RESEARCH-ARTICLE

NeuJeans: Private Neural Network Inference with Joint Optimization of Convolution and FHE Bootstrapping

JAE-HYUNG JU, Seoul National University, Seoul, South Korea

JAIYOUNG PARK, Seoul National University, Seoul, South Korea

JONGMIN KIM, Seoul National University, Seoul, South Korea

MINSIK KANG, Seoul National University, Seoul, South Korea

DONGHWAN KIM, Seoul National University, Seoul, South Korea

JUNG-HEE CHEON, Seoul National University, Seoul, South Korea

View all

Published: 02 December 2024

Citation in BibTeX format

CCS '24: ACM SIGSAC Conference on Computer and Communications Security
October 14 - 18, 2024
UT, Salt Lake City, USA

Conference Sponsors:
SIGSAC

NeuJeans: Private Neural Network Inference with Joint Optimization of Convolution and FHE Bootstrapping

Jae Hyung Ju*
Jaiyoung Park*
Seoul National University
Seoul, Republic of Korea
hpotato@snu.ac.kr
jypark@scale.snu.ac.kr

Jongmin Kim
Seoul National University
Seoul, Republic of Korea
jongmin.kim@snu.ac.kr

Minsik Kang
Seoul National University
Seoul, Republic of Korea
kaiser351@snu.ac.kr

Donghwan Kim
Seoul National University
Seoul, Republic of Korea
eastflame@snu.ac.kr

Jung Hee Cheon
Seoul National University,
CryptoLab Inc.
Seoul, Republic of Korea
jhcheon@snu.ac.kr

Jung Ho Ahn
Seoul National University
Seoul, Republic of Korea
gajh@snu.ac.kr

Abstract

Fully homomorphic encryption (FHE) is a promising cryptographic primitive for realizing private neural network inference (PI) services by allowing a client to fully offload the inference task to a cloud server while keeping the client data oblivious to the server. This work proposes NeuJeans, an FHE-based solution for the PI of deep convolutional neural networks (CNNs). NeuJeans tackles the critical problem of the enormous computational cost for the FHE evaluation of CNNs. We introduce a novel encoding method called Coefficients-in-Slot (CinS) encoding, which enables multiple convolutions in one HE multiplication without costly slot permutations. We further observe that CinS encoding is obtained by conducting the first several steps of the Discrete Fourier Transform (DFT) on a ciphertext in conventional Slot encoding. This property enables us to save the conversion between CinS and Slot encodings as bootstrapping a ciphertext starts with DFT. Exploiting this, we devise optimized execution flows for various two-dimensional convolution (conv2d) operations and apply them to end-to-end CNN implementations. NeuJeans accelerates the performance of conv2d-activation sequences by up to 5.68× compared to state-of-the-art FHE-based PI work and performs the PI of a CNN at the scale of ImageNet within a mere few seconds.

CCS Concepts

• **Security and privacy** → **Privacy-preserving protocols**; *Web application security*; • **Computing methodologies** → **Neural networks**.

*Both authors contributed equally to this research.

CCS '24, October 14–18, 2024, Salt Lake City, UT, USA

ACM ISBN 979-8-4007-0636-3/24/10
https://doi.org/10.1145/3658644.3690375

Keywords

Homomorphic Encryption, Privacy-Preserving Machine Learning, Convolutional Neural Network, Coefficients-in-Slot Encoding

ACM Reference Format:
Jae Hyung Ju, Jaiyoung Park, Jongmin Kim, Minsik Kang, Donghwan Kim, Jung Hee Cheon, and Jung Ho Ahn. 2024. NeuJeans: Private Neural Network Inference with Joint Optimization of Convolution and FHE Bootstrapping. In *Proceedings of the 2024 ACM SIGSAC Conference on Computer and Communications Security (CCS '24), October 14–18, 2024, Salt Lake City, UT, USA.* ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/3658644.3690375

1 Introduction

The advances in machine learning (ML) have opened up a vast pool of cloud-based services, even covering highly private areas such as healthcare [17, 41], finance [1], and home surveillance systems [5]. Such services can be provided simply by sending user data to a cloud server, which runs inference on proprietary neural networks. However, such a method exposes the input user data to the service provider. With the increasing concerns and the resultant regulations [15, 43] on privacy, private inference (PI), which incorporates methods to perform neural network inference while 1) the server remains oblivious to the input user data and 2) the user obtains the inference result without getting any additional information of the proprietary neural network, is garnering immense attention.

Several cryptographic primitives enable PI, among which homomorphic encryption (HE) stands out due to its capability to perform a sequence of operations on encrypted data (i.e., ciphertexts) without any online user intervention. When using HE, the user first generates several public keys required for computation in a one-time offline phase. Once the server holds the generated public keys, the user only needs to perform the encryption of the input data and decryption of the final result. The server is in charge of the whole computation process; it constructs an arithmetic circuit C for the inference task and evaluates the circuit using HE operations (HE ops) on the user's ciphertexts.

However, evaluating complex circuits such as convolutional neural network (CNN) inference, the main target of this work, is challenging due to the critical constraints of HE. First, there exists a

limit to the number of HE ops that can be performed on a ciphertext. In a subclass of HE schemes referred to as fully HE (FHE), a special *bootstrapping* operation resets the limit, enabling more HE ops on the ciphertext. However, bootstrapping adds an enormous amount of computation to the circuit; thus, a large portion of FHE-based PI's execution time is spent on performing bootstrapping rather than useful HE ops.

Moreover, manipulating the order of encrypted data is costly. In HE, a vector is encrypted into a ciphertext and the only viable option for the server to reorder the data inside this vector is to perform an HE op that cyclically rotates the vector (HRot). As HRot is not cheap, prior studies have developed numerous algorithms and data rearrangement methods to minimize the cost for data reorganization in convolutional layers [21, 24, 25, 29, 32, 46].

Despite these efforts, even the state-of-the-art FHE CNN implementations [25, 29, 32] exhibit impractically long latency, taking minutes per inference for relatively simple problems such as CIFAR10/100 and MNIST. This hinders their adoption in real-world ML services.

Henceforth, we will refer to the evaluation of a convolutional layer as **conv2d** to distinguish it from a plain convolution ($*$) operation.

1.1 Contribution

In this paper, we propose **NeuJeans**, a set of algorithmic advancements on ring learning with errors (RLWE)-based FHE to evaluate neural network models. Specifically, we implement our algorithms on CKKS [9], a popular FHE scheme for machine learning. NeuJeans incorporates a novel *CinS encoding* method for CKKS ciphertexts, which enables an efficient conv2d algorithm (§1.1.1). This encoding method, along with the conv2d algorithm, allows for the reformation of the bootstrapping circuit, which reduces the cost of bootstrapping and its adjacent operations by disassembling and reassembling the computational blocks in the circuit (§1.1.2). Additionally, we reorder the components within a CNN layer in a FHE-friendly manner to optimize the execution flows for various conv2d types (§1.1.3). Overall, NeuJeans overcomes the limitations of FHE-based PI of CNNs by minimizing the computational overhead and enabling end-to-end inference of a complex CNN within a few seconds.

1.1.1 CinS encoding and efficient conv2d algorithm. In CKKS, a message vector $\mathbf{m}$ is first encoded into a plaintext $\langle \mathbf{m} \rangle$, which is then encrypted into a ciphertext $[\langle \mathbf{m} \rangle]$. The original encoding method in CKKS is known as *Slot encoding*, which allows for element-wise multiplication ($\odot$ in Eq. 1) and addition between message vectors in the encrypted state. The core part of Slot encoding involves performing an inverse discrete Fourier transform (IDFT) on $\mathbf{m}$. PI studies [21, 25] have later identified that when using *Coefficient encoding*, which skips the IDFT, encrypted multiplication (HMult) of ciphertexts results in convolution ($*$ in Eq. 1) between the messages. Previously, with Slot encoding, HRot operations were necessary to aggregate element-wise multiplication results within a ciphertext. Coefficient encoding reduces the need for HRot operations in conv2d processes, thereby optimizing computational efficiency.

However, even the Coefficient encoding approach faces significant inefficiencies due to two primary reasons. First, it still requires heavy rotations after each conv2d operation. Second, it cannot efficiently evaluate element-wise operations. These limitations arise mainly due to HMult being equivalent to a global convolution across the entire vectors, disregarding the actual convolution length required for a specific conv2d.

To mitigate this issue, we introduce a new encoding, a Coefficients-in-Slot (CinS) encoding, best suited to implement various conv2d evaluations with the CKKS scheme. Our CinS encoding removes the inefficiency by making HMult result in a partial local convolution for each evenly-partitioned slice of the vectors; i.e., given $\mathbf{m} = (\mathbf{m}_0|\mathbf{m}_1|\cdots|\mathbf{m}_{C-1})$ and $\mathbf{m}' = (\mathbf{m}'_0|\mathbf{m}'_1|\cdots|\mathbf{m}'_{C-1})$ for $\mathbf{m}_i, \mathbf{m}'_i$ in a subring, we obtain

$$\begin{aligned}&\langle \mathbf{m} \rangle_{\text{CinS}} \cdot \langle \mathbf{m}' \rangle_{\text{CinS}} = \langle \mathbf{m} *_{\text{local}} \mathbf{m}' \rangle_{\text{CinS}} \\ &= \langle \mathbf{m}_0 *_{\text{part}} \mathbf{m}'_0 | \mathbf{m}_1 *_{\text{part}} \mathbf{m}'_1 | \cdots | \mathbf{m}_{C-1} *_{\text{part}} \mathbf{m}'_{C-1} \rangle_{\text{CinS}},\end{aligned}$$

where $*_{\text{part}}$ denotes the convolution of two polynomials within a subring of the given plaintext space.

We note that the conventional encodings provides the following HE ops:

$$\begin{aligned}\text{Slot encoding: } \langle \mathbf{m} \rangle_{\text{slot}} \cdot \langle \mathbf{m}' \rangle_{\text{slot}} &= \langle \mathbf{m} \odot \mathbf{m}' \rangle_{\text{slot}} \\ \text{Coefficient encoding: } \langle \mathbf{m} \rangle_{\text{coeff}} \cdot \langle \mathbf{m}' \rangle_{\text{coeff}} &= \langle \mathbf{m} * \mathbf{m}' \rangle_{\text{coeff}}.\end{aligned} \tag{1}$$

Based on the CinS encoding method, we can flexibly select the size of a slice in the vector according to the required convolution length of a specific conv2d and develop dedicated conv2d algorithms. Our conv2d proposal using CinS encoding combines the best of both worlds: Coefficient encoding and Slot encoding. In our conv2d algorithm, multiplication induces rotation-less convolution in each slice between input images and kernels as in Coefficient encoding, while rotation induces partial sum of intermediate convolution result as in Slot encoding. Our optimized conv2d algorithms significantly reduce the computational complexity, performing fewer multiplications and rotations (see Table 1).

We note that [8] also suggested leveraging a subring structure to deal with a small number ℓ of slots rather than $N/2$ full slots, which is a widely adopted technique to reduce bootstrapping latency for sparsely-packed ciphertext in CKKS. Compared to [8], our CinS encoding leverages isomorphism in subring, which leads to favorable homomorphic property in conv2d while utilizing full $N/2$ slots.

1.1.2 Fusing conv2d with boostrapping. Although CinS encoding is well-suited to conv2d, it must be converted to Slot encoding during CNN inference to perform element-wise operations, such as ReLU. This conversion is achieved using the DFT ($\mathcal{T}$) and IDFT ($\mathcal{T}^{-1}$) matrices. We first factorize the DFT matrix $\mathcal{T}$ into $\mathcal{T}'_2\mathcal{T}'_1$ using the Cooley-Tukey DFT factorization [13]. Then, the conversion from Slot encoding to CinS encoding is possible by multiplying the ciphertext with $\mathcal{T}'_1$; the opposite is possible by multiplying the ciphertext with $\mathcal{T}^{-1}\mathcal{T}'_2$.

However, evaluating these matrix-vector multiplications is highly costly in the encrypted state. Nevertheless, we identify that these evaluations are already present in the bootstrapping process, which enables us to merge conv2d with bootstrapping to perform the conversions at no additional cost. We go even further by eliminating the need for $\mathcal{T}'_2$ multiplication by fusing the matrix with conv2d

Table 1: Comparison of FHE conv2d using various encoding methods. N is the ring degree in FHE, f is the width/height of kernels, w is the width/height of images, and $C = C_{in} = C_{out}$ is the number of input/output channels that fit in N. dwconv2d denotes depthwise conv2d.

Conv2d type	Encoding method	# of PMult	# of HRot	# of elements in a msg	Level consumption	# of plaintexts
Conv2d	Slot encoding [24]	$2f^2C$	$2f^2 + 2C - 4$	$N/2$	1	$2f^2C$
	Coefficient encoding [25]	$2C - 1$	$C - 1$	N	1	C
	Ours (CinS encoding)	**C**	$\mathbf{2\sqrt{C} - 2}$	**N**	**0**	**C**
Dwconv2d	Slot encoding [24][1]	$2f^2$	$2f^2 - 2$	$N/2$	1	$2f^2C$
	Coefficient encoding [25][1]	$2C - 1$	$C - 1$	N	1	C
	Ours (CinS encoding)	**1**	**0**	**N**	**1**	**C**

[1] Previous work did not include dwconv2d, so we independently implemented dwconv2d based on their encoding methods.

kernel weights using the property that the $\mathcal{T}_2'$ matrix and the kernel weights share a similar structure. As a result, the fused conv2d evaluation has the same cost as a standard conv2d evaluation; thus, $\mathcal{T}_2'$ multiplication becomes effectively free in the online phase.

1.1.3 FHE-friendly execution flows. In applying our conv2d algorithms to end-to-end CNN inference, we carefully design the execution flows such that the number of expensive operations (bootstrapping and HRot in particular) is minimized. Flexibility is required for conv2d variants, including downsampling conv2d and depthwise conv2d (dwconv2d) [12]. Handling downsampling conv2d is especially important because it produces sparsely packed ciphertexts, severely degrading the throughput of HE ops. It is possible to merge multiple sparsely packed ciphertexts into a densely packed ciphertext, but the process involves a lot of bootstrapping. To tackle this problem, we modify the execution flow by introducing a *decomposed downsampling conv2d* algorithm, which involves s^2 times fewer bootstrapping operations for the stride s of a downsampling conv2d. We also devise an efficient dwconv2d algorithm that can be used along with CinS encoding to perform the dwconv2d evaluation with only a single HE op (see Table 1).

1.1.4 Implementation and evaluation. We implemented the conv2d layers and representative CNN models, ResNet18/50, and MobileNetV2 for the ImageNet dataset, using NeuJeans. NeuJeans achieves up to 5.67× improved performance for the conv2d-activation sequence compared to prior state-of-the-art FHE-based CNN work [25]. For the ImageNet dataset, NeuJeans takes **5.35 seconds per inference** with a variant of **ResNet18** and **56.08 seconds** with **ResNet50**. This is remarkably fast considering that prior FHE-based CNN PI studies [25, 29, 32] report minutes to hours of execution time even for smaller networks targeting CIFAR10/100.

1.2 Related Work

FHE-based PI offers robust privacy guarantees by ensuring that the client only learns the final inference result. Moreover, it effectively offloads the computational workload to the server, minimizing the client's role to encrypting and decrypting the input and output data, enhancing its usability in resource-constrained environments such as mobile devices. Additionally, this approach aligns well with hardware acceleration efforts, as evidenced by numerous studies exploring HE hardware acceleration studies [23, 27, 28, 39, 40]. While there exist alternative PI research directions with various cryptographic constructions [21, 24, 33–35, 38, 44], we focus on FHE-based implementation to fully leverage the advantageous properties inherent to FHE.

