PDF Download
3719027.3765022.pdf
25 December 2025
Total Citations: 0
Total Downloads: 934

Published: 19 November 2025

Citation in BibTeX format

CCS '25: ACM SIGSAC Conference on Computer and Communications Security
October 13 - 17, 2025
Taipei, Taiwan

Conference Sponsors:
SIGSAC

RESEARCH-ARTICLE

WPC: Weight Plaintext Compression for CNN Inference based on RNS-CKKS

GUIMING SHI, Tsinghua University, Beijing, China

YUCHEN WEI, Tsinghua University, Beijing, China

SHENGYU FAN, University of Chinese Academy of Sciences, Beijing, China

XIANGLONG DENG, University of Chinese Academy of Sciences, Beijing, China

LIANG KONG, Ant group, Hangzhou, Zhejiang, China

XIANBIN LI, Hong Kong University of Science and Technology, Hong Kong, Hong Kong

View all

Open Access Support provided by:

Tsinghua University

Ant group

University of Chinese Academy of Sciences

Hong Kong University of Science and Technology

WPC: Weight Plaintext Compression for CNN Inference based on RNS-CKKS

Guiming Shi
Tsinghua University
Beijing, China
sgm24@mails.tsinghua.edu.cn

Yuchen Wei
Tsinghua University
Beijing, China
weiyc22@mails.tsinghua.edu.cn

Shengyu Fan
Chinese Academy of Sciences
Beijing, China
University of Chinese Academy of Sciences
Beijing, China
fanshengyu@iie.ac.cn

Xianglong Deng
Chinese Academy of Sciences
Beijing, China
University of Chinese Academy of Sciences
Beijing, China
dengxianglong@iie.ac.cn

Liang Kong
Ant Group
Beijing, China
kongliang.kong@antgroup.com

Xianbin Li
The Hong Kong University of Science and Technology
Hong Kong, China
xligt@connect.ust.hk

Jingwei Cai
Tsinghua University
Beijing, China
caijw21@mails.tsinghua.edu.cn

Shuwen Deng
Department of Electronic Engineering
Tsinghua University
Beijing, China
Zhongguancun Laboratory
Beijing, China
shuwend@tsinghua.edu.cn

Mingzhe Zhang*
Ant Group
Beijing, China
smartzmz@gmail.com

Kaisheng Ma*
Tsinghua University
Beijing, China
kaisheng@tsinghua.edu.cn

Abstract

Convolutional neural network (CNN) inference based on RNS-CKKS enables secure processing on encrypted data but introduces significant weight size overhead. Weight plaintext, weight in RNS-CKKS format, can reach tens to hundreds of gigabytes. Existing compression methods either add high computational cost or yield low compression rates. In this work, we propose WPC, Weight Plaintext Compression, to compress weight plaintext for RNS-CKKS-based CNN inference. We observe that the transformation from the weight in CNN models to the weight plaintext in RNS-CKKS format involves an operation akin to the Discrete Fourier Transform, which shifts data between the time and frequency domains while retaining redundant information from periodic and discrete data. Based on this observation, we first introduce the Periodic Transmit Theorem, which states that periodic patterns can be preserved during the transformation process, thereby enabling compression. We then propose Channel Innermost Packing Scheme and Rotation Padding to rearrange the weight data into periodic patterns for compression. Results show that WPC achieves 1.25 to 2.18 times speedup on an A100 GPU and 46.08 to 139.11 times compression rate.

*Corresponding Author

CCS Concepts

• **Security and privacy → Privacy-preserving protocols**.

Keywords

FHE; RNS-CKKS; CNN; Plaintext Compression; Periodic Transmit

ACM Reference Format:
Guiming Shi, Yuchen Wei, Shengyu Fan, Xianglong Deng, Liang Kong, Xianbin Li, Jingwei Cai, Shuwen Deng, Mingzhe Zhang, and Kaisheng Ma. 2025. WPC: Weight Plaintext Compression for CNN Inference based on RNS-CKKS. In *Proceedings of the 2025 ACM SIGSAC Conference on Computer and Communications Security (CCS '25), October 13–17, 2025, Taipei, Taiwan.* ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/3719027.3765022

1 Introduction

RNS-CKKS [13, 14]-based Convolutional Neural Network (CNN) inference [32] provides a promising approach for secure inference in various applications, including those in financial and healthcare domains [27, 45]. In this approach, the user's sensitive data is encrypted and sent to the cloud, where the CNN model is executed on the encrypted data. The encrypted output is then sent back to the user, who can decrypt it to obtain the final result. This process ensures the privacy of the user's sensitive data while maintaining the confidentiality of the cloud model.

However, the security provided by the RNS-CKKS scheme comes at a significant cost in terms of data expansion, particularly for the weight plaintext, which represents the weights in RNS-CKKS format. For example, the original ResNet-50 model [23] are only 0.11 GB, but the weight plaintext in RNS-CKKS format can reach up to 306 GB, leading to a storage size increase of over 1,000×. This surge in weight plaintext size poses severe storage and memory challenges, especially on hardware with limited memory capacity, such as the A100 GPGPU with 80 GB of memory.

Previous compression techniques attempt to address the surge in weight plaintext size but suffer from significant drawbacks, such as high computational overhead [24, 32] or lower compression rates [26]. For instance, MPCNN [2, 32] generates the weight plaintext on-the-fly, allowing the original CNN weights to be stored; however, this incurs a latency overhead of 2.07× for the ResNet-50 model. NeuJeans [24] employs a Coefficients-in-Slot (CinS) encoding scheme to reduce the size of weight plaintext and achieves the state-of-the-art performance for convolution layers. However, this encoding restricts computational graph optimizations, leading to higher computational overhead compared to MPCNN [32] when the cost of on-the-fly generation is ignored and graph optimizations are feasible. Additionally, HyPHEN [26] exploits weight plaintext reusability, but achieves limited compression rates, which still poses memory limitations for large models.

In this work, we propose WPC, a Weight Plaintext Compression method designed to compress the weight plaintext in RNS-CKKS-based CNN inference. *By analyzing the transformation process from CNN weights to weight plaintext, we observe that this process resembles a Discrete Fourier Transform (DFT)-like operation [47], which facilitates the exchange of periodic and discrete values between the time and frequency domains.* Building on this observation, we exploit the weight-sharing property [31] in CNN models, enabling the identification of periodic patterns prior to the transformation.

Building on this insight, we first propose and prove the *Periodic Transmit Theorem* in the RNS-CKKS scheme, which asserts that periodic patterns in the weight data can be transmitted into the weight plaintext, making it compressible. The transformation from weight data to weight plaintext can be viewed as an Inverse Discrete Fourier Transform (Slot Encoding) following a Negative Wrapped Number Theoretic Transform (NWNTT), both of which satisfy the properties of DFT. *This process converts periodic data into discrete data and then back to periodic data, allowing only a single copy of the final periodic data to be stored, thereby enabling compression, as illustrated in Fig. 1.*

Second, we introduce the *Channel Innermost Packing Scheme* (CIPS) and *Rotation Padding* techniques to rearrange the weight

$$\text{Periodic Data} \xrightarrow[\text{Slot Encoding}]{\text{DFT-like}} \text{Discrete Data} \xrightarrow[\text{NWNTT}]{\text{DFT-like}} \text{Periodic Data}$$

Figure 1: Periodic Transmit Theorem in RNS-CKKS.

data into a periodic pattern, leveraging the Periodic Transmit Theorem. The weight-sharing property [31] causes the weight data to exhibit repetition before transformation, as the same weight is shared by neurons in the output ciphertext across different heights and widths but within the same channel. CIPS arranges the neurons in the output ciphertext by placing the channel dimension as the innermost dimension, while height and width are placed in the outermost dimensions. This arrangement makes part of the data periodic, as the weights are shared across the outermost dimensions. Rotation Padding further ensures that other non-periodic data (caused by zero padding [6]) becomes periodic by padding adjacent neurons in the output ciphertext. Together, CIPS and Rotation Padding ensure that all data is periodic before transmission, enabling compression through the Periodic Transmit Theorem.

We evaluate WPC on the ResNet [23] and VGG [49] CNN models using the ImageNet, Tiny-ImageNet [18], and CIFAR [28] datasets, and assess its performance on both CPU and GPU. Results show that WPC reduces the weight plaintext size by a factor of 46.08 to 139.11, while maintaining or improving model performance by a factor of 1.25 to 2.18 on the A100 GPGPU with 80 GB of memory. By leveraging WPC, even the ResNet-200 model, which requires only 7.90 GB of memory compared to the baseline's 1098.99 GB, can be executed on the A100 GPGPU with 80 GB of memory.

To summarize, the main contributions of this work are as follows:

- We propose and prove the *Periodic Transmit Theorem* in the RNS-CKKS scheme, enabling compression of periodic data.
- We introduce the *Channel Innermost Packing Scheme* and *Rotation Padding* techniques to make weight data in the weight plaintext periodic and compressible, supported by the Periodic Transmit Theorem.
- We demonstrate the effectiveness of WPC through comprehensive evaluations on CNN models, achieving high compression ratios and significant performance improvements.

2 Background

This section introduces Fully Homomorphic Encryption (FHE), RNS-CKKS scheme, the CNN model with its RNS-CKKS-based CNN inference, and the threat model of this work. The notations and parameters of this work are shown in Tab. 1.

2.1 FHE and RNS-CKKS

Fully Homomorphic Encryption (FHE) [22] is a cryptographic technique that allows arbitrary computation on the ciphertext and the result is the same as its unencrypted counterpart computation. CKKS [14] is a widely used FHE scheme as it supports arithmetic fixed-point operations efficiently. This work uses RNS-CKKS [13], the Residue Number System version of CKKS for its high efficiency in arithmetic operations, which is widely used in the RNS-CKKS-based CNN inference [10, 32, 37, 42, 46].

Table 1: Notations and Parameters. c, h, w, c$_o$, c$_i$, k$_h$, k$_w$, i, o **denote the index of channel, height, width, output channel, input channel, kernel height, kernel width, input ciphertext, and output ciphertext, respectively, in CNN Parameter.**

Parameter	Description
N	Polynomial Ring Degree
Q	Coefficient Modulus
P	Key Switching Modulus
l	Ciphertext Level
Δ	Scale Factor
L_{boot}	Maximum Level after Bootstrapping
λ	Security Parameter
Data Types	**Description**
sk, **pk** and **evk**	Secret, Public and Evaluation Key
ct, **pt** and **m**	Ciphertext, Plaintext and Message
slot	Slot Encoding Element of **ct** and **pt**
Plaintext	**Description**
ptM: Message Representation	Message Data Before Encoding
ptC: Coefficient Representation	Plaintext Coefficient
ptE: Evaluation Representation	NTT Coefficient of the Plaintext
RNS-CKKS Operations	**Description**
HAdd($\mathbf{ct}, \mathbf{ct}'$)	Element-wise Addition
HMul($\mathbf{ct}, \mathbf{ct}'$)	Element-wise Multiplication
HRot($\mathbf{ct}, r$)	Cyclic Rotation
CNN Parameter	**Description**
$\mathrm{C_{in}}, \mathrm{H_{in}}, \mathrm{W_{in}}$	Input Channel, Height and Width
$\mathrm{C_{out}}, \mathrm{H_{out}}, \mathrm{W_{out}}$	Output Channel, Height and Width
$\mathrm{K_H}, \mathrm{K_W}, \mathrm{S_H}, \mathrm{S_W}$	Kernel, Stride Height and Width
$\mathrm{P_{HB}}, \mathrm{P_{HE}}, \mathrm{P_{WB}}, \mathrm{P_{WE}}$	Padding on the Beginning and End of the Height and Width Dimension
$x \in \mathbb{R}^{\mathrm{C_{in}} \times \mathrm{H_{in}} \times \mathrm{W_{in}}}$	Input Neuron Tensor
$w \in \mathbb{R}^{\mathrm{C_{out}} \times \mathrm{C_{in}} \times \mathrm{K_H} \times \mathrm{K_W}}$	Weight Tensor
$y \in \mathbb{R}^{\mathrm{C_{out}} \times \mathrm{H_{out}} \times \mathrm{W_{out}}}$	Output Neuron Tensor
$x[\mathrm{c,h,w}]$, $y[\mathrm{c,h,w}]$	Input and Output Tensor Indexing
$w[\mathrm{c_o, c_i, k_h, k_w}]$	Weight Tensor Indexing
ctx[i], **cty**[o]	Input, Output Ciphertext Indexing
n_{in}, n_{out}	Input, Output Ciphertext Number
ctr[i, c, k$_h$, k$_w$]	Rotation Ciphertext Indexing
ptw[o, i, c, k$_h$, k$_w$]	Weight Plaintext Indexing

2.1.1 *RNS-CKKS.*

The ciphertext $\mathbf{ct} \in \mathbb{R}_Q^2$ is a pair of polynomials with degree N and coefficient modulus Q, where $\mathbb{R}_Q = \mathbb{Z}_Q[X]/(X^N+1)$ is the cyclotomic polynomial ring. Q is a product of $l+1$ prime numbers, where l is the ciphertext level. P is the key switching [20] modulus to support the homomorphic operations. Decrypt the ciphertext $\langle \mathbf{ct}, \mathbf{sk} \rangle = \mathbf{pt} + \mathbf{e}$, where **ct**, **sk**, **pt**, **e**, and $\langle \cdot, \cdot \rangle$ are the ciphertext, secret key, plaintext, error term, and dot product operation, respectively. The plaintext **pt** is encoded from the message vector **m** and each element of the message vector in the ciphertext or plaintext is called a slot [14]. All the setting in this work satisfies the security parameter $\lambda \geq 128$ bits.

