log_2 c_i$.

For 3 or more depth consuming rotation optimized convolutions which prioritize minimizing execution time, assuming that operations start at level l, its performance can

be represented as follows: $\min(r_l(f_h f_w - 1) + r_{l-1}(x_1 q) + \sum_{m=2}^{d} r_{l-m}(x_m \frac{q}{\prod_{j=1}^{m-1} q_j}) + r_{l-d-1}(2^x p_c - 1 + \log_2 p_o))$ where $\prod_{m=1}^{d} q_m = q$, $\log_2 c_i - x = \sum_{m=1}^{d} x_m$ and $x_m \geq \log_2 q_m$ for all m.

Once again, in our paper, we do not provide a fixed algorithm of rotation optimized convolution due to its heuristic nature. However, through the equations mentioned above, it is possible to achieve the expected optimal performance for each convolution situation. If the detailed implementation produces the expected performance as predicted by these equations, then it can be considered a correctly implemented rotation optimized convolution.

4.5 Rotation Optimized Downsampling

When implementing ResNet, one of the necessary operations is downsampling for the residual connection when a convolution with a stride greater than 1 is applied. For instance, let's consider data with a width and height of 32 and 16 channels. After passing through a convolutional layer with a stride of 2, the data becomes 16×16 with 32 channels. In this scenario, to implement the residual connection, a downsampling operation is required to reduce the data from 32×32 with 16 channels to 16×16 with 16 channels. While downsampling is a simple operation in plaintext, it becomes somewhat complex in Fully Homomorphic Encryption (FHE) environments. That's why in conventional [18], an algorithm called multiplexed parallel downsampling is introduced to handle this downsampling process efficiently in such environments. Multiplexed parallel downsampling operations are not particularly heavy computations, but similarly, since rotation operations are used during computation, the execution time increases rapidly when performed at higher levels. In this section, similar to rotation optimized convolution, we propose *rotation optimized downsampling*, which optimizes the multiplexed parallel downsampling operation in [18].

Two key ideas were utilized for optimization, with the first being identical to what was used in rotation optimized convolution. The characteristic of the multiplexed parallel packing structure was leveraged, and the intermediate steps were conducted considering the positions of the results. The second key idea leveraged the utilization of downsampling operations in the process of the residual connection. For example, in conventional [18], downsampling was applied to the output before the convolution operation and then added to the output after the convolution operation. Since multiplexed parallel convolution operations consume 2 depths, for instance, downsampling is applied to ciphertexts with a level 2 and produces results that will be added to ciphertexts with a level 0. Due to this characteristic, although multiplexed parallel downsampling originally consumed 1 depth, even when consuming 2 depths, there is no difference in the overall level of consumption. We took advantage of this aspect, and rotation optimized downsampling was designed to consume 2 depths.

The advantage of consuming 2 depths instead of 1 depth in the downsampling process is that it enables rotating channels located in similar positions in the output simultaneously. In other words, with 1-depth consumed downsampling, multiplication simply consumes a level for extracting each channel. However, by using 2 depths, unnecessary values can be removed with a single multiplication, and then channels that are identical to parts of the output are placed through rotation and addition operations into the created empty spaces. Afterward, by extracting each gathered channel within the ciphertext through another multiplication, the number of rotation operations can ultimately be reduced.

For comparison, we conducted experiments in scenarios DOWNSAMP1 and DOWNSAMP2, identical to those in [18]. In these scenarios, we were able to achieve a constant reduction in the number of rotations regardless of the input and output shapes. In other words, unlike the conventional multiplexed parallel downsampling, when using rotation optimized downsampling, the execution times for DOWNSAMP1 and DOWNSAMP2 remained the same. Detailed experimental results can be found in Section 7, and detailed parameters used in DOWNSAMP1 and DOWNSAMP2 are organized in Appendix B. We provide ROTOPTDOWNSAMP algorithm for implementing rotation optimized downsampling for DOWNSAMP1 and DOWNSAMP2.

To define the ROTOPTDOWNSAMP algorithm, we should define new filters, $\overline{S}, \overline{S}'$, which filter out invalid values after rotation and addition operation. $\overline{S} = (\overline{S}[i_1])_{0 \leq i_1 < n_t} \in \mathbb{R}^{n_t}$

$$\overline{S}^{(j)}[i_1] = \begin{cases} 1, & \text{if } (i_1 > 4096j \text{ and } i_1 < 4096(j+1)) \\ & \text{or } (i_1 > 4096(j+4) \text{ and } i_1 < 4096(j+5)) \\ 0, & \text{otherwise,} \end{cases}$$

for $0 \leq i_1 < n_t$. And for $\overline{S}'$, $\overline{S}' = (\overline{S}'[i_1])_{0 \leq i_1 < n_t} \in \mathbb{R}^{n_t}$

$$\overline{S}'^{(j)}[i_1] = \begin{cases} 1, & \text{if } j''\text{mod}8 = j' \text{ and} \\ & (i_1 > 1024j'' \text{ and } i_1 < 1024(j''+1)) \\ 0, & \text{otherwise,} \end{cases}$$

for $0 \leq i_1 < n_t, 0 \leq j'' < 32$. Another variable is defined as $j' = j + 2\lfloor j/2 \rfloor$.

Algorithm 3 ROTOPTDOWNSAMP($\mathsf{ct}''_{\mathsf{a}}$)

```
1:  Input: Parallelly multiplexed tensor ciphertext ct''_a
2:  Output: Parallelly multiplexed tensor ciphertext ct''_c
3:  ct''_c ← ct_zero
4:  ct''_a ← ct''_a ⊙ S''
5:  ct''_a ← ct''_a ⊕ Rot(ct''_a; k_i w_i ⌊k_i mod k_i^2 / k i⌋ + k_i^2 h_i w_i ⌊k_i / k_i^2⌋ − k_i)
6:  ct''_a ← ct''_a ⊕ Rot(ct''_a; 2k_i^2 h_i w_i − k_i^2 w_i)
7:  ct''_b ← ct_zero
8:  for j ← 0 to 3 do
9:      if k_i = 1 then
10:         ct''_b ← ct''_a ⊙ S̄^(j)
11:         ct''_c ← ct''_c ⊕ Rot(ct''_b; k_i^2 w_i h_i (4 k_i i_1 − (i_1 + 2)))
12:     else
13:         i_2 ← i_1 mod k_i + k_i^2 ⌊i_1 / k_i⌋
14:         ct''_b ← ct''_a ⊙ S̄'^(j)
15:         ct''_c ← ct''_c ⊕ Rot(ct''_b; k_i^2 w_i h_i (i_2) − (k_i w_i (i_1 mod 2) + k_i^2 w_i h_i (⌊i_1/2⌋ + 1)))
16:     end if
17: end for
18: ct''_c ← Rot(ct''_c; −n_t / p_o)
19: Return ct''_c
```