The unique algebraic structure of HE prevents direct translation of conventional conv2d algorithms (e.g., im2col-based algorithms) into efficient HE algorithms. To mitigate this issue, Gazelle [24] proposes a HE-specific conv2d algorithm on Slot-encoded ciphertexts to minimize the number of HE ops (see Table 1). Following studies [29, 32, 46] extend Gazelle's algorithm by improving on how multiple channels are packed in a ciphertext. Meanwhile, Cheetah [21] introduces a conv2d algorithm that does not require any HRot ops using the Coefficient-encoding method. [25], the state-of-the-art work in FHE-based PI which we set as our baseline, adapts Cheetah's conv2d algorithm to FHE circumstances and implements end-to-end inference of ResNet models. HyPHEN [26] goes beyond optimizing a single convolution to target an entire block composed of multiple layers (e.g., a residual block in ResNet). By expanding the design space, HyPHEN demonstrates a significant reduction in the number of HE operations and memory footprint, and introduces a novel trade-off between the number of bootstrapping operations and other HE operations. However, HyPHEN requires low-degree polynomial activation to be effective. Despite these advancements, significant inefficiencies still impede the practical adoption of FHE-based PI; [25] requires 368 seconds for the server to perform PI on a single CIFAR10 image with the ResNet20 model.

2 Preliminaries

We represent vectors with bold lower-case letters (e.g., $\mathbf{m}$). All vectors are column vectors. $\lfloor \cdot \rfloor$, $\lceil \cdot \rceil$, and $\lfloor \cdot \rceil$ represent floor, ceiling, and rounding operations. $\langle \cdot \rangle$ and $[\cdot]$ represent encoding and encryption. $\mathbb{Z}$, $\mathbb{R}$, and $\mathbb{C}$ denote the set of integer, real, and complex numbers. $\mathbb{Z}_q = \mathbb{Z}/q\mathbb{Z}$ is a ring of integers modulo q. We denote a $(2N)$-th cyclotomic polynomial ring $\mathbb{Z}_q[X]/(X^N + 1)$ by $\mathcal{R}_q$, whose *degree* N is a power-of-two integer (typically, 2^{16}). Table 6 summarizes the notations we use in this paper.

2.1 Homomorphic Encryption

Homomorphic encryption (HE) is a set of encryption schemes that allow performing operations on encrypted data without decryption, and the result is correct with respect to the computation

on plaintext. An HE scheme is a collection of algorithms $HE = (KeyGen, Enc, Dec, Eval)$ with the following syntax:

- KeyGen(1^λ) $\rightarrow$ (pk, sk). Given the security parameter λ, the KeyGen algorithm outputs a public key pk and a secret key sk.
- Enc($\langle \mathbf{m} \rangle$; pk) $\rightarrow [\langle \mathbf{m} \rangle]$. Given a public key pk and a plaintext $\langle \mathbf{m} \rangle$ as input, the encryption algorithm outputs a ciphertext $[\langle \mathbf{m} \rangle]$.
- Dec($[\langle \mathbf{m} \rangle]$; sk) $\rightarrow \langle \mathbf{m} \rangle$. Given a secret key sk and a ciphertext $[\langle \mathbf{m} \rangle]$ as input, the decryption algorithm outputs the plaintext $\langle \mathbf{m} \rangle$.
- Eval(C; $[\langle \mathbf{m_1} \rangle], \ldots, [\langle \mathbf{m_k} \rangle]$) $\rightarrow [\langle \mathbf{m'} \rangle]$. For an arithmetic circuit C, the evaluation algorithm outputs a new ciphertext $[\langle \mathbf{m'} \rangle]$ encrypting $\mathbf{m'} = C(\mathbf{m_1}, \ldots, \mathbf{m_k})$

CKKS supports an arithmetic circuit C consisting of addition and multiplication on real and complex numbers. The server performs HE ops to evaluate C on the user's input ciphertexts. HE ops can be represented as OpName($[\langle \mathbf{m} \rangle], x$) $\rightarrow [\langle \mathbf{m'} \rangle]$, where x can be a ciphertext (e.g., HAdd/HMult: homomorphic add/mult between ciphertexts), a plaintext (PAdd/PMult), or a constant (CAdd/CMult). Finally, HRot homomorphically performs a cyclic rotation by an integer r on the message (msg) elements, which can be represented as $[\langle (m_r, \ldots, m_{\frac{N}{2}}, m_1 \ldots, m_{r-1}) \rangle]$. We denote HRot($[\langle \mathbf{m} \rangle], r$) to as $[\langle \mathbf{m} \ll r \rangle]$ and also $[\langle \mathbf{m} \rangle] \ll r$ for convenience ($\gg$ for the opposite direction). Among these basic HE ops, HMult and HRot take 1–2 orders of magnitude longer computation time compared to the other basic HE ops on conventional platforms, thereby accounting for the majority of the execution time on most workloads. For a formal description of the RNS-CKKS scheme, an optimized version of CKKS, please refer to the original paper [7].

2.2 CKKS Encoding Methods

Two methods exist for encoding a message vector $\mathbf{m} \in \mathbb{C}^{N/2}$ into a plaintext $\langle \mathbf{m} \rangle \in \mathcal{R}_q$: Slot encoding and Coefficient encoding.

Slot Encoding. We let $\zeta = \exp(\pi\sqrt{-1}/N)$ be a $(2N)$-th primitive root of unity and set ζ_j as $\zeta_j := \zeta^{5^j}$ for $0 \le j < N/2$. For a matrix

$$U = \begin{bmatrix} 1 & \zeta_0 & \zeta_0^2 & \cdots & \zeta_0^{N-1} \\ 1 & \zeta_1 & \zeta_1^2 & \cdots & \zeta_1^{N-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \zeta_{N/2-1} & \zeta_{N/2-1} & \cdots & \zeta_{N/2-1}^{N-1} \end{bmatrix} \in \mathbb{C}^{(N/2)\times N},$$

Slot encoding outputs $\langle \mathbf{m} \rangle := \sum_{i=0}^{N-1} f_i \cdot X^i$ such that $\mathbf{f} = (f_i)_{0 \le i < N} = \frac{1}{N}(\overline{U}^T \cdot \mathbf{m} + U^T \cdot \overline{\mathbf{m}})$. Then it satisfies that $\langle \mathbf{m} \rangle \cdot \langle \mathbf{m'} \rangle = \langle \mathbf{m} \odot \mathbf{m'} \rangle$ for element-wise vector multiplication $\odot$. In ordere to convert convolution into element-wise multiplication, discrete Fourier transform (DFT) is utilized; decoding is equivalent to performing DFT on the coefficients of $\langle \mathbf{m} \rangle$ to obtain $\mathbf{m}$ and encoding is equivalent to the reverse process, which is IDFT. When using Slot encoding, element-wise addition (HAdd/PAdd), element-wise multiplication (HMult/PMult), and cyclic rotation of the msg $\mathbf{m}$ (HRot) is possible by HE ops.

Coefficient Encoding. We first let $\tilde{\mathbf{m}} = (\mathsf{Re}(\mathbf{m}) \mid \mathsf{Im}(\mathbf{m})) \in \mathbb{R}^N$, where $\mathsf{Re}(\mathbf{m})$ and $\mathsf{Im}(\mathbf{m})$ are vectors representing real and imaginary part of $\mathbf{m}$, respectively. We then define Coefficient encoding as setting $\tilde{\mathbf{m}} \in \mathbb{R}^N$ to be the coefficients of $\langle \mathbf{m} \rangle \in \mathcal{R}_q$. Then, $\langle \mathbf{m} \rangle \cdot \langle \mathbf{m'} \rangle = \langle \tilde{\mathbf{m}} * \tilde{\mathbf{m}'} \rangle$ holds for (negacyclic) convolution $*$, defined in $\mathbb{R}^N$. Similar to Slot encoding, element-wise addition is possible; however, multiplication (HMult/PMult) results in convolution of the message elements. Prior work exploits this convolution property to implement matrix-matrix and matrix-vector multiplications [21, 25]. HRot is not supported in Coefficient encoding.

We often use the notation $\langle \cdot \rangle_{\text{slot}}$ and $\langle \cdot \rangle_{\text{coeff}}$ to differentiate between the two encoding methods. For both methods, the encoding results are multiplied by a large *scale factor* Δ and are rounded to convert them into integer polynomials, which can then be embedded into $\mathcal{R}_q$. However, as this follow-up step is not central to our discussion, we often omit it in this paper for brevity.

We make the following observations regarding the encoding methods, which we use extensively in the paper. These properties can be extended to CinS encoding, which we explain in §3.

Remark 1. *We can interpret the same plaintext differently with regard to the encoding method; e.g.,* $\langle \mathbf{m} \rangle_{slot} = \langle IDFT(\mathbf{m}) \rangle_{coeff}$ *and* $\langle \mathbf{m} \rangle_{coeff} = \langle DFT(\mathbf{m}) \rangle_{slot}$.

Remark 2. *We can indeed interpret any HE op as being applied to slot-encoded ciphertexts. For example, HMult for Coefficient encoding can be interpreted as* $\langle \mathbf{m} \rangle_{coeff} \cdot \langle \mathbf{m'} \rangle_{coeff} = \langle DFT(\mathbf{m}) \rangle_{slot} \cdot \langle DFT(\mathbf{m'}) \rangle_{slot} = \langle DFT(\mathbf{m}) \odot DFT(\mathbf{m'}) \rangle_{slot} = \langle DFT(\mathbf{m} * \mathbf{m'}) \rangle_{slot} = \langle \mathbf{m} * \mathbf{m'} \rangle_{coeff}$ *based on the convolution theorem of DFT.*

2.3 Decomposition of DFT matrix

We first define the special DFT matrix as

$$V = \begin{bmatrix} 1 & \zeta_0 & \cdots & \zeta_0^{N/2-1} \\ 1 & \zeta_1 & \cdots & \zeta_1^{N/2-1} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & \zeta_{N/2-1} & \cdots & \zeta_{N/2-1}^{N/2-1} \end{bmatrix},$$

which is an $(N/2)\times(N/2)$ square matrix satisfying $U = [V \mid \sqrt{-1} \cdot V]$. Evaluating this structured square matrix V and its inverse $V^{-1} = \frac{2}{N} \cdot \overline{V}^T$ plays a crucial role in the CKKS scheme, including Slot encoding. There is a line of work [6, 19] to decompose the matrix V into a product of sparse diagonal matrices for efficient matrix evaluation. The main idea is to permute the columns of V using the $(N/2) \times (N/2)$ bit-reversal permutation matrix $\mathcal{P}$ with $\mathcal{P}^{-1} = \mathcal{P}$. After the permutation, V can be decomposed. To be precise, we let $\omega_m = \exp(2\pi\sqrt{-1}/m)$ be a m-th primitive root of unity. (In particular, $\omega_{2N} = \zeta$) and let $\mathcal{T}_n = \left(\omega_{4n}^{5^i \cdot \mathsf{rev}_n(j)}\right)_{0 \le i,j < n}$ for $n \le N/2$, where $\mathsf{rev}_n(j)$ denotes bit-reversal permutation of j with length n. Then we have $\mathcal{T}_{N/2} = V \cdot \mathcal{P}$ and $\mathcal{T}_n$ is decomposed as follows:

$$\mathcal{T}_n = \begin{bmatrix} I_{n/2} & W_{n/2} \\ I_{n/2} & -W_{n/2} \end{bmatrix} \cdot \begin{bmatrix} \mathcal{T}_{n/2} & 0 \\ 0 & \mathcal{T}_{n/2} \end{bmatrix}, \tag{2}$$

where $W_{n/2}$ denotes a diagonal matrix $\mathrm{diag}(\omega_{4n}^{5^i})_{0 \le i < n/2}$. Leveraging this property repeatedly, we can decompose $\mathcal{T}_n$ into $\log n$ number of matrices as:

$$\mathcal{T}_n = S_{n/2}^{(n)} \cdot S_{n/4}^{(n)} \cdots S_1^{(n)}, \quad S_k^{(n)} = \begin{bmatrix} B_k & 0 & \cdots & 0 \\ 0 & B_k & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & B_k \end{bmatrix} \in \mathbb{C}^{n \times n}$$

where $S_k^{(n)}$ is composed of $(n/2k)$ block diagonals of the matrix $B_k = \begin{bmatrix} I_k & W_k \\ I_k & -W_k \end{bmatrix} \in \mathbb{C}^{(2k)\times(2k)}$. For $n = N/2$, we simply write $S_k^{(N/2)} = S_k$ for each k, such that $\mathcal{T}_{N/2} = S_{N/4} \cdot S_{N/8} \cdots S_1$. We refer to [19] for a more detailed description.

2.4 Bootstrapping of FHE

Multiplication between Δ-scaled polynomials produces Δ^2-scaled polynomials. **Rescaling** restores the scale factor to Δ by truncating Δ bits from the least significant bits (LSBs) from the coefficients of the polynomial. In this process, the modulus of the ring decreases from q to q/Δ. As the modulus cannot decrease indefinitely, we define the *level* of a polynomial such that the modulus of the polynomial at level i is defined as $Q(i) = q/\Delta^{L-i}$ for the initial level L, $0 \le i \le L$, and $q = Q(L) > \Delta^L$. Rescaling reduces the level by one.

Bootstrapping is a process to recover the level of a ciphertext reduced by rescaling. Here, we briefly explain the computational flow of bootstrapping and refer the readers to [3, 4, 8] for an explanation of the state-of-the-art CKKS bootstrapping algorithms.

The level of a ciphertext, $[\langle \mathbf{m} \rangle_{\text{coeff}}]$ for instance, reaches zero after L rescaling. If we change the modulus from $Q(0)$ to $Q(L)$ to continue operation, we get $[\langle \mathbf{m} + {}^{Q(0)}/_{\Delta}\mathbf{l} \rangle_{\text{coeff}}]$, where $\mathbf{l}$ is an integer vector holding small values. Bootstrapping removes the unwanted ${}^{Q(0)}/_{\Delta}\mathbf{l}$ term by a complex sequence of HE ops. First, we change the encoding method to Slot encoding by an operation called *coefficient-to-slot* (CtoS) to utilize element-wise multiplication only available in Slot encoding. CtoS is a homomorphic evaluation of multiplication with the matrix V^{-1}, where V is the special DFT matrix defined previously.

Then, through *modular reduction evaluation* (ModEval) operation, it is possible to obtain $[\langle \mathbf{m} \rangle_{\text{slot}}]$ from $[\langle \mathbf{m} + {}^{Q(0)}/_{\Delta}\mathbf{l} \rangle_{\text{slot}}]$. Finally, we return to the original encoding method by performing *slot-to-coefficient* (StoC), which is a homomorphic evaluation of multiplication with the DFT matrix V. Due to Remark 1, the same bootstrapping process applies to $[\langle \mathbf{m} \rangle_{\text{slot}}]$ equally by interpreting it as $[\langle IDFT(\mathbf{m}) \rangle_{\text{coeff}}]$.

