## RESEARCH ARTICLE

# HyPHEN: A Hybrid Packing Method and Its Optimizations for Homomorphic Encryption-Based Neural Networks

DONGHWAN KIM [ID][1], (Graduate Student Member, IEEE),
JAIYOUNG PARK [ID][2], (Graduate Student Member, IEEE),
JONGMIN KIM[1], (Graduate Student Member, IEEE),
SANGPYO KIM [ID][2], (Graduate Student Member, IEEE),
AND JUNG HO AHN [ID][1,2,3], (Senior Member, IEEE)

[1]Interdisciplinary Program in Artificial Intelligence, Seoul National University, Seoul 08826, South Korea
[2]Department of Intelligence and Information, Seoul National University, Seoul 08826, South Korea
[3]Inter-University Semiconductor Research Center, Seoul National University, Seoul 08826, South Korea

Corresponding author: Jung Ho Ahn (gajh@snu.ac.kr)

**ABSTRACT** Convolutional neural network (CNN) inference using fully homomorphic encryption (FHE) is a promising private inference (PI) solution due to the capability of FHE that enables offloading the whole computation process to the server while protecting the privacy of sensitive user data. Prior FHE-based CNN (HCNN) work has demonstrated the feasibility of constructing deep neural network architectures such as ResNet using FHE. Despite these advancements, HCNN still faces significant challenges in practicality due to the high computational and memory overhead. To overcome these limitations, we present HyPHEN, a deep HCNN construction that incorporates novel convolution algorithms (RAConv and CAConv), data packing methods (2D gap packing and PRCR scheme), and optimization techniques tailored to HCNN construction. Such enhancements enable HyPHEN to substantially reduce the memory footprint and the number of expensive homomorphic operations, such as ciphertext rotation and bootstrapping. As a result, HyPHEN brings the latency of HCNN CIFAR-10 inference down to a practical level at 1.4 seconds (ResNet-20) and demonstrates HCNN ImageNet inference for the first time at 14.7 seconds (ResNet-18).

**INDEX TERMS** Convolutional neural network, cryptography, homomorphic encryption, privacy preserving machine learning, secure two party computation, secure inference.

## I. INTRODUCTION

Private inference (PI) has recently gained the spotlight in the machine-learning-as-a-service (MLaaS) domain, allowing cloud companies to comply with privacy regulations such as GDPR [1] and HIPAA [2]. PI enables inference services at the cloud server while protecting both the privacy of the client and the intellectual properties of the service provider. For example, by exploiting PI, hospitals can provide a private medical diagnosis of diseases, and security companies can provide privacy-preserving surveillance systems, each without accessing client's sensitive data [3], [4].

Fully homomorphic encryption (FHE) [5] is a cryptographic primitive that enables direct evaluation of a set of functions on encrypted data, making it suited for PI in terms of security and usability among other cryptographic candidates [6], [7]. FHE-based PI solutions, illustrated in Figure 1, uniquely feature 1) fully offloading compute process to a server, 2) succinct data communication requirement, and 3) non-disclosure of any information about the model

The associate editor coordinating the review of this manuscript and approving it for publication was Wei Huang [ID].
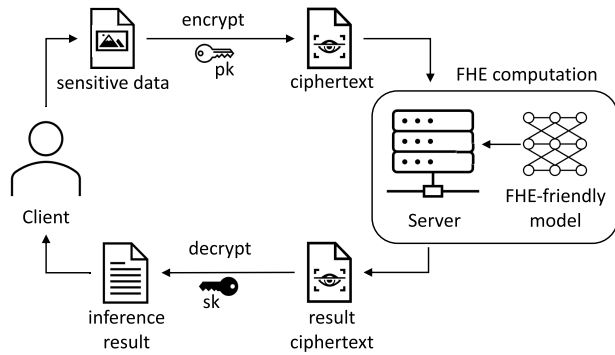
**FIGURE 1.** FHE-based private inference.

except the inference result. Such benefits, enabled by the unique capability of FHE that supports direct computation on ciphertexts (encrypted data), have driven researchers to investigate the FHE-based PI of convolutional neural networks (HCNN) [8], [9], [10], [11], [12]. In particular, this study explores HCNN construction with the CKKS (Cheon-Kim-Kim-Song) FHE scheme [13], which offers higher throughput compared to the other FHE schemes and supports handling real and complex numbers.

Despite these benefits, FHE incurs high computational and memory overhead, which hinders the adoption of HCNN for real-world services. Furthermore, computation with encrypted data in FHE exhibits distinct characteristics compared to its unencrypted counterpart. Particularly, the manipulation of data organization incurs substantial costs when dealing with encrypted data. Thus, to minimize the computational and memory overhead, HCNN requires an optimized convolution algorithm and a distinct data organization tailored to FHE circumstances.

Gazelle [14], a pioneering study in PI, provides an efficient convolutional algorithm that can be used in FHE to reduce the number of homomorphic operations compared to naïvely adopting conventional convolution algorithms used for unencrypted CNN inference. Gazelle avoids the high cost of the FHE evaluation of data rearrangement and nonlinear activation (e.g., ReLU) by combining the use of secure multi-party computation (MPC). Still, Gazelle accumulates data elements dispersed in a ciphertext using FHE, which is also costly. Reference [15] extends Gazelle's algorithm for end-to-end HCNN inference by introducing FHE-based data rearrangement and ReLU evaluation methods. Reference [11] further enhances HCNN performance with a more dense data format, which minimizes the number of ciphertexts. However, prior implementations take tens of minutes [11] to even hours [15] to perform a single HCNN inference for CIFAR-10 (ResNet-20) due to the high cost of accumulation, data rearrangement, and ReLU evaluation. We can significantly reduce the cost for ReLU evaluation by utilizing alternative activation functions proposed in AESPA [16]; however, inefficiency in convolution persists due to the cost of accumulation and data rearrangement.

We tackle this problem by combining multiple flexible data formats with a polymorphic FHE convolution algorithm that performs convolution tailored to each data format. We identify that the high data rearrangement cost stems from the inconsistency in data format between the input and output of a convolution layer. Prior work utilizes a fixed data format throughout HCNN inference, incurring frequent data rearrangement using homomorphic rotation operations. We instead propose using multiple flexible data formats by allowing replication of data elements inside a ciphertext. The input and output of a convolution layer can freely choose from multiple data formats to minimize the cost of data rearrangement. To support different data formats, we also create polymorphic FHE convolution algorithms tailored to each data format. The replication has a positive side effect of reducing the cost of data accumulation during convolution because fewer unique data elements need accumulation.

Furthermore, our data format enables scaling HCNN to real-world images and larger CNN models. In previous works [9], [10], [11], [14], [15], memory expansion of plaintext weight has not received sufficient consideration. However, we identify that the memory footprint of weight plaintext increases significantly with image size and eventually brings a significant bottleneck. Thus, existing techniques do not scale to larger datasets such as ImageNet due to the substantial memory requirement for storing weights. To address this challenge, we introduce a data formatting method that can effectively reduce the plaintext weight size. Our approach involves dividing an image into multiple image segments along the row direction, which ensures that the plaintext size correlates with the image segment size. Consequently, weight plaintexts become smaller and can be reused across these image segments during convolution, alleviating the memory footprint.

We evaluate the real-world scalability of HyPHEN, our HCNN framework combining the previously described solutions. The GPU implementation of HyPHEN achieves 1.40 seconds for encrypted CIFAR-10 inference with the ResNet-20 model. We also demonstrate for the first time end-to-end HCNN inference on the ImageNet dataset with the ResNet-18 model, achieving an execution time of 14.69 seconds.

The main contributions of the paper are as follows:
- We propose a novel data format that enables streamlined data arrangement between consecutive convolutions.
- We devise optimized FHE convolution algorithms, which support various data formats with less computational complexity.
- We identify that the huge memory footprint of HCNN deteriorates its performance and propose an efficient data format that can save hundreds of gigabytes of memory space with negligible overhead.
- We showcase the evaluation of various neural networks within practical execution times. We extend FHE-based PI to complex real-world data, such as ImageNet,

| Operation | AddPt | AddCt | MulPt | MulCt | Rescale | PRot | CRot | Boot |
|---|---|---|---|---|---|---|---|---|
| Time (ms) | 0.169 | 0.202 | 0.506 | 17.3 | 3.90 | 0.102 | 15.5 | 2160 |

by demonstrating the performance with the ResNet-18 model.

## II. BACKGROUND

### A. FULLY HOMOMORPHIC ENCRYPTION (FHE)

FHE is a set of public key encryption schemes that enable computation on encrypted data. Within the array of widely adopted FHE schemes, RNS-CKKS [17] has been favored in the PI domain. This scheme leverages an optimization technique of Residue Number System (RNS) decomposition within the CKKS approximate homomorphic encryption scheme. The popularity of RNS-CKKS stems from its support for fixed-point numbers and its capability to perform *slot batching*.

A *plaintext* in RNS-CKKS is an unencrypted polynomial in a cyclotomic polynomial ring $\mathbb{Z}[X]/(X^N + 1)$ for a power-of-two degree $N$, whose typical values are $2^{15}$–$2^{17}$. Encryption of a plaintext generates a *ciphertext*, a pair of polynomials hiding the plaintext using an additional random polynomial and an obfuscating error term. As a vector containing $N/2$ real (or complex) numbers, referred to as *message*, can be mapped to a plaintext, we can simultaneously operate on $N/2$ numbers by performing homomorphic operations on plaintexts and ciphertexts. This approach, known as slot batching, refers to the specific position within this vector message as a *slot*. Additionally, it is possible to batch a message with a shorter length, typically following a size that is a power of two.

The RNS-CKKS scheme supports slot-wise additive operations (AddPt/AddCt) and multiplicative operations (MulPt/MulCt). These operations, along with their respective execution times, are presented in Table 1. AddCt/MulCt, for instance, receives two ciphertexts as input and returns a ciphertext corresponding to a vector message approximately equal to the element-wise addition/multiplication result between the two vector messages in the input ciphertexts. AddPt/MulPt does a similar job except that it receives a plaintext and a ciphertext as input and involves less computation than AddCt/MulCt. In particular, MulPt takes $34\times$ less execution time compared to MulCt.

A fundamental constraint of HE lies in the limited number of sequential multiplications possible with a ciphertext. Each ciphertext or plaintext is associated with a (multiplicative) level $\ell$, where $0 \leq \ell \leq L$ for the max level $L$. The size of ciphertext or plaintext and the complexity of homomorphic operations increase with the level. Following MulCt or MulPt, a Rescale process is required for the output ciphertext. Rescale is crucial as it reduces the amplified error in the ciphertext caused by multiplication. Rescale reduces the level by one with the constraint that the level cannot drop below zero. Additionally, when performing operations with operands having different levels, Rescale operations are essential to adjust the levels of the operands to match the lowest level among them.

To overcome this constraint, bootstrapping (Boot in Table 1) serves as the solution. Bootstrapping emerges as a unique operation designed to elevate the level of a ciphertext. Bootstrapping elevates a ciphertext's level up to $L'$, which is smaller than the max level $L$ due to the levels bootstrapping consumes for its computation. Therefore, in practice, we can only utilize $L'$ levels for other operations. As can be observed from Table 1, bootstrapping incurs significantly higher costs than basic operations. Therefore, it is crucial to suppress level consumption and perform bootstrapping as few as possible.

