Rotation Keyset Generation Strategy for Efficient Neural Networks using Homomorphic Encryption

Youyeon Joo[1], Hyunyoung Oh[2,*], and Yunheung Paek[1,*]

[1]Department of Electrical and Computer Engineering & ISRC, Seoul National University
[2]Department of AI·Software, Gachon University
{yyjoo@sor.snu.ac.kr, hyoh@gachon.ac.kr, ypaek@snu.ac.kr}

Abstract

Fully Homomorphic Encryption (FHE) enables secure computations on encrypted data for privacy-preserving machine learning. While recent FHE schemes support efficient SIMD-like operations, they require frequent data realignment through rotation, necessitating substantial memory for pre-computed rotation keys. We propose an application-aware, memory-efficient rotation keyset generation method that reduces memory consumption by 4.05–25.02× compared to conventional approaches, while maintaining similar latency. Our method achieves 2.02–2.32× faster rotations and 1.22–1.39× overall performance improvement compared to the power-of-two method.

1 Introduction

Fully Homomorphic Encryption (FHE) enables privacy-preserving computation on encrypted data, crucial for sensitive domains like healthcare and finance. CKKS [6] has become prominent in privacy-preserving machine learning (PPML) due to its support for batching multiple data points in a single ciphertext, allowing SIMD-like computations. A critical operation in CKKS-based PPML is *rotation*, which shifts encrypted data and the corresponding secret key. This operation requires specific keys for each index, presenting a significant challenge in memory consumption, especially for complex models like CNNs. For instance, LeNet-5 [8] requires rotation keys for 288 distinct indices.

Two main approaches have been proposed to address this challenge: the All-required method, which generates keys for all necessary rotation indices, and the Power-of-two method, which generates keys only for power-of-two indices. FHE libraries [9, 22] have adopted the Power-of-two optimization strategy, reducing memory usage but introducing computational overhead through recursive rotation calls. For example, a rotation of 7 positions requires three separate rotations (4 + 2 + 1). The All-required approach, implemented in FHE-MP-CNN [17], generates keys for all required indices, optimizing performance but increasing memory consumption to approximately 307 GB, nearly nine times more than the Power-of-two approach.

* Co-corresponding authors

Table 1: CKKS parameters used in this paper

Parameter	Description	In this paper
N	Polynomial degree	2^{16}
Q_0	Base modulus	2^{58}
Q_{min}	Lowest Q for computation	2^{184}
Q_{comp}	Highest Q for computation	2^{798}
Q_{max}	Maximum Q (during bootstrapping)	2^{1552}
PQ	Entire modulus	2^{1612}

We propose a memory-efficient rotation keyset generation approach for CNNs using FHE, balancing memory efficiency and computational performance. Our method is based on the observation that certain rotation indices are used infrequently or only at lower computation levels. We selectively avoid generating rotation keys for higher levels where certain rotations are less needed, significantly reducing memory consumption without substantially compromising performance. Our evaluation compares our method against the All-required and Power-of-two approaches, assessing their impact on CNN inferences over CKKS with different activation function evaluations. Results show our method achieves 4.05–25.02× less memory consumption than All-required, and 1.39-1.58× speedup compared to Power-of-two, with a modest 1.45-6.71× increase in memory footprint. These findings demonstrate our method's effectiveness in balancing memory efficiency and computational performance for FHE-based CNNs in resource-conscious environments.

2 Background

This section describes the preliminaries of our work. Table 1 lists CKKS parameters used in our work. It satisfies 128-bit

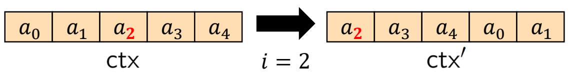

Figure 1: Example of a rotation by index 2

security, and the maximum computation level is 16 in this parameter setting.