2.1.2 *Plaintext Representation and Conversion.*

The plaintext has three representations: message **ptM**, coefficient **ptC**, and evaluation **ptE**, where **ptE** denotes the default representation in RNS-CKKS for efficient homomorphic operations. The conversion of them are:

$$\mathbf{ptM} \overset{Slot\ Encoding}{\rightarrow} \mathbf{ptC} \overset{NWNTT}{\rightarrow} \mathbf{ptE} \tag{1}$$

Slot Encoding The message $\mathbf{ptM} = \mathbf{m} = (m[0], \ldots, m[\frac{N}{2} - 1])$ is encoded into a plaintext polynomial **ptC**, where the message is a length $N/2$ real or complex numbers vector. The conversion between message vector **ptM** and plaintext coefficient **ptC** is performed by computing the Inverse Discrete Fourier Transform (IDFT on $\{\omega^{-5^k}\}_{k=0}^{\frac{N}{2}-1}$, where $\omega^{2N} = 1$, $\frac{N}{2}$ is number of slots) [47] of **ptM**, multiplied with the scale factor Δ, rounding $\lfloor\ \rceil$, splitting the real and imaginary part and placed into the equal interval coefficient in the plaintext polynomial. Slot Encoding (Equation 2):

$$\left.\begin{aligned} \mathbf{M} &= \lfloor \Delta \cdot \mathbf{IDFT}(\mathbf{ptM}) \rceil \\ \mathbf{ptC_i}[j] &= \mathbf{REAL}(\mathbf{M})[j] \mod q_i \\ \mathbf{ptC_i}[j + \frac{N}{2}] &= \mathbf{IMAG}(\mathbf{M})[j] \mod q_i \end{aligned}\right| i \in [0\,..\,\ell], j \in [0\,..\,\frac{N}{2}), \tag{2}$$

where **REAL** and **IMAG** are the real and imaginary part of the complex number, q_i is the modular of plaintext at level i and $\mathbf{ptC_i}[j]$ is the j-th coefficient of plaintext at level i.

Negative Wrapped Number Theoretic Transform Number Theoretic Transform (NTT) with negative wrapped convolution (NWNTT) [38, 44] can speed up the polynomial multiplication in the cyclotomic polynomial ring $\mathbb{Z}_q[X]/(X^N+1)$ in RNS-CKKS, which converts the coefficient representation **ptC** of the plaintext to the evaluation representation **ptE**. NWNTT (Equation 3):

$$\mathbf{ptE_i} = \mathbf{NTT}(\mathbf{ptC_i}[j] \cdot \psi_i^j \mod q_i) \quad \Big| i \in [0\,..\,\ell], j \in [0\,..\,N), \tag{3}$$

where q_i is the modular at level i, ψ_i is the $2N$-th root of unity in $\mathbb{Z}_{q_i}$ and $\mathbf{pt_i}[j]$ is the j-th coefficient of plaintext at level i.

Periodic and Discrete Conversion NTT is the generalization of the DFT in the modular arithmetic domain [7], so these two operations (IDFT and NTT) keep the DFT property:

Lemma 1. Periodic and Discrete Conversion [47]: *For a length n data x with a period of T, the Discrete Fourier Transform of x has discrete values at integer multiples of n/T, and vice versa.*

This Lemma forms the basis for the periodicity of DFT, which plays a key role in the compression of weight plaintext in this work. This demonstrates that periodic or discrete data have the potential to be compressed in the conversion process.

We present the proof of Lemma 1 below. Assume ω is the n-th root of unity ($\omega^n = 1$) and m is an integer.

Lemma 1 (*Periodic* $\overset{DFT}{\rightarrow}$ *Discrete*): $X = \mathrm{DFT}(x)$ is nonzero only at indices $k = m\frac{n}{T}$ if data x is a length n vector with period T:

Proof. $X[k] = \sum_{i=0}^{n-1} x[i]\omega^{ki} = \sum_{j=0}^{\frac{n}{T}-1} \sum_{i=0}^{T-1} x[i]\omega^{k(jT+i)} = \sum_{i=0}^{T-1} x[i]\omega^{ki} \sum_{j=0}^{\frac{n}{T}-1} \omega^{kjT}$, where $x[i] = x[i \bmod T]$. As $1 - \omega^{kn} = 0$. If k is not an integer multiple of $\frac{n}{T}$. $1 - \omega^{kT} \neq 0$ and $X[k] = \sum_{i=0}^{T-1} x[i]\omega^{ki} \frac{1-\omega^{kn}}{1-\omega^{kT}} = 0$, where the final equality holds by applying the formula for the sum of a geometric series. Thus, X is nonzero only at indices $k = m\frac{n}{T}$. □

Lemma 1 (*Discrete* $\overset{DFT}{\rightarrow}$ *Periodic*): $X = \mathrm{DFT}(x)$ is periodic with period T if x is nonzero only at indices $k = m\frac{n}{T}$:

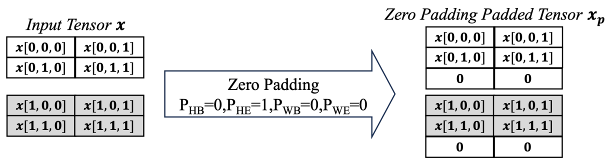

Figure 2: Zero Padding on Input Tensor $x \in \mathbb{R}^{2\times2\times2}$ to Generate the Padded Tensor $x_p \in \mathbb{R}^{C_{in}\times(P_{HB}+H_{in}+P_{HE})\times(P_{WB}+W_{in}+P_{WE})}$ with $P_{HB} = P_{WB} = P_{WE} = 0$ and $P_{HE} = 1$.

Proof. $X[k] = \sum_{i=0}^{n-1} x[i]\omega^{ki} = \sum_{j=0}^{T-1} x[j\frac{n}{T}]\omega^{k\cdot j\frac{n}{T}}$; For $k < n-T$, $X[k+T] = \sum_{j=0}^{T-1} x[j\frac{n}{T}]\omega^{(k+T)\cdot j\frac{n}{T}} = \sum_{j=0}^{T-1} x[j\frac{n}{T}]\omega^{k\cdot j\frac{n}{T}+jn} = X[k]$, where the last equality holds because $\omega^{jn} = 1$. Thus, X is periodic with period T. □

This lemma holds for all DFT-based operations, including the NTT and IDFT, as long as the condition $\omega^n = 1$ is satisfied.

2.1.3 Homomorphic Operations. The encoding of ciphertext before encryption influences the homomorphic operations on the ciphertext. Slot Encoding supports element-wise addition, multiplication, and cyclical rotation operations on each slot of the ciphertext. Coefficient Encoding encodes the message vector into the coefficient of the polynomial [25] and supports element-wise addition and convolution on these coefficients. Assume **m** and **m1** are the message representation of **ct** and **ct1** (**pt1**).

Homomorphic Operation of Slot Encoding

HAdd(**ct**, **ct1** *or* **pt1**) : $(m[0]+m1[0], \ldots, m[\frac{N}{2}-1]+m1[\frac{N}{2}-1])$;

HMul(**ct**, **ct1** *or* **pt1**) : $(m[0]\times m1[0], \ldots, m[\frac{N}{2}-1]\times m1[\frac{N}{2}-1])$;

HRot(**ct**, r) : $(m[r], \ldots, m[\frac{N}{2}-1], \ldots, m[r-1])$,

where r is the rotation step and $r > 0$ means left rotation. HAdd and HMul also support the operation between ciphertext and plaintext. HMul between ciphertexts and HRot with different rotation steps need different evaluation keys **evk**.

2.1.4 Bootstrapping and Management. The ciphertext of RNS-CKKS only supports a limited depth of homomorphic multiplication, which is determined by its level. When the ciphertext level l is reduced to 0, it cannot be used for further homomorphic multiplication. Bootstrapping [12, 34] is a time-consuming operation to recover the ciphertext to a higher level L_{boot} for further computation. The placement of bootstrapping within the computation graph is crucial for reducing RNS-CKKS overhead, which requires careful management, as demonstrated by Dacapo [16].

2.2 CNN Model and Weight-Sharing Property

Convolution Neural Network (CNN) [23, 29, 50] is a widely used deep learning model for image classification [18], object detection [9], and segmentation [39]. CNN consists of multiple layers, including the convolution layer [30], activation layer [41], pooling layer [3, 5], and fully connected layer [4], to extract the features of the input image. The most important and time-consuming layer in CNN is the convolution layer.

2.2.1 Convolution Layer. For an input tensor $x \in \mathbb{R}^{C_{in}\times H_{in}\times W_{in}}$, the convolution layer applies the weight filters $w \in \mathbb{R}^{C_{out}\times C_{in}\times K_H\times K_W}$

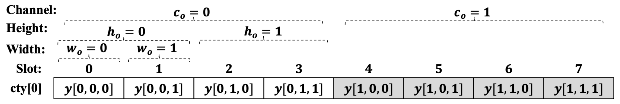

Figure 3: Packing the Output Tensor $y \in \mathbb{R}^{2\times2\times2}$ in the Output Ciphertext cty using the CHW Packing Scheme [17]. The neuron is packed from the ciphertext first slot to the last slot according to the width, height, and channel dimension of y.

to the input tensor to generate the output tensor $y \in \mathbb{R}^{C_{out}\times H_{out}\times W_{out}}$. The parameters of convolution layer are defined in Tab. 1.

Zero Padding The input tensor $x \in \mathbb{R}^{C_{in}\times H_{in}\times W_{in}}$ is usually padded with zeros [6] at the beginning and end of the height and width dimensions to generate the padded tensor $x_p \in \mathbb{R}^{C_{in}\times(P_{HB}+H_{in}+P_{HE})\times(P_{WB}+W_{in}+P_{WE})}$, as shown in Fig. 2.

Convolution Operation For each output tensor data $y[c_o, h_o, w_o]$, $c_o \in [0 \,..\, C_{out})$, $h_o \in [0 \,..\, H_{out})$, $w_o \in [0 \,..\, W_{out})$, the convolution operation is defined as:

$$y[c_o, h_o, w_o] = \sum_{c_i=0}^{C_{in}-1}\sum_{k_h=0}^{K_H-1}\sum_{k_w=0}^{K_W-1} x_p[c_i, h_oS_H + k_h, w_oS_W + k_w] \times w[c_o, c_i, k_h, k_w] + b[c_o], \tag{4}$$

where x_p is the padded input tensor and b is the bias. The total storage overhead of the weights in the convolution layer is: $C_{out}C_{in}K_HK_W \times 8$Byte, where C_{out}, C_{in}, K_H, K_W, and 8Byte represent the number of output channels, input channels, height, and width of the convolutional weight kernel, and the bit length of the float64 number, respectively.

Down-sample Layer A down-sample layer is a type of pooling or convolutional layer that reduces the height and width of the output tensor using a stride $S_H > 1$ or $S_W > 1$.

2.2.2 Weight-Sharing Property. The *weight-sharing property* of the convolution layer is that the same weight is applied to all the height and width of the output tensor with the same channel (e.g., $y[0, h_o, w_o]$, $h_o \in [0 \,..\, H_{out})$, $w_o \in [0 \,..\, W_{out})$), which can be leveraged to reduce the memory and storage overhead.

2.3 RNS-CKKS-based CNN and Weight Plaintext Expansion

The computation pattern of the RNS-CKKS-based CNN model differs from its unencrypted counterpart because of the vectorized homomorphic operations (HAdd, HMul, and HRot) on the ciphertext. Here, we focus on the convolutional layer of the RNS-CKKS-based CNN inference model. All data in Fig.3 and Fig.4 represent the encoded/encrypted weights and neurons indices in the plaintext and the ciphertext within the RNS-CKKS scheme.