Another limitation of HE is that, when using slot batching, it is difficult to arbitrarily change the data order among slots. The only available option is cyclically shifting the slots, which we refer to as rotation. CRot is a homomorphic operation to rotate a ciphertext to the left by a given amount. When computation between data elements in different slots is required, CRot is performed to adjust the positions. As such a computational pattern is extremely common, CRot is one of the most frequently performed operations in RNS-CKKS applications.

Also, as CRot is a relatively expensive operation among basic operations (see Table 1), CRot accounts for a large portion of computational overhead. There is also a rotation operation for plaintexts, PRot, but it is rarely performed and is computationally cheap.

The data organization among the slots is the paramount concern in RNS-CKKS because it determines the number of CRot and bootstrapping operations. The cost of other operations is much less sensitive to the data organization. To reduce the number of bootstrapping, it is advisable to pack as many data elements as possible into a ciphertext to reduce the number of ciphertexts to bootstrap. However, such a dense packing may result in increased amounts of rotations because putting some data elements instead into another ciphertext would remove the need for rotation. As an extreme example, if we pack only one element per ciphertext, we can eliminate all rotation operations; however, this would incur excessively high costs for bootstrapping and other operations. Therefore, to deliver high performance for RNS-CKKS applications, we need to devise an application-specific data organization that strikes a balance between the two objectives, minimizing computation among different slots and maximizing slot usage.

### B. CONVOLUTION ON HE

Table 2 summarizes the symbols and their descriptions used throughout the paper. As aforementioned in the previous section, the symbols $L, L', \ell$ denote the maximum level, multiplicative level, and current level of a ciphertext,
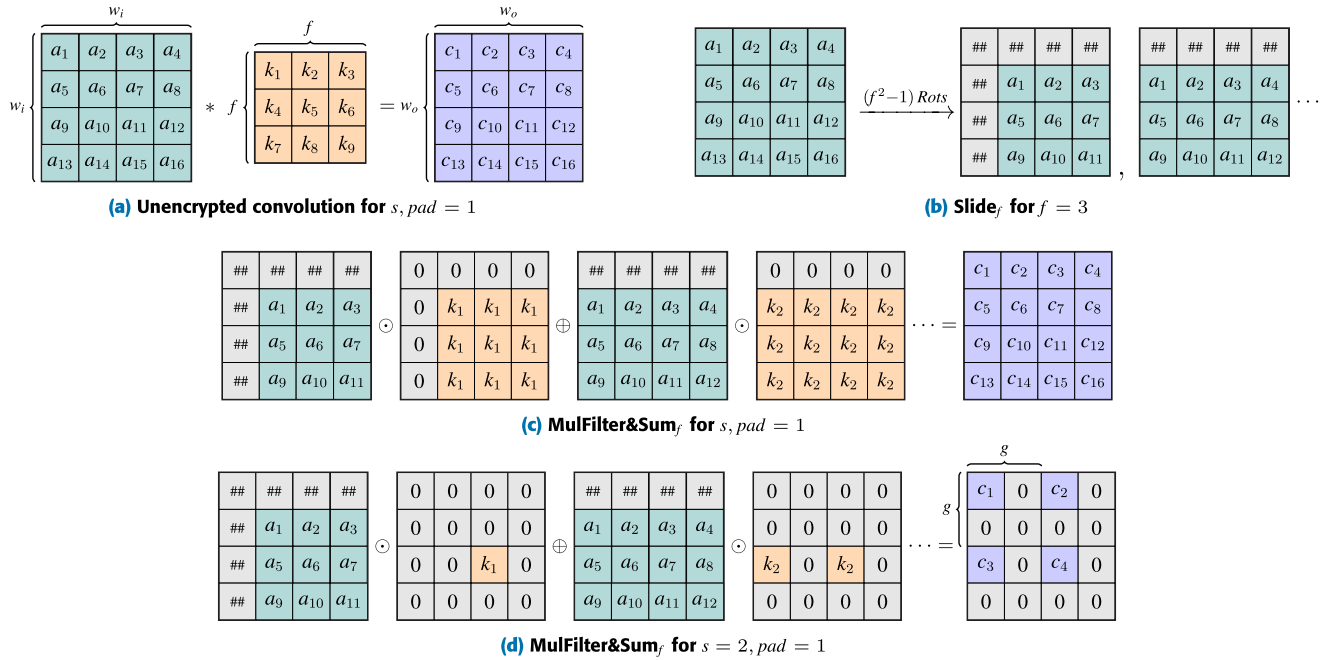
**(a)** Unencrypted convolution for $s, pad = 1$

**(b)** Slide$_f$ for $f = 3$

**(c)** MulFilter&Sum$_f$ for $s, pad = 1$

**(d)** MulFilter&Sum$_f$ for $s = 2, pad = 1$

**FIGURE 2.** Single-input and single-output channel convolution (SISO) [14]. Image ciphertexts and filter plaintexts are illustrated as 2D matrices, but are stored in a 1D manner with each matrix row concatenated. ⊙ symbolizes MulPt and ⊕ symbolizes AddCt.

**TABLE 2.** Notations and descriptions of the symbols used.

| Notation | Description |
| --- | --- |
| $L$ | Maximum level |
| $L'$ | Multiplicative level |
| $\ell$ | Current level |
| $c_i, c_o$ | Input and output channels |
| $w_i, w_o$ | Input and output width of images |
| $f$ | Filter width of convolution |
| $s$ | Stride of convolution |
| $pad$ | Zero padding of convolution |
| $n_i, n_o$ | # of input and output ciphertexts through convolution |
| $g$ | Width (height) of gap |
| $m, d$ | Gap packing configuration – a pair of multiplexed channels and duplicates |
| Dnum | Decomposition number [18] |
| Evk | Evaluation key, utilized at mul or rotation. |

**Algorithm 1** Single-input, single-output channel convolution (SISO)

**Input** $ct_i$: input ciphertext, $W$: plaintext filter
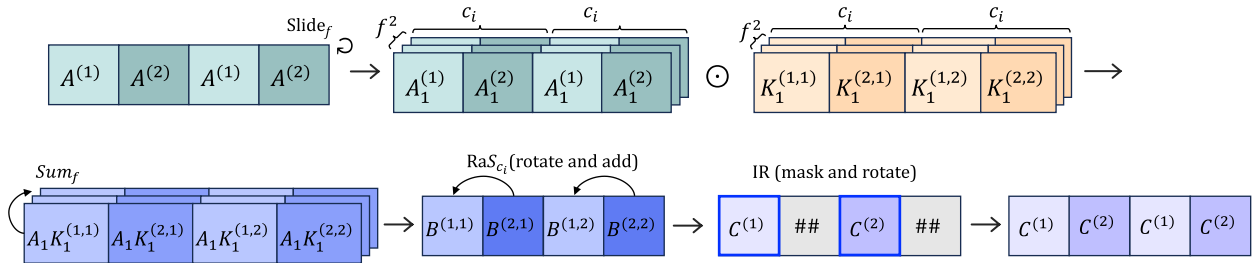**Output** $ct_o$: output ciphertext

1: // SubRoutine: **Slide**$_f(ct_i)$
2: **for** $j_1 = 0, \ldots, f - 1$ **do**
3:     **for** $j_2 = 0, \ldots, f - 1$ **do**
4:        $r \leftarrow w_i(j_1 - \frac{f-1}{2}) + (j_2 - \frac{f-1}{2})$
5:        $ct'[j_1, j_2] \leftarrow \text{CRot}(ct_i; r)$
6: // SubRoutine: **MulFilter&Sum**$_f(ct_i, W)$
7: **for** $j_1 = 0, \ldots, f - 1$ **do**
8:     **for** $j_2 = 0, \ldots, f - 1$ **do**
9:        $ct_o \mathrel{+}= \text{MulPt}(ct'[j_1, j_2], W[j_1, j_2])$
10: **return** $ct_o$

respectively. Regarding the notations for the baseline CNN network, we simplify the notation by assuming square-shaped images and filters. We represent the input and output of a single convolutional layer with tuples {channel, image width} as $\{c_i, w_i\}$ and $\{c_o, w_o\}$, respectively. Parameters for a convolution are represented with tuples {output channel, input channel, filter width} denoted as $\{c_o, c_i, f\}$, stride $s$, and padding $pad$. For other symbols associated with the context of our HCNN implementation, we will provide detailed explanations within the relevant sections of the paper.

*Single-Input, Single-Output Channel Convolution (SISO):* Gazelle [14] proposes an efficient convolution algorithm on HE, referred to as SISO. Figure 2 illustrates SISO with $s$ equal to 1 and 2, where an input ciphertext contains $w_i^2$ pixels.

Although we represent the data format with two dimensions as $(H, W)$ in this example, pixels are stored in the slots in a flattened row-major order regardless of the number of dimensions. $f^2$ plaintexts each storing distinct filter elements need to be prepared for SISO. Each slot of the $i$-th plaintext $(0 \le i < f^2)$ holds a filter element $(k_i)$ or zero (0), depending on whether $k_i$ participates in the computation of the output pixel at the same slot. SISO operation proceeds as follows (see Algorithm 1):

1) *Slide$_f$* rotates an input image placed in encrypted ciphertext with different rotation amounts for each of the $f^2$ plaintexts as shown in Figure 2b.
2) *MulFilter&Sum$_f$* multiplies each rotated input by a filter plaintext and accumulates each of the $f^2$ multiplied results to obtain the output as depicted in Figure 2c.

**FIGURE 3.** Convolution procedure in [11] when $c_i$, $c_o = 2$. A single superscript denotes the channel and a superscript pair denotes (input channel, output channel). $A^{(m)}$ represents the $m$-th channel of the input image contiguously placed in the slots and $K^{(m,n)}$ is the corresponding filter deployed in plaintexts. We simplify the notation of the intermediate SISO result $\sum_{i=1}^{f^2} CRot(A^{(m)}; r(i))K_i^{(m,n)}$ as $B^{(m,n)}$. $C^{(n)}$ represents the $n$-th channel of the output image.

As an input image tuple in CNN usually has multiple channels, multiple ciphertexts are often required per convolutional layer when using SISO. We denote the number of input and output ciphertexts per convolutional layer as $n_i$ and $n_o$, respectively.
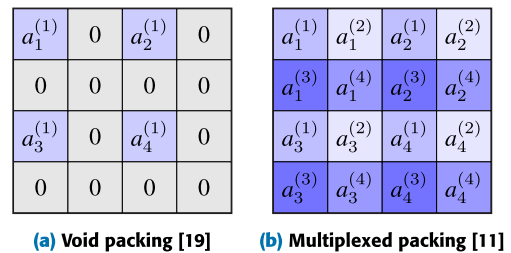
*Convolution and Data Format:* When the image size ($w_i^2$) is smaller than $N/2$, we can reduce $n_i$ and $n_o$ by batching multiple channels into a single ciphertext. Gazelle also proposes a *channel-aligned* batching method where the data format of the input and output ciphertexts adopts a flattened $(C, H, W)$ 3D vector. Reference [15] follows it and utilizes the same $(C, H, W)$ format to implement an end-to-end CNN inference using FHE. By following steps similar to SISO, this format enables performing convolutions on multiple channels simultaneously. Moreover, [11] introduces input repetition to amplify parallelism, which we represent the data format as $(R, C, H, W)$ and the input tuple is repeated $|R|$ times to fill all the slots of a ciphertext. In Figure 3, we present an example when $\{c_o, c_i, w_i\} = \{2, 2, 32\}$ and the number of slots ($N/2$) is 4,096. The input ciphertext ($n_i = 1$) has two channels ($A^{(1)}, A^{(2)}$) repeated twice. $f^2$ filter plaintexts are prepared, where each plaintext holds filter elements with $|C|$ input channels for $|R|$ output channels.