5 Parallel BSGS Matrix-Vector Multiplication

Matrix-vector multiplication, including inference of fully connected layers of neural networks, is a common operation in FHE. Several methods have been devised for matrix-vector multiplication ([19], [26], [27], [28], [29], [30]). One of the famous methods to multiplicate plaintext matrix and ciphertext vector is a diagonal method from [19] and applying the Baby-step Giant-step(BSGS) algorithm to further reduce the number of rotations to reduce latency. [18] also implemented the fully connected layer by applying the BSGS algorithm to the diagonal method (hereinafter referred to as the BSGS diagonal method). [31]

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([32]) images in [18], 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 within a single ciphertext. However, the BSGS diagonal method does not take this into account and operates only on one data, using only around 72 slots for the matrix-vector multiplication. In other words, it does not efficiently utilize the ciphertext.

The scenario where multiple identical data exist within a single ciphertext is not exclusive to multiplexed parallel packing; it applies to various structures in FHE. Bootstrapping is one of the heaviest operations in FHE, and therefore, sparse slot bootstrapping [18], [31] is widely used as an efficient method for utilizing bootstrapping. Since sparse slot bootstrapping requires a structure where multiple identical data are present within a single ciphertext, in many FHE structures utilizing sparse slot bootstrapping, such a structure with multiple identical data within a single ciphertext is employed. Similarly, when using the BSGS diagonal method in such environments, matrix-vector multiplication does not effectively utilize the length of the ciphertext.

One of the problems arising from not effectively utilizing the length of the ciphertext, is the inability to perform operations in parallel, necessitating additional rotation operations. This imposes a burden on using matrix-vector multiplication at higher levels, which reduces its usability. Thus, if the number of rotation operations can be reduced, optimization to decrease it is necessary. An example with a matrix size of 512×512 and vector size is 512×1 further demonstrates the importance of this. Applying the BSGS diagonal method here would require 47 rotations. However, using the algorithm that we will propose, involves creating 32 copies within a single ciphertext, leveraging a substantial portion of the ciphertext for computation, and reducing the number of rotations to 26. The second issue is the excessive amount of plaintext used. When performing multiplication for an $n \times n$ matrix and an $n \times 1$ vector, the required number of plaintexts is n, with approximately n valid values in each plaintext. For the plaintext to ciphertext multiplication, the length of the plaintext must match that of the ciphertext,

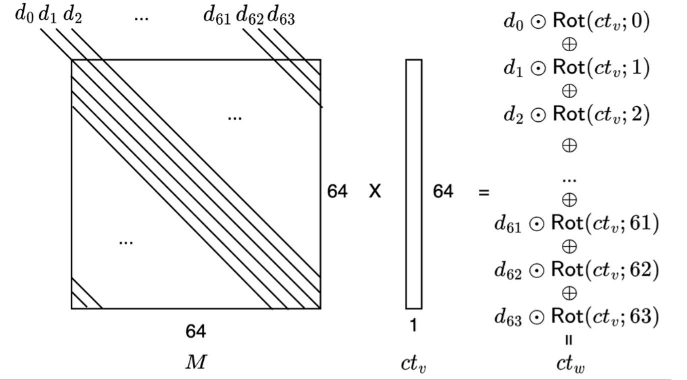

Fig. 9: A simple illustration of diagonal method([19]) for matrix-vector multiplication.

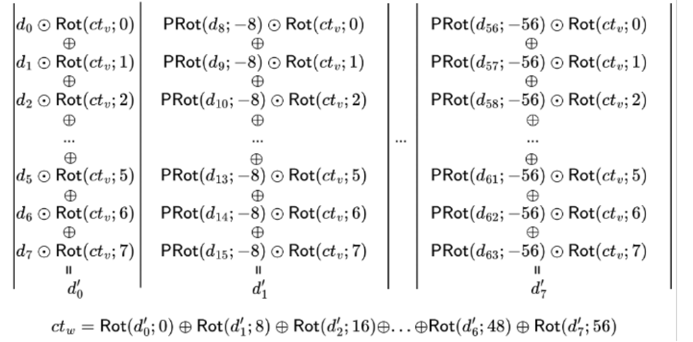

Fig. 10: A simple illustration of BSGS diagonal method.

and since the length of the ciphertext cannot change, the plaintext must also match the ciphertext length. This means that there is a disadvantage of requiring plaintexts with a much larger capacity than the actual valid values available.

Therefore, in this section, we propose parallel BSGS matrix-vector multiplication, which supports the multiplication of a single plaintext square matrix and ciphertext vector. Parallel BSGS matrix-vector multiplication optimizes the number of rotations required compared to BSGS diagonal method, especially when ciphertext can contain numerous identical data parallel, including multiplexed parallel packed ciphertext. To explain parallel BSGS matrix-vector multiplication, knowledge of both the conventional diagonal method and the BSGS diagonal method is required. For an intuitive explanation, we will offer images for an example of 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 actual implementation, a ciphertext that has the value of v in the front is denoted as ct_v and a ciphertext that has the value of w in the front is denoted as ct_w. Both ct_v and ct_w has same n_t length.

Fig.9 illustrates the diagonal method introduced in [19]. For diagonals of M, d is defined as $d_i[j] = A[j][j+i]$ for $0 \le i < n$, $0 \le j < n$. And each d_i 's length is n_t, which is the same as ct_v. As shown in Fig.9, we can get ct_w by integrating the multiplication of each d_i and $\mathsf{Rot}(ct_v; i)$. More precisely, it can be expressed as follows.