2.3.1 CHW [17] Packing Scheme. Figure 3 shows an example of the CHW packing scheme [17] of the output tensor $y \in \mathbb{R}^{2\times2\times2}$. The output tensor $y \in \mathbb{R}^{2\times2\times2}$ is packed into the output ciphertext cty according to the sequence of width, height, and channel dimension of the tensor, where the channel is the outermost dimension and width is the innermost dimension. Figure 4 shows the RNS-CKKS computation of the convolutional layer using the CHW packing

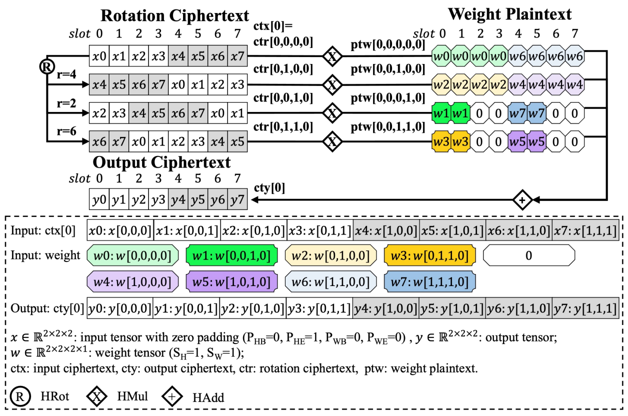
Rotation Ciphertext
Weight Plaintext
Output Ciphertext
ctx[0]=ctr[0,0,0,0]
ctr[0,1,0,0]
ctr[0,0,1,0]
ctr[0,1,1,0]
ptw[0,0,0,0,0]
ptw[0,0,1,0,0]
ptw[0,0,0,1,0]
ptw[0,0,1,1,0]
r=4
r=2
r=6
cty[0]
Input: ctx[0]
Input: weight
Output: cty[0]
$x \in \mathbb{R}^{2\times2\times2}$: input tensor with zero padding (P_{HB}=0, P_{HE}=1, P_{WB}=0, P_{WE}=0), $y \in \mathbb{R}^{2\times2\times2}$: output tensor;
$w \in \mathbb{R}^{2\times2\times2\times1}$: weight tensor ($S_H$=1, S_W=1);
ctx: input ciphertext, cty: output ciphertext, ctr: rotation ciphertext, ptw: weight plaintext.
HRot
HMul
HAdd

Figure 4: RNS-CKKS Computation of the Convolutional Layer using CHW Packing Scheme [17].

scheme [17]. The input ciphertext ctx[0] is rotated (HRot) to generate the rotation ciphertext ctr to match the input channel size (rotation step r=4) and the kernel size (r=2 and r=6) of the weight filter $w \in \mathbb{R}^{2\times2\times2\times1}$. Then these rotation ciphertexts are multiplied (HMul) with the weight plaintext ptw and summation (HAdd) into the output ciphertext cty[0]. The output ciphertext adds (HAdd) the bias plaintext to generate the final output ciphertext, which is ignored in Fig. 4. There are some packing optimizations to further enhance the performance and are integrated in the RNS-CKKS-based CNN inference model. Complex Packing [8] leverages the imaginary part of the weight plaintext to reduce the number of homomorphic operations. Reshaping Layer [48] reshapes the packing scheme after down-sample convolution into CHW to reduce the computation complexity and consumes 2 levels of the ciphertext.

2.3.2 Weight Expansion from Unencrypted CNN to RNS-CKKS-based CNN Model. The weight used in the RNS-CKKS-based CNN model differs from its unencrypted counterpart, requiring two steps to generate the weight plaintext, as shown in Fig. 5. First, the weight of the CNN model is packed into a weight vector, which serves as the message representation of the weight plaintext. In this step, due to the weight-sharing property, the weight is shared across the height and width of the output tensor, causing the weight to be repeated in the weight plaintext $H_{out}W_{out}$ times, where H_{out} and W_{out} represent the height and width of the output tensor. Next, the weight plaintext is converted into the evaluation representation using Slot Encoding and the NWNTT operation. *On one hand, after these steps, the repeated pattern in the weight plaintext is removed, resulting in an increase in storage overhead by a factor of $H_{out}W_{out}$, which accounts for the Weight Repetition expansion. On the other hand, the weight plaintext must match the level of the input ciphertext to perform the homomorphic operation, necessitating expansion to the same level as the input ciphertext. This causes bit expansion by a factor of $l + 1$, where l is the level of the weight plaintext.* Through this process, the weight expansion factor of the weight plaintext is:

$$\text{Expansion}_{\text{weight plaintext}} = \text{H}_{\text{out}}\text{W}_{\text{out}} \times (l + 1), \quad (5)$$

where H_{out}, W_{out}, and $(l + 1)$, represent the height and width of the output tensor, and the weight plaintext level, respectively.

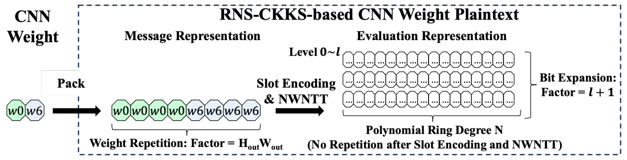
CNN Weight
RNS-CKKS-based CNN Weight Plaintext
Message Representation
Evaluation Representation
Pack
Slot Encoding & NWNTT
Level 0~l
Bit Expansion: Factor = l + 1
Polynomial Ring Degree N
(No Repetition after Slot Encoding and NWNTT)
Weight Repetition: Factor = $H_{out}W_{out}$

Figure 5: Weight Expansion in RNS-CKKS-based CNN.

2.4 Threat Model

RNS-CKKS-based CNN inference can protect the privacy of the user and the cloud during the inference process. The user encrypts the private data in the input ciphertext according to the first layer's input neurons' packing scheme and sends them to the cloud. The cloud performs the RNS-CKKS-based CNN inference on the input ciphertext layer by layer with the weight plaintext and sends the output ciphertext of the final layer back to the user. The user decrypts the output ciphertext to get the inference result according to the final layer's output neurons packing scheme.

This work assumes the semi-honest threat model, where the user and the cloud follow the protocol but try to infer the private data of others [10, 24, 27, 32, 37]. The user has private data and wants to get the inference result from the cloud. The cloud has the CNN model and provides the inference service to the user. The data of the user and the model of the cloud are secure and not leaked to the other party. The privacy of the user is protected by the RNS-CKKS scheme. The model of the cloud is secure as all the homomorphic operations are performed on the cloud [27, 32].

3 Motivation

The storage size surge caused by weight plaintext expansion in the RNS-CKKS scheme is a critical issue in RNS-CKKS-based CNN inference on hardware devices. This issue is particularly significant for large RNS-CKKS-based CNN models and motivates the exploration of Weight Plaintext Compression.

3.1 Drawback of Weight Plaintext Expansion

Slot Encoding supports element-wise operations (i.e., HAdd, HMul) and delivers efficient inference performance, but it suffers from a significant increase in weight size in convolutional layers due to the expansion of weight plaintext, as discussed in §2.3.2. Table 3 presents the weight size and latency for RNS-CKKS-based CNN inference on a CPU, highlighting the drawback of weight plaintext expansion. The size of the weight plaintext is 128 GB for ResNet-18 model [23]. This is significantly larger than the size of the weight data, which is only 0.04 GB. This expansion persists as the CNN model size increases (e.g., ResNet-50, ResNet-101, and larger models), demanding more memory and storage. This may cause out-of-memory issues on hardware with limited memory, such as the A100 GPGPU with 80 GB of memory.

Related Work and Limitations: Previous work employs several methods to reduce the weight size in the RNS-CKKS scheme but faces either increased computational overhead or lower compression rates, as shown in Tab. 2. These methods include Coefficients-in-Slot (CinS) Encoding (NeuJeans) [24], Slot Encoding-MPCNN [32]

Table 2: Comparison of the Overhead from Bootstrapping and the Storage Size of the Convolutional Layer. In this table, M represents the number of convolution-polynomial combinations in the CNN model. $L_{poly}(i)$ and $L_{conv}(i)$ denote the level consumption of the i-th polynomial and convolutional layer in convolution-polynomial combination, respectively. L_{boot} refers to the level that can be used for computing the convolutional or polynomial layer between two bootstrapping operations. C is the number of channels in the input tensor, and n_{out} is the number of output ciphertexts. H_{out} and W_{out} represent the number of height and width of the convolutional layer's output tensor that fits into ciphertext's polynomial ring degree N. K denotes convolutional layer's kernel size. l is ciphertext level, with each level having a size of $N \times 8B$ (UINT64). MPCNN and HyPHEN packing are optimized through advanced techniques such as complex packing [8], CHW [17, 32], and the Reshaping Layer [48]. These methods double the slot encoding packing length by packing tensors into their real and imaginary part and refining the overall packing scheme. CinS Encoding is converted to the Slot Encoding for multiplication polynomial layers, which necessitates bootstrapping for each convolution-polynomial combination. Slot Encoding method allows flexible management of the assignment of bootstrapping operations between the polynomial and convolutional layers. The storage size of CinS-NeuJeans is extended from the NeuJeans [24] to support convolutional layers with multi-ciphertext input and output.

Encoding-Packing Method	Bootstrapping Overhead		Convolutional Layer
	Lower Bound # of Bootstrapping Node	Free Bootstrapping Management	Storage Size (Byte)
CinS-NeuJeans [24]	$M + \sum_{i=0}^{M-1} \lceil \frac{L_{poly}(i) - L_{boot}}{L_{boot}} \rceil$	✗	$n_{out} C \times (l+1) \times 8N$
Slot-MPCNN [32]	$\lceil \frac{\sum_{i=0}^{M-1} (L_{poly}(i) + L_{conv}(i)) - L_{boot}}{L_{boot}} \rceil$	✓	$K^2 \times n_{out} C \times (l+1) \times 8N$
Slot-HyPHEN [26]			$\frac{1}{n_{out}} \times K^2 \times n_{out} C \times (l+1) \times 8N$
Slot-WPC (Ours)			$\frac{2}{H_{out} W_{out}} \times K^2 \times n_{out} C \times (l+1) \times 8N$

Table 3: Weight Size and Latency for RNS-CKKS-based CNN Inference with Slot Encoding-MPCNN [32] on CPU. The activation and dataset used are Hermite [42] and ImageNet [18]. Details about the CNN model, RNS-CKKS parameters, and CPU are provided in §7.1. The Boot, Conv+Poly, and Weight Plaintext Generation latencies represent the time taken for bootstrapping, the computation of convolutional and polynomial layers, and the generation of weight plaintext.

Model	Size (GB)		Latency (s)		
	CNN Weight	Weight Plaintext	Boot	Conv+ Poly	Weight Plaintext Generation
ResNet-18	**0.04**	**128**	**478.5**	**273.6**	**890.8**
ResNet-50	**0.11**	**306**	**1114.0**	**460.9**	**1678.0**

with on-the-fly weight plaintext generation, and Slot Encoding-HyPHEN [26] with weight plaintext reuse. NeuJeans exploits the property that encrypted multiplication is equivalent to the convolution operation on the encoded data, which is leveraged in convolutional layers to reduce the size of weight plaintext and achieve the state-of-the-art performance. As shown in Tab. 2 (Convolutional Layer Storage Size), CinS Encoding generally results in a lower storage size compared to Slot Encoding-MPCNN [32]. However, the bootstrapping overhead of CinS Encoding is higher due to the increased number of bootstrapping operations. CinS Encoding does not support element-wise multiplication. Therefore, each output ciphertext of the convolution layers must be bootstrapped and converted into Slot Encoding to perform element-wise polynomial activation operations. This conversion restricts optimizations in bootstrapping management [16], as shown in Tab. 2 (Bootstrapping Overhead). In contrast, Slot Encoding supports all element-wise operations and can compute convolution and polynomial activation

Table 4: Reasons for Weight Plaintext Expansion in RNS-CKKS-based Convolutional Layers. The term $\mathrm{H_{out}W_{out}}$ represents the number of neurons in one channel of the output tensor. l is the level of the weight plaintext. The typical values are based on the RNS-CKKS-based ResNet-18 [23] model applied to the ImageNet [18] dataset, using the RNS-CKKS parameters from MPCNN [32].

Reason	Expansion Factor	Typical Value
a) Weight Repetition	$\mathrm{H_{out}W_{out}}$	$8^2, 16^2, 32^2, 64^2$
b) Bit Expansion	$l+1$	2 to 17

layers within the same scheme. As a result, bootstrapping optimizations can be applied more freely, reducing computational overhead. However, generating the weight plaintext on-the-fly—by storing only the original CNN weights (MPCNN)—incurs significant latency overhead. As shown in Tab. 3, generating the weight plaintext on-the-fly increases latency by a factor of $\frac{478.5+273.6+890.8}{478.5+273.6} = 2.18\times$ and $\frac{1114.0+460.9+1678.0}{1114.0+460.9} = 2.07\times$ compared to bootstrapping, convolution, and polynomial layer computations, respectively. On the other hand, HyPHEN [26] leverages the weight plaintext reuse method across several ciphertexts in the same convolutional layer by splitting the height dimension. However, it achieves only a modest compression rate, since the number of ciphertexts is typically limited [8, 48]. This still results in out-of-memory issues for large models. Based on this analysis, we observe:

Observation 1. *Slot Encoding [8, 26, 32, 48] results in lower bootstrapping overhead compared to CinS Encoding [24] in RNS-CKKS-based CNN inference because of the freedom in bootstrapping management optimizations, but it requires a large storage and memory size for the weight plaintext.*

The weight plaintext expansion in RNS-CKKS-based convolution layers using Slot Encoding arises from two factors: Weight Repetition and Bit Expansion, as introduced in §2.3.2 and illustrated in Fig. 5. Among these, Weight Repetition is the primary cause of the surge in storage size in the Slot Encoding scheme, as shown in Tab. 4. The typical values for Weight Repetition in ResNet models on the ImageNet dataset [18] are $8^2, 16^2, 32^2, 64^2$, which are significantly larger than the Bit Expansion factor, which ranges from 2 to 17. Based on this analysis, we make the following observation:

Observation 2. *The weight repetition in the message representation of the weight plaintext in RNS-CKKS-based convolution layers primarily causes the increase in storage size in Slot Encoding [32, 48].*

3.2 Computation-Efficient Compression Method

According to Observation 1, Slot Encoding has the computational advantage of supporting element-wise multiplication operations, which are crucial for CNN inference. However, it suffers from a significant increase in weight size due to weight plaintext expansion. As stated in Observation 2, the primary cause of weight plaintext expansion is weight repetition. This expansion negatively affects the performance of RNS-CKKS-based CNN inference on hardware with limited memory capacity, leading to increased storage and memory requirements or higher inference latency due to online weight plaintext generation. Therefore, this work focuses on handling weight repetition and compressing the weight plaintext in RNS-CKKS-based convolution layers using Slot Encoding [32, 48] to reduce storage size and improve inference performance.