Convolution on this ciphertext can be described as follows:

1) SISO: With a single input ciphertext, SISO can be performed to get convolution results for $|C|$ input channels and $|R|$ output channels simultaneously. The resulting ciphertext contains $|R||C|$ intermediate convolution outputs $B^{(x,y)}$ ($1 \le x \le |C|, 1 \le y \le |R|$).
2) $RaS_{c_i}$: To obtain the result for the $y$-th output channel, accumulation of $|C|$ intermediate results are performed ($\sum_{x=1}^{|C|} B^{(x,y)}$) by computing *rotate and sum* (RaS), which requires $\log |C|$ rotations.
3) IR: For subsequent convolutions, data rearrangement aligns the data format to match the subsequent layer's format by masking and rotating data elements, which we refer to as *image realigning* (IR).

Throughout this paper, we refer to this convolution that takes an $(R, C, H, W)$ ciphertext as $Conv_{lc}$.

*Gap and Multiplexed Packing:* Strided convolution ($s > 1$) using SISO generates a *gap* (denoted as $g$) between valid



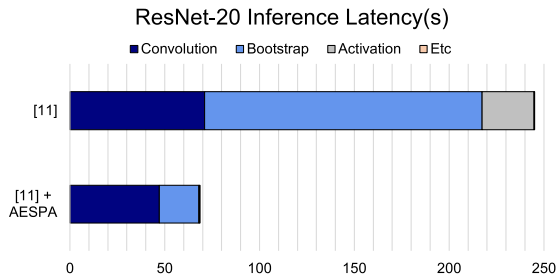**(a) Void packing [19]**    **(b) Multiplexed packing [11]**

**FIGURE 4.** Previous gap packing methods to fill gap induced by downsampling layers. $a_i^{(j)}$ denotes the $i$-th element in the $j$-th channel of an image.

values (see Figure 2d). When using a ciphertext with a gap, slot underutilization degrades the throughput. While interactive protocols such as Gazelle [14] remove the gap by a client-aided re-encryption process, FHE demands heavy masking (MulPt) and rotation (CRot) operations to remove the gap, which incurs additional computation and level consumption. Reference [11] proposes *multiplexed packing* (MP) method on top of $Conv_{lc}$ ($MP$-$Conv_{lc}$) to remedy the slot underutilization.

$MP$-$Conv_{lc}$ adds *repacking process* to the IR process of $Conv_{lc}$, which fills the gap with different channels (see Figure 4b). $MP$-$Conv_{lc}$ uses the data format of $(R, C_a, H, W, C_g)$, adding mutiplexed channel at the innermost dimension $C_g$. The convolution process on an encrypted image with $|C_g| = 4$ is depicted in Figure 8. After applying SISO and RaS on the outer channel dimension $C_a$, additional RaS accumulates channels in the inner dimension $C_g$ within the gap. Multiplexed packing requires a more complex IR process to fill the gap with multiple channels from the output tensor. For further details of $MP$-$Conv_{lc}$, we refer the readers to Section IV of [11].

## C. ACTIVATION FUNCTION ON HE

Nonlinear activation functions, for example ReLU, cannot be used directly in HCNN. They must be replaced by polynomial functions approximating them because FHE only supports additive and multiplicative operations. Maintaining a low polynomial approximation error across a wide range

## ResNet-20 Inference Latency(s)



**FIGURE 5.** ResNet-20 inference latency and their breakdown for [11] and the adoption of low-degree polynomial [16]. Two experiments are conducted under paramter Set$_{lc}$ and Set$_{hyp}$ from 9, respectively.

is crucial to preserving the accuracy of a CNN model. Reference [15] approximates ReLU with a composition of 15-, 15-, and 27-degree polynomials, which keeps L1 norm of approximation error lower than $2^{-13}$ in range [-50, 50]. This approach has the benefit that pretrained CNN models can be directly used without modification. However, the evaluation of high-degree polynomials imposes a significant runtime overhead; a series of Rescale during the evaluation incurs a lot of level consumption, resulting in an increased number of bootstrapping. One may attempt to mitigate this overhead by using a CKKS parameter with more levels (i.e., higher $L'$); however, such a parameter set has an extremely large memory footprint as presented in Table 9, and thus each operation becomes much more expensive.

Another approach is to retrain neural networks with low-degree polynomial activation functions as in [20], [21], [22], [23], and [24]. By retraining, the operational cost drops significantly. Recently, AESPA [16] has shown that CNNs trained with low-degree polynomials can achieve equivalent accuracy to the original ReLU-based networks across various CNN architectures and image datasets. AESPA replaces ReLU and batch normalization (BN) with the composition of orthogonal basis polynomials and basis-wise BN. During inference, vertical layer fusion transforms the composition into a simple square function, drastically reducing the runtime of activation.

Due to the use of high-degree polynomials, the primary performance bottleneck of prior work [11] stems from bootstrapping operations. In contrast, our analysis reveals that, if we adopt AESPA, the portion of bootstrapping in the entire HCNN inference time becomes small and convolution operations dominate the execution time (see Figure 5). We also analyze that rotation operations (CRot) account for the most of computation in convolution. Therefore, in this work, we focus on enhancing the performance of convolution by proposing convolution algorithms and packing methods that effectively mitigate the substantial rotation overhead in HCNN inference.

## III. HYPHEN CONSTRUCTION
We introduce HyPHEN, our HCNN solution that focuses on reducing the memory footprint and the number of

resource-intensive homomorphic operations, including rotation and bootstrapping. We propose convolution algorithms and data formats that can streamline data arrangement between consecutive convolutions. Our method consists of two data formats: channel-aligned (CA) and replication-aligned (RA) formats, which we denote as $\pi_{CA}$ and $\pi_{RA}$. Both formats can be formally described as $\pi_{CA} = \{C_a, H, W, R_g, C_g\}$, $\pi_{RA} = \{R_a, H, W, C_g, R_g\}$. In the outermost dimension, $\pi_{CA}$ aligns images with different channels ($C_a$) similar to [14] and [19], whereas $\pi_{RA}$ aligns with the replications ($R_a$) of images. We introduce two distinct convolutions for the two data formats in the following section.

### A. CONVOLUTION ALGORITHMS OF HYPHEN
We devise two convolution algorithms that start with input ciphertexts in the format of $\pi_{CA}$ and $\pi_{RA}$, respectively (see Figure 6). For the simplicity of illustration, we assume that the size of the last two dimensions is one, implying that no gap exists, in the figure; gap packing is handled by additional computation during RaS and IR, which will be discussed in Section III-B. We focus on how the outermost $C_a$ ($R_a$) dimension is handled for $\pi_{CA}$ ($\pi_{RA}$).

**Channel-aligned convolution (CAConv)** is designed for the $\pi_{CA}$ data format, resembling *MP-Conv$_{lc}$* but without requiring the IR process. Recall that, in *MP-Conv$_{lc}$* targeting the data format $\{R, C_a, H, W, C_g\}$, $|C_a|$ ($|C_g| = 1$ for simplicity) intermediate SISO results are accumulated into the first channel's position and the rest of the slots contains meaningless values (## in Figure 3). Subsequent IR rearranges the ciphertext for the next convolution. In contrast, by relocating the input repetition to inside the gap ($\pi_{CA} = \{C_a, H, W, R_g, C_g\}$), we can position $C_a$ as the outermost dimension. Executing RaS on this ciphertext automatically leads to the replication of the output result because homomorphic rotation is cyclic. The resulting ciphertexts follow the data format of $\pi_{RA}$ as depicted in Figure 6a. By using CAConv, our image realignment does not require any rotation at the cost of an increase in the number of output ciphertexts by $c_o$.

**Replication-aligned convolution (RAConv)** goes even further by eliminating most of the RaS computation in addition. RAConv performs a sequence of $\{Slide_f, MulFilter, Sum_{f,c_i}\}$ operations on $\pi_{RA}$-formatted inputs and returns $\pi_{CA}$-formatted results. $Slide_f$ expands input ciphertexts with $f^2$ rotations and *MulFilter* multiplies slided ciphertexts with plaintext filters using MulPt operations. $Sum_{f,c_i}$ accumulates the intermediate results of $f^2 \cdot c_i$ ciphertexts using only additions (AddCt) without costly rotation operations. The resulting ciphertexts follow the $\pi_{CA}$ format, allowing the next CAConv to directly utilize them as input.

As the number of input ciphertexts ($n_i$) of RAConv has increased by the number of input channels $c_i$, additional rotation cost ($c_i \cdot (f^2 - 1)$ rotations) is required for $Slide_f$. The increase in the $Slide_f$ cost undermines the performance enhancement from reduced rotations in RaS and
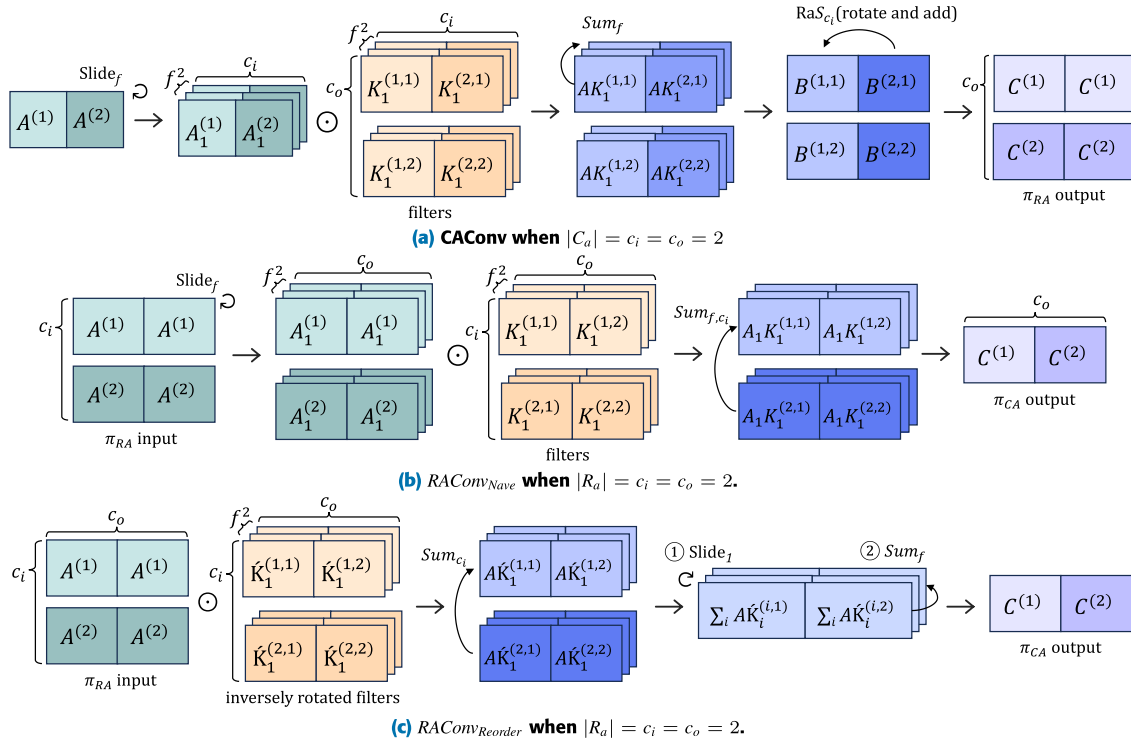
**FIGURE 6.** CAConv and two variants of RAConv. A single superscript denotes the channel and a superscript pair denotes (input channel, output channel). We simplify the notation of $M^{(a)}K^{(a,b)}$ as $MK^{(a,b)}$.
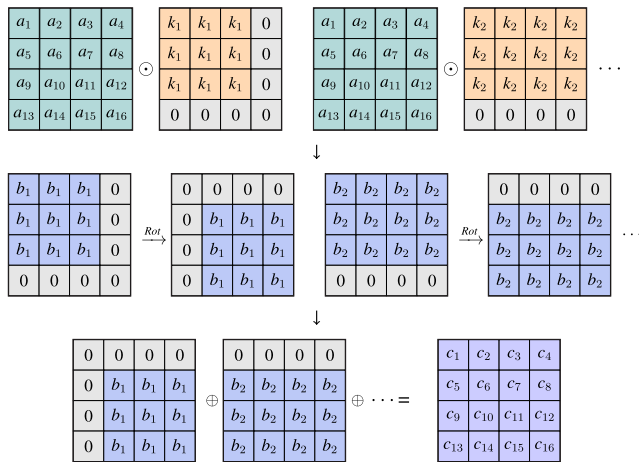