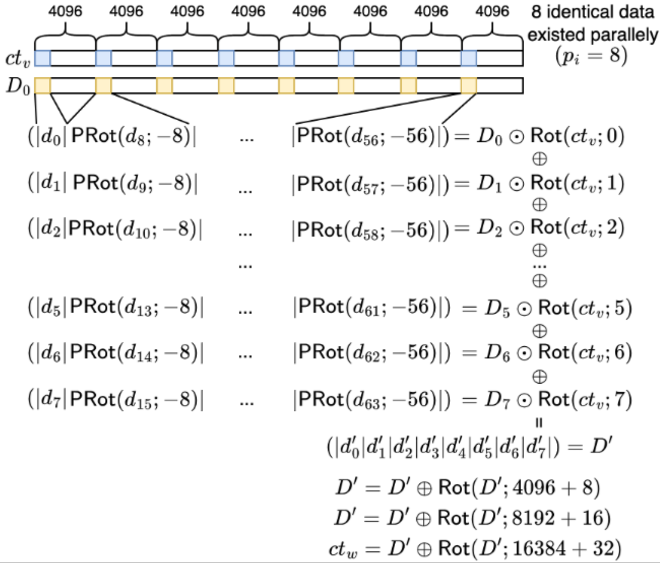

Fig. 11: A simple illustration of parallel BSGS diagonal method.

$ct_w = \bigoplus_{i=0}^{n-1}(d_i \odot \mathsf{Rot}(ct_v;i))$. 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 Fig.9, we can examine examples of BSGS diagonal method as Fig.10. The idea leverages the fact that d_is are plaintexts and precomputation of plaintext is possible. Therefore, optimization is achieved by pre-rotating d_is. To represent BSGS diagonal method, several additional parameters need to be defined. For n_1, n_2, $n = n_1 n_2$. And for d', $d'_i = \bigoplus_{j=n_1 i}^{n_1 i+n_1-1}(\mathsf{PRot}(d_j;-n_1 i) \odot \mathsf{Rot}(ct_v; j \bmod n_1))$ for $0 \le i < n_2$. ct_w can also be defined as follows. $ct_w = \bigoplus_{i=0}^{n_2-1}\mathsf{Rot}(d'_i; n_1 i)$. The number of rotations needed 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 needs $n = 64$ plaintexts.

The example applying our proposed algorithm, parallel BSGS matrix-vector multiplication, can be seen in Fig.11. In Fig.11, we showcase the scenario where parallel BSGS matrix-vector multiplication is applied to the fully connected layer of Resnet (for CIFAR-10 Images) implementation of [18]. In the input ciphertext ct_v, 8 identical data are located parallelly, and the length of the ciphertext is $n_t = 32768$. Unlike the conventional BSGS diagonal method, which only utilizes the leftmost data, we utilize all 8 data. To further clarify the concept, additional definitions are necessary to explain Fig.11. Here, p_i signifies that there are p_i identical data within one ciphertext, specifically, p_i within ct_v is 8. $(|v_0|v_1|v_2|v_3|v_4|v_5|v_6|v_7|)$ indicates that within one ciphertext, the value of v_i is located at positions $in_t/8$. D represents the arrangement of multiple d's within one ciphertext. It can be precisely defined as follows:

$$D_i = (|d_i|\mathsf{PRot}(d_{i+n_1};-n_1)|\mathsf{PRot}(d_{i+2n_1};-2n_1)|\dots \dots|\mathsf{PRot}(d_{i+(n_2-1)n_1};-(n_2-1)n_1)|) \tag{3}$$

for $0 \le i < n_1$. In other words, D is obtained by arranging n_2 number of d's within one ciphertext at equal intervals. D' represents a similar arrangement where multiple d's exist within the ciphertext: $D' = (|d'_0|d'_1|\dots|d'_{n_2-1}|)$ As observed in Fig.11, D' can be obtained as the sum of the multiplications of D and the rotated ct_v. Ultimately, 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 of the concept 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 D. This idea increases the number of valid values within the plaintext and reduces the total number of plaintexts required. Additionally, by repetitively 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 Fig.11 where the rotation is reduced to $2\log_2 n_2 + n_1 - \log_2 p_i = 11$ where $n = 64 = n_1 n_2 = 8 \times 8$. Furthermore, the required number of plaintexts is reduced to $n_1 = 8$.

In Fig.11, using the structure where there are 8 identical data within the ciphertext remains unchanged during the process. However, increasing the number of identical data within the ciphertext as needed could be a method to reduce the number of rotation operations. For instance, let's consider increasing the number of identical data 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.

$$\underset{n_2}{\operatorname{argmin}}(2\log_2(n_2) + (n/n_2) - \log_2 p_i) \tag{4}$$

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 that can exist within the ciphertext is also limited. In other words, in the equation 4, 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 algorithm, ParBSGSMatVecMul. ParBSGSMatVecMul's input, the parallel diagonal plaintext D can be further generalized as follows: $D_i = \sum_{j=0}^{n_2-1}\mathsf{PRot}(d[jn_1+i]; -jn_1 - (n_t/n_2)j)$ for $0 \le i < n_1$.

Finally, in terms of guidance for practical usage, to effectively utilize ParBSGSMatVecMul, it is necessary for there to be multiple identical data 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 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. For instance, in a scenario where $p_i = 1$, ParBSGSMatVecMul can still be effectively utilized as long as the ciphertext size is sufficient, even though there may be a performance difference of three rotations compared to the case where $p_i = 8$. Therefore, ParBSGSMatVecMul can be effectively utilized even in cases where $p_i = 1$, as long as the ciphertext size is adequate. Applying ParBSGSMatVecMul could be even easier in other situations that already utilize BSGS diagonal method.([33], [34]) It also should be noted that the presence of p_i identical data within a single ciphertext in algorithm ParBSGSMatVecMul is assumed to occur with each data positioned at equal intervals, as expressed in ct_v of Fig.11

Algorithm 4 ParBSGSMatVecMul($D, \mathsf{ct_v}, n_1, n_2, p_i$)

1: **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 $n = n_1 n_2$.
2: **Output:** Parallelly multiplexed tensor ciphertext that contains n valid value $\mathsf{ct_w}$.
3: $\mathsf{ct_w} \leftarrow \mathsf{ct_{zero}}$
4: $\mathsf{ct_v} \leftarrow \mathsf{ct_v} \oplus \mathsf{Rot}(\mathsf{ct_v}; -n_1 n_2)$
5: **for** $i_1 \leftarrow 1$ **to** $\log_2(n_2/p_i)$ **do**
6: $\quad \mathsf{ct_v} \leftarrow \mathsf{ct_v} \oplus \mathsf{Rot}(\mathsf{ct_v}; -(n_t/p_i)/2^{i_1})$
7: **end for**
8: **for** $i_1 \leftarrow 0$ **to** $n_1 - 1$ **do**
9: $\quad \mathsf{ct_w} \leftarrow \mathsf{ct_w} \oplus D[i_1] \odot \mathsf{Rot}(ct_v; i_1)$
10: **end for**
11: **for** $i_1 \leftarrow 0$ **to** $\log_2(n_2) - 1$ **do**
12: $\quad \mathsf{ct_w} \leftarrow \mathsf{ct_w} \oplus \mathsf{Rot}(\mathsf{ct_w}; 2^{i_1}(n_1 + n_t/n_2))$
13: **end for**
14: **Return** $\mathsf{ct_w}$