Due to its complexity, bootstrapping is an extremely expensive process that takes most of the execution time in FHE CKKS workloads, with CtoS and StoC operations comprising a significant portion of its runtime. To alleviate this burden, we use the bit-reversal DFT matrix $\mathcal{T}_{N/2}$ and its inverse $\mathcal{T}_{N/2}^{-1}$ instead of V and V^{-1} in StoC and CtoS, respectively. Then, we can interpret CtoS and StoC all in Slot encoding based on Remark 2 to obtain

$$\text{CtoS} : \langle \mathbf{m} \rangle_{\text{coeff}} = \langle \mathcal{T}_{N/2}(\mathcal{P}\mathbf{m}) \rangle_{\text{slot}} \xmapsto{\mathcal{T}_{N/2}^{-1}} \langle \mathcal{P}\mathbf{m} \rangle_{\text{slot}},$$

$$\text{StoC} : \langle \mathcal{P}\mathbf{m} \rangle_{\text{slot}} \xmapsto{\mathcal{T}_{N/2}} \langle \mathcal{T}_{N/2}(\mathcal{P}\mathbf{m}) \rangle_{\text{slot}} = \langle \mathbf{m} \rangle_{\text{coeff}},$$

which implies that $\mathcal{P}\mathbf{m}$, rearranged from $\mathbf{m}$ in bit-reversal order, is encoded in slots instead. We refer to [19] for more details about the advantages of using $\mathcal{T}_{N/2}$. The order of the slots after CtoS does not play any role in the bootstrapping. Therefore, once the message $\mathbf{m}$ is given, we may assume that its Coefficient encoding $\langle \mathbf{m} \rangle_{\text{coeff}}$ is encoded in the given order, but its Slot encoding $\langle \mathbf{m} \rangle_{\text{slot}}$ is encoded in bit-reversal order for brevity of explanation.

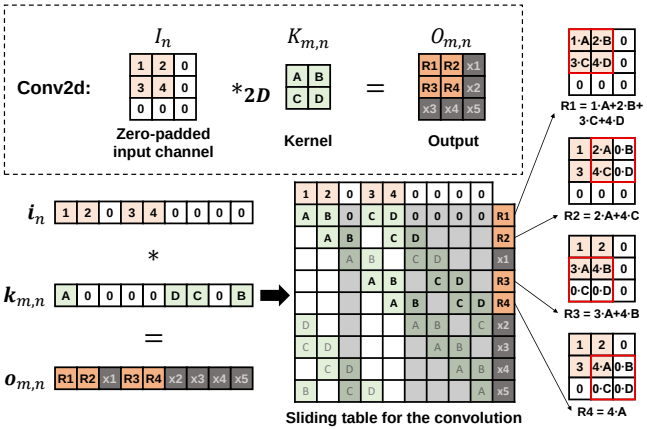


Figure 1: Converting conv2d ($*_{2D}$) between a 2×2 input channel I_n and a 2×2 kernel $K_{m,n}$, which produces a partial conv2d result $O_{m,n}$, into convolution between vectors: $\mathbf{i}_n * \mathbf{k}_{m,n} = \mathbf{o}_{m,n}$.

As the resulting bootstrapping circuit consists of many HE ops, it consumes multiple levels (L_{boot}) and produces an output ciphertext at level $L' = L - L_{\text{boot}}$, leaving only a small number of levels on which to perform multiplication operations.

2.5 Prior HE Implementations of CNNs

2.5.1 Handling conv2d. Conv2d processes an input feature map $I \in \mathbb{R}^{C_{in}\times w\times w}$ comprising C_{in} channels each of dimension $w \times w$. For a predetermined stride (s) and padding, a set of kernels represented by $K \in \mathbb{R}^{C_{out}\times C_{in}\times f\times f}$ is used to produce the output feature map $O \in \mathbb{R}^{C_{out}\times w'\times w'}$. We denote the $\mathbf{n}$-th channel of I as I_n, the kernel corresponding to the $\mathbf{n}$-th input and $\mathbf{m}$-th output channel as $K_{m,n}$, and the $\mathbf{m}$-th channel of O as O_m. Then, conv2d is formalized as

$$O_m = \sum_{n=1}^{C_{in}} O_{m,n} = \sum_{n=1}^{C_{in}} K_{m,n} *_{2\text{D}} I_n.$$

Prior studies have identified that plain convolution can be utilized to perform higher-dimensional convolutions, such as $*_{2D}$, by zero-padding and flattening the data. This is shown in Figure 1, where I_n and $K_{m,n}$ are converted into vectors $\mathbf{i}_n$ and $\mathbf{k}_{m,n}$. It can be observed from the figure that the convolution result $\mathbf{o}_{m,n} = \mathbf{i}_n * \mathbf{k}_{m,n}$ is equivalent to $O_{m,n} = K_{m,n} *_{2\text{D}} I_n$, flattened. Imitating higher-dimensional convolution (2D in the figure) using plain convolution creates unnecessary values (x1 to x5, painted black) in the output $O_{m,n}$. However these values appear precisely in the locations of zero-padding, and can be easily removed through an element-wise multiplication with a mask vector.

Based on this conversion from $*_{2D}$ to convolution, Cheetah [21] proposes an efficient HE conv2d algorithm on Coefficient-encoded ciphertexts, on which our baseline [25] improves. For example, when $C = N/w^2$ (suppose w is the padded input width/height and is a power-of-two number) channels can be placed in a ciphertext and $C = C_{in} = C_{out}$, the entire I can be packed as a single ciphertext $[\langle \mathbf{i} \rangle]$ and $\{K_{m,n} | n \in [1, C]\}$ can be packed together in a plaintext $\langle \mathbf{k}_m \rangle$ for each m. Prior studies devise a data organization method inside $[\langle \mathbf{i} \rangle]$ and $\langle \mathbf{k}_m \rangle$'s for Coefficient encoding, such that

PMult($[\langle \mathbf{i} \rangle]$, $\langle \mathbf{k}_m \rangle$) results in a batch computation of $*_{2D}$ to finally produce a Coefficient-encoded ciphertext that encrypts the data of O_m along with some unnecessary values. The method is highly efficient compared to prior methods based on Slot encoding [24].

However, conv2d with Coefficient encoding still needs to gather C ciphertexts each encrypting the data for O_m ($m \in [1, C]$) into a single ciphertext that encrypts the entire O for subsequent operations, incurring additional C-1 PMult and C-1 HRot ops for the data rearrangement. Please refer to prior studies [21, 24, 25] for the implementation details and cost analysis of each conv2d method; we summarize the final cost of the conv2d methods in Table 1.

2.5.2 Activation for FHE. FHE only supports arithmetic circuits consisting of multiplication and addition, and non-polynomial functions for activation cannot be directly evaluated. [31] proposes ReLU and MaxPool implementations based on high-degree polynomial approximation. These approximation-based methods can directly replace the non-linear operations, but they incur huge computational overhead due to a lot of level consumption and frequent bootstrapping resulting from it. For example, [29] reports that over 83% of the total inference time is spent on ReLU and bootstrapping.

To mitigate this overhead, [2, 36] propose retraining approaches that can construct CNNs with low-degree polynomial activation. Contrary to the findings in [16, 20, 22], AESPA [36] shows that, by adopting special training methods, deep CNNs can be effectively trained using low-degree polynomials. With the retrained network, AESPA allows using a simple square function (x^2) for activation during inference without compromising the inference accuracy.

The type and implementation of the activation function are orthogonal to our contributions. We refer to the homomorphic evaluation of the activation function as HActivation($\mathbf{ct}$) and only specify its type when presenting experiment results. Our only assumption is that the input and output ciphertexts are slot-encoded for HActivation and that HActivation preserves the slot index of each value.

2.6 HE Matrix-Vector Multiplication Algorithm

We introduce a widely used matrix-vector multiplication method in HE [18], which utilizes diagonal grouping. Suppose the server wants to multiply a known $M \times M$ matrix W to a length-M vector encrypted as $[\langle \mathbf{m} \rangle_{\text{slot}}]$. For the elements of W, $w_{i,j} (i, j \in [1, M])$, we refer to the vector of diagonally grouped elements starting with $w_{1,k}$ as k-th *cyclic diagonal* of the matrix and write it as $\mathbf{w}_k^{\text{diag}}$. Then, the multiplication of W with $\mathbf{m}$ can be performed as the following:

$$W\mathbf{m} = \sum_{k=1}^{M} \mathbf{w}_k^{\text{diag}} \odot (\mathbf{m} \ll (k-1)). \tag{3}$$

When evaluating Eq. 3 in the encrypted state, it requires $M-1$ HRot and M PMult ops. As HRot ops are expensive, we can reduce the cost by using the baby-step giant-step (BSGS) algorithm [18]. For B and G that satisfies $B \cdot G = M$, BSGS modifies Eq. 3 into a nested loop:

$$\sum_{j=1}^{G} \left(\sum_{i=1}^{B} \mathbf{w}_{(j-1)B+i}^{\text{diag}'} \odot (\mathbf{m} \ll (i-1)) \right) \ll (j-1)B.$$

By reusing $m \ll (i-1)$ values, the total number of HRot ops is reduced to $B + G - 2$; we can select $B = G = \sqrt{M}$ to minimize it to $2\sqrt{M} - 2$. The notation $\mathbf{w}_{(j-1)B+i}^{\text{diag}'}$ indicates $\mathbf{w}_{(j-1)B+i}^{\text{diag}} \gg (j-1)B$. Since W is known to the server, the plaintexts $\{\langle \mathbf{w}_k^{\text{diag}'} \rangle_{\text{slot}}\}_{1 \le k \le BG}$ can be created in the preprocessing phase.

The BSGS algorithm can be extended for a sparse matrix with a few evenly-spaced cyclic diagonals. For example, if a matrix has M' cyclic diagonals $\{\mathbf{w}_1^{\text{diag}}, \mathbf{w}_{1+\ell}^{\text{diag}}, \cdots, \mathbf{w}_{1+(M'-1)\ell}^{\text{diag}}\}$, BSGS can be applied to perform the matrix-vector multiplication with $2\sqrt{M'} - 2$ HRot ops.

The cost of matrix-vector multiplication can be further reduced if the matrix can be decomposed into a product of sparser matrices, as in §2.3, where the BSGS algorithm can be applied for each decomposed sparse matrix. As we decompose a matrix into a greater number of sparser matrices, the total number of HRot ops decreases at the cost of more level consumption.

The optimal point in this trade-off differs depending on the FHE parameters and the ciphertext level [6]. Deciding the optimal point is mostly orthogonal to our contribution, so we simply write HMatmul(W, $[\langle \mathbf{m} \rangle_{\text{slot}}]$) to represent a series of BSGS operations resulting in $[\langle W\mathbf{m} \rangle_{\text{slot}}]$.

3 CinS Encoding

We introduce CinS encoding, which is the central method for our efficient CNN implementation. We recall that StoC in the bootstrapping process is a homomorphic evaluation of $\mathcal{T}_{N/2}$, the DFT evaluation with bit-reversal permutation, as described in §2.4. Then StoC converts the encoding type of $\langle \mathbf{m} \rangle_{\text{slot}} \in \mathcal{R}_q$ as

$$\text{StoC} : \langle \mathbf{m} \rangle_{\text{slot}} \mapsto \langle \mathcal{T}_{N/2} \cdot \mathbf{m} \rangle_{\text{slot}} = \langle \mathbf{m} \rangle_{\text{coeff}}.$$

We decompose $\mathcal{T}_{N/2}$ as $\mathcal{T}_{N/2} = \mathcal{T}_2' \cdot \mathcal{T}_1'$ and propose an intermediate encoding between Slot encoding and Coefficient encoding as

$$\langle \mathbf{m} \rangle_{\text{CinS}} := \langle \mathcal{T}_1' \cdot \mathbf{m} \rangle_{\text{slot}},$$

which would lead to $\langle \mathcal{T}_2' \cdot (\langle \mathbf{m} \rangle_{\text{CinS}}) \rangle_{\text{slot}} = \langle \mathbf{m} \rangle_{\text{coeff}}$. We then show that $\langle \mathbf{m} \rangle_{\text{CinS}}$ indeed contains the coefficient of each $\langle \mathbf{m}_i \rangle_{\text{coeff}}$ in its slots when given a proper concatenation $\mathbf{m} = (\mathbf{m}_0 | \mathbf{m}_1 | \cdots | \mathbf{m}_{C-1})$.

Remark 3. *One might consider the dual encoding method referred as Slots-in-Coefficient (SinC) encoding $\langle \mathbf{m} \rangle_{SinC}$, which contains the context about each $\langle \mathbf{m}_i \rangle_{slot}$ at its coefficients. However, we do not address the details of SinC encoding in this paper since we can not find any suitable applications for it.*

3.1 Exploiting Partial DFT

We provide technical details for our CinS encoding method. Leveraging the decomposition of $\mathcal{T}_{N/2}$ described in §2.3, we have

$$\langle \mathbf{m} \rangle_{\text{coeff}} = \langle \mathcal{T}_{N/2} \cdot \mathbf{m} \rangle_{\text{slot}} = \langle S_{N/4} \cdot S_{N/8} \cdots S_1 \cdot \mathbf{m} \rangle_{\text{slot}}.$$

Suppose that we are given power-of-two integers C, ℓ with $N/2 = C \cdot \ell$. If we apply the decomposition property (Eq. 2 in §2.3) $\log N/2\ell$

times to the initial matrix $\mathcal{T}_{N/2}$, then we have

$$\mathcal{T}_{N/2} = S_{N/4} \cdot S_{N/8} \cdots S_\ell \cdot \begin{bmatrix} \mathcal{T}_\ell & 0 & \cdots & 0 \\ 0 & \mathcal{T}_\ell & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \mathcal{T}_\ell, \end{bmatrix}$$

with C number of block-diagonal matrices $\mathcal{T}_\ell$. Concatenating the message $\mathbf{m} \in \mathbf{C}^{N/2}$ as $\mathbf{m} = (\mathbf{m}_0 \mid \mathbf{m}_1 \mid \cdots \mid \mathbf{m}_{C-1})$, then we have

$$\begin{aligned}(S_{\ell/2} \cdots S_1)(\mathbf{m}) &= (S_\ell^{-1} \cdots S_{N/4}^{-1} \cdot \mathcal{T}_{N/2})(\mathbf{m}) \\ &= (\mathcal{T}_\ell(\mathbf{m}_0) \mid \mathcal{T}_\ell(\mathbf{m}_1) \mid \cdots \mid \mathcal{T}_\ell(\mathbf{m}_{C-1})).\end{aligned}$$

If we use a shorthand notation $S_{j \leftarrow i} = S_{j/2} S_{j/4} \cdots S_i$, one can write the bit-reversal DFT matrix as $\mathcal{T}_{N/2} = S_{N/2 \leftarrow \ell} \cdot S_{\ell \leftarrow 1}$. Now, we provide a formal definition of CinS encoding.

Definition 1 (CinS Encoding). Let C, ℓ be power-of-two integers satisfying $N/2 = C \cdot \ell$. For a message vector $\mathbf{m} \in \mathbb{C}^{N/2}$ in bit-reversal order and a matrix decomposition $\mathcal{T}_{N/2} = S_{N/2 \leftarrow \ell} \cdot S_{\ell \leftarrow 1}$, we define a CinS encoding of $\mathbf{m}$ as

$$\langle \mathbf{m} \rangle_{\mathrm{CinS}_\ell} := \langle S_{\ell \leftarrow 1} \mathbf{m} \rangle_{\mathrm{slot}}.$$

Here, the matrix $\mathcal{T}_\ell$ is obtained after permuting the columns in the DFT matrix of length ℓ, denoted as DFT_ℓ. We will often abbreviate $\langle \cdot \rangle_{\mathrm{CinS}_\ell}$ as $\langle \cdot \rangle_{\mathrm{CinS}}$ when the context ℓ is clear, and write $\mathcal{T}_\ell$ as DFT_ℓ for clarity. Then, we conclude the following theorem.