As introduced in §2.1.2, the data in the message representation of the weight plaintext is converted into the evaluation representation for homomorphic operations through the IDFT and NTT operations during the Slot Encoding and NWNTT processes. **These operations leverage the Periodic and Discrete Conversion Property of DFT [7, 47] (Lemma 1), which can be used for compression if the input data exhibits periodic patterns or discrete values at fixed intervals.** This enables the compression of the weight plaintext in RNS-CKKS-based convolution layers, as the current message representation contains redundant information due to weight repetition. However, the current message representation of the weight plaintext does not fully satisfy the compressed pattern of the Periodic and Discrete Conversion Property. Compressing the repeated weights in the message representation of the weight plaintext presents the following challenges:

- **Theoretical Proof of Compression Pattern:** How can we theoretically demonstrate which data patterns in the message representation can be compressed through the conversion process from the message representation to the evaluation representation in the RNS-CKKS Slot Encoding scheme? §5 details the solution.
- **Practical Implementation of Compression:** How can we implement the identified compression patterns in the message representation of the weight plaintext in RNS-CKKS-based CNN inference models to reduce the storage size and improve inference performance? This is solved in §6.

In the following sections, we introduce the Weight Plaintext Compression (WPC) to address these challenges to reduce the size of the weight plaintext in RNS-CKKS-based CNN inference.

4 Overview

WPC enables computational efficiency through flexible bootstrapping management and storage size reduction by leveraging the properties of DFT, as shown in Tab. 2 (Slot-WPC). The storage size of the weight plaintext for WPC is given by:

$$\begin{aligned}&\#\text{ of weight plaintext} \times \textit{Storage Size of each weight plaintext}\\ &= n_{out} 2K^2 C \times \frac{(l+1)\times 8N}{H_{out}W_{out}} = \frac{2}{H_{out}W_{out}} \times Storage_{\text{MPCNN [32]}}\end{aligned} \tag{6}$$

where # of weight plaintext and *Storage Size of each weight plaintext* are defined in Eq. 7 and Eq. 8, respectively. Compared to Slot Encoding with the state-of-the-art packing scheme [32], our method achieves a weight plaintext compression rate of $\frac{H_{out}W_{out}}{2}$, where H_{out} and W_{out} are the height and width of the output tensor in the output ciphertexts, respectively.

4.1 Optimization Flow of WPC

WPC focuses on compressing the weight plaintext in RNS-CKKS-based CNN inference to reduce memory footprint. As discussed in §3.2, the weight repetition in the message representation of the weight plaintext, combined with the Periodic and Discrete Conversion Property of the DFT [7, 47], enables the compression of the weight plaintext. First, we introduce and formally prove the Periodic Transmit Theorem in §5, which demonstrates that the periodicity in the message representation of the weight plaintext can be preserved and transmitted to the evaluation representation within the RNS-CKKS Slot Encoding scheme and NWNTT. This preserves the possibility for compression in the evaluation representation. Second, based on this theorem, we design the Channel Innermost Packing Scheme (CIPS) in §6, incorporating the optimization technique of Rotation Padding. The key idea is to pack original repeated weights in the weight plaintext into the periodic pattern by repacking the original inner dimension into the outer dimension of the weight plaintext. Through this repacking, the original repeated weights are packed into a constant interval and the final weight-plaintext has a periodic structure. Through the CIPS and Rotation Padding techniques, the evaluation representation of the weight plaintext can be compressed, leveraging the Periodic Transmit Theorem, addressing the expansion factor of weight repetition shown in Tab. 4 and Eq. 5 ($H_{out}W_{out}$), and achieving a significantly reduced storage size. This enables the storage of all weight plaintext in memory and eliminates the need for the computationally expensive online weight plaintext generation.

5 Theoretical Proof of Compression Pattern

As analyzed in §3.1, the expansion of weight repetition in the evaluation representation of the weight plaintext is the primary cause of the surge in storage size, which in turn increases the inference latency or memory footprint. In this section, we analyze the conversion process (Slot Encoding scheme and NWNTT operation, introduced in §2.1.2) between different representations in the RNS-CKKS [13, 14] plaintext and propose that **the periodic data pattern in the message representation of the weight plaintext can be transmitted to the evaluation representation**, enabling

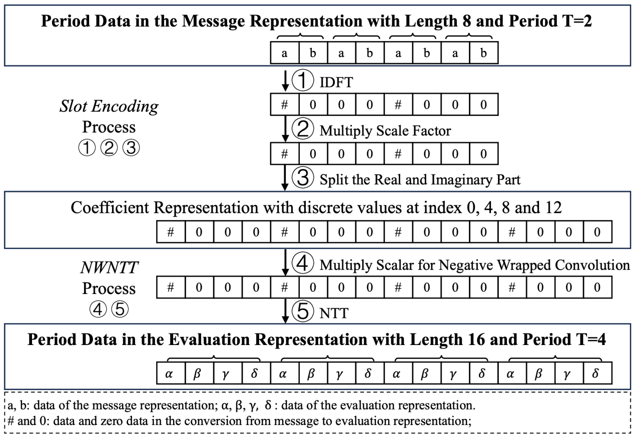
Period Data in the Message Representation with Length 8 and Period T=2
① IDFT
Slot Encoding Process ①②③
② Multiply Scale Factor
③ Split the Real and Imaginary Part
Coefficient Representation with discrete values at index 0, 4, 8 and 12
NWNTT Process ④⑤
④ Multiply Scalar for Negative Wrapped Convolution
⑤ NTT
Period Data in the Evaluation Representation with Length 16 and Period T=4
a, b: data of the message representation; α, β, γ, δ : data of the evaluation representation.
# and 0: data and zero data in the conversion from message to evaluation representation;

Figure 6: Slot Encoding and NWNTT Process of Periodic Data with a Length of 8 and a Period of 2 in the RNS-CKKS Scheme. When the message representation of the plaintext contains periodic data with a period of 2, the corresponding evaluation representation, after applying Slot Encoding and NWNTT, results in periodic data with a period of 4.

storage size reduction. This compression opportunity is a key finding of this work and supports the weight plaintext compression for the RNS-CKKS-based CNN inference discussed in §6.

5.1 Periodic Transmit

DFT-based Conversion Process in RNS-CKKS from Message to Evaluation Representation: The Slot Encoding process in the RNS-CKKS scheme transforms the data in the message representation into the coefficient representation using the IDFT operation (Eq. 2). The NTT operation (Eq. 3) then converts the coefficient representation into the evaluation representation, which is the final form of the plaintext used for homomorphic operations. Both the IDFT and NTT operations exhibit the periodic and discrete conversion property (Lemma 1). Periodic data in the message representation is transformed into discrete values in the coefficient representation, and then these discrete values are converted into periodic data in the evaluation representation, which can be compressed. By leveraging this lemma, we derive the following theorem:

Theorem 1. *Periodic Transmit: In the Slot Encoding and NWNTT process of RNS-CKKS, if the plaintext data in the message representation is periodic with length n and period T, then the corresponding plaintext data in the evaluation representation will retain periodicity with length $2n$ and period $2T$.*

We use a toy example in Fig. 6 to illustrate the Slot Encoding process, showing how periodic data is transferred from the message representation to the coefficient representation, and how the NWNTT process further transforms the data into the evaluation representation. This example helps validate the theorem.

Proof. Let the data be a sequence of length n with a period of T. The following steps outline how the periodic data is processed through the Slot Encoding and NWNTT operations: ① IDFT: Transforms the periodic data in the message representation into discrete values at integer multiples of n/T, as defined in Lemma 1. ② Multiply Scale Factor: Maintains the discrete values at integer multiples of n/T. ③ Split the Real and Imaginary Part: Further transforms the discrete values into the discrete values at integer multiples of $2n/2T$ in the coefficient representation. The length of the data is doubled. ④ Multiply scalar for Negative Wrapper Convolution: Maintains the discrete values at integer multiples of $2n/2T$. ⑤ NTT: Converts the discrete values into periodic data with a period of $\frac{2n}{2n/2T} = 2T$ in the evaluation representation, as described in Lemma 1. The length of the final result is $2n$, and the period is $2T$. □

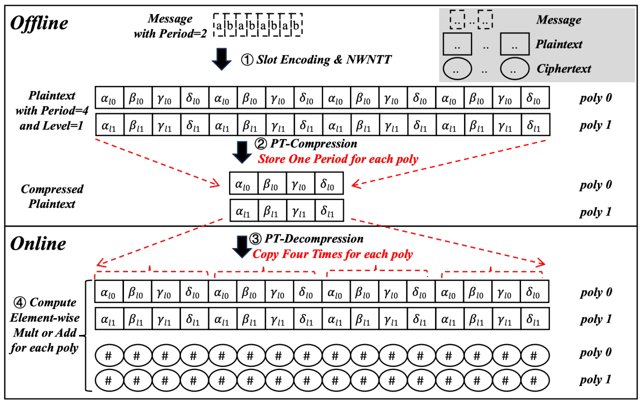

Figure 7: Offline Compression, Online Decompression, and Polynomial Operations on the Periodic Transmit Plaintext with Ring Degree $N = 16$ and Level $l = 1$.

Remark 1. *Benefit of Periodic Transmit: Theorem 1 shows that a periodic pattern exists in the evaluation representation of the RNS-CKKS plaintext if the data in the message representation is periodic. Thus, for each polynomial of the plaintext, the weight repetition expansion ratio of the data in the evaluation representation is reduced to a factor of $\frac{2T}{T}\times = 2\times$, instead of $\frac{2n}{T}\times$, assuming the data is purely periodic, where n is the slot number of the plaintext data in the message representation, $2n = N$ is the polynomial ring degree, and T is the period of the data in message representation. Specifically, if the message representation contains complex data, there is no expansion in the evaluation representation. This significant reduction in data expansion helps mitigate the storage size surge.*

5.2 Compression and Decompression

The plaintext of Periodic Transmit in the evaluation representation exhibits a periodic structure, which enables the compression process to store only a single period of the data. During decompression, the compressed plaintext is read and replicated $\frac{N}{2T}$ times to reconstruct its full evaluation representation, where N and $2T$ denotes the polynomial ring degree and period of the data in the evaluation representation.

Figure 7 illustrates an example of the compression and decompression process with a polynomial ring degree of $N = 16$ and level $l = 1$. The message (message representation) and plaintext polynomials (evaluation representation) exhibit periodicities of 2 and 4, respectively. This results in a compression rate of 4 for the plaintext.

The following describes the Periodic Transmit (PT) compression and decompression with polynomial operations:

PT-Compression: The message is first encoded and transformed into the evaluation representation using Slot Encoding and NWNTT, respectively. This step converts a period-2 structure in the message representation into a period-4 structure in the evaluation representation, as shown in Fig. 7 ①. The evaluation representation of the plaintext consists of $l + 1 = 2$ polynomials. Next, compression is applied by retaining only one period of data for each polynomial, as shown in Fig. 7 ②. The resulting compressed data occupies 4× less memory than the full evaluation representation.

PT-Decompression: In the online phase, the compressed plaintext is decompressed by copying the stored period four times for each polynomial, thereby reconstructing the complete evaluation representation, as illustrated in Fig. 7 ③.

Polynomial Operations: Subsequent polynomial operations, such as element-wise multiplication and addition, are executed directly on the decompressed evaluation representation in conjunction with the ciphertext polynomials, as depicted in Fig. 7 ④.

Remark 2. *Decompression Pattern and Polynomial Indexing: Different RNS-CKKS libraries may use different polynomial indexing schemes and lead to different decompression patterns. Figure 7 illustrates only the case of the standard (vanilla) indexing scheme, where coefficients are ordered as* $0, 1, 2, \ldots, N-1$. *For other indexing schemes, the decompression ratio remains the same, but the decompression pattern must be adjusted to account for differences in indexing.*

6 Practical Implementation of Compression

Theorem 1, Periodic Transmit, shows that plaintext with a periodic message representation can be compressed in its evaluation representation. However, this cannot be directly applied to the weight plaintext in the RNS-CKKS-based CNN inference model due to the non-periodic structure of its message representation. As shown in Fig. 4, the message representation of the weight plaintext in the CHW packing scheme [17] (introduced in §2.3) exhibits repetition in certain slot positions other than periodic data (e.g., Fig. 4 slot 0 to 3 of ptw[0,0,0,0,0] and ptw[0,0,1,0,0]).