6 Rotation Key Reduction

While proposing rotation optimized convolution, we significantly reduced the number of rotations and effectively decreased the number of rotation keys by about one-third. 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 introduce the hierarchical rotation key system from [35]. 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 computation keys with key-level 0 are required for operations at the server, the client only sends computation keys with key-level 1. Then, the server generates 0 key-level keys from the 1 key-level keys. This system, known as the two-level hierarchical rotation key system, allowed us to further reduce the size of computation keys. We applied this technique to the rotation keys required for rotation optimized convolution, where the rotation keys needed for rotation optimized convolution correspond to the 0-level keys. Since 0-level keys can be generated by 1-level keys, the keys that the client needs to transmit to the server are indeed 1-level keys. By applying this, we were able to additionally reduce the size of rotation keys.

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 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 rotation optimized convolution technology, by default, 2 levels are consumed during rotation optimized convolution 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 *small level key system*. We ultimately extended this to a two-level hierarchical rotation key system.

In all 2-level key systems, the size of a 1-level key is proportional to the size of a 0-level key. Since the size of small level keys is smaller than that of the original 0-level key, the size of the 1-level key that needs to be transmitted from the client to the server has also decreased. A simple illustration of this is in Fig. 12. 0-level key in Fig, 12 represents 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. L', the combined length of L and K, representing the highest level, constitutes the form of a 1-level key. This characterizes the hierarchical rotation key system, and an example of applying it to a small level key system is also provided. Unlike the previous requirement of rotation keys for all L levels, the small level key system contains only the rotation keys for the specific level where the rotation operation will be used. This significantly reduces its size compared to before. By applying the hierarchical rotation key system to the small level key in a situation consuming 2-depth in rotation optimized convolution, the l value of the 1-level key is $K + 2$, which confirms that the size of the 1-level key can be greatly reduced.

TABLE 2 represents the experiment results of applying a two-level hierarchical rotation key system and small level key system to reduce the size of rotation keys required for rotation optimized convolution. As mentioned in Subsection 4.3, utilizing additional depth is unrelated to the number of rotation keys. Therefore, experiments were conducted using the 2-depth Conv **blueprint**. By applying the hierarchical rotation key system and small level key system to the rotation keys transmitted in rotation optimized convolution, we were able to reduce the size of the final transmitted rotation keys by approximately 29× compared to [18].

7 Experimental Results

In this section, we will compare the execution time and size of rotation keys of the various operations proposed by

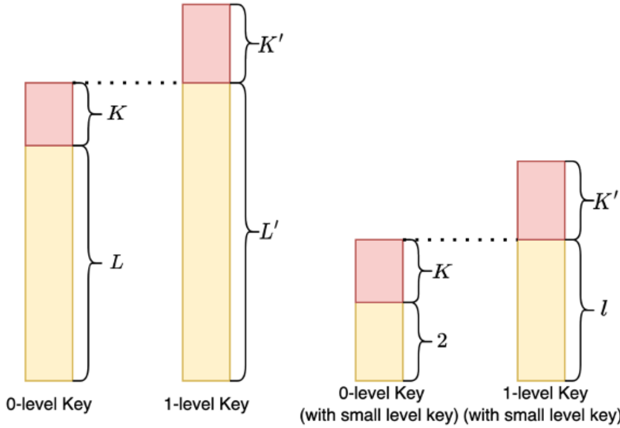

Fig. 12: A simple and abstract illustration of applying the hierarchical rotation key system and small level key system to rotation key.

Method	Conventional [18]	RotOptConv	RotOptConv +Hierarchical rotation key system	RotOptConv +Hierarchical rotation key system+ Small level key system
Rotation key size	29100 MB	8250 MB	2100 MB	1100 MB

TABLE 2: Comparison of rotation key size

us with those of the conventional method. [1] The numerical analyses are conducted on the representative Lattigo v5(lattice-based homomorphic encryption library, [36]) on AMD Ryzen 9 7900X at 3.2 GHz (24 cores) with 124 GB RAM, running the Ubuntu 20.04 operating system. We set Lattigo main CKKS parameter as follows: $N = 2^{16}, \log Q = [51, 46, 46, 46, 46, 46, 46, 46, 46, 46, 46, 46, 46, 46, 46, 46, 46, 46, 46, 46, 46, 46, 46, 46, 46], \log P = [60, 60, 60, 60, 60]$. P denotes the special modulus and length of special modulus correspondence to the length of $\log P$. Using terminology borrowed from [18], six representative examples of multiplexed parallel convolution- CONV1, CONV2, CONV3s2, CONV3, CONV4s2, CONV4—appear, depending on the size of the convolution's input, the number of kernels, and the stride value. We measured and compared the performance of these six convolution examples to evaluate rotation optimized convolution.

We measured the performance of each convolution using four **blueprint**s, 2-depth Conv, 3-depth Conv, 4-depth Conv, and 5-depth Conv. Each **blueprint** corresponds to minimizing execution time in consuming each depth. Fig. 13 summarizes the execution time according to the starting mult level of the operation for each type of convolution. As seen in the graph, the effect of time reduction is evident, and the difference in execution time is more pronounced at higher levels due to the characteristics of the rotation operation. Speaking in precise numbers, in experiments using 2-depth Conv, the decrease in execution time ranged from a minimum of 20% to a maximum of 43%, with an average of 36%. With 3-depth Conv, the decrease ranged from a minimum of 40% to a maximum of 66%, with an average of 51%. For 4-depth Conv, the decrease ranged from a minimum of 41% to a maximum of 70%, with an average of 56%. It was observed that the effect of rotation optimized convolution depends on the situation of the convolution implemented as well as the implementation environment. In our implementation, 5-depth Conv showed little improvement compared to 4-depth Conv. Although the reduction rate increased from 4-depth Conv to 3-depth Conv, it was not significant enough. Therefore, in our implementation environment, using 3-depth Conv seems to provide the most significant reduction in execution time compared to the consumption of depth.