Theorem 2 (Partial DFT Property). *Let integers C, ℓ, and a message $\mathbf{m} \in \mathbb{C}^{N/2}$ be given as described in Definition 1. If we concatenate the message $\mathbf{m}$ as $\mathbf{m} = (\mathbf{m}_0 \mid \mathbf{m}_1 \mid \cdots \mid \mathbf{m}_{C-1})$, which is composed of C slices having the length ℓ, then we have*

$$\begin{aligned}\langle \mathbf{m} \rangle_{CinS} &= \langle (S_{N/2 \leftarrow \ell}^{-1} \cdot DFT) \cdot \mathbf{m} \rangle_{slot} \\ &= \langle (\hat{\mathbf{m}}_0 \mid \hat{\mathbf{m}}_1 \mid \cdots \mid \hat{\mathbf{m}}_{C-1}) \rangle_{slot},\end{aligned}$$

where $\hat{\mathbf{m}}_i = DFT_\ell(\mathbf{m}_i)$ for a partial DFT matrix DFT_ℓ of length ℓ.

3.2 Homomorphic Property of CinS Encoding

We interpret CinS encoding from an algebraic perspective. From Theorem 2, CinS encoding of $\mathbf{m} = (\mathbf{m}_0 | \mathbf{m}_1 | \cdots | \mathbf{m}_{C-1}) \in (\mathbb{C}^\ell)^C$ is

$$\begin{aligned}&\langle \mathbf{m} \rangle_{\mathrm{CinS}} = \langle (\hat{\mathbf{m}}_0 \mid \hat{\mathbf{m}}_1 \mid \cdots \mid \hat{\mathbf{m}}_{C-1}) \rangle_{\mathrm{slot}} \\ &= \langle (DFT_\ell(\mathbf{m}_0) \mid DFT_\ell(\mathbf{m}_1) \mid \cdots \mid DFT_\ell(\mathbf{m}_{C-1})) \rangle_{\mathrm{slot}},\end{aligned} \tag{4}$$

a concatenation of component-wise DFT vectors of $\mathbf{m}_i \in \mathbb{C}^\ell$. Here, we consider another cyclotomic polynomial ring $\mathbb{Z}_q[Y]/(Y^{2\ell}+1)$, denoted as $\mathcal{R}'_q$. Then we can regard CinS encoding $\mathbf{m}$ as C-tuples of plaintexts in $\mathcal{R}'_q$, each of which can be viewed as

$$\langle DFT_\ell(\mathbf{m}_i) \rangle_{\mathrm{slot}} = \langle \mathbf{m}_i \rangle_{\mathrm{coeff}} \in \mathcal{R}'_q.$$

Letting $Y = X^C$, the ring $\mathcal{R}'_q = \mathbb{Z}_q[Y]/(Y^{2\ell}+1)$ can be regarded as a subring of $\mathcal{R}_q = \mathbb{Z}_q[X]/(X^N+1)$, and there exists a relation between two algebraic objects from the following theorem.

Theorem 3 (Isomorphism). *Let $\mathcal{R}_q$ and $\mathcal{R}'_q$ denote the $2N$-th and 2ℓ-th cyclotomic polynomial ring, respectively, as described above. Then we have a module isomorphism as*

$$\mathcal{R}_q = \mathbb{Z}_q[X]/(X^N+1) \simeq \left(\mathbb{Z}_q[Y]/(Y^{2\ell}+1)\right)^C = (\mathcal{R}'_q)^C.$$

Proof. For an element $\mathbf{a} = \sum_{i=0}^{N-1} a_i X^i$ in $\mathcal{R}_q$, one can write it in the form of

$$\mathbf{a} = \sum_{i=0}^{N-1} a_i X^i = \sum_{k=0}^{C-1} \sum_{j=0}^{2\ell-1} a_{Cj+k} \cdot X^{Cj+k} = \sum_{k=0}^{C-1} \left(\sum_{j=0}^{2\ell-1} a_{Cj+k} \cdot Y^j \right) \cdot X^k.$$

If we set each $\mathbf{a}_{C,k}$ as $\mathbf{a}_{C,k} = \sum_{j=0}^{\ell-1} a_{Cj+k} \cdot Y^j \in \mathcal{R}'_q$, then the mapping $\mathbf{a} \mapsto (\mathbf{a}_{C,0}, \mathbf{a}_{C,1}, \cdots, \mathbf{a}_{C,C-1})$ induces an isomorphism $\mathcal{R}_q \simeq (\mathcal{R}'_q)^C$. □

From the view of Coefficient encoding, the plaintext $\langle \mathbf{m} \rangle_{\mathrm{coeff}} \in \mathcal{R}_q$ is mapped onto $(\langle \mathbf{m}_0 \rangle_{\mathrm{coeff}}, \cdots, \langle \mathbf{m}_{C-1} \rangle_{\mathrm{coeff}}) \in (\mathcal{R}'_q)^C$. Thus, one may regard CinS encoding as a local Coefficient encoding, which can be viewed as

$$\begin{aligned}\langle \mathbf{m} \rangle_{\mathrm{CinS}} &= \langle (\hat{\mathbf{m}}_0 \mid \hat{\mathbf{m}}_1 \mid \cdots \mid \hat{\mathbf{m}}_{C-1}) \rangle_{\mathrm{slot}} \\ &= \langle (\mathbf{m}_0 \mid \mathbf{m}_1 \mid \cdots \mid \mathbf{m}_{C-1}) \rangle_{\text{local-coeff}}.\end{aligned}$$

3.2.1 *Local Convolution.*

From the above observations, we now define a local convolution $*_{\mathrm{local}}$ as follows. For two message vectors $\mathbf{m} = (\mathbf{m}_0 | \mathbf{m}_1 | \cdots | \mathbf{m}_{C-1})$ and $\mathbf{m}' = (\mathbf{m}'_0 | \mathbf{m}'_1 | \cdots | \mathbf{m}'_{C-1})$ given in bit-reversal order, we define a mapping $*_{\mathrm{local}}$ as

$$\mathbf{m} *_{\mathrm{local}} \mathbf{m}' := (\mathbf{m}_0 *_\ell \mathbf{m}'_0 | \mathbf{m}_1 *_\ell \mathbf{m}'_1 | \cdots | \mathbf{m}_{C-1} *_\ell \mathbf{m}'_{C-1}),$$

where $*_\ell$ denotes the negacyclic convolution defined in a subring $\mathcal{R}'_q = \mathbb{Z}_q[Y]/(Y^\ell+1)$. It can be viewed as component-wise convolution $*_\ell$ between two elements lying in $(\mathcal{R}'_q)^C$. Thus, we have

$$\begin{aligned}&\langle \mathbf{m} *_{\mathrm{local}} \mathbf{m}' \rangle_{\mathrm{CinS}} \\ &= \langle (\mathbf{m}_0 *_\ell \mathbf{m}'_0 | \mathbf{m}_1 *_\ell \mathbf{m}'_1 | \cdots | \mathbf{m}_{C-1} *_\ell \mathbf{m}'_{C-1}) \rangle_{\text{local-coeff}}.\end{aligned}$$

Now, we show the homomorphic property of CinS encoding via the following theorem.

Theorem 4 (Homomorphism for CinS encoding). *For two message vectors $\mathbf{m}, \mathbf{m}' \in \mathbb{C}^{N/2}$, we have*

$$\langle \mathbf{m} *_{local} \mathbf{m}' \rangle_{CinS} = \langle \mathbf{m} \rangle_{CinS} \cdot \langle \mathbf{m}' \rangle_{CinS}$$

Proof. By the convolution theorem at subring $\mathcal{R}'_q$, we have $\mathbf{m}_0 *_\ell \mathbf{m}'_0 = \widehat{\mathbf{m}_0 *_\ell \mathbf{m}'_0}$ for each i. Utilizing the theorem, one can show the following:

$$\begin{aligned}&\langle \mathbf{m} *_{\mathrm{local}} \mathbf{m}' \rangle_{\mathrm{CinS}} \\ &= \langle (\mathbf{m}_0 *_\ell \mathbf{m}'_0 | \mathbf{m}_1 *_\ell \mathbf{m}'_1 | \cdots | \mathbf{m}_{C-1} *_\ell \mathbf{m}'_{C-1}) \rangle_{\text{local-coeff}} \\ &= \langle (\widehat{\mathbf{m}_0 *_\ell \mathbf{m}'_0} | \widehat{\mathbf{m}_1 *_\ell \mathbf{m}'_1} | \cdots | \widehat{\mathbf{m}_{C-1} *_\ell \mathbf{m}'_{C-1}}) \rangle_{\mathrm{slot}} \\ &= \langle (\hat{\mathbf{m}}_0 \odot \hat{\mathbf{m}}'_0 | \hat{\mathbf{m}}_1 \odot \hat{\mathbf{m}}'_1 | \cdots | \hat{\mathbf{m}}_{C-1} \odot \hat{\mathbf{m}}'_{C-1}) \rangle_{\mathrm{slot}} \\ &= \langle (\hat{\mathbf{m}}_0 | \hat{\mathbf{m}}_1 | \cdots | \mathbf{m}_{\hat{C}-1}) \rangle_{\mathrm{slot}} \cdot \langle (\hat{\mathbf{m}}'_0 | \hat{\mathbf{m}}'_1 | \cdots | \mathbf{m}'_{\hat{C}-1}) \rangle_{\mathrm{slot}} \\ &= \langle \mathbf{m} \rangle_{\mathrm{CinS}} \cdot \langle \mathbf{m}' \rangle_{\mathrm{CinS}}\end{aligned} \tag{5}$$

□

3.2.2 *Cyclic Rotation.*

Another important functionality that CinS encoding supports is a cyclic rotation of the sequence of slices. When interpreting a CinS-encoded ciphertext as a slot-encoded ciphertext following Eq. 4, we can perform HRot by a multiple of ℓ (e.g., $k\ell$) on the ciphertext and obtain $[\langle \mathbf{m} \ll k\ell \rangle_{\mathrm{CinS}}] = [\langle (\mathbf{m}_{\hat{k}+1} | \cdots | \mathbf{m}_{\hat{C}} | \hat{\mathbf{m}}_1 | \cdots | \hat{\mathbf{m}}_k) \rangle_{\mathrm{slot}}]$.

3.3 CinS Encoding and Efficient Conv2d

Previous homomorphic conv2d algorithms, which utilizre HE ops provided by Slot and Coefficient encodings, are not well-suited to the computational patterns of conv2d. These methods incur additional data relocation steps involving an excessive amount of HRot ops. By leveraging the homomorphic properties of CinS encoding, we instantiate a more efficient conv2d algorithm, which results in much fewer HRot ops.

3.3.1 Real CinS Encoding. We have explained the case where CinS encoding uses the message domain $\mathbb{C}^{N/2}$ in accordance with the message domain of Slot encoding. However, as most CNN models utilize only real values, half of the available space remains unused. To address this, we present a modification to the definition of CinS encoding, which effectively utilizes $\mathbb{R}^N$ as the message domain, enabling us to pack two times more data in a message.

For a length-2ℓ real vector $\mathbf{x} \in \mathbb{R}^{2\ell}$, we define a folded vector $\check{\mathbf{x}} \in \mathbb{C}^{\ell}$, whose i-th element $(\check{\mathbf{x}})_i$ is defined as $(\check{\mathbf{x}})_i = (\mathbf{x})_i + \sqrt{-1}(\mathbf{x})_{i+\ell}$. Then, for a real vector $\mathbf{m} = (\mathbf{m}_1|\mathbf{m}_2|\cdots|\mathbf{m}_C) \in \mathbb{R}^N$ composed of C length-2ℓ slices, we redefine CinS encoding to be:

$$\begin{aligned}\langle \mathbf{m} \rangle_{\mathrm{CinS}} &= \langle S_{\ell \leftarrow 1} \mathcal{P}_\ell (\check{\mathbf{m}}_1 | \cdots | \check{\mathbf{m}}_C) \rangle_{\mathrm{slot}} \\ &= \langle (DFT'_{2\ell}(\mathbf{m}_1) | \cdots | DFT'_{2\ell}(\mathbf{m}_C)) \rangle_{\mathrm{slot}}\end{aligned} \tag{6}$$

Here, $\mathcal{P}_\ell$ is a block diagonal matrix of total size $N/2 \times N/2$, where each block is P_ℓ, a bit-reversal permutation matrix of size $\ell \times \ell$. Then, the following property holds.

$$\begin{aligned}&\langle (\mathbf{m}_1|\cdots|\mathbf{m}_C) \rangle_{\mathrm{CinS}} \cdot \langle (\mathbf{m}'_1|\cdots|\mathbf{m}'_C) \rangle_{\mathrm{CinS}} \\ =&\langle (\mathbf{m}_1 *_{2\ell} \mathbf{m}'_1 | \cdots | \mathbf{m}_C *_{2\ell} \mathbf{m}'_C) \rangle_{\mathrm{CinS}}\end{aligned} \tag{7}$$

Here, $DFT'_{2\ell}$ is a real-to-complex DFT [42] that accepts real vectors of length 2ℓ and outputs complex vectors of length ℓ. We present the proof in Appendix A. This explains why the number of elements in a message is N for our methods in Table 1.