In this section, we propose the *Channel Innermost Packing Scheme* (CIPS) and *Rotation Padding* to compress the weight plaintext in the RNS-CKKS-based CNN model. These techniques make the message representation of the weight plaintext periodic, enabling the use of the Periodic Transmit Theorem. CIPS generates periodic data in the message representation of the weight plaintext by exploiting the weight-sharing property (introduced in §2.2.2). In the CHW packing scheme [17], the height and width dimensions of the tensor are packed into the innermost dimension of the input and output ciphertexts. This arrangement leads to the weight repetition of the weight plaintext, as the same weight is shared across different neurons within the same channel, as illustrated in Fig. 4. To leverage both Periodic Transmit Theorem and the weight-sharing property, we propose the CIPS: *CIPS packs the channel dimension into the innermost dimension and the height and width dimensions into the outermost dimension in the message representation of the ciphertext.* Using CIPS, part of the weight plaintext consists of periodic data. We further propose the Rotation Padding to make the weight plaintext

Algorithm 1 Channel Innermost Packing Scheme (CIPS)
ct[i][j] denotes the j-th slot in the ciphertext ct[i].
Notations are defined in Tab. 1.

```
Input  Tensor $A \in \mathbb{R}^{C_A \times H_A \times W_A}$ and packing length n.
Output  Ciphertext ct with pack parameters H, W, C.
1: Set all slot in the ciphertext ct to zero.
2: $H = 2^{\lceil \log_2^{H_A} \rceil}$, $W = 2^{\lceil \log_2^{W_A} \rceil}$, $C = \frac{n}{HW}$
3: for i ∈ [0 .. $\lceil \frac{C_A}{C} \rceil$) do
4:     for j ∈ [0 .. n) do
5:         c = j  mod C
6:         w = $\lfloor \frac{j}{C} \rfloor$  mod W, h = $\lfloor \frac{j}{WC} \rfloor$  mod H
7:         if h < $H_A$ and w < $W_A$ and c + iC < $C_A$ then
8:             ct[i][j] = $A$[c + iC, h, w]
```

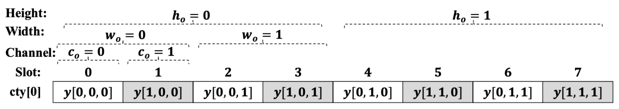

Figure 8: Packing the Output Tensor $y \in \mathbb{R}^{2\times2\times2}$ in the Output Ciphertext cty using the CIPS. The neuron is packed from the ciphertext first slot to the last slot according to $x's$ channel, width, and height dimension.

fully periodic in the message representation by *padding adjacent neurons in the input ciphertext to the input tensor.*

6.1 CIPS Packing and Computation Scheme

This subsection first defines the packing and computation scheme of CIPS, which determines how the weight data is encoded in the weight plaintext. All data in Fig.8 and Fig.9 represent the encoded/encrypted weights and neurons indices in the plaintext and the ciphertext within the RNS-CKKS scheme.

6.1.1 Pack Neuron Data in Ciphertext Slot using CIPS. Algorithm 1 shows the packing algorithm of the CIPS. For the tensor $A \in \mathbb{R}^{C_A \times H_A \times W_A}$ and the packing length n, Alg. 1 shows the case that one channel of the tensor ($H_A W_A$ neurons) can be packed into the n slot of the ciphertext for simplicity. *line 2*: First, the height and width dimensions are aligned to the power of 2 to calculate the pack parameters H and W. The number of channels in one ciphertext is $C = \frac{n}{HW}$ and the total number of ciphertexts is $\lceil \frac{C_A}{C} \rceil$. *line 3-8*: For each ciphertext, the neuron data is packed into the slot of ciphertext channel by channel (*line 8*), where the channel dimension is packed into the innermost dimension (*line 5*) and the height and width dimensions are packed into the outermost dimension (*line 6*). Figure 8 shows an example that CIPS packs the output tensor $y \in \mathbb{R}^{2\times2\times2}$ into the ciphertext with the packing length $n = 8$. The pack parameters are H = 2, W = 2, and C = 2. The channel dimension is packed into the innermost dimension of the ciphertext, so there are continues neurons in the ciphertext with the same height and width. For example, the cty[0] in Fig. 8 has the same height and width for each 2 slots (e.g., slot 0, 1 or slot 2, 3). For the case that one channel of the tensor can not be packed into one ciphertext (e.g., only the first layer of the ResNet model [23] on

Algorithm 2 CIPS RNS-CKKS-based Convolutional Layer
Notations are defined in Tab. 1 and Alg. 1.

```
Input  Input Ciphertexts ctx with pack parameters H, W, C
Input  Weight Plaintext ptw with kernel size $K_H$, $K_W$
Input  Padding Parameters $P_{HB}$, $P_{WB}$
Input  Stride Parameters $S_H$, $S_W$
Output  Output Ciphertexts cty
 1: Encrypt cty with all zero slots
 2: $r_{base} = P_{HB}WC + P_{WB}C$
 3: for i ∈ [0 .. #ctx) do
 4:     for $k_h$ ∈ [0 .. $K_H$); $k_w$ ∈ [0 .. $K_W$) do
 5:         for c ∈ (−C .. C) do
 6:             $r = k_hWC + k_wC + c - r_{base}$
 7:             $ctr[i, c, k_h, k_w] = \text{HRot}(ctx[i], r)$
 8: for o ∈ [0 .. #cty) do
 9:     for i ∈ [0 .. #ctx) do
10:         for $k_h$ ∈ [0 .. $K_H$); $k_w$ ∈ [0 .. $K_W$) do
11:             for c ∈ (−C .. C) do
12:                 $ctt = \text{HMul}(ctr[i, c, k_h, k_w], ptw[o, i, c, k_h, k_w])$
13:                 $cty[o] = \text{HAdd}(cty[o], ctt)$
```

the ImageNet [18]), CIPS splits the height and width dimensions into several parts in several ciphertexts. After the down-sample layer, the packing scheme changes [32], with the channel dimension inserted between the height and width dimensions. To restore the CIPS scheme, we apply the reshaping layer proposed by [48] to transform the output ciphertext of the down-sample layer. This reshaping layer consumes one level of the ciphertext.

6.1.2 Set RNS-CKKS Computation. Algorithm 2 shows the computation algorithm of the CIPS convolution layer in the RNS-CKKS scheme. The input are the input ciphertext ctx, the weight plaintext ptw, and parameters. *line 2-7*: the input ciphertext ctx are rotated for kernel size (*line 4*) and input channel (*line 5*) to generate the rotation ciphertext ctr (*line 7*). The padding operations may misalign the input and output ciphertexts, which affects the rotation step (*line 6*). The rotation base r_{base} (*line 2*) is calculated by the padding parameters P_{HB} and P_{WB} to handle the alignment of the input ciphertext and the output ciphertext. *line 8-13*: the output ciphertext cty are computed by multiplication these rotation ciphertext ctr with the weight plaintext ptw and summation. We can use the baby step giant step algorithm [12] to reduce the homomorphic rotation operations HRot. Specifically, the baby step giant step algorithm decomposes the rotation operation into two parts: the baby step and the giant step, with the giant step following the multiplication operation. Figure 9 shows CIPS RNS-CKKS-based convolution layer. The ciphertext ctx[0] and cty[0] are packed using the CIPS with the vector length $n = 8$. The input ciphertext ctx[0] is rotated to generate the rotation ciphertext ctr, which are multiplied with the weight plaintext ptw to generate the output ciphertext cty[0]. The total number of weight plaintext for the CIPS is:

$$\#\text{ of weight plaintext} \approx n_{out} \times 2 \times K^2 \times C, \tag{7}$$

where n_{out} is the number of output ciphertexts (cty), K is the kernel size (assumed $K_H = K_W$), and C is the number of input channels.

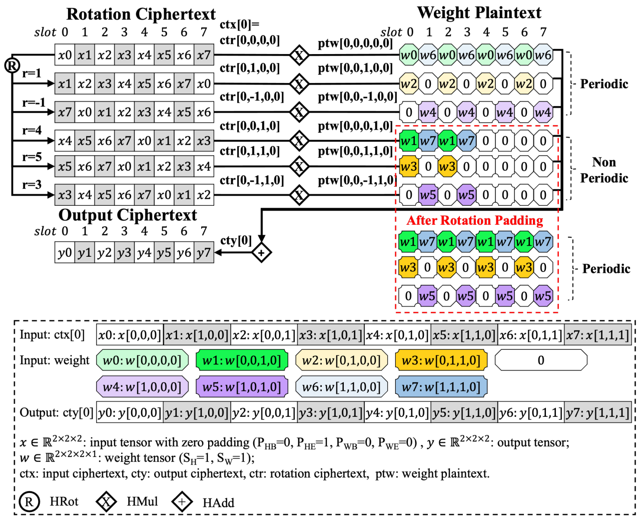

Figure 9: CIPS RNS-CKKS Computation of the Convolution Layer. Part of the message representation of the weight plaintext is periodic using zero padding. After applying Rotation Padding, the weight plaintext becomes fully periodic.

6.1.3 Make Periodic Message Representation Data of Weight Plaintext. Similar to unencrypted CNN model, the weight data in RNS-CKKS-based CNN model (weight plaintext) is independent of the input data. It is generated in the offline phase and used in the online inference phase. In the RNS-CKKS-based CNN model, the positions of neurons in the rotation ciphertext and output ciphertext determine the encoded data (Message Representation) in the weight plaintext. Figure 9 illustrates the Message Representation of the weight plaintext in the CIPS RNS-CKKS-based convolution layer, which can be inferred from the rotation ciphertext, output ciphertext, and the computation. For example, the slot 1 of the weight plaintext filter ptw[0,0,0,0,0] is multiplied with the slot 1 of the rotation ciphertext ctr[0,0,0,0] to generate the slot 1 of the output ciphertext cty[0]. The input and output neurons are $x1 = x[1, 0, 0]$ and $y1 = y[1, 0, 0]$. So the corresponding weight filter is $w6 = w[1, 1, 0, 0]$ according to the convolution Equation 4.

6.1.4 Leverage Periodic Transmit Theorem. Part of the weight plaintext before encoding is periodic data with a period of 2 in Fig. 9 (e.g., ptw[0,0,0,0,0], ptw[0,0,1,0,0] and ptw[0,0,-1,0,0]), which can be used to compress the plaintext in the RNS-CKKS scheme for the Theorem 1 Periodic Transmit. The weight plaintext is store in the evaluation representation by keeping only a single copy of the periodic data. Using the CIPS packing scheme, the storage size of one weight plaintext is:

$$\textit{Storage Size of each weight plaintext} = \frac{(l+1) \times 8N}{H_{out}W_{out}}, \tag{8}$$

where H_{out} and W_{out} are the number of height and width of the convolutional layer's output tensor that fits into ciphertext's polynomial ring degree N. l is the level of the weight plaintext, with each level having a storage size of $N \times 8B$ (UINT64). Each weight plaintext is compressed by $\frac{n}{T} = H_{out}W_{out}$ times, where n, T, and

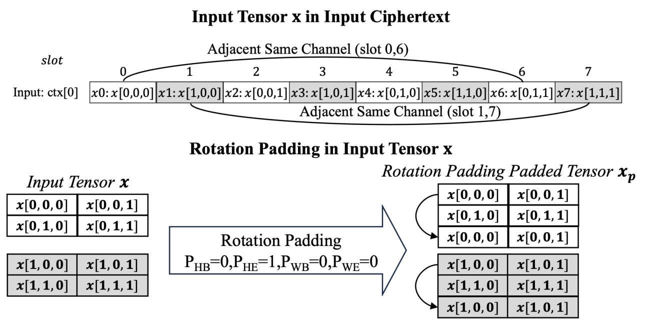

Figure 10: Rotation Padding on the Input $x \in \mathbb{R}^{2\times 2\times 2}$ to Generate the Padded Tensor $x_p \in \mathbb{R}^{\mathrm{C_{in}}\times(\mathrm{P_{HB}}+\mathrm{H_{in}}+\mathrm{P_{HE}})\times(\mathrm{P_{WB}}+\mathrm{W_{in}}+\mathrm{P_{WE}})}$ with $\mathrm{P_{HB}} = \mathrm{P_{WB}} = \mathrm{P_{WE}} = 0$ and $\mathrm{P_{HE}} = 1$.

$H_{out}W_{out}$ are the packing length, the period length, and the height and width of the output tensor, respectively.