Furthermore, the extent of performance improvement in convolution varies depending on the convolution situation. As highlighted in Subsection 4.1, when the Combine process is modified to CrossCombine, the number of rotations required changes from c_o to $p_c - 1$. Therefore, we can observe that the greater the difference, the greater the degree of performance improvement, particularly in convolution situations where this difference is significant. In other words, since the values of p_c are not extremely large, typically around 2, 4, or 8, the effectiveness of rotation optimized convolution tends to be greater as the output channel count, represented by c_o, increases. Hence, we can note that the performance improvement tends to be more significant for CONV1 and CONV2 with c_o of 16, CONV3s2 and CONV3 with c_o of 32, and CONV4s2 and CONV4 with c_o of 64, in ascending order. This also applies equally to convolutions in other scenarios. For instance, in ResNet targeting ImageNet, there exists a convolution with an input and output size of $7 \times 7 \times 512$, and due to the large number of channels (512), our convolution demonstrates even greater effectiveness. By simply calculating the number of rotations using formulas, while multiplexed parallel convolution requires 2825 rotations, rotation optimized convolution requires only about 776 rotations without additional depth consumption. This once again confirms the potential of rotation optimized convolution for utilization in various AI models.

We also provide TABLE 3, which contains information on the number of rotation operations performed at each stage of rotation optimized convolution. The difference between multiplexed parallel convolution and rotation optimized convolution lies in the substitution of subprocess Combine of ZeroOutCombine process with subprocess CrossCombine. In TABLE 3, for the sake of simplicity in comparison, the processes of ZeroOut and Combine in multiplexed parallel convolution and the subprocesses of ZeroOut and CrossCombine in rotation optimized convolution are both denoted as ZeroOutCombine.

The execution time tests for algorithms RotOptDownSamp is also provided in Fig.14 We tested two downsampling operations within a ResNet targeting CIFAR-10 images, which corresponds to DOWNSAMP1 and DOWNSAMP2 in the conventional method [18] for accurate comparison. The precise situations for DOWNSAMP1 and DOWNSAMP2 can be found in TABLE 8. In both scenarios, meaningful time reductions were observed using the proposed method for downsampling operations. Moreover,

1. All experiments we conducted in this paper are reproducible by referencing the README in our github link: https://github.com/byeongseomin51/FHE_Conv_MatVecMul

Convolution	Process	Multiplexed parallel convolution	2-depth Conv	3-depth Conv	4-depth Conv	5-depth Conv
CONV1	SISOConv	8	8	-	-	-
	RotationSum	4	4	-	-	-
	ZeroOutCombine	17	2	-	-	-
	Total	29	14	-	-	-
CONV2	SISOConv	8	8	8	8	-
	RotationSum	32	24	16	14	-
	ZeroOutCombine	17	4	4	4	-
	Total	57	36	28	26	-
CONV3s2	SISOConv	8	8	8	8	8
	RotationSum	64	64	40	32	30
	ZeroOutCombine	34	5	5	5	5
	Total	106	77	53	45	43
CONV3	SISOConv	8	8	8	8	-
	RotationSum	40	32	22	18	-
	ZeroOutCombine	34	9	5	5	-
	Total	82	49	35	31	-
CONV4s2	SISOConv	8	8	8	8	8
	RotationSum	80	64	44	36	32
	ZeroOutCombine	67	10	6	6	6
	Total	155	82	58	50	46
CONV4	SISOConv	8	8	8	8	-
	RotationSum	48	48	24	20	-
	ZeroOutCombine	67	10	10	10	-
	Total	123	66	42	38	-

TABLE 3: Comparison of number of rotations used in each process of convolution.

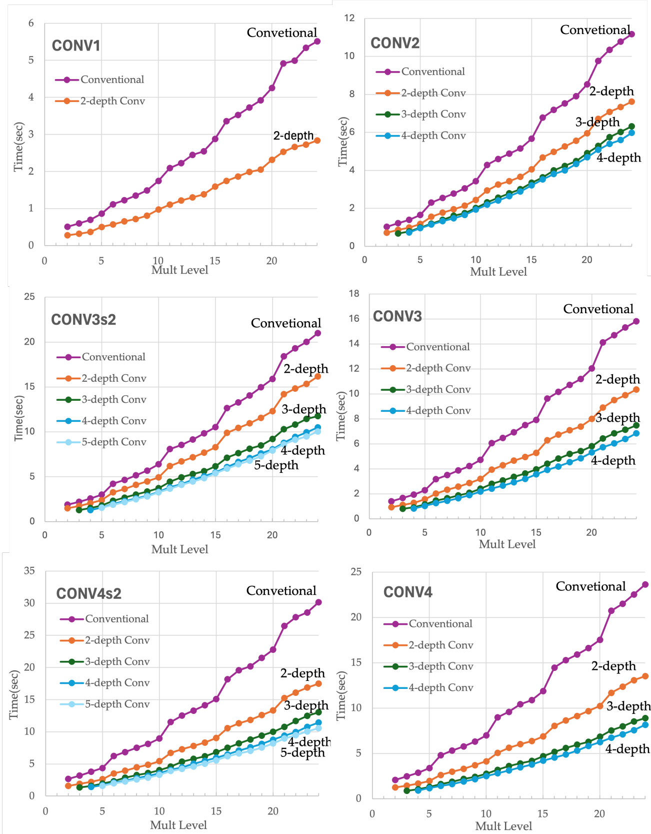

Fig. 13: Execution time comparison of proposed(rotation optimized convolution) and conventional(multiplexed parallel convolution in [18]) convolution.

the proposed method achieved a constant runtime reduction in DOWNSAMP1 and DOWNSAMP2, ensuring that the execution times for downsampling operations in both scenarios were equal. Downsampling is not an excessively time-consuming operation, so the difference may not be easily noticeable. However, it can be observed that employing downsampling at higher levels is less burdensome with the RotOptDownsamp algorithm, which mitigated the usage burden across various levels. Downsampling is associated with convolution operations, therefore, it can also contribute to increasing the flexibility of convolution operation usage at various levels.