2.1 CKKS

CKKS [4, 6] is a homomorphic encryption scheme that offers arithmetic operation over encrypted data. It gets a vector of complex numbers and encodes it into a plaintext, which is in a polynomial ring. A polynomial ring of degree N with modulus Q is defined as $\mathcal{R}_{Q,N} = \mathbb{Z}_Q[X]/(X^N+1)$ for a Power-of-two integer $N \geq 2$. The encryption of an encoded plaintext $\mathbf{m} \in \mathcal{R}_{Q,N}$ using secret key $\mathbf{s} \in \mathcal{R}_{Q,N}$ is computed as $\mathsf{ctx} = (c_0, c_1) = (\mathbf{m}+\mathbf{e}-\mathbf{a}\cdot\mathbf{s},\ \mathbf{a}) \in \mathcal{R}^2_{Q,N}$, where ctx denotes ciphertext, $\mathbf{a}$ and $\mathbf{e}$ are randomly sampled from a uniform distribution and an error distribution with standard deviation (e.g. gaussian distribution), respectively. The decryption of a ciphertext ctx is defined as $\langle \mathsf{ctx}, \mathbf{s} \rangle = c_0 + c_1 \cdot \mathbf{s} = \mathbf{m} + \mathbf{e}$.
Ciphertext Level. For the efficient computation, the ciphertext modulus Q is divided into a number of smaller moduli, and the ciphertext is also divided into smaller polynomials based on the Chinese remainder theorem (CRT). The ciphertext modulus is denoted as $Q = \Pi_{i=0}^{l} q_i$, where q_0 is a base modulus, and the others are prime moduli, which form a residue number system (RNS) [4]. In this representation, the number of non-base primes (i.e. l) denotes the ciphertext level, refers the available number of multiplication. In general, we need to switch the moduli chain down after multiplication and this process is to remove the last modulus of Q, dividing Q by a scaling factor Δ (i.e. $Q' = Q/\Delta$), then the level of ciphertext goes down l to $l-1$. This implies that the number of polynomials in a ciphertext is also reduced.
Basic Operations. CKKS supports three basic operations, Addition, Multiplication, and Rotation. The two arithmetics guarantee that given two ciphertexts $\mathsf{ctx}_1 = Enc(v_1)$ and $\mathsf{ctx}_2 = Enc(v_2)$, the decryption of $\mathsf{ctx}_1 \circledast \mathsf{ctx}_2$ is same as the $v_1 * v_2$, where $\circledast$ is a homomorphically same operator as an operator $*$ (i.e. $+/\times$) between two vectors. Rotation offers a cyclic shift of a ciphertext for a given index idx, which rotates the order of a vector v by idx slots. Figure 1 depicts an example of rotating a ciphertext with 5 slots by index 2.
Bootstrapping. When the modulus reaches Q_{min}, a ciphertext should recover the exhausted levels for more multiplications. Bootstrapping enables this process by starting at Q_{min}, raising the modulus to Q_{max} and performing sub-operations such as Rotations and Multiplications. The modulus becomes Q_{comp} with successive Multiplications during bootstrapping. As it consists of lots of sub-operations, it is the most time-consuming operation, taking 79.46× longer than the multiplication between ciphertexts at level 16.

Table 2: Relative latency of rotations for $N = 2^{16}$

Hamming weight	1	2	3	4	5	6	7
Relative latency	1.0	2.6	3.8	5.3	5.8	7.6	8.3

2.2 Rotation and Rotation Key

As the Rotation shifts both encrypted vector v and corresponding secret key $\mathbf{s}$ by given idx, the ciphertext needs to switch the secret key for correct decryption using rotation keys. For a given secret key $\mathbf{s}$, we denote $\mathbf{s_1}$ as a rotated secret key and $\mathbf{s_2}$ as $\mathbf{s}$. We assume our parameter uses a single special prime p (in Table 1). We first sample $\mathbf{a}^{(i)}$ from a uniform distribution for $0 \leq i \leq (1+l)$ and $\mathbf{e}'$ from error distribution. Then a rotation key rk is defined as $\mathsf{rk} = (\mathsf{rk}^{(0)}, \ldots, \mathsf{rk}^{(1+l)})$, where $\mathsf{rk}^{(0)} = (-\mathbf{a}^{(0)} \cdot \mathbf{s_2} + \mathbf{e}', \mathbf{a}^{(0)}) \pmod p$ and $\mathsf{rk}^{(1+j)} = (-\mathbf{a}^{(1+j)} \cdot \mathbf{s_2} + [p]_{q_j} \cdot \mathbf{s_1} + \mathbf{e}', \mathbf{a}^{(1+j)}) \pmod{q_j}$ for $0 \leq j \leq l$.