**FIGURE 7.** Plaintexts inversely rotated for SISO reordering.

---

**Algorithm 2** Reordered SISO for RAConv

**Input** $ct_i$: input ciphertext, $W'$: filter plaintexts inversely rotated

**Output** $ct_o$: output ciphertext

1: // **MulFilter&Sum$_{c_i}$**$(ct_i, W)$
2: **for** $j_1 = 0, \ldots, f-1$ **do**
3:     **for** $j_2 = 0, \ldots, f-1$ **do**
4:        $ct'[j_1, j_2] \mathrel{+}= \text{MulPt}(ct_i, W'[j_1, j_2])$
5: // **Slide$_1$&Sum$_f$**$(ct')$
6: **for** $j_1 = 0, \ldots, f-1$ **do**
7:     **for** $j_2 = 0, \ldots, f-1$ **do**
8:        $r \leftarrow w_i(j_1 - \frac{f-1}{2}) + (j_2 - \frac{f-1}{2})$
9:        $ct_o \mathrel{+}= \text{CRot}(ct'[j_1, j_2], -r)$
10: **return** $ct_o$

---

IR. We resolve this issue by reordering the operations in RAConv, which significantly reduces the $Slide_f$ cost.

*Reordered RAConv:* We *rearrange* the RAConv sequence to {$MulFilter$, $Sum_{c_i}$, $Slide_1$, $Sum_f$} under the observation that the order of sliding and filter multiplication can be reversed if we prepare filter plaintexts to be inversely rotated. We describe the procedure of reordered SISO in Figure 7 and Algorithm 2. Unlike $Slide_f$ that generates $f^2$ rotated ciphertext from a single input ciphertext, $Slide_1$ gathers $f^2$ ciphertexts into one by single rotations and addition. By reordering these operations, we can perform rotations after $n_i$ input ciphertexts are accumulated as shown in Figure 6c, effectively reducing the number of rotations required for sliding from $c_i \cdot (f^2 - 1)$ to $(f^2 - 1)$.

We also propose an inter-layer optimization aimed at reducing the memory footprint required for ciphertexts in CAConv and RAConv, as outlined in Algorithm 3. In homomorphic convolutions, the number of input, output, and temporary ciphertexts is determined by their data formats. Our optimization utilizes aggressive forwarding which aims to avoid states where the intermediate data formats occupy large memory space (e.g. $ct_2$, $ct_3$ in Algorithm 3). We denote

**TABLE 3.** Cost of homomorphic convolutions. We compare our convolutions with [11] when $n_i = n_o = 1$ in their setting. ($c_n$: channel multiplexing in $C_a$ dimension), (m: channel multiplexing in $C_g$ dimension), (d: input repetition in R dimension).

| Method | $n_i$ | $n_o$ | Slide | RaS | $RaS_g$ | IR | $IR_g$ |
|---|---|---|---|---|---|---|---|
| **MP-Conv$_{lc}$ [11]** | 1 | 1 | $f^2$-1 | $\frac{mc_n}{d}\log(c_n)$ | $\frac{mc_n}{d}\log(m)$ | $(mc_n$-1+$\log_2(d))$ | |
| **CAConv** | 1 | $\frac{mc_n}{d}$ | $f^2$-1 | $\frac{mc_n}{d}\log(c_n)$ | $\frac{mc_n}{d}\log(m)$ | 0 | $\frac{mc_n}{d}\log(m)$ |
| **RAConv$_{Naive}$** | $\frac{mc_n}{d}$ | 1 | $\frac{mc_n}{d}(f^2$-1) | 0 | $\log(m)$ | 0 | $\log(m)$ |
| **RAConv$_{Reorder}$** | $\frac{mc_n}{d}$ | 1 | $f^2$-1 | 0 | $\log(m)$ | 0 | $\log(m)$ |

---

**Algorithm 3** CAConv and RAConv Fused Block
___
**Input** $ct_i$: input ciphertext vector, $W_{CA}$, $W_{RA}$: weight plaintexts for CAConv, RAConv, $n_{CA}$, $n_{RA}$: number of ciphertexts with data format $\pi_{CA}$, $\pi_{RA}$, $ct_k$: temporary ciphertexet vectors
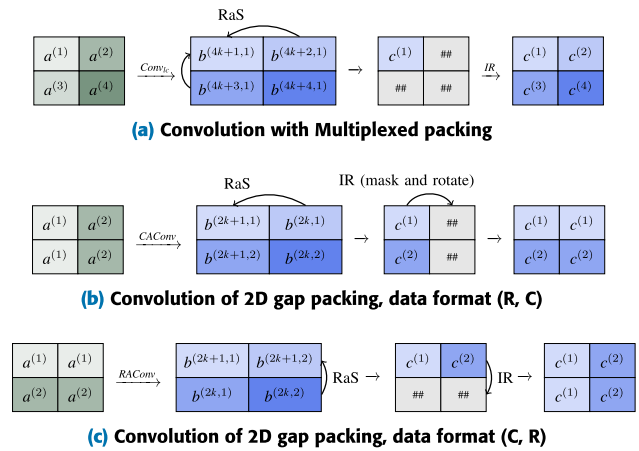**Output** $ct_o$: output ciphertext vector
___
1: **for** $k = 0, \ldots, n_{CA}$-1 **do**
2:    $ct_1[k] \leftarrow$ **Slide**$_f(ct_i[k])$
3: **for** $j = 0, \ldots, n_{RA}$-1 **do**
4:    **for** $k = 0, \ldots, n_{CA}$-1 **do**
5:       $ct_2[j, k]$ +=**MulFilter&Sum**$_f(ct_1[k], W_{CA}[j, k])$
6:       $ct_3[j] \leftarrow$ **RaS**$_{c_i}(ct_2[j, k])$
7:       $ct_3[j] \leftarrow$ **Square**$(ct_3[j])$
8:       **for** $l = 0, \ldots, n_{CA}$-1 **do**
9:          $ct_4[l]$ += **MulFilter&Sum**$_{c_i}(ct_3[j], W_{RA}[l, j])$
10: **for** $k = 0, \ldots, n_{CA}$-1 **do**
11:     $ct_o[k] \leftarrow$ **Slide$_1$** & **Sum**$_f(ct_4[k])$
12: **return** $ct_o$
___



**(a) Convolution with Multiplexed packing**



**(b) Convolution of 2D gap packing, data format (R, C)**



**(c) Convolution of 2D gap packing, data format (C, R)**

**FIGURE 8.** The procedure of convolution on single pixel of image with gap size $g$=2. $a_i^{(j)}$ denotes the $i$-th element in the $j$-th channel of an input image tuple $a$. $b^{(mk+n,l)}$ represents the channels are accumulated by stride m $\sum_{k=1}^{\frac{c_i}{m}} b^{(mk+n,l)}$.

the number of inputs for CAConv and RAConv as $n_{CA}$ and $n_{RA}$, respectively, to prevent confusion incurred by fusing these two convolutions. By fusing loops that yield $ct_2$ and $ct_3$ into $ct_4$, we forward each ciphertext from $ct_2$, $ct_3$ to subsequent operations until obtaining $ct_4$ (see line number 4-9). Our forwarding optimization retains the number of ciphertexts to $n_{CA} \cdot f^2$, which is much smaller than $n_{RA} \cdot f^2$.

### B. DATA FORMATS FOR 2D GAP PACKING

We propose a gap packing method that collaboratively employs duplication and channel multiplexing to alleviate the high cost of the repacking process. In the innermost two dimensions, referred to as *2D gap packing*, $C_g$ and $R_g$ of $\pi_{CA}$ and $\pi_{RA}$ represent duplication and channel multiplexing, respectively. Unlike *MP-Conv$_{lc}$*, which maintains a single $C_g$ for convolution, $\pi_{CA}$ and $\pi_{RA}$ act together in a complementary manner such that each convolution transforms $C_g$ ($R_g$) into $R_g$ ($C_g$). Our key observation is that introducing this heterogeneity can significantly reduce the number of rotations being invoked. Specifically, in *MP-Conv$_{lc}$*, image repacking inside the gap ($IR_g$) spends $O(g^2)$ rotations to maintain the gap packing by $C_g$, when $g$ denotes the gap width (height). In contrast, when using our 2D gap packing, IR only

requires $O(\log g^2)$ rotations in the $R_g$ direction as shown in Figure 8b and Figure 8c.

### C. COMPLEXITY ANALYSIS

We describe the complexity of rotation counts of convolution algorithms in Table 3. We first start with the baseline [11], and denote $c_n$, $m$ and $d$ as $|C_a|$, $|C_g|$ and $|R|$ of *MP-Conv$_{lc}$*. In CAConv, d turns into $|R_g|$ as we relocate input repetition to the gap dimension, then RAConv proceeds on the output of CAConv. Except for the RaS of CAConv, our convolution does not require any rotation for RaS and IR. As elaborated in Section III-A, our convolution does not require any rotations except for RaS of CAConv aside from the gap. By reorganizing the order of sliding rotation, Slide of RAConv$_{Reorder}$ requires $\frac{mc_n}{d} \times$ fewer rotations compared to RAConv$_{Naive}$ We also separately present rotation amounts to retain our 2D gap packing method ($RaS_g$ and $IR_g$). Overall, our construction halves the rotation cost of RaS and avoids the high rotation cost of IR.