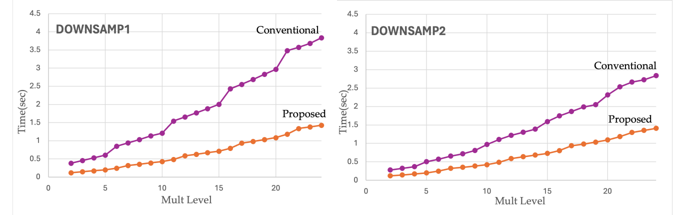

Fig. 14: Execution time comparison of proposed(rotation optimized downsampling) and conventional(multiplexed parallel downsampling in [18]) downsampling for DOWNSAMP1 and DOWNSAMP2 situation.

To clearly observe the effectiveness of parallel BSGS matrix-vector multiplication, we conducted computations under various scenarios, and the results can be seen in Fig.15. In the first scenario, we compared the execution time of applying the conventional BSGS diagonal method and the proposed parallel BSGS matrix-vector multiplication to a fully connected layer as used in [18]. In this scenario, $n = 64$, $p_i = 8$ and $n_t = 32768$. Across most levels, it was observed that the execution time differed by approximately a factor of two, indicating a clear advantage of the proposed method. In another scenario, we experimented with the multiplication of an $n \times n$ matrix and an $n \times 1$ vector across various values of n. In this case, the proposed algorithm was executed with $p_i = 1$ for all scenarios. This implies that if p_i values were higher, we could potentially observe even greater reductions in execution time compared to the graphs. For instance, if the implementation already includes

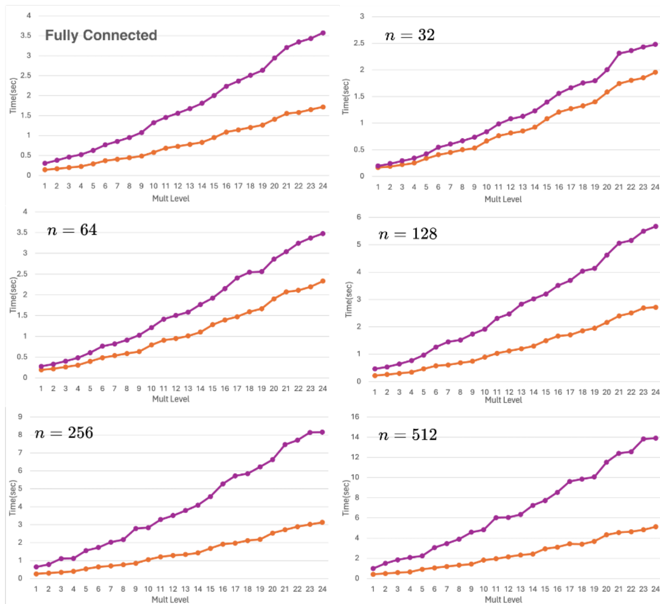

Fig. 15: Execution time comparison of parallel BSGS matrix-vector multiplication and BSGS diagonal method.

Scenario	Criteria	BSGS diagonal method	Proposed
Fully Connected in [18]	# 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

TABLE 4: Comparison of the number of rotation and plaintext required during matrix-vector multiplication between BSGS diagonal method and proposed algorithm, PARBSGSMATVECMUL. The scenario is same as Fig.15. Assume matrix size as $n \times n$, and vector size as $n \times 1$

α identical data parallelly within the ciphertext, we can predict a reduction in the number of rotations by $\log_2 \alpha$ compared to the depicted graph.

When $n = n_1 n_2$ the number of rotations differs between the BSGS diagonal method and the parallel BSGS diagonal method. In the former, it's approximately $n_1 + n_2 - 1$ while in the latter, it's $2 \log_2 (n_2) + n_1 - \log_2 p_i$. As evidenced by the equation, as n becomes larger, the effect becomes more pronounced due to the logarithmic nature of the expression. This property is clearly observed in Fig.15, where the execution time decreases as n increases. To provide an accurate performance comparison of the parallel BSGS diagonal method, we've summarized the number of rotations and plaintexts required for each scenario presented in Fig.15 in TABLE 4.

The number of rotation keys used at each stage of rotation optimized convolution is summarized in TABLE 5. We compared the number of rotation keys for 2-depth Conv

Convolution	Process	Conventional	2-depth Conv
CONV1	SISOConv	8	8
	RotationSum	2	3
	ZeroOutCombine	16	2
	Total	26	13
CONV2	SISOConv	8	8
	RotationSum	4	6
	ZeroOutCombine	16	4
	Total	28	18
CONV3s2	SISOConv	8	8
	RotationSum	4	8
	ZeroOutCombine	33	5
	Total	45	21
CONV3	SISOConv	8	8
	RotationSum	5	7
	ZeroOutCombine	33	9
	Total	46	24
CONV4s2	SISOConv	8	8
	RotationSum	5	8
	ZeroOutCombine	66	10
	Total	79	26
CONV4	SISOConv	8	8
	RotationSum	6	9
	ZeroOutCombine	66	10
	Total	80	27

TABLE 5: Comparison of number of rotation keys used in each process of convolution.

blueprint with the multiplexed parallel convolution. It is evident that compared to conventional methods, a significant reduction in rotation keys has been achieved during the ZeroOutCombine process. **blueprint** for convolution with additional depth is not included, as the usage of convolution with additional depth is unrelated to the number of rotation keys.

8 Conclusion and Future Works

In this paper, we proposed rotation optimized convolution, which reduces the number of rotation operations in multiplexed parallel convolution in [18]. The decrease in the number of rotation operations led to a reduction in latency. As the execution time of rotation operations significantly increases as the ciphertext level rises, rotation optimized convolution also alleviates the burden of performing convolution operations at various levels. We also define the relationship between the consumed depth and execution time of rotation optimized convolution. By reducing the number of rotation keys while using a hierarchical rotation key system and small level key system, we can also reduce the size of rotation keys transmitted between the client and server for rotation optimized convolution. Rotation optimized downsampling and parallel BSGS matrix-vector multiplication operations are also proposed. Specifically, parallel BSGS matrix-vector multiplication was utilized in this paper to implement a fully connected layer on ciphertexts in a state of multiplexed parallel packing. However, even if the ciphertexts are not multiplexed and parallel-packed, the technique can be directly applied as long as multiple identical data are present in parallel within a single ciphertext.

Recently, various AI models, including SSM or convolution operations integrated transformers, have been leveraging convolution operations. Consequently, we opted to

optimize the multiplexed parallel convolution for general applicability. Through the optimization process, we reduced the number of rotation operations, thereby alleviating the burden of conducting convolutions at different levels. This opens up many possibilities for implementing a wide range of AI models as private AI in the future. We plan to utilize rotation optimized convolution to implement cutting-edge models like SSM or convolution operations integrated transformers as private AI going forward.