Rotation consists of three steps, modUp, multKey, modDown and generated rk is used at multKey step. Note that the vector v is encoded and encrypted, Rotation does not indicate the simple shift of coefficients of a polynomial. For a given ciphertext ctx that already rotated composed polynomials at level l, modUp raises the modulus from $Q = \Pi_{i=0}^{l} q_i$ to $PQ = p \cdot \Pi_{i=0}^{l} q_i$. We write the modulus raised ctx as ctx' and it is multiplied by rk. In details, ctx' is written as $(\mathsf{ctx}'^{(0)}, \ldots, \mathsf{ctx}'^{(1+l)})$ where $\mathsf{ctx}'^{(i)} = (c_0'^{(i)}, c_1'^{(i)})$ for $0 \leq i \leq (1+l)$. Then $\mathsf{ctx}'^{(0)} = \mathsf{ctx}'^{(0)} \cdot \mathsf{rk}^{(0)} \pmod p$ and $\mathsf{ctx}'^{(i)} = \mathsf{ctx}'^{(i)} \cdot \mathsf{rk}^{(i)} \pmod{q_i}$ for $0 \leq i \leq l$. After the modDown, the modulus becomes Q, and the secret key is also turned back to s. For cryptographic details, refer to the literature [11].

3 Motivation

Existing Approaches and Their Limitations. The generation of rotation keys for all possible indices $(N/2-1)$ is impractical due to excessive memory requirements, exceeding 30 TB in our parameter settings (Table 1). To address this, FHE libraries [9, 22] have implemented the power-of-two approach, reducing the number of keys to $2\log N - 1$ and decreasing memory consumption to 33.28 GB. However, this method introduces computational overhead through recursive rotation calls. Table 2 shows the relative latency increase for rotations based on their Hamming weight. For example, rotating by index 3 (2+1) is 2.6× slower than a single rotation.
Impact on FHE-based CNNs. In FHE-based Convolutional Neural Networks (CNNs), which require various rotation indices, this performance degradation is particularly pronounced. Many rotations in CNNs necessitate multiple recursive calls, leading to significant slowdowns. Our evaluations show performance degradation of up to 1.38× in CNN inference tasks using the power-of-two method. These findings highlight the need for a balanced approach that optimizes

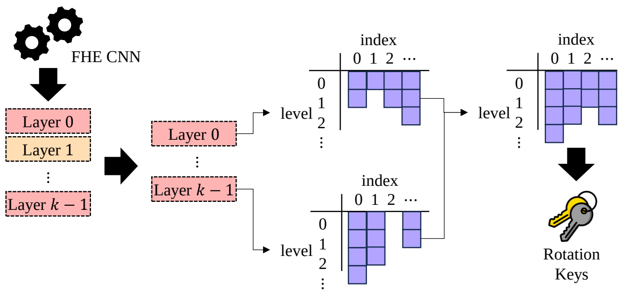

Figure 2: Overview of our approach

both memory efficiency and computational performance in FHE-based neural networks, which is the focus of our proposed method.

4 Optimizing Rotation Keyset Generation

4.1 Key Observation and Approach

While recent work like FHE-MP-CNN [17] has improved performance by generating all required indices, this approach still incurs significant memory overhead. Our method builds upon this foundation, introducing a crucial optimization based on a key observation: ciphertext levels for rotations are often set lower than the maximum level (e.g. only required at level=3), despite rotation keys being generated up to the maximum level by defined parameter set [1, 3, 9, 22].