3.3.2 Data organization. As we have described, CinS encoding can be utilized to perform convolution on multiple input slices of power-of-two length at once. First, each $w_0 \times w_0$-sized channel of the input feature map (I_i) is zero-padded to a $w \times w$ shape, where $w = 2^{\lceil \log_2 w_0 \rceil}$ is a power-of-two value. The padded channel is flattened in the row-major order to form a vector $\mathbf{i}_i$. Then, we can use CinS encoding to pack $C = N/w^2$ padded channels together in a ciphertext. For the entire feature map, we need C_{in}/C ciphertexts:

$$\{ [\langle \mathbf{i}_{C(i-1)+1} | \mathbf{i}_{C(i-1)+2} | \cdots | \mathbf{i}_{Ci} \rangle_{\mathrm{CinS}}] \mid i \in [1, C_{in}/C] \}.$$

For each of the $C_{out} \cdot C_{in}$ kernels of conv2d ($K_{i,j}$), we use the method in Figure 1 to prepare a $w \times w$-sized vector $\mathbf{k}_{i,j}$, which can be directly used to perform convolution with the feature map. Then, in the same way as the feature map, C kernels can be packed into a ciphertext. However, we first need to decide the order of kernel packing. We make the following observation, which forms the core of our conv2d method:

Remark 4. *If we regard the feature map as a length-C_{in} vector of channels $\mathcal{I}$ and conv2d weights as a $C_{out} \times C_{in}$ matrix of kernels $\mathcal{K}$, conv2d is equivalent to performing a matrix-vector multiplication $\mathcal{K}\mathcal{I}$, where the multiplication between a channel and a kernel is replaced by convolution between them.*

Based on Remark 4 and inspired by the diagonal grouping method in §2.6, we devise a diagonal packing method for CinS-encoding-based conv2d. We first present an example for a simple case when $C_{out} = C_{in} = C$. $\mathbf{k}_{i,j}$ represents the padded kernel corresponding to the i-th ($i \in [1, C]$) input channel and the j-th ($j \in [1, C]$) output channel. Then, we prepare C plaintexts each encoding a cyclic diagonal from the matrix of $\mathbf{k}_{i,j}$'s; i.e., we prepare $\{\langle \mathbf{k}_k^{\mathrm{diag}} \rangle_{\mathrm{CinS}} \mid k \in [1, C]\}$, where

$$\mathbf{k}_k^{\mathrm{diag}} = (\mathbf{k}_{1,k}|\mathbf{k}_{2,k+1}|\cdots|\mathbf{k}_{C-k+1,C}|\mathbf{k}_{C-k+2,1}|\cdots|\mathbf{k}_{C,k-1}).$$

3.3.3 Computation of conv2d. With the prepared data organization, we can use a similar method to §2.6 to perform conv2d when $C_{out} = C_{in} = C$. For the input ciphertext $[\langle \mathbf{i} \rangle_{\mathrm{CinS}}] = [\langle \mathbf{i}_1 | \mathbf{i}_2 | \cdots | \mathbf{i}_C \rangle_{\mathrm{CinS}}]$, we compute

$$\sum_{k=1}^{C} \mathrm{PMult}([\langle \mathbf{i} << (k-1)w^2 \rangle_{\mathrm{CinS}}], \langle \mathbf{k}_k^{\mathrm{diag}} \rangle_{\mathrm{CinS}}). \tag{8}$$

Eq. 8 has the same computational flow as the matrix-vector multiplication introduced in §2.6. This produces the conv2d output ciphertext $[\langle \mathbf{o} \rangle_{\mathrm{CinS}}] = [\langle \mathbf{o}_1 | \mathbf{o}_2 | \cdots | \mathbf{o}_C \rangle_{\mathrm{CinS}}]$ such that $\mathbf{o}_i = \sum_{j=1}^{C} \mathbf{k}_{i,j} * \mathbf{i}_j$, which is what we want. Eq. 8 requires C PMult and $C-1$ HRot ops when computed naïvely, but applying the BSGS algorithm reduces the number of HRot ops to $2\sqrt{C} - 2$ as shown in Table 1. As HRot dominates the execution time of conv2d, our conv2d algorithm roughly features $2/(\sqrt{C}+1)$ times the complexity of the coefficient-packing-based algorithm, which incurs $C-1$ HRot ops.

3.3.4 Generalization to many channels. For a more general case where there are more input and output channels than C, we perform block matrix multiplication for the $\mathcal{K}\mathcal{I}$ matrix-vector multiplication. For example, when $C_{out} = C_{in} = 2C$,

$$\mathcal{K}\mathcal{I} = \left(\begin{array}{c|c} \mathcal{K}_{11} & \mathcal{K}_{12} \\ \hline \mathcal{K}_{21} & \mathcal{K}_{22} \end{array}\right) \left(\begin{array}{c} \mathcal{I}_1 \\ \hline \mathcal{I}_2 \end{array}\right) = \left(\begin{array}{c} \mathcal{K}_{11}\mathcal{I}_1 + \mathcal{K}_{12}\mathcal{I}_2 \\ \hline \mathcal{K}_{21}\mathcal{I}_1 + \mathcal{K}_{22}\mathcal{I}_2 \end{array}\right).$$

By dividing $\mathcal{K}$ into size-$C \times C$ blocks and $\mathcal{I}$ into length-C blocks, we can use the conv2d method for the $C_{out} = C_{in} = C$ case repeatedly to compute the final result.

3.3.5 Depthwise Conv2d and Average Pooling. Whereas dwconv2d with Coefficient encoding is performed in the same way as for regular conv2d, CinS encoding enables a more efficient implementation of dwconv2d due to its structure. When $C_{\mathrm{out}} = C_{in} = C$ as in §3.3.3, we can pack the entire kernels into a single plaintext as $\langle \mathbf{k}_1 | \mathbf{k}_2 | \cdots | \mathbf{k}_C \rangle_{\mathrm{CinS}}$. Then, dwconv2d is equivalent to simply performing a single PMult between the input ciphertext and the kernel plaintext. Also, average pooling can be regarded as a special variant of downsampling dwconv2d where the kernels are equivalent across channels.

4 Fusing Conv2d with Bootstrapping

As described in §3.1, conversions between Slot, CinS, and Coefficient encoding are inherent in FHE bootstrapping process. Thus, incorporating efficient conv2d in CinS encoding leads to a favorable latency reduction. We show that the cost of StoC computation can further be reduced by jointly evaluating CinS conv2d with StoC.

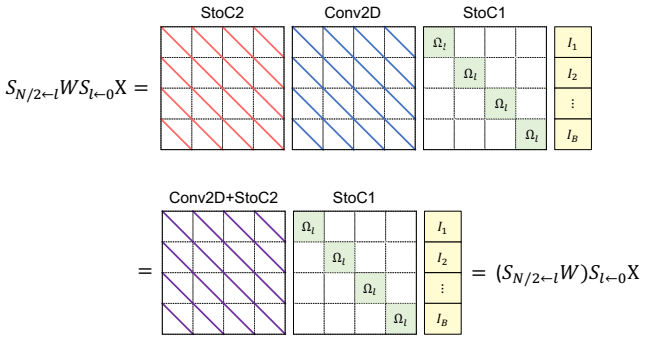


Figure 2: Merging conv2D with StoC. The resulting composite operation simultaneously executes conv2D and StoC, and maintains the same computational cost as the original conv2D while canceling out the remaining StoC steps. Non-zero values exist only at the colored positions of the matrices.

We optimize the operational flow by merging conv2d into the later part of StoC, StoC2 ($S_{N/2\leftarrow\ell}$), to reduce the overall level consumption. We first interpret conv2d operations on CinS-encoded ciphertexts as if being applied to slot-encoded ciphertexts based on Remark 2. We show that the StoC2 matrix $S_{N/2\leftarrow\ell}$ can be fused with the conv2d kernel matrix ($\mathcal{K}$ in Remark 4) without increasing the amount of computation in terms of the number of HE ops for Slot-encoded ciphertexts.

4.1 StoC2 matrix

$S_{N/2\leftarrow\ell}$ corresponds to a set of butterfly operations with strides greater than or equal to ℓ. Therefore, the matrix $S_{N/2\leftarrow\ell}$ only operates in the granularity of a length-ℓ slice of the message vector. In other words, for length-ℓ slices $\mathbf{s}_{i,j}$ that are composed of DFT twiddle factors and $\mathbf{m} = (\mathbf{m}_1|\mathbf{m}_2|\cdots|\mathbf{m}_C)$, the following holds:

$$(S_{N/2\leftarrow\ell}\mathbf{m})_i = \sum_{j=1}^{C} \mathbf{s}_{i,j} \odot \mathbf{m}_j, \tag{9}$$

where $(S_{N/2\leftarrow\ell}\mathbf{m})_i$ is the i-th slice of $S_{N/2\leftarrow\ell}\mathbf{m}$ and $\mathbf{m}_j$ is the j-th slice of $\mathbf{m}$. We can regard $S_{N/2\leftarrow\ell}$ as a $C\times C$ matrix $\mathcal{S}'$ with each element being a length-ℓ slice ($\mathbf{s}_{i,j}$). $\mathbf{m}$ can be similarly regarded as a length-C vector $\mathcal{M}$ with each element being a length-ℓ slice ($\mathbf{m}_j$). Then, $S_{N/2\leftarrow\ell}\mathbf{m}$ can be rewritten as a matrix-vector multiplication $\mathcal{S}'\mathcal{M}$ as in Remark 4, where the multiplication between slices are replaced by element-wise multiplication between them.

4.2 Conv2d matrix

If we reinterpret CinS-encoded ciphertexts as Slot-encoded, the slices $\mathbf{m}_j$ are converted to $\hat{\mathbf{m}}_j$, and the local convolution $*_{\text{local}}$ is converted to element-wise multiplication $\odot$ (see Eq. 5). As a result, Eq. 8 is converted to the following equation in Slot encoding:

$$\hat{\mathbf{o}}_i = \sum_{j=1}^{C} \hat{\mathbf{k}_{i,j}} \odot \hat{\mathbf{i}_j}. \tag{10}$$

As in the previous section, we can regard this as a matrix-vector multiplication $\hat{\mathcal{K}}\hat{\mathcal{I}}$. $\hat{\mathcal{K}}$ is a $C\times C$ matrix of length-ℓ slices ($\hat{\mathbf{k}_{i,j}}$) and $\hat{\mathcal{I}}$ is a length-C vector of length-ℓ slices ($\hat{\mathbf{i}_j}$). Then, the process

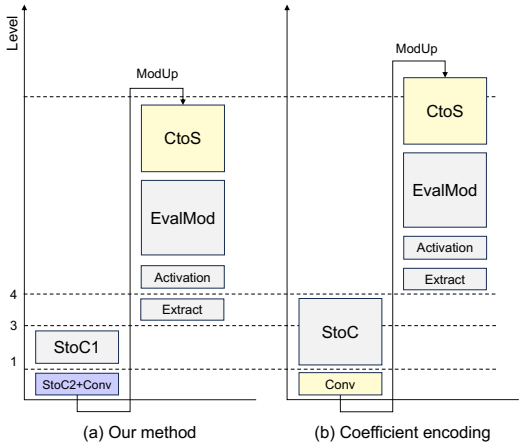


Figure 3: Levels of ciphertexts when evaluating conv2d and activation along with boostrapping. We color operational blocks according to the encoding type of input ciphertexts: Slot- op , Coefficient- op , and CinS- op

of computing conv2d and then StoC2 can be represented as the following:

$$\mathcal{S}'\hat{\mathcal{K}}\hat{\mathcal{I}} = (\mathcal{S}'\hat{\mathcal{K}})\hat{\mathcal{I}} = \hat{\mathcal{K}}'\hat{\mathcal{I}}. \tag{11}$$

Linear algebra still holds even when each element in a matrix or a vector is replaced with a length-ℓ slice, and multiplication is replaced with element-wise multiplication between the slices. Thus, the product of the two $C\times C$ matrices $\mathcal{S}'\hat{\mathcal{K}}$ can be computed first.

As $\mathcal{S}'$ and $\hat{\mathcal{K}}$ are composed of only twiddle factors and kernel weights, which are known to the server, the multiplication $\hat{\mathcal{K}}' = \mathcal{S}'\hat{\mathcal{K}}$ can be precomputed by the server in the offline phase. The resulting $\hat{\mathcal{K}}'$ is a $C\times C$ matrix of length-ℓ slices, having the same form as $\hat{\mathcal{K}}$. Thus, the same computation process in §3.3.3 can be used for the online phase.

That is, the cost of evaluating $\mathcal{S}'\hat{\mathcal{K}}\hat{\mathcal{I}}$ is identical to the cost of evaluating $\hat{\mathcal{K}}\hat{\mathcal{I}}$, completely eliminating the cost of StoC2. Hence, the cost of bootstrapping is significantly reduced by the fusion.

This fusion also reduces the level consumption by one and performs conv2d at the lowest level possible (level 1). This minimizes the memory space required for storing kernel weights, as plaintext sizes are proportional to the level. Algorithm 1 describes the operational flow of our conv2d algorithm followed by activation.

We emphasize that fusion is only beneficial when the two matrices have the same granularity of ℓ. We provide an alternative explanation for the fusion in Figure 2. In the figure, ℓ-granular matrices are represented as a $C\times C$ matrix of blocks, where each block is shown as a $\ell\times\ell$ diagonal matrix, signifying element-wise operations on each slice. Merging two ℓ-granular matrices does not increase the number of cyclic diagonals, thereby retaining the evaluation cost equivalent to that of a single matrix. It is not reasonable to additionally merge StoC1 because it will produce a dense matrix with a large number of non-zero cyclic diagonals, increasing the overall cost for StoC.

4.2.1 Generalization of the fusion. The fusion technique can be easily extended to encompass other linear layers within CNNs. In

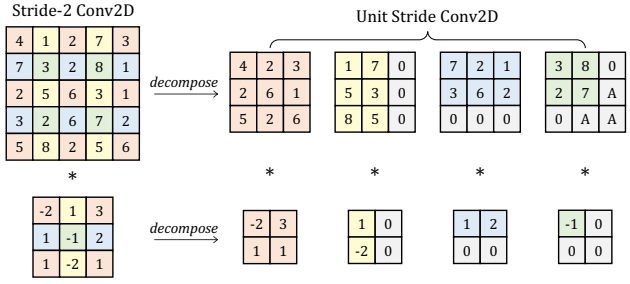


Figure 4: Downsampling conv2d decomposition method for stride $s = 2$.

particular, a fully-connected (FC) layer that performs matrix-vector multiplication utilizing a weight matrix, can be merged with the entire StoC matrix. For FC layers, we use Slot encoding. As Slot encoding is indeed equivalent to CinS encoding with a slice size of $\ell = 1$, the same formulation is applicable for the fusion process. Also, for specific matrix shapes (e.g., $\mathcal{P}_\ell$ in Equation 6), we can even merge it with StoC1 and leave StoC2 for the use in conv2d.

Algorithm 1 Default Conv2d and Activation

Input: A ciphertext of $C\ W \times W = 2\ell$ sized input images with CinS encoding $\mathbf{ct}_{\text{CinS, in}}$, C plaintexts of $C \times C$ preprocessed kernels with CinS encoding $\{\langle \mathbf{k}_i^{\text{diag}'} \rangle_{\text{CinS}}\}_{1 \le i \le C}$, two plaintexts to restore zero padding $\langle \mathbf{mask}_{\text{Re}} \rangle_{\text{slot}}$, $\langle \mathbf{mask}_{\text{Im}} \rangle_{\text{slot}}$

Output: A ciphertext of C output images with CinS encoding $\mathbf{ct}_{\text{CinS, out}}$

1: $\mathbf{ct}_{\text{coeff}} \leftarrow \text{BSGS}(\mathbf{ct}_{\text{CinS, in}}, \{\langle \mathbf{k}_i^{\text{diag}'} \rangle_{\text{CinS}}\}_{1 \le i \le C}, \ell)$
2: $\mathbf{ct}_{\text{slot,Re}}, \mathbf{ct}_{\text{slot,Im}} \leftarrow \text{ModEval}(\text{CtoS}(\mathbf{ct}_{\text{coeff}}))$
3: $\mathbf{ct}_{\text{slot,Re}} \leftarrow \text{HActivation}(\text{PMult}(\mathbf{ct}_{\text{slot,Re}}, \langle \mathbf{mask}_{\text{Re}} \rangle_{\text{slot}}))$
4: $\mathbf{ct}_{\text{slot,Im}} \leftarrow \text{HActivation}(\text{PMult}(\mathbf{ct}_{\text{slot,Im}}, \langle \mathbf{mask}_{\text{Im}} \rangle_{\text{slot}}))$
5: $\mathbf{ct}_{\text{slot}} \leftarrow \text{HAdd}(\mathbf{ct}_{\text{slot,Re}}, \text{iMult}(\mathbf{ct}_{\text{slot,Im}}))$
6: $\mathbf{ct}_{\text{CinS, out}} \leftarrow \text{HMatmul}(S_{\ell \leftarrow -1}, \mathbf{ct}_{\text{slot}})$

5 Execution Flow for Downsampling Conv2d

Our proposed methods apply to any conv2d shape (e.g. pointwise conv2d), but special care is required for downsampling conv2d. We show how we rearrange downsampling conv2d to reduce the number of bootstrapping operations, and significantly reduce the overall computational cost.