We highlight that $n_i$ and $n_o$ have a direct impact on the total number of activation functions and bootstrappings. For instance, $n_i$ of RAConv is typically larger than $n_i$ of CAConv, increasing the number of activation functions or
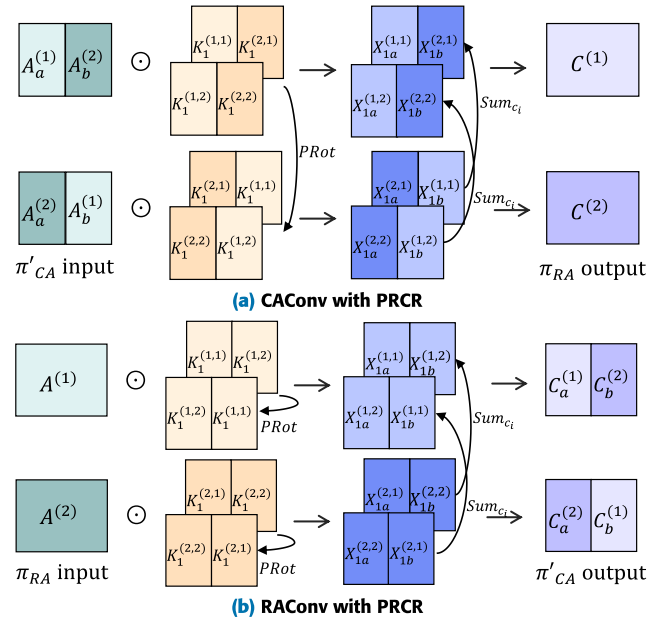
bootstrappings on $\pi_{RA}$. Unlike the square activation function which has minor impacts on overall performance, HCNN performance is critically dependent on the number of bootstrapping. Therefore, we conduct bootstrapping when the number of ciphertexts is minimal, particularly in the format of $\pi_{CA}$. Furthermore, during end-to-end network implementation, we fine-tune $(m, d)$ on each block, aiming for the least total execution time for rotation and bootstrapping. An in-depth performance analysis of the choice of $(m, d)$ is provided in Section IV-E.

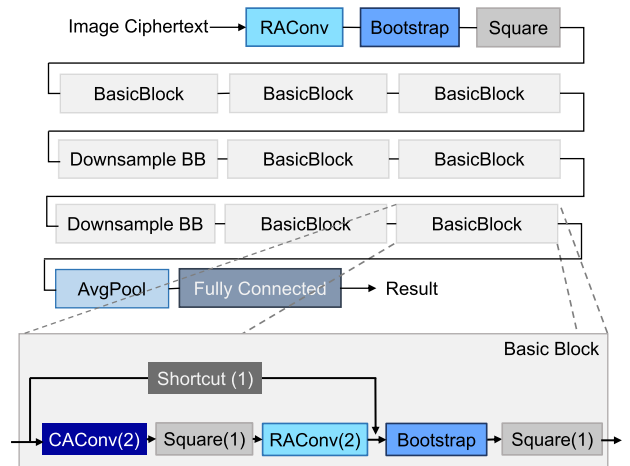### D. PLAINTEXT SIZE REDUCTION THROUGH IMAGE TO CIPHERTEXT REARRANGEMENT

When extending HCNN to high-resolution images (e.g., $224 \times 224$ images in ImageNet), HCNN encounters a significant surge in memory footprint. This spike is especially pronounced in the weight plaintext, with each filter element occupying the fragment-sized $w_i h_i$ slots as shown in Figure 2c. In total, filters require $w_i h_i f^2 c_i c_o$ slots to produce weight plaintexts. For instance, when running ResNet-18 on ImageNet, weight plaintexts alone occupy a substantial 364.8GB of memory space (see Table 5). This exceeds the memory capacity of a single cutting-edge GPU, limited by the current memory technology.

We propose an optimization named PRCR (**P**laintext size **R**eduction through the image to **C**iphertext **R**earrangement) to mitigate the data expansion of plaintexts. Our method involves splitting an image into multiple image segments along the row direction. Thus, the data format of CAConv is adjusted to $\pi_{CA'} = \{C_a, S, H', W, R_g, C_g\}$, where $S$ denotes the dimension of sub-images and $H'$ denotes the dimension of rows of an image segment. This rearrangement can reduce the fragment size of a filter element to the image segment size. Furthermore, These fragments are organized circularly, allowing the pieces to be reused by permutation. In Figure 9, we depict PRCR method when $slot = w_i h_i$, $f = 1$, and $c_i = c_o = 2$. CAConv reuses a single weight plaintext $|S|$ times, by multiplying with an input ciphertext and rotating by a fragment size. Subsequent $Sum_{c_i}$ accumulates input channels, restoring image-size fragments and generating replication-aligned ciphertexts. While $\pi_{RA}$ remains unchanged, PRCR rearranges weight plaintexts into smaller fragments, placing different output channels circularly. As CAConv, RAConv reuses the weight plaintexts $|S|$ times with multiplication and rotation and returns $\pi_{CA'}$.

PRCR can be independently applied along with any form of packing technique to reduce plaintext memory. The extra computation of PRot is not significant to the overall performance, because PRot is much cheaper than other FHE operations as presented in Table 1. For our implementation, we use $|S| = 8$ for ImageNet. However, as GPU memory is capable of CIFAR-10 workloads, we do not apply PRCR to CIFAR-10.



**FIGURE 9.** Simplified convolution processes with PRCR. We assume $c_i = c_o = 2$ but with images per ciphertext $c_n$ and filter size $f$ equals one for this figure. Ciphertexts are filled with subscript $a$ and $b$ represents upper half and lower half of an image, respectively (e.g., $\pi'_{CA} A$ of (a) has single image, but each composed of different half of channel image). PRCR reuses weight plaintexts through plaintext rotation (PRot).



**FIGURE 10.** ResNet basic block built on HyPHEN. The level consumption per block is written in each parentheses. In the downsampling block, pointwise convolution is added to the critical path. Otherwise, a simple shortcut is added.

### E. RESNET CONSTRUCTION WITH HYPHEN

HyPHEN combines RAConv, 2D gap packing and PRCR to build the entire CNN model. Figure 10 illustrates the ResNet implemented in HyPHEN. There are three considerations when deciding the placement of operations. First, it is effective to place bootstrapping after RAConv, and not CAConv, due to the smaller number of ciphertexts involved. Second, to match the levels between the shortcut

**TABLE 4.** Model architecture detail of ResNet-20/32/44 for CIFAR-10 and ResNet-18 for ImageNet. Downsampling Convolution with stride 2 is represented as dsconv. Pointwise convolution with filter size 1 is denoted as pconv.

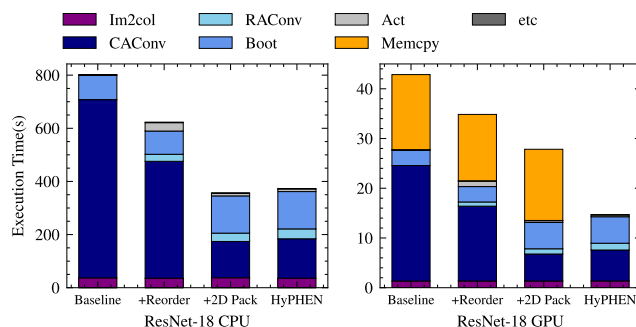| | ResNet-20/32/44 | | | | | | | ResNet-18 | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Layer1 | Layer2 | | | Layer3 | | | Layer1 | Layer2 | | | Layer3 | | | Layer4 | | | | |
| | conv | dsconv | pconv | conv | dsconv | pconv | conv | conv | dsconv | pconv | conv | dsconv | pconv | conv | dsconv | pconv | conv | | |
| $c_i$ | 16 | 16 | 16 | 32 | 32 | 32 | 64 | 64 | 64 | 64 | 128 | 128 | 128 | 256 | 256 | 256 | 512 | | |
| $c_o$ | 16 | 32 | 32 | 32 | 64 | 64 | 64 | 64 | 128 | 128 | 128 | 256 | 256 | 256 | 512 | 512 | 512 | | |
| $w_i$ | 32 | 32 | 32 | 16 | 16 | 16 | 8 | 56 | 56 | 56 | 28 | 28 | 28 | 14 | 14 | 14 | 7 | | |
| $w_o$ | 32 | 16 | 16 | 16 | 8 | 8 | 8 | 56 | 28 | 28 | 28 | 14 | 14 | 14 | 7 | 7 | 7 | | |
| $f$ | 3 | 3 | 1 | 3 | 3 | 1 | 3 | 3 | 3 | 1 | 3 | 3 | 1 | 3 | 3 | 1 | 3 | | |
| $s$ | 1 | 2 | 2 | 1 | 2 | 2 | 1 | 1 | 2 | 2 | 1 | 2 | 2 | 1 | 2 | 2 | 1 | | |

path and the main CAConv-RAConv path, bootstrapping should be positioned either before the divergence of residual connections or after their convergence. Lastly, it is beneficial to perform convolutional layers at the lowest level possible because the complexity of FHE operations such as rotation, is proportional to the ciphertext level $l$.

Putting everything together, our ResNet basic block consumes a total of 6 levels. The level consumption of each layer is represented in the parentheses of each block. CAConv and RAConv use 2D gap packing and consume one level for each of SISO and IR. For convolution with $m > 1$ need level consumption for multiplication with mask. We adopted AESPA for activation which consumes one level. AESPA is a quadratic polynomial with different coefficients for each channel. During inference, we fuse the coefficients into nearby layers, resulting in a simple square function $x^2$ for activation. We set the ciphertext level after bootstrapping ($L'$ in Section II-A) to six and perform bootstrapping every time when the level becomes zero.

## IV. EVALUATION

### A. EXPERIMENTAL SETUP
We conducted HCNN inference in both CPU and GPU environments using the RNS-CKKS library, HEaaN [25]. The CPU system is equipped with two AMD EPYC 7452 CPUs running at 2.35GHz (32 cores per socket) and 480GB of DRAM. GPU experiments were carried out on the same system with an additional NVIDIA A100 GPU with 80GB of memory. In our HCNN inference experiments, we used ResNet-20/32/44 for CIFAR-10 [26] and ResNet-18 for ImageNet [27] datasets. The training was conducted with AESPA on PyTorch under normal a supervised setting following the original paper. We utilized the Kaiming-normal initialization method to initialize convolution and fully connected layers. After the training process, we applied a fusion technique to combine the weights and biases of batch normalization (BN) and the coefficients of AESPA with convolution layers. Our RNS-CKKS parameters satisfy a 128-bit security level [28] with a polynomial degree $N = 2^{16}$ and a hamming weight of 192. Table 4 displays the parameters used in the convolution layers of ResNet-20/32/44/18. All the parameters ($c_i, c_o, w_i, w_o, f, s$) are determined following the original



**FIGURE 11.** HCNN execution time of ResNet-18 (ImageNet) on our CPU and GPU systems.

**TABLE 5.** Memory footprint of ResNet-18 on ImageNet. Reference [11] is an extended implementation to ImageNet. Baseline is [11] augmented with AESPA, where HyPHEN improves it with PRCR.

| Implementation | Filter plaintexts | Evaluation keys |
|---|---|---|
| [11] | 285.1GB | 69.7GB |
| Baseline | 364.8GB | 11.1GB |
| **HyPHEN** | 45.6GB | 11.1GB |

ResNet paper [29]. Execution time measurement begins once all weight plaintexts and input ciphertexts are loaded into either host or device memory. It ends once the encrypted inference result is returned. The client's decrypt and encrypt processes are excluded as they are not considered critical bottlenecks in HCNN inference.