This insight allows us to propose a novel approach that considers both rotation indices and ciphertext levels when generating keys. By analyzing the neural network computation to identify the specific ciphertext level status at rotation calls, we can generate only the keys that are actually needed, significantly reducing memory requirements without compromising performance.

4.2 Optimized Rotation Keyset Generation Process

Our approach, illustrated in Figure 2, focuses on minimizing the rotation keyset size through a systematic analysis of FHE CNN. We assume the availability of pre-generated FHE CNN computation code, which is standard in this field [7, 20].

Layer-Specific Rotation Analysis. We begin by identifying layers that require rotations, focusing on non-SIMD operations such as convolution, pooling, and fully-connected layers. This targeted analysis allows us to exclude layers like batch normalization and activation, which operate in a SIMD manner and do not require rotations.

Comprehensive Rotation Table Generation. The core of our optimization lies in creating a detailed rotation table. This table maps rotation indices against ciphertext levels, providing a comprehensive view of rotation requirements across the network. By deterministically identifying the specific ciphertext levels and indices at which rotations are called in each layer, we construct a minimal set of required rotation keys. This process reveals patterns and opportunities for optimization that are not apparent when considering indices alone. For instance, we may find that certain high-level rotations are never used in practice, or that some indices are only required at lower ciphertext levels.

Efficient Rotation Key Generation. Based on the rotation table, we implement a level-aware key generation strategy. This approach ensures that we only generate keys for the levels at which they are actually needed. Importantly, we account for the RNS representation by generating keys for all levels up to and including the required level for each index. For example, if a layer requires a rotation at index *idx* at level k, we generate rotation keys for this index from level 0 to k. This strategy ensures compatibility with the RNS structure while avoiding the generation of unnecessary high-level keys. This level-aware approach represents a significant advancement over previous methods. By precisely tailoring the key generation to the actual needs of the network, we achieve substantial memory savings without sacrificing the ability to perform any required rotation. Moreover, this method provides valuable insights into the rotation patterns within FHE CNNs, potentially informing future optimizations in network design and implementation.

5 Evaluation

Environment and Benchmarks. We used a machine with two Intel Xeon Gold 6326 CPUs (2.9 GHz) and 1 TB main memory. All experiments were conducted using the HEaaN library [9], with a multi-threaded preset compile option (64 threads). We evaluate ResNet-20 [12] and SqueezeNet [14] using the CIFAR-10 dataset. To show the effectiveness of our approach, we compare ours to two different FHE CNNs, one using low-depth ReLU (`AESPA`) and the other using high-depth ReLU (`MPCNN`). AESPA is re-trained using the method presented in their literature [21]. MPCNN does not re-train the model but simply replaces the ReLUs with composite approximated polynomials [18] consuming multiple multiplicative depths. Note that we evaluate only the rotation keys required during the CNN computation, thus we exclude the rotation keys called during bootstrapping.

Compared Baselines. We compare our method against the two baselines that have been previously introduced, Power-of-two and All-required. Power-of-two is a method that generates a default rotation keyset of power-of-two indices back-and-forth, while All-required generates all the keys for all rotations called during inference. Note that consequently, the keyset generated by All-required depends on the benchmark, while Power-of-two always generates the same keyset for all the benchmarks.

Table 3: Comparison of Memory Consumption

Memory (GB)	AESPA		MPCNN	
	ResNet-20	SqueezeNet	ResNet-20	SqueezeNet
Power-of-Two	33.29	33.29	33.29	33.29
All-Required	307.09	241.39	307.09	241.39
Ours	63.07	49.96	20.79	9.65

Table 4: Comparison of Latency

Latency (s)	ResNet-20		SqueezeNet	
	Power-of-Two	Ours	Power-of-Two	Ours
Rotation	55.04	27.15	48.73	20.94
Bootstrapping	23.52	23.97	55.86	56.13
Others	20.19	19.94	49.54	48.97
Total	98.75	71.06	154.13	126.04

5.1 Memory Consumption

All-required. Our approach generates all required indices but selective levels. Thus, ours reduced the memory usage by 4.05–4.86× for AESPA applied models and 14.77–25.02× for MPCNN models compared to All-required. As we mentioned earlier, MPCNN runs all rotation-containing layers at a low level, requiring less memory for keys compared to AESPA.