Downsampling layers reduce the image size, and thus previous homomorphic conv2d algorithms output sparsely packed ciphertexts, which leads to slot underutilization. This can have a large impact on performance since the expansion of the number of ciphertexts necessitates a larger number of bootstrappings to process the same amount of data. Furthermore, a sparsely packed ciphertext cannot be reorganized to a densely packed ciphertext by simply changing memory addresses as in unencrypted data. It requires a *densify* operation, which is essentially an HMatmul op with a matrix ($\mathcal{G}_\ell$) that rearranges the scattered data into a sequential format. In more detail, the HMatmul consists of PMult ops with mask plaintexts that encode a vector composed of 0 and 1 to extract

Algorithm 2 Downsampling Conv2d and Activation

Input: Ciphertexts of $2C\ W \times W = 2\ell$ sized input images with Slot encoding $\mathbf{ct}_{\text{slot, 1}}$, $\mathbf{ct}_{\text{slot, 2}}$, $8C$ plaintexts of $4C \times 8C$ decomposed and preprocessed kernels with CinS encoding $\{\langle \mathbf{k}_i^{\text{diag}'} \rangle_{\text{CinS}}\}_{1 \le i \le 8C}$, two plaintexts to restore zero padding $\langle \mathbf{mask}_{\text{Re}} \rangle_{\text{slot}}$, $\langle \mathbf{mask}_{\text{Im}} \rangle_{\text{slot}}$

Output: A ciphertext of $4C\ W/2 \times W/2$ output images with CinS encoding $\mathbf{ct}_{\text{CinS, out}}$

1: $\mathbf{r} \leftarrow [\ell/2, W/4, \ell/2 + W/4]$
2: **for** $i \leftarrow 1$ **to** 2 **do**
3: $\quad \mathbf{ct}_{\text{slot,ds},i} \leftarrow \text{HMatmul}(\mathcal{G}_\ell, \mathbf{ct}_{\text{slot},i})$
4: $\quad$ **for** $j \leftarrow 1$ **to** 3 **do**
5: $\quad\quad \mathbf{ct}_{\text{slot,tmp}} \leftarrow \text{HMatmul}(\mathcal{G}_\ell, \mathbf{ct}_{\text{slot},i} \ll \mathbf{r}[j])$
6: $\quad\quad \mathbf{ct}_{\text{slot,tmp}} \leftarrow \mathbf{ct}_{\text{slot,tmp}} \gg (j \times \ell/4)$
7: $\quad\quad \mathbf{ct}_{\text{slot,ds},i} \leftarrow \text{HAdd}(\mathbf{ct}_{\text{slot,ds},i}, \mathbf{ct}_{\text{slot,tmp}})$
8: $\quad$ **end for**
9: $\quad \mathbf{ct}_{\text{CinS},i} \leftarrow \text{HMatmul}(S_{\ell/4 \leftarrow -1}, \mathbf{ct}_{\text{slot,ds},i})$
10: **end for**
11: $\mathbf{ct}_{\text{coeff},1} \leftarrow \text{BSGS}(\mathbf{ct}_{\text{CinS},1}, \{\langle \mathbf{k}_i^{\text{diag}'} \rangle_{\text{CinS}}\}_{1 \le i \le 4C}, \ell/4)$
12: $\mathbf{ct}_{\text{coeff},2} \leftarrow \text{BSGS}(\mathbf{ct}_{\text{CinS},2}, \{\langle \mathbf{k}_i^{\text{diag}'} \rangle_{\text{CinS}}\}_{4C+1 \le i \le 8C}, \ell/4)$
13: $\mathbf{ct}_{\text{coeff}} \leftarrow \text{HAdd}(\mathbf{ct}_{\text{coeff},1}, \mathbf{ct}_{\text{coeff},2})$
14: $\mathbf{ct}_{\text{slot,Re}}, \mathbf{ct}_{\text{slot,Im}} \leftarrow \text{ModEval}(\text{CtoS}(\mathbf{ct}_{\text{coeff}}))$
15: $\mathbf{ct}_{\text{slot,Re}} \leftarrow \text{HActivation}(\text{PMult}(\mathbf{ct}_{\text{slot,Re}}, \langle \mathbf{mask}_{\text{Re}} \rangle_{\text{slot}}))$
16: $\mathbf{ct}_{\text{slot,Im}} \leftarrow \text{HActivation}(\text{PMult}(\mathbf{ct}_{\text{slot,Im}}, \langle \mathbf{mask}_{\text{Im}} \rangle_{\text{slot}}))$
17: $\mathbf{ct}_{\text{slot}} \leftarrow \text{HAdd}(\mathbf{ct}_{\text{slot,Re}}, \text{iMult}(\mathbf{ct}_{\text{slot,Im}}))$
18: $\mathbf{ct}_{\text{CinS,out}} \leftarrow \text{HMatmul}(S_{\ell/4 \leftarrow -1}, \mathbf{ct}_{\text{slot}})$

relevant slots, and many HRot ops to change the order [25]. The resulting densify operation has high computational requirements and is often more expensive than the actual conv2d computation. We defer the details of $\mathcal{G}_\ell$ to Appendix B.

It becomes especially problematic when using the Coefficient or CinS encoding methods. To perform the fine-grained HRot ops of densify, conversion to Slot encoding is required, resulting in the operational flow shown in Figure 6(b). This requirement forces bootstrapping on a large number of sparse ciphertexts. For example, with a stride of $s = 2$, a ciphertext stores only one valid value per four slots, thus four times more bootstrapping evaluations are required than necessary.

We devise a method to reduce the number of bootstrapping in downsampling conv2d by rearranging stride s downsampling convolution into s^2 unit-stride conv2d operations as shown in Figure 4.

As shown in Algorithm 2, this method executes a decomposition step (lines 1-8), similar to densify, prior to conv2d. Afterwards, It converts each ciphertext to CinS encoding and performs unit-stride convolutions (line 9-12), and aggregates the convolution results to generate the densely packed ciphertexts embedding the downsampling convolution result (line 13). Prepending the decomposition step significantly reduces the bootstrapping cost by facilitating an execution flow that avoids the need to bootstrap sparse ciphertexts. The resulting execution flow is shown in Figure 6(a). The decomposition initially resizes $w \times w$ images and $f \times f$ kernels to s^2 $\frac{w}{s} \times \frac{w}{s}$ images and $\frac{f}{s} \times \frac{f}{s}$ kernels. During this process, the number of

Table 2: NeuJeans inference time for ResNet18, ResNet50, and MobileNetV2 models using a single ImageNet image. We implement the idea proposed in [25] and set it as our baseline. For ResNet18-AESPA, we also implemented the idea proposed in [26]. Polynomial approximation is used for activation functions, except in the case of ResNet18-AESPA.

Execution time (s)	ResNet18-AESPA			ResNet18		ResNet50		MobileNetV2	
	[25]	[26]	NeuJeans	[25]	NeuJeans	[25]	NeuJeans	[25]	NeuJeans
CtoS+ModEval	10.57	3.27	4.08	14.23	5.48	33.71	20.76	23.24	16.70
StoC	0.68	0.35	0.46	0.60	0.48	2.63	1.84	1.85	1.68
Conv2d	2.26	3.94	0.25	2.30	0.21	14.82	1.92	12.45	0.41
Activation	0.17	0.36	0.08	7.55	7.44	29.87	30.61	21.02	21.14
Densify/Decomp	0.77	0.00	0.46	1.22	0.48	1.47	0.91	1.01	0.79
ETC	0.02	1.82	0.02	0.02	0.02	0.03	0.03	0.04	0.01
Total	14.47	9.74	5.35	25.92	14.11	82.53	56.08	60.01	40.76
Speedup	-	1.49×	2.70×	-	1.84×	-	1.47×	-	1.47×

Table 3: CKKS parameters used for the evaluation of NeuJeans. N is the ring degree, L is the max level, Δ is the scale factor for encoding, and λ is the security level.

Param set	N	L	$dnum$	Δ	h	$\log pq$	λ
Set1	2^{16}	24	5	2^{42}	192	1555	> 128
Set2	2^{16}	19	4	2^{42}	192	1345	> 128

ciphertexts and plaintexts remains constant, as they are all densely packed, fully utilizing the slots.

This modification has a large performance impact for complex networks like ResNet18 for ImageNet; we were able to reduce the number of total bootstrapping operations by more than two times for ResNet18. We also further optimize densify/decompose operations by applying the BSGS optimization in §2.6.

6 Evaluation

6.1 Experimental Setup & CNN Model Training

We implemented NeuJeans using the HEaaN CKKS library with the support for GPU operation [14]. We select the parameters for CKKS as in Table 3, which guarantees over 128 bits of security [11]. We also implemented the state-of-the-art FHE-based PI methods proposed in [25, 26] using the same library to compare with NeuJeans. Specifically, we selected [25] as the representative method for coefficient encoding and [26] for slot encoding. However, [26] is not designed for CNNs that require high-degree polynomial approximations, thus we could not test it in such cases. We measured the execution time of server-side tasks on a system with an AMD EPYC 7452 CPU, an NVIDIA A100 GPU, and 480GB of DDR4-3200 memory. Before execution, we load weight plaintexts and evaluation keys into GPU memory. For the evaluation of client-side tasks, we used a laptop computer, Apple MacBook Air M1 with 8GB memory. All client-side tasks use a single CPU thread and do not use GPU.

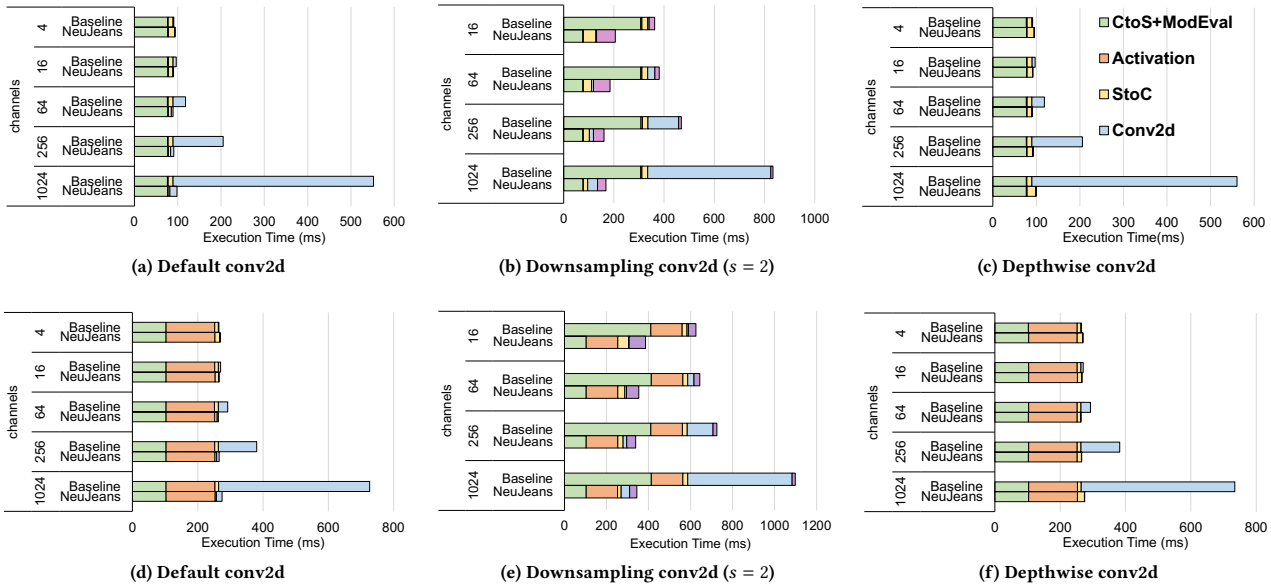
Using NeuJeans, we implemented ResNet18, ResNet50, and MobileNetV2 for the ImageNet dataset using ReLU **poly approx** (polynomial approximation) and additionally implemented ResNet18 using **AESPA** [36]. As described in §2.5.2, we can use a simple square function for activation after training with AESPA. For AESPA, we trained the model for 200 epochs using standard supervised learning methods and employed the same hyperparameters as stated in the paper. For poly approx, we used high-degree polynomial approximations of ReLU based on [10]. We also attempted to employ ReLU approximation in [30] but decided not to use it as it resulted in higher approximation errors and produced unstable results. For all networks, MaxPool is replaced by AvgPool. Whereas this replacement is handled within the training process for AESPA, we use the method described in [2] for poly approx, which gradually replaces MaxPool with AvgPool by fine-tuning a pre-trained model for 20 epochs. We used the pre-trained network from [45] and trained the models using the PyTorch framework [37].

6.2 End-to-End CNN Inference Result

NeuJeans reduces the execution time of FHE-based CNN inference to as low as 5.35 seconds, as shown in Table 2. Compared to the baseline, NeuJeans achieves speedups of 1.84×, 1.47×, and 1.47× for ResNet18, ResNet50, and MobileNetV2 networks based on poly approx, while NeuJeans achieves 2.70× speedup for ResNet18-AESPA network. Larger speedups are achieved for AESPA because activation, which NeuJeans does not accelerate, constitutes a smaller portion in the execution time for AESPA.

The overall performance gains are attributed to 7.72–30.37× speedups on conv2d and 1.39–2.59× speedups on CtoS+ModEval compared to the baseline. NeuJeans's conv2d algorithm based on CinS encoding leads to faster conv2d execution by significantly reducing the number of HE ops. Also, our optimized execution flow for downsampling conv2d reduces the number of bootstrapping evaluations, which in turn decreases the CtoS+ModEval time.

Similar to [25], NeuJeans performs conv2d and activation on different encodings, requiring bootstrapping for each conv2d-activation sequence to avoid additional costs from encoding transitions. In contrast, [26] exclusively uses slot encoding and performs bootstrapping after every two conv2d layers, thereby reducing the overall bootstrapping time. However, the significant number of rotations required for conv2d in [26] outweigh the benefits of the reduced bootstrapping time. Consequently, NeuJeans achieves a 1.82× speedup for ResNet18-AESPA compared to [26].



(a) Default conv2d (b) Downsampling conv2d ($s = 2$) (c) Depthwise conv2d

(d) Default conv2d (e) Downsampling conv2d ($s = 2$) (f) Depthwise conv2d

Figure 5: Latency of default, downsampling ($s = 2$), and depthwise conv2d followed by activation using the conv2d methods in baseline [25] and NeuJeans for ResNet18 networks using the activation method of (a)(b)(c) AESPA and (d)(e)(f) polynomial approximation.

Table 4: ImageNet image classification Top-1 and Top-5 Accuracy (%) with 3000 tested images.

Network type	Unencrypted accuracy		NeuJeans accuracy	
	Top-1	Top-5	Top-1	Top-5
ResNet18-AESPA	66.4	87.3	66.4	87.2
ResNet18	67.2	87.7	64.8	85.4
ResNet50	76.4	92.9	74.1	90.5

6.3 Microbenchmark Result

To further analyze how NeuJeans accelerates conv2d, we conducted microbenchmarks for various types of conv2d. Figure 5 presents the results for default, downsampling, and depthwise conv2d, each followed by an activation function. We used the baseline case of $C = C_{in} = C_{out}$, performing conv2d on a single ciphertext while varying C, the number of channels that can be packed into a ciphertext. For downsampling conv2d, we used the case of $C = 2C_{in} = C_{out}$, where two ciphertexts are downsampled to fit into a single ciphertext. These results can be generalized to other cases based on §3.3.4.