### B. IMPACT OF HYPHEN'S COMPONENTS ON PERFORMANCE
To analyze the effectiveness of each component of HyPHEN, we gradually apply alternation between CAConv and RAConv with reordering (+ Reorder), 2D gap packing (+ 2P), and PRCR (HyPHEN). We tested with the CPU/GPU implementations of ResNet-18 for ImageNet. The results are shown in Figure 11. By Alternating between CAConv and RAConv with Reorder, we achieved a 1.29× speedup and an additional 1.75× speedup due to 2D gap packing in the CPU. The benefits of both Reorder and 2D gap packing also extend to GPU, resulting in a 2.05× speedup of computation

**TABLE 6.** HyPHEN inference time of a single CIFAR-10 image using ResNet20/32/44 and a single ImageNet image using ResNet-18 on CPU and GPU. As FC and pooling layers have a tiny execution time, we gather them at Others. For ResNet-18, we implement the initial downsampling convolution by im2col, which is also summed in Others.

| Execution time (s) | CPU (64 threads) | | | | GPU | | | |
|---|---|---|---|---|---|---|---|---|
| | ResNet-20 | ResNet-32 | ResNet-44 | ResNet-18[1] | ResNet-20 | ResNet-32 | ResNet-44 | ResNet-18[2] |
| **CAConv** | 12.85 | 19.39 | 24.42 | 148.33 | 0.46 | 0.66 | 0.87 | 6.22 |
| **RAConv** | 1.68 | 3.02 | 4.01 | 37.16 | 0.06 | 0.09 | 0.13 | 1.37 |
| **Bootstrap** | 21.40 | 34.23 | 45.90 | 140.57 | 0.83 | 1.32 | 1.83 | 5.32 |
| **Activation** | 1.41 | 2.41 | 3.27 | 9.65 | 0.05 | 0.09 | 0.12 | 0.36 |
| **Others** | 0.23 | 0.30 | 0.44 | 37.70 | <0.01 | <0.01 | 0.01 | 1.42 |
| **Total** | $37.57 \pm 0.7$ | $59.35 \pm 0.9$ | $78.04 \pm 0.8$ | $373.41 \pm 3.9$ | $1.40 \pm 0.04$ | $2.17 \pm 0.04$ | $2.96 \pm 0.04$ | $14.69 \pm 0.01$ |

[1,2] ResNet-18 experiments are conducted on the ImageNet dataset.

**TABLE 7.** Comparison of the classification accuracy between unencrypted execution (Backbone) and execution employing our HCNN implementation (HyPHEN) of the ResNet models. While ResNet-20/32/44 results are accuracy of CIFAR-10, ResNet-18 is the accuracy of ImageNet classification.

| Top-1 Acc (%) | ResNet-20 | ResNet-32 | ResNet-44 | ResNet-18 |
|---|---|---|---|---|
| **Backbone** | 92.18 | 93.36 | 94.04 | 65.25 |
| **HyPHEN** | 92.17 | 93.35 | 94.08 | 65.25 |

time, excluding memory access latency. Meanwhile, PRCR results in 1.89× latency reduction in GPU, but leads to a slight slowdown (4.6%) in CPU due to the added computation of plaintext permutation. Although PRCR reduces the aggregate size of weight plaintexts by $s_i = 8$ times, it does not lead to latency reduction for our CPU system since its memory capacity can accommodate the whole working set (see Table 5). However, for the GPU system, PRCR enables the entire working set to fit in the 80GB GPU memory, reducing latency by eliminating the runtime overhead of copying data from the host CPU to the GPU (Memcpy). In summary, PRCR offers the benefits of substantial memory footprint reduction and latency reduction, especially for memory-constrained cases.

## C. EXECUTION TIME BREAKDOWN

Table 6 displays the runtimes of ResNet-20/32/44 for the inference of a single CIFAR-10 image and ResNet-18 for a single ImageNet image. Our ResNet-20/32/44 implementations on GPU take a few seconds to complete. While the majority of inference time for ResNet20/32/44 is spent on bootstrapping, for ResNet-18, 49.6% (CPU) and 51.7% (GPU) of inference time is spent on convolution. This is because ResNet-18 has four times more channels than ResNet-20/32/44. Table 6 also demonstrates that our RAConv, which replaces about half of the convolution layers effectively reduces the overall runtime of the convolutional layers.

## D. ACCURACY

In Table 7, we measured the classification accuracies of the validation set for CIFAR-10 running ResNet models

with HyPHEN. Near-zero accuracy degradation ($\le 0.01\%$) is observed for ResNet-20/32/44. HyPHEN proves to be more robust to accuracy degradation than [11], which exhibits 0.09% to 0.21% accuracy degradation for ResNet-20/32/44 on CIFAR-10. The difference in accuracy drop can be explained by whether the original network is executed as is (using AESPA) or an approximation has been made (using ReLU approximation). HyPHEN shows the same accuracy as the backbone accuracy even for a wider network (ResNet-18) and a larger dataset (ImageNet).

## E. PARAMETER STUDY

We conducted a parameter study to determine the optimal 2D gap packing setting for ResNet models to minimize latency. The choice of $(m, d)$ is crucial as it determines the number of rotations and bootstrappings to run the networks and thus execution time. Table 8 shows the representative $(m, d)$ instances, along with the resulting operation counts and execution times. We only present the $(m, d)$ of CAConv for simplicity, as $m$ and $d$ are exchanged at RAConv, and we omit ResNet-32/44 as ResNet-20/32/44 share the same optimal points.

In ResNet-20, [11] uses input repetition because the size of the input tensor in the first layer ($32 \times 32 \times 16$) is smaller than the ciphertext slots ($2^{15}$). To avoid input repetition, we start with $(m, d) = (1, 2)$ in ResNet-20, instead of using input repetition. In ResNet-18, we start with the default $(m, d) = (1, 1)$. When input repetition occurs, it is more efficient to increase $d$ instead, but further increasing $m$ or $d$ does not yield better performance because it leads to more bootstrapping, as shown in our proposed architecture (see Figure 10). As the input ciphertexts go through the downsampling layer, $m \cdot d$ gets quadrupled and the size of the intermediate tensor gets halved. 2D gap packing, which doubles $m$ and $d$ every downsampling layer demonstrates optimal performance, corresponding to the minimal bootstrapping setting. In ResNet-18, the minimal bootstrapping setting shows better performance than MP-CAConv. However, thanks to the flexibility of 2D gap packing we can find more efficient settings. The optimal setting requires 27 more bootstrappings and 20641 fewer rotations

**TABLE 8.** Runtime for the ResNet instances with different $(m, d)$ parameters and packing strategies. † We decompose rotation indices of unloaded evks into loaded evks. Effective rotations, the number of rotations after decomposition are represented as eff.total.

| Model | Packing | (m,d) | | | | Conv Rotations | | | | eff. total† | Boot | CPU Runtime(s) |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | L1 | L2 | L3 | L4 | SISO | RaS | IR | total | | | |
| ResNet-20 | Baseline | | | | | 152 | 924 | 800 | 1876 | 3638 | 10 | 68.58 ± 1.0 |
| | Optimal | (1,2) | (2,4) | (4,8) | - | 152 | 580 | 187 | 919 | 1002 | 10 | **37.57 ± 0.7** |
| | Min Rot | (1,2) | (1,8) | (2,16) | - | 240 | 407 | 142 | 789 | 881 | 15 | 44.51 ± 0.5 |
| ResNet-18 | Baseline | | | | | 536 | 32384 | 4669 | 37589 | 43672 | 38 | 802.08 ± 7.6 |
| | Min Boot | (1,1) | (4,1) | (16,1) | (64,1) | 536 | 17920 | 9544 | 28000 | 30072 | 38 | 623.11 ± 12.3 |
| | Optimal | (1,1) | (2,2) | (4,4) | (8,8) | 1024 | 4512 | 1823 | 7359 | 9095 | 65 | **356.97 ± 5.9** |

than the one with the minimum bootstrapping. 2D gap packing method helps balance the amount of rotation and bootstrapping to derive better performance.
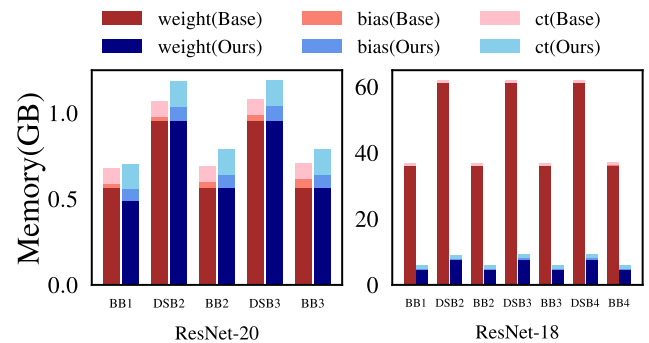
## F. TRAINING DETAILS

The Models used in this paper are all trained using PyTorch [30]. For ResNet-18 and 20, our training settings are mostly identical to AESPA; specifically, networks are trained for 200 epochs using an SGD optimizer. We also use soft labels as in [16] to achieve higher accuracy. For ResNet-32 and 44, we use knowledge distillation [31] to enhance the accuracy, using pre-trained ResNet-32/44 with 93.4% and 94.1% accuracies as teacher models. We adopt additional $l_2$ loss ($L_{kd} = \|f_t - f_s\|_2^2$) for distillation and trained for 240 epochs using the SGD optimizer.

## G. MEMORY FOOTPRINT ANALYSIS

Memory-capacity requirement for HCNN depends on FHE parameters and data representations, such as packing schemes. In FHE, data size expands during encoding and encryption procedures. The resulting plaintexts and ciphertexts are typically orders of magnitude larger than the original messages. Table 9 presents the actual size of ciphertexts, plaintexts, and evaluation keys on two FHE parameter settings. We denote the RNS-decomposition number introduced in [18] as Dnum. Given N, the degree of a cyclotomic polynomial ring, a large dnum increases L, the maximum level of a ciphertext. We assume $\mathtt{Set}_{lc}$ is a setting used in [11]. As [11] approximates ReLU with a high degree polynomial for activation, $\mathtt{Set}_{lc}$ adopts the maximum dnum to have $L' = 16$. $\mathtt{Set}_{hyp}$ is the parameter set used in this work. Adopting AESPA allows us to select a smaller RNS-CKKS parameter ($L' = 6$), as activation consumes one level. Certain FHE operations, such as MulCt, Rotate, and Conjugate, require the key-switching procedure. Evk denotes the public evaluation key used during this process. The size of a single Evk is 1,056MB and 176MB in $\mathtt{Set}_{lc}$ and $\mathtt{Set}_{hyp}$, respectively. For bootstrapping, one relinearization key for MulCt, one conjugation key, and 48 rotation keys are required. Frequently used rotation keys for Slide are loaded for convolution. For instance, loading 66 unique Evks in ResNet-18 takes up 69.7GB and 11.1GB in $\mathtt{Set}_{lc}$ and $\mathtt{Set}_{hyp}$, respectively. Other irregular rotation keys used in IR

**TABLE 9.** FHE parameter settings. dnum is tuned to support 16, 6 levels required in $\mathtt{Set}_{lc}$, and $\mathtt{Set}_{hyp}$. Each ciphertext and plaintext memory size is represented when bootstrap refresh the level.

| | L + 1 | L' | Dnum | Ctxt (MB) | Ptxt (MB) | Evk (MB) |
|---|---|---|---|---|---|---|
| $\mathtt{Set}_{lc}$ | 32 | 16 | 32 | 17 | 8.5 | 1056 |
| $\mathtt{Set}_{hyp}$ | 24 | 6 | 6 | 10 | 5 | 168 |



**FIGURE 12.** Total memory size (GB) of each object in residual blocks. We abbreviated Down-sampling Block and Basic Block to DSB and BB with each layer number as a postfix number. Weight plaintexts, bias plaintexts, and input ciphertexts configure the memory footprint of each residual block. Base and Ours represent the baseline and HyPHEN, respectively.

are not loaded; instead, these rotation indices are synthesized using the already loaded key indices.