Power-of-two. Compared to Power-of-two, our approach for AESPA requires more memory. AESPA consists of Rotations at the various levels because it consumes only one multiplicative depth to run ReLU, the rotation contained layers are at various levels. Therefore, it requires more rotation keys for higher levels, 1.83–1.45× more memory. We will discuss the related effect of latency later. Meanwhile, ours generates fewer rotation keys for MPCNN, reduced by 1.65–3.56×. This is because it requires all indices, but we only generate lower-level keys.

5.2 End-to-End Inference Latency

Table 4 shows the end-to-end inference latencies using our approach and Power-of-Two. The latency of our approach is nearly identical to All-required because neither requires recursive Rotation calls. As we generated specific indices for rotations during bootstrapping, the bootstrapping latency of both Power-of-two and ours are almost same. The main difference is in Rotation, we reduced the latency of Rotation 2.02×, 2.32×, respectively. This indicates that many rotations in both models are not power-of-two indices and thus incur recursive Rotation calls. By generating all indices used for a model, ours improves Rotation latency, which in turn enhances the performance of FHE CNN model inference, 1.39× for ResNet-20 and 1.22× for SqueezeNet.

Memory Efficiency and Latency Trade-offs. Our approach ensures that the total size of rotation keys is smaller than (or equal to) that of All-required. Because prior approaches generate keys up to the maximum level that defined at parameter setting, we selectively generate keys by levels as much as the model needed. Compared to Power-of-two, ours does not guarantee lower memory usage, but offers a comparable trade-off in inference latency.

6 Related Works and Discussion

Key Extension. Several works [13, 16] adopted methods that extend the key size in order to enhance the performance. They decompose certain keys (including rotation keys) that induce more memory overhead but reduce the NTT/INTT complexity. Our work is orthogonal to theirs in that their works increase the size of each rotation key, while our work aims to reduce the number of selections. In fact, our work can be applied on top of their works to reduce their memory overhead while benefiting from their performance enhancements.

Application to diverse neural networks. Just as AESPA and MPCNN, researchers devised numerous techniques for computing neural networks using FHE. For example, AutoFHE [2] dynamically allocates different polynomial approximations to different ReLUs in a CNN selected by running a neural architecture search that predicts its impact on accuracy and performance. Our work can also enhance those other neural network inferences over FHE in that there are no limitations in analyzing the used rotation indices and generate only those that are required considering the levels. We envision that a future work extending ours by generalizing the rotation key index selection process could provide a general solution to this matter.

7 Conclusion

In this work, we introduced a memory-efficient rotation keyset generation approach for FHE CNNs. By selectively generating rotation keys only for specific indices and levels required by the model, our approach achieves up to 25.02× reduction in memory consumption without sacrificing performance. Experimental results show that ours maintains comparable inference latency to baseline methods, with up to 2.32× faster rotations than the Power-of-two method. We envision that our approach enables the practical deployment of FHE CNNs in memory-constrained environments, such as edge devices, making PPML more feasible in real-world applications.

Acknowledgments

This work was supported by the BK21 FOUR program of the Education and Research Program for Future ICT Pioneers, Seoul National University in 2024. This work was supported by the National Research Foundation of Korea(NRF) grant funded by the Korea government(MSIT) (No. RS-2022-00166529, RS-2023-00277326). This work was supported by the Institute of Information & Communications Technology Planning & Evaluation (IITP) grant funded by the Korea government(MSIT) (No. 2021-0-00528, Development of Hardware-centric Trusted Computing Base and Standard Protocol for Distributed Secure Data Box, No. RS-2023-00277060, Development of open edge AI SoC hardware and software platform, No. RS-2024-00337414, Binary Micro-Security Patch Technology Applicable with Limited Reverse Engineering Capability under SW Supply Chain Environments, No.RS-2024-00438729, Development of Full Lifecycle Privacy-Preserving Techniques using Anonymized Confidential Computing). This work was supported by Institute of Information & communications Technology Planning & Evaluation (IITP) under the artificial intelligence semiconductor support program to nurture the best talents (IITP-2023-RS-2023-00256081) grant funded by the Korea government(MSIT). This work was supported by the Korea Planning & Evaluation Institute of Industrial Technology(KEIT) grant funded by the Korea Government(MOTIE) (No. RS-2024-00406121, Development of an Automotive Security Vulnerability-based Threat Analysis System(R&D)).