For default conv2d (Figures 5(a) and 5(d)), we observe that the total execution time remains relatively constant when using NeuJeans, whereas the execution time of the baseline rapidly increases with C. This is due to two main factors. First, the HRot complexity of conv2d increases slowly, in proportion to $\sqrt{C}$ for NeuJeans, while it increases in proportion to C for the baseline (see Table 1). Second, when many channels fit in a ciphertext, StoC time decreases by a large margin. For the DFT matrix $\mathcal{T}_{N/2} = S_{N/2\leftarrow\ell} S_{\ell\leftarrow 0}$, NeuJeans fuses $S_{N/2\leftarrow\ell}$ with the evaluation of conv2d, leaving only $S_{\ell\leftarrow 1}$ multiplication for StoC. For a large C, $\ell = N/C$ becomes small, which makes the evaluation of $S_{\ell\leftarrow 1}$ multiplication much cheaper. Overall, up to 5.62× (respectively, 2.65×) speedups are achieved for conv2d that uses the AESPA (poly approx) activation method.

In downsampling conv2d (Figure 5(b) and 5(e)), the difference in execution time is more pronounced even for low-C cases. This is primarily due to the optimized decomposition-based execution flow for downsampling conv2d, which requires performing bootstrapping on only one-fourth the number of ciphertexts compared to the baseline. As a result, CtoS+ModEval time decreases by 3.98–4.01×. Also, the aforementioned benefits of using NeuJeans for default conv2d apply in a similar manner for downsampling conv2d.

Finally, depthwise conv2d (Figure 5(c) and 5(f)) shows a similar trend to default conv2d. In fact, the baseline performs an identical job for both cases. Meanwhile, NeuJeans further reduces the execution time of depthwise conv2d by requiring only a single PMult op for its evaluation. Overall, up to 5.68× (respectively, 2.68×) speedups are achieved for conv2d that uses the AESPA (poly approx) method.

6.4 Accuracy

In Table 4, we present the FHE classification accuracy results for the ResNet18 and ResNet50 networks, based on 3,000 random samples from the ImageNet validation set, along with unencrypted classification results. For the ResNet18 model trained using AESPA, we observed accuracy comparable to that of the backbone model, achieving 66.4% of top-1 and 87.2% of top-5 accuracy for the encrypted classification. In contrast, the use of polynomial approximations results in 2.3–2.4% drop in accuracy.

To evaluate the impact of our approach on precision, we tested a single conv2d operation with varying channel counts, up to 1024,

Table 5: The cost of client-side tasks on a laptop computer for ImageNet inference. Client-side tasks stay the same regardless of the parameter set or the activation method.

Implementation (encoding, domain)	Encryption Time	Encryption Ctxt size	Decryption Time	Decryption Ctxt size
[25] (coefficient, $\mathbb{R}^N$)	17.3ms	6MB	2.11ms	1MB
NeuJeans (CinS, $\mathbb{C}^{N/2}$)	45.7ms	12MB	2.11ms	1MB
NeuJeans (CinS, $\mathbb{R}^N$)	22.1ms	6MB	2.11ms	1MB

combined with StoC, and analyzed the errors from FHE execution. The largest errors occurred when 1024 channels were packed per ciphertext, with both NeuJeans and [25] yielding mean absolute errors around 1.4e-6. Given that the error from ReLU polynomial approximation [31] is around 1.5e-4, the conv2d errors are minimal and do not significantly impact accuracy. Therefore, the disparity in accuracy drop can be attributed to whether the original network operates as-is (using AESPA) or uses approximation. Prior FHE-based CNN inference studies [25, 29] have also experienced non-negligible accuracy drops due to errors in approximation. However, the impact appears more pronounced in our evaluation, likely due to the larger size of our network.

6.5 Client Overhead

Although the client system used for the evaluation has much lower computational capability than the server system, client computation time accounts for only a tiny portion in the end-to-end inference time. Table 5 shows the execution time of client-side tasks, which only include encryption and decryption. Compared to [25] using Coefficient encoding, which has negligible encoding cost, NeuJeans involves additional computations for CinS encoding, adding 4.8ms to the encryption time. It is important to use the dense packing for real numbers in §3.3.1 because it reduces the number of ciphertexts to encrypt for a fixed input size, also reducing the total encryption time. For decryption, Slot encoding is used for all the implementations, resulting in the same decryption time. Regardless of the implementation method, the use of FHE incurs small overhead for the clients, adding only dozens of milliseconds to the end-to-end inference time.

Table 5 also shows the total size of ciphertexts that are sent to and received from the server for a single inference. The use of FHE enables NeuJeans to perform CNN inference with only 7MB of total data communication,

7 Conclusion

NeuJeans is an optimized solution for private inference of deep convolutional neural networks (CNNs) based on a cryptographic primitive, fully homomorphic encryption (FHE). NeuJeans incorporates efficient algorithms for the evaluation of convolutional layers (conv2d), enabled by a dedicated encoding method for FHE ciphertexts. Through leveraging common patterns in conv2d and bootstrapping, NeuJeans eliminates the computational redundancy by fusing conv2d with bootstrapping. Finally, NeuJeans provides FHE-friendly execution flows that minimize the cost of bootstrapping and various conv2d layers when applying our methods to the private end-to-end inference of deep CNNs. Our experiments exhibit up to 5.68× of speedups achieved by NeuJeans across various types of conv2d. We demonstrate that NeuJeans performs ResNet18 ImageNet classification within 5.35 seconds on an A100 GPU system, while inducing minimal computation and communication overhead for the client.

Acknowledgments

This work was supported by Samsung Electronics Co., Ltd(IO201207-07812-01) and Institute of Information & communications Technology Planning & Evaluation (IITP) grant funded by the Korea government (MSIT) (No. 2021-0-01343, IITP-2023-RS-2023-00256081, and IITP-2024-RS-2024-00469698). The ICT at Seoul National University (SNU) provides research facilities for this study. Jung Ho Ahn, the corresponding author, is with the Department of Intelligence and Information and the Interdisciplinary Program in Artificial Intelligence, SNU.

References

[1] Peter Martey Addo, Dominique Guegan, and Bertrand Hassani. 2018. Credit Risk Analysis Using Machine and Deep Learning Models. *Risks* 6, 2 (2018), 38. https://doi.org/10.3390/risks6020038

[2] Moran Baruch, Nir Drucker, Gilad Ezov, Eyal Kushnir, Jenny Lerner, Omri Soceanu, and Itamar Zimerman. 2023. Sensitive Tuning of Large Scale CNNs for E2E Secure Prediction using Homomorphic Encryption. arXiv preprint arXiv:2304.14836.

[3] Jean-Philippe Bossuat, Christian Mouchet, Juan Ramón Troncoso-Pastoriza, and Jean-Pierre Hubaux. 2021. Efficient Bootstrapping for Approximate Homomorphic Encryption with Non-sparse Keys. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT)*. Springer. https://doi.org/10.1007/978-3-030-77870-5_21

[4] Jean-Philippe Bossuat, Juan Ramón Troncoso-Pastoriza, and Jean-Pierre Hubaux. 2022. Bootstrapping for Approximate Homomorphic Encryption with Negligible Failure-Probability by Using Sparse-Secret Encapsulation. In *International Conference on Applied Cryptography and Network Security (ACNS)*. Springer. https://doi.org/10.1007/978-3-031-09234-3_26

[5] William Bowditch, Will Abramson, William J. Buchanan, Nikolaos Pitropakis, and Adam J. Hall. 2020. Privacy-preserving Surveillance Methods using Homomorphic Encryption. In *International Conference on Information Systems Security and Privacy (ICISSP)*. https://doi.org/10.5220/0008864902400248

[6] Hao Chen, Ilaria Chillotti, and Yongsoo Song. 2019. Improved Bootstrapping for Approximate Homomorphic Encryption. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT)*. https://doi.org/10.1007/978-3-030-17656-3_2

[7] Jung Hee Cheon, Kyoohyung Han, Andrey Kim, Miran Kim, and Yongsoo Song. 2018. A Full RNS Variant of Approximate Homomorphic Encryption. In *Selected Areas in Cryptography (SAC)*. https://doi.org/10.1007/978-3-030-10970-7_16

[8] Jung Hee Cheon, Kyoohyung Han, Andrey Kim, Miran Kim, and Yongsoo Song. 2018. Bootstrapping for Approximate Homomorphic Encryption. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT)*. https://doi.org/10.1007/978-3-319-78381-9_14

[9] Jung Hee Cheon, Andrey Kim, Miran Kim, and Yong Soo Song. 2017. Homomorphic Encryption for Arithmetic of Approximate Numbers. In *International Conference on the Theory and Application of Cryptology and Information Security, (ASIACRYPT)*. https://doi.org/10.1007/978-3-319-70694-8_15

[10] Jung Hee Cheon, Dongwoo Kim, and Duhyeong Kim. 2020. Efficient Homomorphic Comparison Methods with Optimal Complexity. In *International Conference on the Theory and Application of Cryptology and Information Security, (ASIACRYPT)*. https://doi.org/10.1007/978-3-030-64834-3_8

[11] Jung Hee Cheon, Yongha Son, and Donggeon Yhee. 2021. Practical FHE Parameters Against Lattice Attacks. Cryptology ePrint Archive, Paper 2021/039.

[12] Francois Chollet. 2017. Xception: Deep Learning With Depthwise Separable Convolutions. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. https://doi.org/10.1109/CVPR.2017.195

[13] James W. Cooley and John W. Tukey. 1965. An Algorithm for the Machine Calculation of Complex Fourier Series. *Math. Comp.* 19, 90 (1965), 297–301.

[14] Crypto Lab. 2022. HEaaN Library. https://heaan.it/

[15] EU. 2016. Regulation (EU) 2016/679 of the European Parliament and of the Council. http://data.europa.eu/eli/reg/2016/679/oj

[16] Karthik Garimella, Nandan Kumar Jha, and Brandon Reagen. 2021. Sisyphus: A Cautionary Tale of Using Low-Degree Polynomial Activations in Privacy-Preserving Deep Learning. arXiv preprint arXiv:2107.12342.
[17] Varun Gulshan, Lily Peng, Marc Coram, Martin C Stumpe, Derek Wu, Arunachalam Narayanaswamy, Subhashini Venugopalan, Kasumi Widner, Tom Madams, Jorge Cuadros, et al. 2016. Development and Validation of a Deep Learning Algorithm for Detection of Diabetic Retinopathy in Retinal Fundus Photographs. *JAMA* 316, 22 (2016), 2402–2410. https://doi.org/10.1001/jama.2016.17216
[18] Shai Halevi and Victor Shoup. 2018. Faster Homomorphic Linear Transformations in HElib. In *Annual International Cryptology Conference (CRYPTO)*. https://doi.org/10.1007/978-3-319-96884-1_4
[19] Kyoohyung Han, Minki Hhan, and Jung Hee Cheon. 2019. Improved homomorphic discrete fourier transforms and fhe bootstrapping. *IEEE Access* 7 (2019), 57361–57370.
[20] Ehsan Hesamifard, Hassan Takabi, and Mehdi Ghasemi. 2017. CryptoDL: Deep Neural Networks over Encrypted Data. arXiv preprint arXiv:1711.05189.
[21] Zhicong Huang, Wen jie Lu, Cheng Hong, and Jiansheng Ding. 2022. Cheetah: Lean and Fast Secure Two-Party Deep Neural Network Inference. In *USENIX Security Symposium*.
[22] Takumi Ishiyama, Takuya Suzuki, and Hayato Yamana. 2020. Highly Accurate CNN Inference Using Approximate Activation Functions over Homomorphic Encryption. In *IEEE International Conference on Big Data (BigData)*. https://doi.org/10.1109/BigData50022.2020.9378372
[23] Wonkyung Jung, Sangpyo Kim, Jung Ho Ahn, Jung Hee Cheon, and Younho Lee. 2021. Over 100x Faster Bootstrapping in Fully Homomorphic Encryption through Memory-centric Optimization with GPUs. *IACR Trans. Cryptogr. Hardw. Embed. Syst.* (2021). https://doi.org/10.46586/tches.v2021.i4.114-148
[24] Chiraag Juvekar, Vinod Vaikuntanathan, and Anantha P. Chandrakasan. 2018. GAZELLE: A Low Latency Framework for Secure Neural Network Inference. In *USENIX Security Symposium*.
[25] Dongwoo Kim and Cyril Guyot. 2023. Optimized Privacy-Preserving CNN Inference With Fully Homomorphic Encryption. *IEEE Transactions on Information Forensics and Security (TIFS)* 18 (2023), 2175–2187. https://doi.org/10.1109/TIFS.2023.3263631
[26] Donghwan Kim, Jaiyoung Park, Jongmin Kim, Sangpyo Kim, and Jung Ho Ahn. 2023. HyPHEN: A Hybrid Packing Method and Its Optimizations for Homomorphic Encryption-Based Neural Networks. *IEEE Access* (2023). https://ieeexplore.ieee.org/abstract/document/10376063
[27] Jongmin Kim, Gwangho Lee, Sangpyo Kim, Gina Sohn, Minsoo Rhu, John Kim, and Jung Ho Ahn. 2022. ARK: Fully Homomorphic Encryption Accelerator with Runtime Data Generation and Inter-Operation Key Reuse. In *2022 55th IEEE/ACM International Symposium on Microarchitecture (MICRO)*. 1237–1254. https://doi.org/10.1109/MICRO56248.2022.00086
[28] Sangpyo Kim, Jongmin Kim, Michael Jaemin Kim, Wonkyung Jung, John Kim, Minsoo Rhu, and Jung Ho Ahn. 2022. BTS: an accelerator for bootstrappable fully homomorphic encryption. In *ISCA '22: The 49th Annual International Symposium on Computer Architecture, New York, New York, USA, June 18 - 22, 2022*. https://doi.org/10.1145/3470496.3527415
[29] Eunsang Lee, Joon-Woo Lee, Junghyun Lee, Young-Sik Kim, Yongjune Kim, Jong-Seon No, and Woosuk Choi. 2022. Low-Complexity Deep Convolutional Neural Networks on Fully Homomorphic Encryption Using Multiplexed Parallel Convolutions. In *International Conference on Machine Learning (ICML)*.
[30] Eunsang Lee, Joon-Woo Lee, Jong-Seon No, and Young-Sik Kim. 2022. Minimax Approximation of Sign Function by Composite Polynomial for Homomorphic Comparison. *IEEE Transactions on Dependable and Secure Computing* 19, 6 (2022), 3711–3727. https://doi.org/10.1109/TDSC.2021.3105111
[31] Junghyun Lee, Eunsang Lee, Joon-Woo Lee, Yongjune Kim, Young-Sik Kim, and Jong-Seon No. 2021. Precise Approximation of Convolutional Neural Networks for Homomorphically Encrypted Data. arXiv preprint arXiv:2105.10879.
[32] Joon-Woo Lee, Hyungchul Kang, Yongwoo Lee, Woosuk Choi, Jieun Eom, Maxim Deryabin, Eunsang Lee, Junghyun Lee, Donghoon Yoo, Young-Sik Kim, and Jong-Seon No. 2022. Privacy-Preserving Machine Learning With Fully Homomorphic Encryption for Deep Neural Network. *IEEE Access* 10 (2022), 30039–30054. https://doi.org/10.1109/ACCESS.2022.3159694
[33] Jian Liu, Mika Juuti, Yao Lu, and N. Asokan. 2017. Oblivious Neural Network Predictions via MiniONN Transformations. In *ACM SIGSAC Conference on Computer and Communications Security (CCS)*. https://doi.org/10.1145/3133956.3134056
[34] Pratyush Mishra, Ryan Lehmkuhl, Akshayaram Srinivasan, Wenting Zheng, and Raluca Ada Popa. 2020. Delphi: A Cryptographic Inference Service for Neural Networks. In *USENIX Security Symposium*.
[35] Lucien K. L. Ng and Sherman S. M. Chow. 2021. GForce: GPU-Friendly Oblivious and Rapid Neural Network Inference. In *USENIX Security Symposium*.
[36] Jaiyoung Park, Michael Jaemin Kim, Wonkyung Jung, and Jung Ho Ahn. 2022. AESPA: Accuracy Preserving Low-degree Polynomial Activation for Fast Private Inference. arXiv preprint arXiv:2201.06699.
[37] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. 2019. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In *Advances in Neural Information Processing Systems*. Vol. 32. Curran Associate, Inc, 8024–8035.
[38] Deevashwer Rathee, Mayank Rathee, Nishant Kumar, Nishanth Chandran, Divya Gupta, Aseem Rastogi, and Rahul Sharma. 2020. CrypTFlow2: Practical 2-Party Secure Inference. In *ACM SIGSAC Conference on Computer and Communications Security (CCS)*. https://doi.org/10.1145/3372297.3417274
[39] M. Sadegh Riazi, Kim Laine, Blake Pelton, and Wei Dai. 2020. HEAX: An Architecture for Computing on Encrypted Data. In *ASPLOS '20: Architectural Support for Programming Languages and Operating Systems, Lausanne, Switzerland, March 16-20, 2020*. https://doi.org/10.1145/3373376.3378523
[40] Nikola Samardzic, Axel Feldmann, Aleksandar Krastev, Nathan Manohar, Nicholas Genise, Srinivas Devadas, Karim Eldefrawy, Chris Peikert, and Daniel Sánchez. 2022. CraterLake: a hardware accelerator for efficient unbounded computation on encrypted data. In *ISCA '22: The 49th Annual International Symposium on Computer Architecture, New York, New York, USA, June 18 - 22, 2022*. https://doi.org/10.1145/3470496.3527393
[41] Akhilesh Kumar Sharma, Shamik Tiwari, Gaurav Aggarwal, Nitika Goenka, Anil Kumar, Prasun Chakrabarti, Tulika Chakrabarti, Radomir Gono, Zbigniew Leonowicz, and Michał Jasiński. 2022. Dermatologist-level Classification of Skin Cancer Using Cascaded Ensembling of Convolutional Neural Network and Handcrafted Features Based Deep Neural Network. *IEEE Access* 10 (2022), 17920–17932. https://doi.org/10.1109/ACCESS.2022.3149824
[42] Henrik V. Sorensen, Douglas L. Jones, Michael T. Heideman, and C. Sidney Burrus. 1987. Real-Valued Fast Fourier Transform Algorithms. *IEEE Transactions on Acoustics, Speech, and Signal Processing* 35, 6 (1987), 849–863. https://doi.org/10.1109/TASSP.1987.1165220
[43] U.S. Department of Health & Human Services. 1996. Health Insurance Portability and Accountability Act of 1996.
[44] Jean-Luc Watson, Sameer Wagh, and Raluca Ada Popa. 2022. Piranha: A GPU Platform for Secure Computation. In *USENIX Security Symposium*.
[45] Ross Wightman, Hugo Touvron, and Herve Jegou. 2021. ResNet Strikes Back: An Improved Training Procedure in Timm. arXiv preprint arXiv:2110.00476.
[46] Qiao Zhang, Chunsheng Xin, and Hongyi Wu. 2021. GALA: Greedy ComputAtion for Linear Algebra in Privacy-Preserved Neural Networks. In *Annual Network and Distributed System Security Symposium, (NDSS)*.