REFERENCES

[1] R. Dathathri, O. Saarikivi, H. Chen, K. Laine, K. Lauter, S. Maleki, M. Musuvathi, and T. Mytkowicz, "Chet: an optimizing compiler for fully-homomorphic neural-network inferencing," in *Proceedings of the 40th ACM SIGPLAN conference on programming language design and implementation*, 2019, pp. 142–156.

[2] F. Boemer, Y. Lao, R. Cammarota, and C. Wierzynski, "ngraph-he: a graph compiler for deep learning on homomorphically encrypted data," in *Proceedings of the 16th ACM international conference on computing frontiers*, 2019, pp. 3–13.

[3] Q. Lou and L. Jiang, "Hemet: a homomorphic-encryption-friendly privacy-preserving mobile neural network architecture," in *International conference on machine learning*. PMLR, 2021, pp. 7102–7110.

[4] C. Juvekar, V. Vaikuntanathan, and A. Chandrakasan, "{GAZELLE}: A low latency framework for secure neural network inference," in *27th USENIX security symposium (USENIX security 18)*, 2018, pp. 1651–1669.

[5] A. Brutzkus, R. Gilad-Bachrach, and O. Elisha, "Low latency privacy preserving inference," in *International Conference on Machine Learning*. PMLR, 2019, pp. 812–821.

[6] R. Gilad-Bachrach, N. Dowlin, K. Laine, K. Lauter, M. Naehrig, and J. Wernsing, "Cryptonets: Applying neural networks to encrypted data with high throughput and accuracy," in *International conference on machine learning*. PMLR, 2016, pp. 201–210.

[7] E. Chou, J. Beal, D. Levy, S. Yeung, A. Haque, and L. Fei-Fei, "Faster cryptonets: leveraging sparsity for real-world encrypted inference. corr abs/1811.09953 (2018)," *arXiv preprint arXiv:1811.09953*, 2018.

[8] A. Al Badawi, C. Jin, J. Lin, C. F. Mun, S. J. Jie, B. H. M. Tan, X. Nan, K. M. M. Aung, and V. R. Chandrasekhar, "Towards the alexnet moment for homomorphic encryption: Hcnn, the first homomorphic cnn on encrypted data with gpus," *IEEE Transactions on Emerging Topics in Computing*, vol. 9, no. 3, pp. 1330–1343, 2020.

[9] J. Lee, E. Lee, Y.-S. Kim, Y. Lee, J.-W. Lee, Y. Kim, and J.-S. No, "Optimizing layerwise polynomial approximation for efficient private inference on fully homomorphic encryption: a dynamic programming approach," *arXiv preprint arXiv:2310.10349*, 2023.

[10] H. Chabanne, A. De Wargny, J. Milgram, C. Morel, and E. Prouff, "Privacy-preserving classification on deep neural network," *Cryptology ePrint Archive*, 2017.

[11] P. Mishra, R. Lehmkuhl, A. Srinivasan, W. Zheng, and R. A. Popa, "Delphi: A cryptographic inference system for neural networks," in *Proceedings of the 2020 Workshop on Privacy-Preserving Machine Learning in Practice*, 2020, pp. 27–30.

[12] J. Park, M. J. Kim, W. Jung, and J. H. Ahn, "Aespa: Accuracy preserving low-degree polynomial activation for fast private inference," *arXiv preprint arXiv:2201.06699*, 2022.

[13] D. Kim and C. Guyot, "Optimized privacy-preserving cnn inference with fully homomorphic encryption," *IEEE Transactions on Information Forensics and Security*, vol. 18, pp. 2175–2187, 2023.

[14] 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.

[15] Y. Liu, G. Sun, Y. Qiu, L. Zhang, A. Chhatkuli, and L. Van Gool, "Transformer in convolutional neural networks," *arXiv preprint arXiv:2106.03180*, vol. 3, 2021.

[16] H. Yan, Z. Li, W. Li, C. Wang, M. Wu, and C. Zhang, "Contnet: Why not use convolution and transformer at the same time?" *arXiv preprint arXiv:2104.13497*, 2021.

[17] A. Gu and T. Dao, "Mamba: Linear-time sequence modeling with selective state spaces," *arXiv preprint arXiv:2312.00752*, 2023.

[18] E. Lee, J.-W. Lee, J. Lee, Y.-S. Kim, Y. Kim, J.-S. No, and W. Choi, "Low-complexity deep convolutional neural networks on fully homomorphic encryption using multiplexed parallel convolutions," in *International Conference on Machine Learning*. PMLR, 2022, pp. 12 403–12 422.

[19] S. Halevi and V. Shoup, "Algorithms in helib," in *Advances in Cryptology–CRYPTO 2014: 34th Annual Cryptology Conference, Santa Barbara, CA, USA, August 17-21, 2014, Proceedings, Part I 34*. Springer, 2014, pp. 554–571.

[20] 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*, vol. 10, pp. 30 039–30 054, 2022.

[21] D. Kim, J. Park, J. Kim, S. Kim, and J. H. Ahn, "Hyphen: A hybrid packing method and its optimizations for homomorphic encryption-based neural networks," *IEEE Access*, 2023.

[22] H. Peng, R. Ran, Y. Luo, J. Zhao, S. Huang, K. Thorat, T. Geng, C. Wang, X. Xu, W. Wen *et al.*, "Lingcn: Structural linearized graph convolutional network for homomorphically encrypted inference," *Advances in Neural Information Processing Systems*, vol. 36, 2024.

[23] R. Ran, X. Luo, W. Wang, T. Liu, G. Quan, X. Xu, C. Ding, and W. Wen, "Spencnn: orchestrating encoding and sparsity for fast homomorphically encrypted neural network inference," in *International Conference on Machine Learning*. PMLR, 2023, pp. 28 718–28 728.

[24] H. Chen, I. Chillotti, and Y. Song, "Improved bootstrapping for approximate homomorphic encryption," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2019, pp. 34–54.

[25] E. Lee, J.-W. Lee, J.-S. No, and Y.-S. Kim, "Minimax approximation of sign function by composite polynomial for homomorphic comparison," *IEEE Transactions on Dependable and Secure Computing*, vol. 19, no. 6, pp. 3711–3727, 2021.