References

[1] Lattigo v5. Online: https://github.com/tuneinsight/lattigo, November 2023. EPFL-LDS, Tune Insight SA.

[2] Wei Ao and Vishnu Naresh Boddeti. {AutoFHE}: Automated adaption of {CNNs} for efficient evaluation over {FHE}. In *33rd USENIX Security Symposium (USENIX Security 24)*, pages 2173–2190, 2024.

[3] Ahmad Al Badawi, Andreea Alexandru, Jack Bates, Flavio Bergamaschi, David Bruce Cousins, Saroja Erabelli, Nicholas Genise, Shai Halevi, Hamish Hunt, Andrey Kim, Yongwoo Lee, Zeyu Liu, Daniele Micciancio, Carlo Pascoe, Yuriy Polyakov, Ian Quah, Saraswathy R.V., Kurt Rohloff, Jonathan Saylor, Dmitriy Suponitsky, Matthew Triplett, Vinod Vaikuntanathan, and Vincent Zucca. OpenFHE: Open-source fully homomorphic encryption library. Cryptology ePrint Archive, Paper 2022/915, 2022. https://eprint.iacr.org/2022/915.

[4] Jung Hee Cheon, Kyoohyung Han, Andrey Kim, Miran Kim, and Yongsoo Song. A full rns variant of approximate homomorphic encryption. In *Selected Areas in Cryptography–SAC 2018: 25th International Conference, Calgary, AB, Canada, August 15–17, 2018, Revised Selected Papers 25*, pages 347–368. Springer, 2019.

[5] Jung Hee Cheon, Minsik Kang, Taeseong Kim, Junyoung Jung, and Yongdong Yeo. Batch inference on deep convolutional neural networks with fully homomorphic encryption using channel-by-channel convolutions. *IEEE Transactions on Dependable and Secure Computing*, 2024.

[6] Jung Hee Cheon, Andrey Kim, Miran Kim, and Yongsoo Song. Homomorphic encryption for arithmetic of approximate numbers. In *Advances in Cryptology–ASIACRYPT 2017: 23rd International Conference on the Theory and Applications of Cryptology and Information Security, Hong Kong, China, December 3-7, 2017, Proceedings, Part I 23*, pages 409–437. Springer, 2017.

[7] Seonyoung Cheon, Yongwoo Lee, Dongkwan Kim, Ju Min Lee, Sunchul Jung, Taekyung Kim, Dongyoon Lee, and Hanjun Kim. {DaCapo}: Automatic bootstrapping management for efficient fully homomorphic encryption. In *33rd USENIX Security Symposium (USENIX Security 24)*, pages 6993–7010, 2024.

[8] Hyunmin Choi, Jihun Kim, Seungho Kim, Seonhye Park, Jeongyong Park, Wonbin Choi, and Hyoungshick Kim. Unihenn: Designing faster and more versatile homomorphic encryption-based cnns without im2col. *IEEE Access*, 2024.

[9] CrytptoLab. Official HEaaN Library, 2023. https://heaan.it/.

[10] Roshan Dathathri, Olli Saarikivi, Hao Chen, Kim Laine, Kristin Lauter, Saeed Maleki, Madanlal Musuvathi, and Todd 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*, pages 142–156, 2019.

[11] Kyoohyung Han and Dohyeong Ki. Better bootstrapping for approximate homomorphic encryption. In *Cryptographers' Track at the RSA Conference*, pages 364–390. Springer, 2020.