6.2 Rotation Padding for Non-Periodic Plaintext

Part of the weight plaintext is periodic data in the message representation using the CIPS packing scheme. However, it is challenging to apply the Periodic Transmit Theorem in the RNS-CKKS-based CNN inference model to all the weight plaintext due to the non-periodic data before encoding, as shown in Fig. 9 ptw[0,0,0,1,0], ptw[0,0,1,1,0] and ptw[0,0,-1,1,0]. The zero value in slot 4-7 of these weight plaintext blocks the periodic data in the weight plaintext and makes the weight plaintext non-periodic. In this subsection, we propose the Rotation Padding to overcome this problem and achieve the periodic data in all the weight plaintext.

6.2.1 Root Cause of Non-Periodic Weight Plaintext. Part of the message representation of the weight plaintext consists of non-periodic data in the RNS-CKKS-based CNN model, which is caused by 1): The non-power-of-2 height and width dimensions of the input and output tensor; 2): Zero padding and the cyclical rotation (HRot) in the RNS-CKKS scheme. First, the non-power-of-2 dimensions of the input and output tensor cause a series of zero-weight data in the message representation of the weight plaintext for non-packed slots (which do not satisfy the condition of *line 7* in Alg. 1). Second, the cyclical rotation in the RNS-CKKS scheme shifts the positions of the zero-padded data with adjacent data in the input ciphertext, resulting in non-periodic data in the weight plaintext. As shown in Fig. 9, the weight plaintext values ptw[0,0,0,0,0], ptw[0,0,1,0,0], and ptw[0,0,-1,0,0] exhibit periodic pattern with a period of 2, while other values are non-periodic. These non-periodic values in the plaintext are caused by slots 4-7, which correspond to the output neurons $y_4 = y[0, 1, 0]$, $y_5 = y[1, 1, 0]$, $y_6 = y[0, 1, 1]$, and $y_7 = y[1, 1, 1]$ in the output ciphertext cty[0]. The required input neurons for these four output neurons are $\{x_4 = x[0, 1, 0], x_5 = x[1, 1, 0]\}$, $\{x_4, x_5\}$, $\{x_6 = x[0, 1, 1], x_7 = x[1, 1, 1]\}$, and $\{x_6, x_7\}$, with zero padding. If zero padding exists in slots 4-7 of the rotation ciphertext, the weight plaintext in slots 4-7 can be set to the same values as slots 0-3, thereby making the weight plaintext periodic. However, due to the cyclical rotation, these slots in the rotation ciphertext are packed with the first row of the input tensor x $\{x_0 = x[0, 0, 0]$, $x_1 = x[1, 0, 0]\}$, $\{x_0, x_1\}$, $\{x_2 = x[0, 0, 1], x_3 = x[1, 0, 1]\}$, $\{x_2, x_3\}$, resulting in the weight plaintext in slots 4-7 being zero, thus making the weight plaintext non-periodic.

6.2.2 Rotation Padding. We propose the Rotation Padding to make the weight plaintext periodic in the RNS-CKKS-based CNN model: *First, Rotation Padding expands the height and width dimensions of the input and output tensor to powers of 2. Second, the adjacent same-channel neurons in the input ciphertext are padded to the input tensor.* Figure 10 illustrates the second step of Rotation Padding. The adjacent neurons of $x6 = x[0, 1, 1]$ and $x7 = x[1, 1, 1]$ in the input ciphertext ctx[0] are $x0 = x[0, 0, 0]$ and $x1 = x[1, 0, 0]$, which are padded to the end of the input tensor x. The same analysis can be applied to the other neurons and the padded tensor x_p is shown in Fig. 10. The needed input neurons of the slot 4-5, 6-7 neurons in output ciphertext cty[0] in Fig. 9 ($y[:, 1, 0]$ and $y[:, 1, 1]$) are changed to $\{x4 = x[0, 1, 0], x5 = x[1, 1, 0], x0 = x[0, 0, 0], x1 = x[1, 0, 0]\}$ and $\{x6 = x[0, 1, 1], x7 = x[1, 1, 1], x2 = x[0, 0, 1], x3 = x[1, 0, 1]\}$ with Rotation Padding. The slot 4-7 of the weight plaintext can be set to the same value as the slot 0-3 to make the weight plaintext periodic data. As shown in Fig. 9, by applying the Rotation Padding to the input tensor, the weight plaintext are all periodic in the message representation and can be compressed in the evaluation representation using PT Compression in **§**5.2.

7 Evaluation

7.1 Experimental Setup

7.1.1 RNS-CKKS Parameters. The paper uses RNS-CKKS parameters with $N = 2^{16}$, $\log QP = 1531$, $\Delta = 2^{45}$ and $L_{boot} = 9$ for the implemented baselines, MPCNN and HyPHEN, as well as for WPC. The RNS-CKKS parameters of NeuJeans are based on the original paper [24]. These parameters satisfy a security level of $\lambda > 128$.

7.1.2 CNN Models, Datasets, and Training Configuration. We evaluate the performance of WPC on the ResNet [23] and VGG [49] models, using the ImageNet and Tiny-ImageNet datasets [18], as well as the CIFAR dataset [28]. The ResNet architectures—ResNet-50, ResNet-101, and ResNet-200—use the *BasicBlock* [23] in stages 1 through 4 with {3, 6, 12, 3}, {3, 12, 30, 3}, and {3, 24, 66, 3} layers, respectively. The VGG architecture increases the number of channels in the convolutional layers to fully utilize all slots in the RNS-CKKS scheme, ensuring that the product of each layer's CHW is a multiple of 32768. The ImageNet dataset consists of 1,200,000 training images, 50,000 validation images, and 1,000 classes. For the encrypted accuracy evaluation, we randomly select 1,000 validation images from the ImageNet validation dataset. We modify the CNN models by replacing the maximum pooling layer [5] with a convolutional layer. We replace the ReLU activation function with the Degree-2 Hermite polynomial activation function [42] or use the Minimax polynomial activation function [33] to approximate it (PolyReLU). The PolyReLU function consumes 14 levels of RNS-CKKS ciphertext. We train the models using the PyTorch framework [43] on an NVIDIA A100 GPGPU. The AdamW optimizer [36] is used with a learning rate of 4e-3 and a weight decay of 5e-2. A batch size of 128 is employed, and the model is trained for 200 epochs. The learning rate follows a cosine annealing schedule [35]. The fine-tuning is same as the training procedure, with the learning rate set to 1e-4 to

Table 5: Weight Plaintext Size and Inference Latency of Different Convolutional Layers in MPCNN and WPC. The RNS-CKKS-based convolutional layers are computed at ciphertext level 2. The hardware is the Intel 8480+ CPU. Cout, Cin, K, S, H and W represent the number of output channels, input channels, kernel size, stride, height and width of the input tensor, respectively.

Layer ID	Layer Parameters					Weight Plaintext Size (MB)			Latency (s)									
									MPCNN					WPC				SpeedUp
	Cout	Cin	K	S	H, W	MPCNN	WPC	Compression Rate	HRot	HMul	Other	Generate	Total	HRot	HMul	Other	Total	
0	64	3	3	2	256	**865.69**	**5.32**	**162.72×**	1.51	0.75	0.46	9.96	**12.68**	2.14	1.41	0.86	**4.41**	**2.88×**
1	64	64	3	1	64	**3456.70**	**2.62**	**1319.35×**	2.01	1.69	1.02	22.41	**27.13**	2.94	3.17	1.90	**8.01**	**3.39×**
2	128	64	3	2	64	**8801.56**	**293.96**	**29.94×**	2.96	3.38	2.03	44.82	**53.19**	4.66	8.87	5.33	**18.86**	**2.82×**
3	128	64	1	2	64	**2656.32**	**59.82**	**44.41×**	1.10	0.38	0.23	4.98	**6.69**	1.66	0.99	0.60	**3.25**	**2.06×**
4	128	128	3	1	32	**3456.70**	**6.05**	**571.36×**	1.99	1.69	1.01	22.41	**27.10**	3.43	4.91	2.95	**11.29**	**2.40×**
5	256	128	3	2	32	**7467.12**	**580.52**	**12.86×**	2.88	3.38	2.03	44.82	**53.11**	5.13	11.09	6.65	**22.87**	**2.32×**
6	256	128	1	2	32	**1321.88**	**103.33**	**12.79×**	1.01	0.38	0.23	4.98	**6.60**	1.76	1.23	0.74	**3.73**	**1.77×**
7	256	256	3	1	16	**3456.70**	**19.58**	**176.54×**	1.97	1.69	1.01	22.41	**27.08**	3.68	5.82	3.49	**12.99**	**2.08×**
8	512	256	3	2	16	**7057.29**	**1155.81**	**6.11×**	2.87	3.38	2.03	44.82	**53.10**	5.39	12.28	7.36	**25.03**	**2.12×**
9	512	256	1	2	16	**912.05**	**197.08**	**4.63×**	1.00	0.38	0.23	4.98	**6.59**	1.84	1.36	0.82	**4.02**	**1.64×**
10	512	512	3	1	8	**6913.40**	**145.68**	**47.46×**	3.03	3.38	2.03	44.82	**53.26**	5.91	12.58	7.54	**26.03**	**2.05×**

Table 6: Weight Plaintext Compression Comparison of WPC and Baseline Methods.

Model	VGG-11 Hermite			ResNet-34 Hermite			ResNet-101 Hermite			ResNet-200 Hermite		
Dataset	CIFAR-100			Tiny-ImageNet			ImageNet			ImageNet		
Method	MPCNN	HyPHEN	WPC	MPCNN	HyPHEN	WPC	MPCNN	HyPHEN	WPC	MPCNN	HyPHEN	WPC
Weight Plaintext Size (GB)	23.04	23.04	**0.50**	204.51	151.6	**3.83**	576.54	458.1	**5.53**	1098.99	900.2	**7.90**
Compression Rate	1.00×	1.00×	**46.08×**	1.00×	1.35×	**53.40×**	1.00×	1.26×	**104.26×**	1.00×	1.22×	**139.11×**

enhance accuracy. The ResNet architecture modification is necessary for training stability based on our experiments, as arithmetic replacement causes some of the original model architectures to diverge during training.

7.1.3 Inference Devices. We evaluate WPC on CPU and GPU devices. The CPU is an Intel Xeon 8480+ with 512 GB of memory, while the GPU is an NVIDIA A100 with 80 GB of memory. The RNS-CKKS library used on CPU and GPU are Lattigo [40] and [19, 21].

7.1.4 Baseline Implementation. We compare three baselines: Slot-MPCNN [32], Slot-HyPHEN [26], and CinS-NeuJeans [24]. All baseline CNN models use zero-padding. Slot-MPCNN and Slot-HyPHEN represent state-of-the-art Slot Encoding methods. Slot-HyPHEN additionally incorporates a plaintext reuse technique. Our implementation of Slot-MPCNN applies the CHW packing scheme [17], a Reshaping Layer with two-level consumption [48], and Complex Packing [8] to reduce the number of output ciphertexts and optimize packing efficiency. The original HyPHEN scheme [26] achieves a compression rate of 8 on ResNet-18. We improve HyPHEN's inference performance by independently applying the CHW packing scheme, the Reshaping Layer with two-level consumption, and Complex Packing. The implemented HyPHEN reduces the weight plaintext size by splitting convolutional layers into height-wise subtensor convolutions that share the same weight plaintext. These optimizations limit the number of output ciphertexts, resulting in a lower compression rate but improved inference efficiency. The consumed level of the Reshaping Layer in CHW is two, as the reshaping operation in the CHW packing scheme incurs higher costs for a one level reshaping. CinS-NeuJeans represents the state-of-the-art CinS Encoding [24] and its evaluation data is sourced from the original publication [24].

7.1.5 WPC Implementation. WPC employs Rotation Padding (§6.2) as its padding function. It adopts the CIPS packing scheme (§6.1), a Reshaping Layer with one-level consumption [48], and Complex Packing [8]. WPC incorporates the Periodic Transmit Compression method (§5.2) to compress the weight plaintext.

7.2 Results

7.2.1 Microbenchmark Result. Table 5 presents the weight plaintext size and inference latency of the convolutional layers in MPCNN and WPC. WPC achieves compression of the weight plaintext by a factor ranging from 4.63× to 1319.35× across different convolutional layers. The compression rate varies depending on the parameters of each convolutional layer, primarily the stride (S), and the height (H)&width (W) of the input tensors:

- For $S = 1$, $Compression_{Rate} \propto HW$.
- For $S = 2$, $Compression_{Rate} \propto H$.

The height and width of the tensors decrease with increasing layer ID because of the stride2-convolution, leading to a reduced compression rate in deeper layers of the CNN model. This reduction occurs because fewer repeated weights are encoded into the weight plaintext, as the weight-sharing property (§2.2.2) of convolutional layers spans the height and width dimensions. For instance, the compression rate for Layer ID 1 ($H, W = 64$) is 1319.35×, while for Layer ID 4 ($H, W = 32$), it decreases to 571.36×. The stride parameter also affects the compression rate. In strided convolution layers ($S = 2$), parts of the encoded data in the weight plaintext become zero, which increases the periodic length of the weight plaintext and reduces the compression rate. The zero data is padded between the height and width dimensions within the weight plaintext, resulting in larger periodic length. Thus, the compression rate for Layer ID 7 with $S = 1$ is larger than that for Layer ID 8 with $S = 2$ ($Compression_{Rate}$ = 176.54× vs. 6.11×).