Once FHE parameters are determined, the packing scheme determines the number of ciphertexts and plaintexts required to run each ResNet block. In the SISO-based HCNN filter, the size of filter plaintexts increases by a factor of $w_i h_i$ as each filter element is duplicated to the size of an input image. The number of slots for weight plaintexts is $w_i h_i f^2 c_i c_o$. Thus, weight plaintexts significantly outweigh ciphertexts in terms of memory footprint, which only requires $w_i h_i c_i d$ slots. Figure 12 illustrates the total memory capacity of ciphertexts and plaintexts of each CNN residual block. In the case of ResNet-20, our implementation shows up to a 14.75% memory-capacity overhead compared to the baseline. This overhead is due to the increase in the number of intermediate ciphertexts and bias plaintexts when using CAConv and RAConv, as HyPHEN does not apply PRCR for CIFAR-10. However, for ResNet-18, our implementation achieves 6.15-6.81× memory reduction compared to the baseline.

## V. RELATED WORK

### A. HE-BASED PRIVACY PRESERVING MACHINE LEARNING

Several variants of homomorphic encryption (HE) have been explored for Privacy-Preserving Machine Learning (PPML). Early literatures [8], [9], [10], [12], and [32] employed Leveled HE (LHE), which lacks support managing ciphertext errors and can only perform a limited number of operations, restricting its use to shallow networks. Convolutions in these prior works can still be adopted for FHE but are less efficient than convolutions specifically designed for FHE. For example, [12] devised a general tensor framework based on tiling, which can serve as an alternative to a SISO-based framework for specific HE parameter settings and image sizes. We observe that tiling incurs excessive bootstrapping for FHE-based CNNs in general, so we mainly focus on prior FHE CNN implementations for comparison. By contrast, SHE [33] implements CNN based on a TFHE scheme [34] and evaluates non-linear functions through table lookups, but it requires a long latency per operation. Using RNS-CKKS, [11], [15] have demonstrated ResNet implementation on CIFAR-10, employing high-degree polynomial approximation of ReLU. Reference [11] reported a single thread implementation of ResNet-20, which took 2271 seconds by efficiently utilizing ciphertext slots. A concurrent work [35] utilizes coefficient-packed ciphertexts that deploy values as the coefficient of a ring polynomial. This method reduces the number of rotations in convolution at the cost of limiting flexibility, as a single convolution must be paired with a single bootstrapping.

### B. HYBRID PPML

To address the high computational complexity in PPML, the HE-MPC hybrid PI protocol has gained attention as an alternative solution. In this protocol, client-aided MPC handles non-linear functions such as ReLU while HE operations compute linear functions. HE-MPC protocols [14], [36], [37], [38] have made significant progress. Cheetah [38] introduced an efficient packing scheme that removed rotations. However, it is challenging to make a fair comparison between HCNN and HE-MPC protocols due to differences in their security models. HE-MPC protocols also reveal network architecture. The hybrid approach places less computation burden on the server but assumes 1) continuous network communication and 2) comparable client-side computation power for optimal performance. In contrast, FHE only requires succinct communication for transmitting and receiving small input and output ciphertexts. Contrary to the common belief that HE-MPC protocols are significantly faster than FHE-based implementations, our results show comparable performance to prior HE-MPC protocols.

### C. FHE HARDWARE ACCELERATION

FHE-based applications are promising especially when combined with hardware acceleration. In response to the IT industry's need for privacy-preserving services with realistic quality of service (QoS), prior studies [39], [40] have introduced and analyzed the characteristics of FHE operations from a computer architectural perspective. This sufficient analysis has led to well-suited solutions for various hardware platforms such as CPU [41], GPU [42], [43], FPGA [44], [45], and ASIC [46], [47], [48]. While our HCNN implementation demonstrates performance on CPU and GPU platforms, specialized FHE accelerators can achieve 2–3 orders of magnitude performance improvements.

## VI. DISCUSSION

In our implementation, HyPHEN, we introduced several significant enhancements to boost the performance of HCNN. First, we incorporated a low-degree polynomial activation function obtained from AESPA. Additionally, we developed HCNN based on the GPU implementation of RNS-CKKS. Most importantly, we proposed novel algorithms to tackle two key challenges of HCNN: computational complexity and memory footprint.

### A. COMPUTATIONAL COMPLEXITY

We conducted a comprehensive bottleneck analysis of previous HCNN implementations and discovered that a performance limitation arises from FHE rotations. We found out that the majority of the rotation is attributed to the summation of channels within a ciphertext (RaS) and the adjustment of data format between two convolution layers (IR). To address this challenge, we developed novel algorithms, RAConv and CAConv, which enable encrypted convolution without the need for IR and effectively reduce RaS rotations. We also proposed a hybrid packing method capable of efficiently managing gaps introduced by strided convolution while minimizing the required rotations. Our implementation, HyPHEN, demonstrated a substantial reduction in execution time, from tens of minutes to just a few seconds.

### B. MEMORY FOOTPRINT

In addition to the memory expansion resulting from encryption and encoding procedures in the FHE scheme, we identified another significant source of memory expansion: duplicated data in packing methods. Specifically, while plaintext has lower memory expansion compared to ciphertext, it suffers from additional memory expansion because each weight element occupies the same number of slots as the input image size. Our experiments on the ImageNet dataset highlighted that loading weight plaintexts from host memory can significantly hinder the performance in CNN models such as ResNet-18. The available High Bandwidth Memory (HBM) in accelerators may not be capable of accommodating the entire model weights. Our proposed solution, PRCR, addressed this problem by introducing a novel data arrangement that eliminates the need to duplicate each weight element to match the size of an input image. For larger models like ResNet-50, the memory reduction ratio achieved by PRCR becomes even more significant. The problem of memory footprint becomes particularly crucial

when utilizing specialized accelerators, which are reported to be one or two orders of magnitude faster than GPUs but still share the same HBM technology and, therefore, have the same capacity limitations.

## C. LIMITATION

While HyPHEN has made significant strides in advancing HCNNs, several limitations should be considered. Despite achieving a remarkable reduction in execution times, a few seconds for inference are still challenging for deployment in real-world scenarios. Further, as image classification models evolve to achieve higher accuracy, they require larger memory capacity. Advanced memory footprint reduction algorithms could further facilitate the broad adoption of PPML.

## VII. CONCLUSION

In this paper, we have proposed HyPHEN, an efficient private inference construction of FHE-based CNN (HCNN). Combining two convolution methods with reordering and 2D gap packing enables fast inference by significantly reducing the number of homomorphic rotations in convolution. Additionally, PRCR enables HyPHEN to reduce the memory footprint for high-resolution image classification tasks, which is especially beneficial for memory-constrained devices. Our experiments with HyPHEN on CPU systems show $1.83\times/2.15\times$ lower latency compared to the prior state-of-the-art algorithm baseline in ResNet-20/18. Using GPU acceleration, HyPHEN achieves 1.40s/2.17s/2.96s execution time for running ResNet-20/32/44 for CIFAR-10, and we also demonstrated HCNN inference of ResNet-18 for ImageNet in 14.69s for the first time. We have showcased the practicality of utilizing FHE as a solution to achieve private inference through HyPHEN, which exhibits reasonable execution time while enabling clients to benefit from succinct computation and communication processes.

## REFERENCES

[1] *Regulation (EU) 2016/679 of the European Parliament and of the Council*, Eur. Parliament Council Eur. Union, Brussels, Belgium, 2016.

[2] *Health Insurance Portability and Accountability Act of 1996*, U.S. Dept. Health Hum. Services, Washington, DC, USA, 1996.

[3] N. Kumar, M. Rathee, N. Chandran, D. Gupta, A. Rastogi, and R. Sharma, "CrypTFlow: Secure TensorFlow inference," in *Proc. IEEE Symp. Secur. Privacy (SP)*, San Francisco, CA, USA, May 2020, pp. 336–353.

[4] W. Bowditch, W. Abramson, W. J. Buchanan, N. Pitropakis, and A. J. Hall, "Privacy-preserving surveillance methods using homomorphic encryption," in *Proc. 6th Int. Conf. Inf. Syst. Secur. Privacy (ICISSP)*, Valletta, Malta, Feb. 2020, pp. 240–248.

[5] C. Gentry, "Fully homomorphic encryption using ideal lattices," in *Proc. 41st Annu. ACM Symp. Theory Comput. (STOC)*, Bethesda, MD, USA, May/Jun. 2009, pp. 169–178.

[6] A. C.-C. Yao, "Protocols for secure computations (extended abstract)," in *Proc. 23rd Annu. Symp. Found. Comput. Sci.*, Chicago, IL, USA, Nov. 1982, pp. 3–5.

[7] V. Costan and S. Devadas, "Intel SGX explained," Int. Assoc. Cryptologic Res. (IACR), Tech. Rep. 86, 2016.

[8] R. Gilad-Bachrach, N. Dowlin, K. Laine, K. E. Lauter, M. Naehrig, and J. Wernsing, "CryptoNets: Applying neural networks to encrypted data with high throughput and accuracy," in *Proc. 33rd Int. Conf. Mach. Learn. (ICML)*, New York, NY, USA, Jun. 2016, pp. 201–210.

[9] A. Brutzkus, R. Gilad-Bachrach, and O. Elisha, "Low latency privacy preserving inference," in *Proc. 36th Int. Conf. Mach. Learn. (ICML)*, Long Beach, CA, USA, Jun. 2019, pp. 9–15.

[10] R. Dathathri, B. Kostova, O. Saarikivi, W. Dai, K. Laine, and M. Musuvathi, "EVA: An encrypted vector arithmetic language and compiler for efficient homomorphic computation," in *Proc. 41st ACM SIGPLAN Conf. Program. Lang. Design Implement.*, London, U.K., Jun. 2020, pp. 546–561.

[11] E. Lee, J.-W. Lee, J. Lee, Y.-S. Kim, Y. Kim, J.-S. No, and W. Choi, "Low-complexity deep convolutional neural networks on fully homomorphic encryption using multiplexed parallel convolutions," in *Proc. Int. Conf. Mach. Learn. (ICML)*, Baltimore, MD, USA, Jul. 2022, pp. 17–23.