A Real CinS Encoding

Consider a real message vector $\mathbf{m} = (\mathbf{m}_1|\mathbf{m}_2|\cdots|\mathbf{m}_C) \in \mathbb{R}^N$ that is CinS encoded as in §3.3.1. Then from the matrix properties of §3.1, the following holds.

$$\begin{aligned}\langle \mathbf{m} \rangle_{\text{CinS}} &= \langle S_{\ell \leftarrow 1} \mathcal{P}_\ell (\check{\mathbf{m}}_1 | \cdots | \check{\mathbf{m}}_C) \rangle_{\text{slot}} \\ &= \langle S_{\ell \leftarrow 1} (P_\ell \check{\mathbf{m}}_1 | \cdots | P_\ell \check{\mathbf{m}}_C) \rangle_{\text{slot}} \\ &= \langle (\mathcal{T}_\ell P_\ell \check{\mathbf{m}}_1 | \cdots | \mathcal{T}_\ell P_\ell \check{\mathbf{m}}_C) \rangle_{\text{slot}}\end{aligned} \tag{12}$$

Let $DFT'_{2\ell}(\mathbf{m}_i) = \mathcal{T}_\ell P_\ell \check{\mathbf{m}}_i$ as in shown Equation 6. Since P_ℓ is an $\ell \times \ell$ bit-reversal permutation matrix, we can simplify $\mathcal{T}_\ell P_\ell$.

$$\mathcal{T}_\ell P_\ell = \left(\omega_{4\ell}^{5^j \cdot \text{rev}_\ell(k)}\right)_{0 \le j,k < \ell} P_\ell = \left(\omega_{4\ell}^{5^j \cdot k}\right)_{0 \le j,k < \ell} \tag{13}$$

$\omega_{4\ell}^{5^j}$ has the following property.

$$(\omega_{4\ell}^{5^j})^\ell = e^{2\pi\sqrt{-1}\ell 5^j/4\ell} = (e^{\pi\sqrt{-1}/2})^{5^j} = (\sqrt{-1})^{5^j} = \sqrt{-1}$$

Then, $(DFT'_{2\ell}(\mathbf{m}_i))_j$ becomes

$$\begin{aligned}&(DFT'_{2\ell}(\mathbf{m}_i))_j \\ =&\left(\left(\omega_{4\ell}^{5^j \cdot k}\right)_{0 \le j,k < \ell} \check{\mathbf{m}}_i\right)_j \\ =&\sum_{k=0}^{\ell-1} (\mathbf{m}_i)_k (\omega_{4\ell}^{5^j})^k + \sum_{k=0}^{\ell-1} \sqrt{-1} (\mathbf{m}_i)_{\ell+k} (\omega_{4\ell}^{5^j})^k \\ =&\sum_{k=0}^{\ell-1} (\mathbf{m}_i)_k (\omega_{4\ell}^{5^j})^k + \sum_{k=0}^{\ell-1} (\mathbf{m}_i)_{\ell+k} (\omega_{4\ell}^{5^j})^{\ell+k} \\ =&\sum_{k=0}^{2\ell-1} (\mathbf{m}_i)_k (\omega_{4\ell}^{5^j})^k\end{aligned}$$

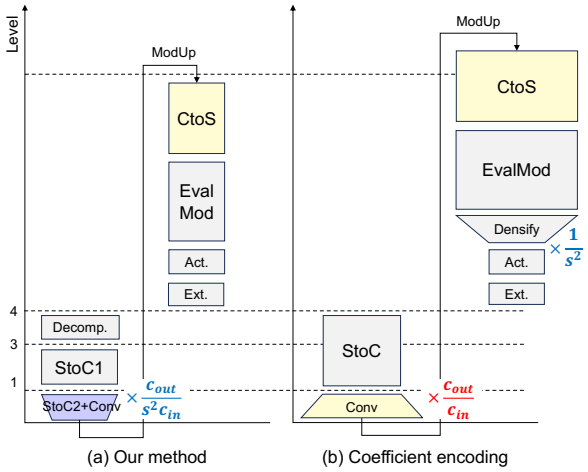


Figure 6: Levels of ciphertexts when evaluating downsampling conv2d and activation along with boostrapping. The width of a block indicates the number of ciphertexts in each operation of a ResNet downsampling layer.

Given this definition of $DFT'_{2\ell}$, the following holds for real vectors $\mathbf{x}, \mathbf{y} \in \mathbb{R}^{2\ell}$ ($*_{2\ell}$ is the negacyclic convolution).

$$
\begin{aligned}
&(DFT'_{2\ell}(\mathbf{x}) \odot DFT'_{2\ell}(\mathbf{y}))_j \\
=&(DFT'_{2\ell}(\mathbf{x}))_j \cdot (DFT'_{2\ell}(\mathbf{y}))_j \\
=&(\sum_{k=0}^{2\ell-1} (\mathbf{x})_k (\omega_{4\ell}^{5^j})^k) \cdot (\sum_{k=0}^{2\ell-1} (\mathbf{y})_k (\omega_{4\ell}^{5^j})^k) \\
=&\sum_{k=0}^{2\ell-1} \sum_{s=0}^{k} (\mathbf{x})_s (\mathbf{y})_{k-s} (\omega_{4\ell}^{5^j})^k + \sum_{k=2\ell}^{4\ell-1} \sum_{s=k-2\ell+1}^{2\ell-1} (\mathbf{x})_s (\mathbf{y})_{k-s} (\omega_{4\ell}^{5^j})^k \\
=&\sum_{k=0}^{2\ell-1} \sum_{s=0}^{k} (\mathbf{x})_s (\mathbf{y})_{k-s} (\omega_{4\ell}^{5^j})^k + \sum_{k=0}^{2\ell-1} \sum_{s=k+1}^{2\ell-1} (\mathbf{x})_s (\mathbf{y})_{2\ell+k-s} (\omega_{4\ell}^{5^j})^k (\omega_{4\ell}^{5^j})^{2\ell} \\
=&\sum_{k=0}^{2\ell-1} \left(\sum_{s=0}^{2\ell-1} (-1)^{\lfloor \frac{k-s}{2\ell} \rfloor} (\mathbf{x})_s (\mathbf{y})_{(k-s) \bmod 2\ell} \right) (\omega_{4\ell}^{5^j})^k \\
=&\sum_{k=0}^{2\ell-1} (\mathbf{x} *_{2\ell} \mathbf{y})_k (\omega_{4\ell}^{5^j})^k \\
=&(DFT'_{2\ell}(\mathbf{x} *_{2\ell} \mathbf{y}))_j
\end{aligned}
$$

Thus we can derive the core property of real CinS encoding.

$$
\begin{aligned}
&\langle \mathbf{m} \rangle_{\text{CinS}} \cdot \langle \mathbf{m}' \rangle_{\text{CinS}} \\
=&\langle (DFT'_{2\ell}(\mathbf{m}_1) | \cdots | DFT'_{2\ell}(\mathbf{m}_C)) \odot (DFT'_{2\ell}(\mathbf{m}'_1) | \cdots | DFT'_{2\ell}(\mathbf{m}'_C)) \rangle_{\text{slot}} \\
=&\langle (DFT'_{2\ell}(\mathbf{m}_1 *_{2\ell} \mathbf{m}'_1) | \cdots | DFT'_{2\ell}(\mathbf{m}_C *_{2\ell} \mathbf{m}'_C)) \rangle_{\text{slot}} \\
=&\langle (\mathbf{m}_1 *_{2\ell} \mathbf{m}'_1 | \cdots | \mathbf{m}_C *_{2\ell} \mathbf{m}'_C) \rangle_{\text{CinS}}
\end{aligned}
$$

B Downsampling Conv2d

We explain the decompose/densify methods from §5 in more detail. Both decompose and densify operations require the image to be packed sparsely in slots, and execute fine-grained masking (PMult) and rotation (HRot). However, decompose prevents the mass generation of unwanted result by changing the image and kernel arrangement before evaluating conv2d, while densify removes unwanted result after it has been created. The computation of bootstrapping after single-stride conv2d, and before densify, greatly amplifies this difference, as shown in Figure 6(b).

Computationally, decompose and densify are surprisingly similar. This is because the pattern of the data which densify extracts is almost identical to the rearrangement pattern of decompose. In detail, the densify operation extracts and rearranges only the red colored data from Figure 4. Decompose is essentially repeating the densify operation (HMatmul($\mathcal{G}_\ell, \cdot$)) four times, but with rotations before and after (line 5, 6 of Algorithm 2) for applying densify to the right spot, and combining the results.

The exact form of matrix $\mathcal{G}_\ell$ is rather complex because the arrangement of the pixels in Figure 4 undergoes bit-reversal permutation when being converted between different encodings. Here we write a brief description of the matrix $\mathcal{G}_\ell$ from Algorithm 2, and refer the reader to [25] for a more detailed explanation.

$\mathcal{G}_\ell$ is a block diagonal matrix of total size $N/2 \times N/2$. Each block is a $\ell \times \ell$ matrix G_ℓ. The matrix G_ℓ itself is also a $2W \times 2W$ block matrix $((G_\ell)_{i,j})_{0 \le i,j < 2W}$, with each block being a matrix of size $W/4 \times W/4$. $(G_\ell)_{i,j}$ can be expressed as the following, where I_n is an $n \times n$ identity matrix and O_n is an $n \times n$ zero matrix.

$$
(G_\ell)_{i,j} = \begin{cases} I_{W/4} & \text{if } j = 2i \text{ and } i \le W/2 \\ O_{W/4} & \text{otherwise} \end{cases}
$$

Table 6: Notations Summary

Notation	Description
N	Degree of plaintext polynomial.
ℓ	Length of a vector slice, which can be any power-of-two smaller than N.
C	Number of vector slices that fit in a message. N/ℓ for real CinS encoding and $(N/2)/\ell$ for complex.
P_ℓ	Length ℓ bit-reversal permutation matrix.
$\mathcal{P}_\ell$	Block diagonal matrix of a total size of $N/2 \times N/2$, where each block is P_ℓ.
$\mathcal{P}$	$\mathcal{P}_{N/2}$. Length $N/2$ bit-reversal permutation matrix.
$\mathcal{T}_n$	Length n DFT matrix with bit-reversal permutation.
S_k	$\mathcal{T}_{N/2} = S_{N/4} \cdot S_{N/8} \cdots S_1$.
$S_{j \leftarrow i}$	$S_{j/2} S_{j/4} \cdots S_i$. $S_{N/2 \leftarrow 1} = \mathcal{T}_{N/2}$.
$\hat{\mathbf{x}}$	$DFT_\ell(\mathbf{x})$, where $\mathbf{x} \in \mathbb{C}^\ell$.
$\check{\mathbf{x}}$	$(\check{\mathbf{x}})_i := (\mathbf{x})_i + \sqrt{-1}(\mathbf{x})_{i+\ell}$, where $\mathbf{x} \in \mathbb{R}^{2\ell}$.
$\odot$	Element-wise multiplication.
$*_\ell$	Length ℓ (negacyclic) convolution.
$*_{\text{local}}$	$\mathbf{m} *_{\text{local}} \mathbf{m}' := (\mathbf{m}_0 *_\ell \mathbf{m}'_0 \| \cdots \| \mathbf{m}_{C-1} *_\ell \mathbf{m}'_{C-1})$.
$\langle \mathbf{m} \rangle_{type}$	Plaintext encoding $\mathbf{m}$, using $type$ encoding.
$[\mathbf{pt}]$	Ciphertext encrypting plaintext $\mathbf{pt}$.
s	Stride of a strided conv2d.
C_{in}	Number of input channels in conv2d.
C_{out}	Number of output channels in conv2d.