Despite the higher computational overhead in the convolutional layers, WPC achieves lower inference latency than the Slot-MPCNN

Table 7: Inference Latency Comparison on A100 GPGPU and ImageNet Dataset. The Boot, Conv, Poly, and Weight Plaintext Generation times represent the latency of bootstrapping, convolutional, polynomial layers, and weight plaintext generation, respectively. For NeuJeans, the Poly, Conv, and Boot times correspond to the activation, Conv2d, and the sum of all other components as reported in [24]. MPCNN, HyPHEN, and WPC use the PolyReLU and RNS-CKKS library from [33] and [19, 21]. NeuJeans employs the PolyReLU and RNS-CKKS library from [15] and [1]. HyPHEN uses the first layer with a kernel size of 3 to fit the ResNet-18 on the A100 and encounters an out-of-memory (OoM) issue with larger models.

Inference Latency (s)	ResNet-18 Hermite				ResNet-18 PolyReLU				ResNet-50 PolyReLU			ResNet-200 Hermite	
	MPCNN	NeuJeans	HyPHEN	WPC	MPCNN	NeuJeans	HyPHEN	WPC	MPCNN	HyPHEN	WPC	MPCNN	WPC
Boot	1.77	5.02	1.77	**1.36**	7.84	6.46	7.84	**7.59**	18.39		**18.14**	13.38	**13.33**
Conv	1.83	0.25	2.17	**2.29**	1.51	0.21	2.23	**2.15**	2.62		**5.08**	8.48	**20.39**
Poly	0.06	0.08	0.06	**0.03**	0.51	7.44	0.51	**0.49**	1.26	OoM	**1.25**	0.52	**0.41**
Weight Plaintext Generation	4.35	-	-	-	4.34	-	-	-	8.26		-	24.95	-
Total	8.01	5.35	4.00	**3.68**	14.20	14.11	10.58	**10.23**	30.53		**24.47**	47.33	**34.04**
Speedup	1.00×	-	2.00×	**2.18×**	1.00×	-	1.34×	**1.39×**	1.00×	OoM	**1.25×**	1.00×	**1.39×**

Table 8: ImageNet Classification Top-1 Accuracy. The plain accuracy with full validation set and 1,000 randomly selected validation images is evaluated on the NVIDIA A100 GPU using the PyTorch framework. The encrypted accuracy with the same 1,000 randomly selected validation images is evaluated on an Intel 8480+ CPU and the Lattigo library [40].

CNN Model	Polynomial Activation	Full Val Set		1,000 Samples Images			
		Plain Accuracy (%)		Plain Accuracy (%)		Encrypt Accuracy (%)	
		MPCNN	WPC	MPCNN	WPC	MPCNN	WPC
ResNet-18	Hermite	69.36	70.38	68.50	70.60	68.50	70.60
	PolyReLU	71.42	71.13	71.60	71.30	71.60	71.20
ResNet-50	Hermite	71.19	71.65	71.70	71.80	71.60	71.70
	PolyReLU	76.31	77.83	76.50	78.70	76.50	78.70
ResNet-101	Hermite	71.94	71.89	72.20	71.90	72.20	71.90
ResNet-200	Hermite	74.45	74.57	74.50	74.40	74.50	74.40

baseline because of the elimination of on-the-fly weight plaintext generation. On one hand, the high computational cost of generating weight plaintext during inference significantly increases the latency in Slot-MPCNN, whereas WPC avoids this overhead. On the other hand, if the baseline MPCNN precomputes the weight plaintext to reduce latency, it incurs substantial storage and memory overhead to store the full plaintext representation.

7.2.2 Weight Plaintext Compression Comparison. Table 6 shows the weight plaintext size and compression rate of WPC compared to the baseline methods. The baseline MPCNN methods require between 23.04 GB and 1098.99 GB of weight plaintext, which is significantly larger than the A100 GPU memory (80 GB) and even exceeds the CPU memory (512 GB). The HyPHEN method reduces the weight plaintext size by up to 1.35× compared to the baseline MPCNN method. The lower compression rate arises because the implemented HyPHEN shares the weight plaintext across output ciphertexts within the same convolutional layer, whereas the number of ciphertexts is constrained by CHW [17, 32], Complex Packing [8], and Reshaping Layer [48] optimizations. For example, all the convolutional layers in VGG-11 use a single ciphertext, meaning the weight plaintext size is not reduced. A significant portion of the ResNet-101 and ResNet-200 models also includes layers with only one ciphertext, so the weight plaintext size is not reduced in these cases. The WPC method achieves a reduction in weight plaintext size by a factor of 46.08× to 139.11× compared to the baseline MPCNN method. For larger models, the down-sample layers occupy a smaller portion of the model, which leads to a lower compression rate, as discussed in detail in §7.2.1. Thus, for larger models, the compression rate increases, demonstrating the effectiveness of the weight plaintext compression of WPC.

7.2.3 CNN Inference Latency Comparison. Table 7 shows the inference latency of WPC and the baseline methods on the A100 GPGPU. The Slot Encoding Scheme-MPCNN exhibits lower performance compared to others, primarily because of the cost associated with weight plaintext generation. The CinS Encoding scheme of NeuJeans incurs higher computational overhead due to bootstrapping operation (e.g., ResNet-18 Hermite). This is because CinS Encoding does not support element-wise multiplication, necessitating bootstrapping to convert the output ciphertext of the convolutional layers into Slot Encoding for polynomial activation operations. HyPHEN incurs higher convolutional overhead due to support for weight plaintext reuse. Additionally, HyPHEN encounters an OoM issue on the ResNet-50 and ResNet-200 models because the limited weight plaintext compression rate. WPC benefits from the computational advantages of the Slot Encoding and significantly compresses the weight plaintext (as detailed in §7.2.2). Thus, WPC achieves the best performance across all models, despite the increased convolutional layer overhead.

7.2.4 Image Classification Top-1 Accuracy. Table 8 shows the ImageNet classification top-1 accuracy for both WPC and the Slot-MPCNN baseline. The plain accuracy on the full ImageNet validation dataset of both the baseline and WPC are similar, indicating that the modification to the CNN architecture (Rotation Padding) does not affect the CNN's accuracy. For the 1,000 randomly selected validation images, part of the encrypted accuracy is lower than the plain accuracy due to the inference error introduced by the RNS-CKKS scheme. Specifically, the plain accuracy of Slot-MPCNN ResNet-50 with Hermite, WPC ResNet-18 with PolyReLU, and ResNet-50 with Hermite is lower than the corresponding plain accuracy because of the inference error caused by the RNS-CKKS.

8 Discussion

This section presents key insights from our findings, outlines the limitations of our approach, and suggests directions for future work.

Insights: This paper introduces a new perspective on plaintext compression that targets the storage size of individual plaintext in RNS-CKKS. Unlike prior works that reduce the number of plaintext, WPC leverages the formalized periodicity arising from DFT-like Slot Encoding and NWNTT to minimize the size of each plaintext.

Limitations and Future Work: First, this work focuses on RNS-CKKS-based CNN inference, leaving other types of applications unexplored (e.g., Transformers [51]). However, as demonstrated in this paper, the compression opportunity is not exclusive to CNNs. The Periodic Transmit Theorem (Theorem 1) is a general characteristic of RNS-CKKS. Therefore, it is potentially applicable to other neural network architectures. Challenges in this direction include exploring the periodicity of weight plaintext and designing efficient packing schemes tailored to different architectures. Second, the proposed CIPS packing scheme reduces the size of weight plaintext at the cost of increased computational complexity in convolutional layers. Developing more efficient packing strategies may help mitigate this overhead while preserving the compression benefits.

Extensions to Other FHE Schemes: The Periodic Transmit Theorem also applies to other FHE schemes, such as BGV [11], although the relationship between the period and the message length may vary. These schemes typically employ a similar conversion process that maps the message representation to the evaluation representation of the plaintext.

9 Conclusion

In this work, we have proposed WPC, an efficient weight plaintext compression method for RNS-CKKS-based convolutional neural network inference. By leveraging insights from the transformation of CNN weights into weight plaintext and exploiting the periodic nature of the data, we have introduced a compression approach that minimizes storage and memory overhead while preserving the accuracy and security of the inference process. We have presented the Periodic Transmit Theorem in the RNS-CKKS scheme, which enables the compression of periodic data, and introduced the Channel Innermost Packing Scheme and Rotation Padding techniques to rearrange weight data into a format that can be efficiently compressed. Our experimental evaluations demonstrate that WPC significantly reduces weight plaintext size, achieving compression ratios of dozens to hundreds, while maintaining or even improving model performance on standard datasets such as ImageNet, Tiny-ImageNet, and CIFAR. In summary, WPC provides an effective solution to the challenges of large memory consumption in encrypted inference, offering both compression efficiency and computational practicality.

Acknowledgments

This work was supported by the Tsinghua University Dushi Program (20251080047). We thank Guang Fan and Yi Chen for providing the GPGPUs RNS-CKKS library.

References

[1] 2022. HEaaN Library. https://heaan.it/.
[2] 2022. MPCNN. https://github.com/snu-ccl/FHE-MP-CNN.
[3] 2023. AvgPool2d. https://pytorch.org/docs/stable/generated/torch.nn.AvgPool2d.html.
[4] 2023. Fully Connected Layer. https://pytorch.org/docs/stable/generated/torch.nn.Linear.html.
[5] 2023. MaxPool2d. https://pytorch.org/docs/stable/generated/torch.nn.MaxPool2d.html.
[6] 2023. ZeroPad2d. https://pytorch.org/docs/stable/generated/torch.nn.ZeroPad2d.html#torch.nn.ZeroPad2d.
[7] Ramesh C Agarwal and C Sidney Burrus. 1975. Number theoretic transforms to implement fast digital convolution. *Proc. IEEE* 63, 4 (1975), 550–560.
[8] Ehud Aharoni, Nir Drucker, Gilad Ezov, Hayim Shaul, and Omri Soceanu. 2022. Complex Encoded Tile Tensors: Accelerating Encrypted Analytics. *IEEE Secur. Priv.* 20, 5 (2022), 35–43. doi:10.1109/MSEC.2022.3181689
[9] Ayoub Benali Amjoud and Mustapha Amrouch. 2023. Object Detection Using Deep Learning, CNNs and Vision Transformers: A Review. *IEEE Access* 11 (2023), 35479–35516. doi:10.1109/ACCESS.2023.3266093
[10] Wei Ao and Vishnu Naresh Boddeti. 2024. AutoFHE: Automated Adaption of CNNs for Efficient Evaluation over FHE. In *33rd USENIX Security Symposium, USENIX Security 2024, Philadelphia, PA, USA, August 14-16, 2024*, Davide Balzarotti and Wenyuan Xu (Eds.). USENIX Association. https://www.usenix.org/conference/usenixsecurity24/presentation/ao
[11] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. 2011. Fully Homomorphic Encryption without Bootstrapping. *Electron. Colloquium Comput. Complex.* TR11-111 (2011). ECCC:TR11-111 https://eccc.weizmann.ac.il/report/2011/111
[12] Jung Hee Cheon, Kyoohyung Han, Andrey Kim, Miran Kim, and Yongsoo Song. 2018. Bootstrapping for Approximate Homomorphic Encryption. In *Advances in Cryptology - EUROCRYPT 2018 - 37th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Tel Aviv, Israel, April 29 - May 3, 2018 Proceedings, Part I (Lecture Notes in Computer Science, Vol. 10820)*, Jesper Buus Nielsen and Vincent Rijmen (Eds.). Springer, 360–384. doi:10.1007/978-3-319-78381-9_14
[13] 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 2018 - 25th International Conference, Calgary, AB, Canada, August 15-17, 2018, Revised Selected Papers (Lecture Notes in Computer Science, Vol. 11349)*, Carlos Cid and Michael J. Jacobson Jr. (Eds.). Springer, 347–368. doi:10.1007/978-3-030-10970-7_16
[14] Jung Hee Cheon, Andrey Kim, Miran Kim, and Yong Soo Song. 2017. Homomorphic Encryption for Arithmetic of Approximate Numbers. In *Advances in Cryptology - ASIACRYPT 2017 - 23rd International Conference on the Theory and Applications of Cryptology and Information Security, Hong Kong, China, December 3-7, 2017, Proceedings, Part I (Lecture Notes in Computer Science, Vol. 10624)*, Tsuyoshi Takagi and Thomas Peyrin (Eds.). Springer, 409–437. doi:10.1007/978-3-319-70694-8_15
[15] Jung Hee Cheon, Dongwoo Kim, and Duhyeong Kim. 2020. Efficient Homomorphic Comparison Methods with Optimal Complexity. In *Advances in Cryptology - ASIACRYPT 2020 - 26th International Conference on the Theory and Application of Cryptology and Information Security, Daejeon, South Korea, December 7-11, 2020, Proceedings, Part II (Lecture Notes in Computer Science, Vol. 12492)*, Shiho Moriai and Huaxiong Wang (Eds.). Springer, 221–256. doi:10.1007/978-3-030-64834-3_8
[16] Seonyoung Cheon, Yongwoo Lee, Dongkwan Kim, Ju Min Lee, Sunchul Jung, Taekyung Kim, Dongyoon Lee, and Hanjun Kim. 2024. DaCapo: Automatic Bootstrapping Management for Efficient Fully Homomorphic Encryption. In *33rd USENIX Security Symposium, USENIX Security 2024, Philadelphia, PA, USA, August 14-16, 2024*, Davide Balzarotti and Wenyuan Xu (Eds.). USENIX Association. https://www.usenix.org/conference/usenixsecurity24/presentation/cheon
[17] Roshan Dathathri, Olli Saarikivi, Hao Chen, Kim Laine, Kristin E. Lauter, Saeed Maleki, Madanlal Musuvathi, and Todd Mytkowicz. 2019. CHET: an optimizing compiler for fully-homomorphic neural-network inferencing. In *Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI 2019, Phoenix, AZ, USA, June 22-26, 2019*, Kathryn S. McKinley and Kathleen Fisher (Eds.). ACM, 142–156. doi:10.1145/3314221.3314628
[18] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. 2009. ImageNet: A large-scale hierarchical image database. In *2009 IEEE Computer Society Conference on Computer Vision and Pattern Recognition CVPR 2009, 20-25 June 2009, Miami, Florida, USA*. IEEE Computer Society, 248–255. doi:10.1109/CVPR.2009.5206848
[19] Guang Fan, Mingzhe Zhang, Fangyu Zheng, Shengyu Fan, Tian Zhou, Xianglong Deng, Wenxu Tang, Liang Kong, Yixuan Song, and Shoumeng Yan. 2025. WarpDrive: GPU-Based Fully Homomorphic Encryption Acceleration Leveraging Tensor and CUDA Cores. In *IEEE International Symposium on High Performance Computer Architecture, HPCA 2025, Las Vegas, NV, USA, March 1-5, 2025*. IEEE, 1187–1200. doi:10.1109/HPCA61900.2025.00091
[20] Junfeng Fan and Frederik Vercauteren. 2012. Somewhat Practical Fully Homomorphic Encryption. *IACR Cryptol. ePrint Arch.* (2012), 144. http://eprint.iacr.org/2012/144
[21] Shengyu Fan, Zhiwei Wang, Weizhi Xu, Rui Hou, Dan Meng, and Mingzhe Zhang. 2023. TensorFHE: Achieving Practical Computation on Encrypted Data Using GPGPU. In *IEEE International Symposium on High-Performance Computer*