[12] E. Aharoni, A. Adir, M. Baruch, N. Drucker, G. Ezov, A. Farkash, L. Greenberg, R. Masalha, G. Moshkowich, D. Murik, H. Shaul, and O. Soceanu, "HeLayers: A tile tensors framework for large neural networks on encrypted data," 2020, *arXiv:2011.01805*.

[13] J. H. Cheon, A. Kim, M. Kim, and Y. S. Song, "Homomorphic encryption for arithmetic of approximate numbers," in *Proc. 23rd Int. Conf. Theory Appl. Cryptol. Inf. Secur.*, Hong Kong, Dec. 2017, pp. 409–437.

[14] C. Juvekar, V. Vaikuntanathan, and A. P. Chandrakasan, "GAZELLE: A low latency framework for secure neural network inference," in *Proc. 27th USENIX Secur. Symp. (USENIX Secur.)*, Baltimore, MD, USA, Aug. 2018, pp. 1651–1669.

[15] J.-W. Lee, H. Kang, Y. Lee, W. Choi, J. Eom, M. Deryabin, E. Lee, J. Lee, D. Yoo, Y.-S. Kim, and J.-S. No, "Privacy-preserving machine learning with fully homomorphic encryption for deep neural network," *IEEE Access*, vol. 10, pp. 30039–30054, 2022.

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

[17] J. H. Cheon, K. Han, A. Kim, M. Kim, and Y. Song, "A full RNS variant of approximate homomorphic encryption," in *Proc. 25th Int. Conf. Sel. Areas Cryptogr. (SAC)*, Calgary, AB, Canada, Aug. 2018, pp. 347–368.

[18] K. Han and D. Ki, "Better bootstrapping for approximate homomorphic encryption," Int. Assoc. Cryptologic Res. (IACR), Tech. Rep. 688, 2019.

[19] J. Lee, E. Lee, J.-W. Lee, Y. Kim, Y.-S. Kim, and J.-S. No, "Precise approximation of convolutional neural networks for homomorphically encrypted data," 2021, *arXiv:2105.10879*.

[20] T. Ishiyama, T. Suzuki, and H. Yamana, "Highly accurate CNN inference using approximate activation functions over homomorphic encryption," in *Proc. IEEE Int. Conf. Big Data (Big Data)*, Atlanta, GA, USA, Dec. 2020, pp. 3989–3995.

[21] H. Chabanne, A. de Wargny, J. Milgram, C. Morel, and E. Prouff, "Privacy-preserving classification on deep neural network," Int. Assoc. Cryptologic Res. (IACR), Tech. Rep. 35, 2017.

[22] S. Obla, X. Gong, A. Aloufi, P. Hu, and D. Takabi, "Effective activation functions for homomorphic evaluation of deep neural networks," *IEEE Access*, vol. 8, pp. 153098–153112, 2020.

[23] E. Hesamifard, H. Takabi, and M. Ghasemi, "Deep neural networks classification over encrypted data," in *Proc. 9th ACM Conf. Data Appl. Secur. Privacy*, Richardson, TX, USA, Mar. 2019, pp. 97–108.

[24] P. Thaine, S. Gorbunov, and G. Penn, "Efficient evaluation of activation functions over encrypted data," in *Proc. IEEE Secur. Privacy Workshops (SPW)*, San Francisco, CA, USA, May 2019, pp. 57–63.

[25] *HEaaN Library*, CryptoLab Inc., Seoul, South Korea, 2022.

[26] A. Krizhevsky and G. Hinton, "Learning multiple layers of features from tiny images," Univ. Toronto, Toronto, ON, Canada, Tech. Rep., 2009.

[27] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei, "ImageNet large scale visual recognition challenge," *Int. J. Comput. Vis.*, vol. 115, no. 3, pp. 211–252, Dec. 2015.

[28] J. H. Cheon, M. Hhan, S. Hong, and Y. Son, "A hybrid of dual and meet-in-the-middle attack on sparse and ternary secret LWE," *IEEE Access*, vol. 7, pp. 89497–89506, 2019.

[29] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Las Vegas, NV, USA, Jun. 2016, pp. 770–778.

[30] A. Paszke et al., "PyTorch: An imperative style, high-performance deep learning library," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 32, 2019, pp. 8024–8035.

[31] G. E. Hinton, O. Vinyals, and J. Dean, "Distilling the knowledge in a neural network," 2015, *arXiv:1503.02531*.

[32] Q. Lou and L. Jiang, "HEMET: A homomorphic-encryption-friendly privacy-preserving mobile neural network architecture," in *Proc. 38th Int. Conf. Mach. Learn. (ICML)*, Jul. 2021, pp. 18–24.

[33] Q. Lou and L. Jiang, "SHE: A fast and accurate deep neural network for encrypted data," in *Proc. Annu. Conf. Neural Inf. Process. Syst. (NIPS)*, Vancouver, BC, Canada, Dec. 2019, pp. 10035–10043.

[34] I. Chillotti, N. Gama, M. Georgieva, and M. Izabachène, "TFHE: Fast fully homomorphic encryption over the torus," *J. Cryptol.*, vol. 33, no. 1, pp. 34–91, Jan. 2020.

[35] D. Kim and C. Guyot, "Optimized privacy-preserving CNN inference with fully homomorphic encryption," *IEEE Trans. Inf. Forensics Security*, vol. 18, pp. 2175–2187, 2023.

[36] P. Mishra, R. Lehmkuhl, A. Srinivasan, W. Zheng, and R. A. Popa, "Delphi: A cryptographic inference service for neural networks," in *Proc. 29th USENIX Secur. Symp. (USENIX Secur.)*, Aug. 2020, pp. 2505–2522.

[37] D. Rathee, M. Rathee, N. Kumar, N. Chandran, D. Gupta, A. Rastogi, and R. Sharma, "CrypTFlow2: Practical 2-party secure inference," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur. (CCS)*, USA, Nov. 2020, pp. 325–342.

[38] Z. Huang, W.-J. Lu, C. Hong, and J. Ding, "Cheetah: Lean and fast secure two-party deep neural network inference," in *Proc. 31st USENIX Secur. Symp. (USENIX Security)*, Boston, MA, USA, K. R. B. Butler and K. Thomas, Eds., Aug. 2022, pp. 809–826.

[39] W. Jung, E. Lee, S. Kim, J. Kim, N. Kim, K. Lee, C. Min, J. H. Cheon, and J. H. Ahn, "Accelerating fully homomorphic encryption through architecture-centric analysis and optimization," *IEEE Access*, vol. 9, pp. 98772–98789, 2021.

[40] S. Kim, W. Jung, J. Park, and J. H. Ahn, "Accelerating number theoretic transformations for bootstrappable homomorphic encryption on GPUs," in *Proc. IEEE Int. Symp. Workload Characterization (IISWC)*, Beijing, China, Oct. 2020, pp. 264–275.

[41] F. Boemer, S. Kim, G. Seifu, F. D. M. de Souza, and V. Gopal, "Intel HEXL: Accelerating homomorphic encryption with Intel AVX512-IFMA52," in *Proc. 9th Workshop Encrypted Comput. Appl. Homomorphic Cryptogr. (WAHC)*, South Korea, Nov. 2021, pp. 57–62.

[42] W. Jung, S. Kim, J. H. Ahn, J. H. Cheon, and Y. Lee, "Over 100x faster bootstrapping in fully homomorphic encryption through memory-centric optimization with GPUs," *IACR Trans. Cryptograph. Hardw. Embedded Syst.*, vol. 4, pp. 114–148, Aug. 2021.

[43] S. Fan, Z. Wang, W. Xu, R. Hou, D. Meng, and M. Zhang, "TensorFHE: Achieving practical computation on encrypted data using GPGPU," in *Proc. IEEE Int. Symp. High-Perform. Comput. Archit. (HPCA)*, Montreal, QC, Canada, Feb./Mar. 2023, pp. 922–934.

[44] M. S. Riazi, K. Laine, B. Pelton, and W. Dai, "HEAX: An architecture for computing on encrypted data," in *Proc. Archit. Support Program. Lang. Oper. Syst. (ASPLOS)*, Lausanne, Switzerland, Mar. 2020, pp. 1295–1309.

[45] R. Agrawal, L. de Castro, G. Yang, C. Juvekar, R. Yazicigil, A. Chandrakasan, V. Vaikuntanathan, and A. Joshi, "FAB: An FPGA-based accelerator for bootstrappable fully homomorphic encryption," in *Proc. IEEE Int. Symp. High-Perform. Comput. Archit. (HPCA)*, Montreal, QC, Canada, Feb./Mar. 2023, pp. 882–895.

[46] S. Kim, J. Kim, M. J. Kim, W. Jung, J. Kim, M. Rhu, and J. H. Ahn, "BTS: An accelerator for bootstrappable fully homomorphic encryption," in *Proc. 49th Annu. Int. Symp. Comput. Archit. (ISCA)*, New York, NY, USA, Jun. 2022, pp. 711–725.

[47] J. Kim, G. Lee, S. Kim, G. Sohn, M. Rhu, J. Kim, and J. H. Ahn, "ARK: Fully homomorphic encryption accelerator with runtime data generation and inter-operation key reuse," in *Proc. 55th IEEE/ACM Int. Symp. Microarchitecture (MICRO)*, Oct. 2022, pp. 1237–1254.

[48] N. Samardzic, A. Feldmann, A. Krastev, N. Manohar, N. Genise, S. Devadas, K. Eldefrawy, C. Peikert, and D. Sánchez, "Crater-Lake: A hardware accelerator for efficient unbounded computation on encrypted data," in *Proc. 49th Annu. Int. Symp. Comput. Archit.*, New York, NY, USA, Jun. 2022, pp. 173–187.

**DONGHWAN KIM** (Graduate Student Member, IEEE) received the B.S. degree in electrical engineering from Seoul National University, in 2022, where he is currently pursuing the master's degree with the Interdisciplinary Program in Artificial Intelligence. His research interests include privacy-preserving machine learning and architecture-centric optimization for emerging applications.

**JAIYOUNG PARK** (Graduate Student Member, IEEE) received the B.S. degree in electrical and computer engineering from Seoul National University, in 2020, where he is currently pursuing the Ph.D. degree with the Graduate School of Convergence Science and Technology. His research interests include privacy-preserving machine learning and architecture-centric optimization for emerging applications.

**JONGMIN KIM** (Graduate Student Member, IEEE) received the B.S. degree in electrical and computer engineering from Seoul National University, in 2021, where he is currently pursuing the Ph.D. degree with the Interdisciplinary Program in Artificial Intelligence. His research interests include architecture-centric optimization and hardware accelerator design for emerging applications, especially cryptographic applications.

**SANGPYO KIM** (Graduate Student Member, IEEE) received the B.S. degree in naval architecture and ocean engineering from Seoul National University, in 2019, where he is currently pursuing the Ph.D. degree with the Graduate School of Convergence Science and Technology, with a focus on computer architecture for accelerating emerging applications.

**JUNG HO AHN** (Senior Member, IEEE) received the B.S. degree in electrical engineering from Seoul National University and the M.S. and Ph.D. degrees in electrical engineering from Stanford University, Stanford, CA, USA. He is currently the Dean of the Graduate School of Convergence Science and Technology, Seoul National University. His research interests include bridging the gap between the performance demand of emerging applications and the performance potential of modern and future massively parallel systems.

● ● ●