[12] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[13] Intak Hwang, Jinyeong Seo, and Yongsoo Song. Optimizing he operations via level-aware key-switching framework. In *Proceedings of the 11th Workshop on Encrypted Computing & Applied Homomorphic Cryptography*, pages 59–67, 2023.

[14] Forrest N Iandola. Squeezenet: Alexnet-level accuracy with 50x fewer parameters and< 0.5 mb model size. *arXiv preprint arXiv:1602.07360*, 2016.

[15] Chiraag Juvekar, Vinod Vaikuntanathan, and Anantha Chandrakasan. {GAZELLE}: A low latency framework for secure neural network inference. In *27th USENIX security symposium (USENIX security 18)*, pages 1651–1669, 2018.

[16] Miran Kim, Dongwon Lee, Jinyeong Seo, and Yongsoo Song. Accelerating he operations from key decomposition technique. In *Annual International Cryptology Conference*, pages 70–92. Springer, 2023.

[17] Eunsang Lee, Joon-Woo Lee, Junghyun Lee, Young-Sik Kim, Yongjune Kim, Jong-Seon No, and Woosuk Choi. Low-complexity deep convolutional neural networks on fully homomorphic encryption using multiplexed parallel convolutions. In *International Conference on Machine Learning*, pages 12403–12422. PMLR, 2022.

[18] Eunsang Lee, Joon-Woo Lee, Jong-Seon No, and Young-Sik Kim. Minimax approximation of sign function by composite polynomial for homomorphic comparison. *IEEE Transactions on Dependable and Secure Computing*, 19(6):3711–3727, 2021.

[19] Joon-Woo Lee, HyungChul Kang, Yongwoo Lee, Woosuk Choi, Jieun Eom, Maxim Deryabin, Eunsang Lee, Junghyun Lee, Donghoon Yoo, Young-Sik Kim, et al. Privacy-preserving machine learning with fully homomorphic encryption for deep neural network. *iEEE Access*, 10:30039–30054, 2022.

[20] Yongwoo Lee, Seonyeong Heo, Seonyoung Cheon, Shinnung Jeong, Changsu Kim, Eunkyung Kim, Dongyoon Lee, and Hanjun Kim. Hecate: Performance-aware scale optimization for homomorphic encryption compiler. In *2022 IEEE/ACM International Symposium on Code Generation and Optimization (CGO)*, pages 193–204. IEEE, 2022.

[21] Jaiyoung Park, Michael Jaemin Kim, Wonkyung Jung, and Jung Ho Ahn. Aespa: Accuracy preserving low-degree polynomial activation for fast private inference. *arXiv preprint arXiv:2201.06699*, 2022.

[22] Microsoft SEAL (release 4.1). https://github.com/Microsoft/SEAL, January 2023. Microsoft Research, Redmond, WA.

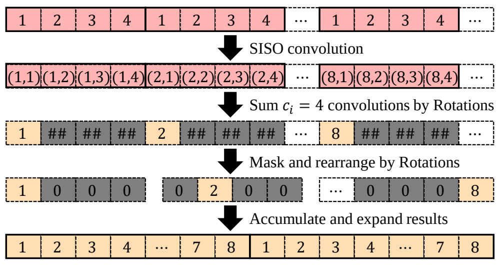

Figure 3: Example of MPConv [17]

A Rotations in Encrypted CNN

Several works [5, 10, 15, 19] proposed efficient convolution methods to compute convolutional neural networks (CNN). Multiplexed parallel convolution (MPConv) [17] aims to use a minimal number of ciphertexts considering bootstrapping. It utilizes SISO convolution [15] and packs as many pixels as possible in a single ciphertext between layers, which is achieved by Rotations. Figure 3 shows an example of MPConv with 4 input channels and 8 output channels by strides 2. For slot utilization, MPConv fills useless slots with valid values by masking and Rotations even if convolution output is obtained after the second step. This is necessary because a CNN is composed of successive layers (i.e. The output of layer k is going to be the input of layer $k+1$), pixels should be re-arranged to apply same convolution method.