[26] X. Jiang, M. Kim, K. Lauter, and Y. Song, "Secure outsourced matrix computation and application to neural networks," in *Proceedings of the 2018 ACM SIGSAC conference on computer and communications security*, 2018, pp. 1209–1222.

[27] W.-j. Lu, S. Kawasaki, and J. Sakuma, "Using fully homomorphic encryption for statistical analysis of categorical, ordinal and numerical data," *Cryptology ePrint Archive*, 2016.

[28] S. Wang and H. Huang, "Secure outsourced computation of multiple matrix multiplication based on fully homomorphic encryption," *KSII Transactions on Internet and Information Systems (TIIS)*, vol. 13, no. 11, pp. 5616–5630, 2019.

[29] W.-j. Lu, S. Kawasaki, and J. Sakuma, "Using fully homomorphic encryption for statistical analysis of categorical, ordinal and numerical data," *Cryptology ePrint Archive*, 2016.

[30] H. Huang and H. Zong, "Secure matrix multiplication based on fully homomorphic encryption," *The Journal of Supercomputing*, vol. 79, no. 5, pp. 5064–5085, 2023.

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

[32] A. Krizhevsky, G. Hinton *et al.*, "Learning multiple layers of features from tiny images," 2009.

[33] J. H. Ju, J. Park, J. Kim, D. Kim, and J. H. Ahn, "Neujeans: Private neural network inference with joint optimization of convolution and bootstrapping," *arXiv preprint arXiv:2312.04356*, 2023.

[34] A. Ebel, K. Garimella, and B. Reagen, "Orion: A fully homomorphic encryption compiler for private deep neural network inference," *arXiv preprint arXiv:2311.03470*, 2023.

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

[36] "Lattigo," https://github.com/tuneinsight/lattigo/tree/v5.0.2, 2023, ePFL-LDS, Tune Insight SA.

Appendix A
Rotation Optimzied Convolution blueprint

This section provides four **blueprint**s for use in the rotation optimized convolution algorithm: 2-depth Conv, 3-depth Conv, 4-depth Conv, and 5-depth Conv. All **blueprint**s are designed with consideration for the relationship between the RotationSum and CrossCombine process in rotation optimized convolution, and all **blueprint**s are designed with the goal of minimizing time. These blueprints may not be the only blueprints in each convolution scenario. There could be other blueprints with different values, but they are unlikely to have better performance than our blueprint. In other words, there are many correct blueprints, but not all of them are optimal. You can test other various correct blueprints in our github link.

Rotation optimized convolution has some heuristics nature, so here's a simple guide for creating **blueprint**s. Firstly, rather than focusing on the correlation between processes RotationSum and CrossCombine, create the **blueprint** backward from the final output of the convolution. In other words, decide how to divide the final output of the convolution for CrossCombine, then determine the order for performing RotationSum to move each channel to its position, and finally, decide on the order of kernel weights. This approach makes it easier to create the initial **blueprint**. Subsequently, you can further optimize the **blueprint** using the correlation between processes RotationSum and CrossCombine. Or if consuming additional depth is allowed, you can combine ciphertext that requires the same shift of rotation during RotationSum for further optimization on execution time.

The reason why there is no information available for some CONV1 situations and some convolution situations in 5-depth Conv **blueprint** is that the maximum depth that can be consumed in techniques using additional depth in convolution is $\log_2 q + 1$. Therefore, blueprints that cannot exist according to this criterion have not been defined.

It should be noted that in all expressions related to **blueprint** algorithms, the x-th element of a 1-dimensional **blueprint** A is denoted as A[x], and for some 2-dimensional **blueprint** B, the y-th element of B[x] is denoted as B[x][y]. The index of the first value of **blueprint** is 0, for instance, the first value of 1-dimensional **blueprint** A is A[0]. KernelBP is set to be consistent regardless of how many depths the convolution uses, for the sake of simplicity.

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 6: 2-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 7: 3-depth Conv **blueprint**

Appendix 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 determined differently for each component such as rotation optimized convolution algorithm and downsampling. TABLE 8 shows the values of parameters that are used in each component of the proposed algorithm. These parameters are also defined to represent the multiplexed parallel convolution described in [18], to avoid confusion, a similar notation as in [18] was 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 stride value of convolution. k_i, k_o are gaps of input and output ciphertext of convolution, respectively. Other variable can be defined as $t_i = \lceil \frac{c_i}{k_i^2} \rceil$, $t_o = \lceil \frac{c_o}{k_o^2} \rceil$, $p_i = 2^{\lfloor \log_2 (\frac{n_t}{k_i^2 h_i w_i t_i}) \rfloor}$, $p_o = 2^{\lfloor \log_2 (\frac{n_t}{k_o^2 h_o w_o t_o}) \rfloor}$, and $q = \lceil \frac{c_o}{p_i} \rceil$. Specifically, p_i and p_o represent the number of identical parallelly existing data within input ciphertext, and output ciphertext, respectively.

The parameter p_c is a new parameter proposed to denote the essential number of parallel data to be merged in Cross Combine. It is influenced by p_i and p_o, which can be defined as follows:

$$p_c = \begin{cases} p_i, & \text{if } s > 1 \text{ and } c_i p_i^2 / c_o p_o \geq 1 \\ p_o, & \text{otherwise,} \end{cases}$$

Component	CONV1	CONV2	CONV3s2	CONV3	CONV4s2	CONV4	DOWNSAMP1	DOWNSAMP2
f_h	3	3	3	3	3	3	-	-
f_w	3	3	3	3	3	3	-	-
s	1	1	2	1	2	1	-	-
h_i	32	32	32	16	16	8	32	16
h_o	32	32	16	16	8	8	16	8
w_i	32	32	32	16	16	8	32	16
w_o	32	32	16	16	8	8	16	8
c_i	3	16	16	32	32	64	16	32
c_o	16	16	32	32	64	64	32	64
k_i	1	1	1	2	2	4	1	2
k_o	1	1	2	2	4	4	2	4
t_i	3	16	16	8	8	4	16	8
t_o	16	16	8	8	4	4	8	4
p_i	8	2	2	4	4	8	2	4
p_o	2	2	4	4	8	8	4	8
p_c	2	2	4	4	4	8	-	-
q	2	8	16	8	16	8	-	-

TABLE 8: Parameters that are used in each ROTOPTCONV or ROTOPTDOWNSAMP process

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: 4-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 10: 5-depth Conv **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 11: KernelBP **blueprint**