Architecture, HPCA 2023, Montreal, QC, Canada, February 25 - March 1, 2023. IEEE, 922–934. doi:10.1109/HPCA56546.2023.10071017

[22] Craig Gentry. 2009. Fully homomorphic encryption using ideal lattices. In *Proceedings of the 41st Annual ACM Symposium on Theory of Computing, STOC 2009, Bethesda, MD, USA, May 31 - June 2, 2009*, Michael Mitzenmacher (Ed.). ACM, 169–178. doi:10.1145/1536414.1536440

[23] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep Residual Learning for Image Recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016.* IEEE Computer Society, 770–778. doi:10.1109/CVPR.2016.90

[24] 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 on ACM SIGSAC Conference on Computer and Communications Security, CCS 2024, Salt Lake City, UT, USA, October 14-18, 2024*, Bo Luo, Xiaojing Liao, Jun Xu, Engin Kirda, and David Lie (Eds.). ACM, 4361–4375. doi:10.1145/3658644.3690375

[25] Dongwoo Kim and Cyril Guyot. 2023. Optimized Privacy-Preserving CNN Inference With Fully Homomorphic Encryption. *IEEE Trans. Inf. Forensics Secur.* 18 (2023), 2175–2187. doi:10.1109/TIFS.2023.3263631

[26] Donghwan Kim, Jaiyoung Park, Jongmin Kim, Sangpyo Kim, and Jung Ho Ahn. 2024. HyPHEN: A Hybrid Packing Method and Its Optimizations for Homomorphic Encryption-Based Neural Networks. *IEEE Access* 12 (2024), 3024–3038. doi:10.1109/ACCESS.2023.3348170

[27] Miran Kim, Xiaoqian Jiang, Kristin Lauter, Elkhan Ismayilzada, and Shayan Shams. 2022. Secure human action recognition by encrypted neural network inference. *Nature communications* 13, 1 (2022), 4799.

[28] Alex Krizhevsky. 2009. Learning Multiple Layers of Features from Tiny Images. (2009).

[29] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. 2012. ImageNet Classification with Deep Convolutional Neural Networks. In *Advances in Neural Information Processing Systems 25: 26th Annual Conference on Neural Information Processing Systems 2012. Proceedings of a meeting held December 3-6, 2012, Lake Tahoe, Nevada, United States*, Peter L. Bartlett, Fernando C. N. Pereira, Christopher J. C. Burges, Léon Bottou, and Kilian Q. Weinberger (Eds.). 1106–1114. https://proceedings.neurips.cc/paper/2012/hash/c399862d3b9d6b76c8436e924a68c45b-Abstract.html

[30] Yann LeCun, Bernhard E. Boser, John S. Denker, Donnie Henderson, Richard E. Howard, Wayne E. Hubbard, and Lawrence D. Jackel. 1989. Handwritten Digit Recognition with a Back-Propagation Network. In *Advances in Neural Information Processing Systems 2, [NIPS Conference, Denver, Colorado, USA, November 27-30, 1989]*, David S. Touretzky (Ed.). Morgan Kaufmann, 396–404. http://papers.nips.cc/paper/293-handwritten-digit-recognition-with-a-back-propagation-network

[31] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. 1998. Gradient-based learning applied to document recognition. *Proc. IEEE* 86, 11 (1998), 2278–2324. doi:10.1109/5.726791

[32] 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 2022, 17-23 July 2022, Baltimore, Maryland, USA (Proceedings of Machine Learning Research, Vol. 162)*, Kamalika Chaudhuri, Stefanie Jegelka, Le Song, Csaba Szepesvári, Gang Niu, and Sivan Sabato (Eds.). PMLR, 12403–12422. https://proceedings.mlr.press/v162/lee22e.html

[33] 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 Trans. Dependable Secur. Comput.* 19, 6 (2022), 3711–3727. doi:10.1109/TDSC.2021.3105111

[34] Joon-Woo Lee, Eunsang Lee, Yongwoo Lee, Young-Sik Kim, and Jong-Seon No. 2021. High-Precision Bootstrapping of RNS-CKKS Homomorphic Encryption Using Optimal Minimax Polynomial Approximation and Inverse Sine Function. In *Advances in Cryptology - EUROCRYPT 2021 - 40th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zagreb, Croatia, October 17-21, 2021, Proceedings, Part I (Lecture Notes in Computer Science, Vol. 12696)*, Anne Canteaut and François-Xavier Standaert (Eds.). Springer, 618–647. doi:10.1007/978-3-030-77870-5_22

[35] Ilya Loshchilov and Frank Hutter. 2017. SGDR: Stochastic Gradient Descent with Warm Restarts. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings.* OpenReview.net. https://openreview.net/forum?id=Skq89Scxx

[36] Ilya Loshchilov and Frank Hutter. 2019. Decoupled Weight Decay Regularization. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019.* OpenReview.net. https://openreview.net/forum?id=Bkg6RiCqY7

[37] Qian Lou and Lei Jiang. 2021. HEMET: A Homomorphic-Encryption-Friendly Privacy-Preserving Mobile Neural Network Architecture. In *Proceedings of the 38th International Conference on Machine Learning, ICML 2021, 18-24 July 2021, Virtual Event (Proceedings of Machine Learning Research, Vol. 139)*, Marina Meila and Tong Zhang (Eds.). PMLR, 7102–7110. http://proceedings.mlr.press/v139/lou21a.html

[38] Ahmet Can Mert, Emre Karabulut, Erdinç Öztürk, Erkay Savas, and Aydin Aysu. 2022. An Extensive Study of Flexible Design Methods for the Number Theoretic Transform. *IEEE Trans. Computers* 71, 11 (2022), 2829–2843. doi:10.1109/TC.2020.3017930

[39] Shervin Minaee, Yuri Boykov, Fatih Porikli, Antonio Plaza, Nasser Kehtarnavaz, and Demetri Terzopoulos. 2022. Image Segmentation Using Deep Learning: A Survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 44, 7 (2022), 3523–3542. doi:10.1109/TPAMI.2021.3059968

[40] Christian Vincent Mouchet, Jean-Philippe Bossuat, Juan Ramón Troncoso-Pastoriza, and Jean-Pierre Hubaux. 2020. Lattigo: A multiparty homomorphic encryption library in go. In *Proceedings of the 8th Workshop on Encrypted Computing and Applied Homomorphic Cryptography.* 64–70.

[41] Vinod Nair and Geoffrey E. Hinton. 2010. Rectified Linear Units Improve Restricted Boltzmann Machines. In *Proceedings of the 27th International Conference on Machine Learning (ICML-10), June 21-24, 2010, Haifa, Israel*, Johannes Fürnkranz and Thorsten Joachims (Eds.). Omnipress, 807–814. https://icml.cc/Conferences/2010/papers/432.pdf

[42] Jaiyoung Park, Michael Jaemin Kim, Wonkyung Jung, and Jung Ho Ahn. 2022. AESPA: Accuracy Preserving Low-degree Polynomial Activation for Fast Private Inference. *CoRR* abs/2201.06699 (2022). arXiv:2201.06699 https://arxiv.org/abs/2201.06699

[43] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Z. 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 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, Hanna M. Wallach, Hugo Larochelle, Alina Beygelzimer, Florence d'Alché-Buc, Emily B. Fox, and Roman Garnett (Eds.). 8024–8035. https://proceedings.neurips.cc/paper/2019/hash/bdbca288fee7f92f2bfa9f7012727740-Abstract.html

[44] Thomas Pöppelmann and Tim Güneysu. 2012. Towards Efficient Arithmetic for Lattice-Based Cryptography on Reconfigurable Hardware. In *Progress in Cryptology - LATINCRYPT 2012 - 2nd International Conference on Cryptology and Information Security in Latin America, Santiago, Chile, October 7-10, 2012. Proceedings (Lecture Notes in Computer Science, Vol. 7533)*, Alejandro Hevia and Gregory Neven (Eds.). Springer, 139–158. doi:10.1007/978-3-642-33481-8_8

[45] Luis Bernardo Pulido-Gaytan, Andrei Tchernykh, Jorge M. Cortés-Mendoza, Mikhail G. Babenko, Gleb I. Radchenko, Arutyun Avetisyan, and Alexander Yu. Drozdov. 2021. Privacy-preserving neural networks with Homomorphic encryption: Challenges and opportunities. *Peer-to-Peer Netw. Appl.* 14, 3 (2021), 1666–1691. doi:10.1007/S12083-021-01076-8

[46] Ran Ran, Xinwei Luo, Wei Wang, Tao Liu, Gang Quan, Xiaolin Xu, Caiwen Ding, and Wujie Wen. 2023. SpENCNN: Orchestrating Encoding and Sparsity for Fast Homomorphically Encrypted Neural Network Inference. In *International Conference on Machine Learning, ICML 2023, 23-29 July 2023, Honolulu, Hawaii, USA (Proceedings of Machine Learning Research, Vol. 202)*, Andreas Krause, Emma Brunskill, Kyunghyun Cho, Barbara Engelhardt, Sivan Sabato, and Jonathan Scarlett (Eds.). PMLR, 28718–28728. https://proceedings.mlr.press/v202/ran23b.html

[47] Mark Richardson. 1978. Fundamentals of the discrete Fourier transform. *Sound & Vibration Magazine* 12 (1978), 40–46.

[48] Lorenzo Rovida and Alberto Leporati. 2024. Encrypted Image Classification with Low Memory Footprint Using Fully Homomorphic Encryption. *Int. J. Neural Syst.* 34, 5 (2024), 2450025:1–2450025:16. doi:10.1142/S0129065724500254

[49] Karen Simonyan and Andrew Zisserman. 2015. Very Deep Convolutional Networks for Large-Scale Image Recognition. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, Yoshua Bengio and Yann LeCun (Eds.). http://arxiv.org/abs/1409.1556

[50] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jonathon Shlens, and Zbigniew Wojna. 2016. Rethinking the Inception Architecture for Computer Vision. In *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016.* IEEE Computer Society, 2818–2826. doi:10.1109/CVPR.2016.308

[51] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is All you Need. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, Isabelle Guyon, Ulrike von Luxburg, Samy Bengio, Hanna M. Wallach, Rob Fergus, S. V. N. Vishwanathan, and Roman Garnett (Eds.). 5998–6008. https://proceedings.neurips.cc/paper/2017/hash/3f5ee243547dee91fbd053c1c4a845aa-Abstract.html