CHIP: Efficient Homomorphic Encryption-Based CNN Batch Inference Using Channel-Interleaved Packing with Small Rotation Key Set

1st Huan-Chih Wang
National Taiwan University
Taipei, Taiwan
whcjimmy@cmlab.csie.ntu.edu.tw

2nd Ja-Ling Wu
National Taiwan University
Taipei, Taiwan
wjl@cmlab.csie.ntu.edu.tw

***Abstract*—As privacy concerns rise, numerous laws require machine learning-based applications to comply with stringent privacy regulations. While Homomorphic Encryption (HE) allows computation directly on encrypted data, existing HE-based inference solutions suffer from significant computational and memory overhead for both single and multiple samples. Additionally, current methods require many rotation keys, which limits their practicality in a broader range of scenarios.**

To address these challenges, we propose channel-interleaved packing (CHIP) to embed three-dimensional (3-D) data into 2-D ciphertexts, enabling 3-D HE convolution to be performed as a 2-D HE convolution combined with channel aggregations via ciphertext rotations. To further improve the performance of CHIP-based convolution, we introduce an efficient 2-D convolution that halves the number of HE multiplications. For computationally intensive inference tasks, we employ partial-kernel and mini-batch strategies that iteratively process sliced kernels and subsets of samples, aggregating the results to produce the final output.

Experimental results demonstrate the superior efficiency of our method compared to the state-of-the-art HE-based approaches by Lee et al. (ICML'22) and Cheon et al. (IEEE TDSC'24) in both single-sample and multi-sample scenarios. Using ResNet18, VGG11, and VGG16 with a batch size of 64, our solution achieves speedups of up to 4.7×. When processing a single test sample, the speedup increases to 60×. Moreover, our method requires only 29 rotation keys for evaluation, which is at least 35% fewer than previous works, resulting in an overall memory reduction of up to 45%. Code is available at: https://github.com/whcjimmy/chip.

***Index Terms*—Batch Inference, Fast Convolution Algorithm, Homomorphic Encryption, Privacy-Preserving Machine Learning**

I. Introduction

THE ML as a Service (MLaaS) paradigm is increasingly adopted in industry, enabling training and deployment of ML models on third-party cloud platforms [2]. Among ML models, CNNs excel at extracting spatial features and patterns [3] and are widely applied in image recognition and computer vision [4]. However, privacy concerns and regulations, such as GDPR, present significant challenges to MLaaS, as data transmission to cloud services raises privacy issues for data owners [5]. This necessitates novel approaches to safeguard privacy while maintaining MLaaS utility.

The emergence of privacy-preserving ML (PPML) focuses on creating secure, efficient ML applications balancing data use and privacy [6]. Many PPML techniques employ homomorphic encryption (HE), which enables arithmetic operations on ciphertexts [7], providing end-to-end security during computation and communication for private inference on encrypted data [8], [9]. Reference [10] classifies HE-based approaches into two categories: single-sample inference and batch inference. Single-sample inference methods [11]–[16] minimize end-to-end inference time for individual inputs, whereas batch inference techniques [17]–[22] support concurrent processing of multiple inputs for efficient batch handling.

Unlike single-sample inference, where runtime scales linearly with sample count, batch inference has gained traction due to its high throughput and flexible batch size selection based on scenario and resources [23]–[29]. Though batch inference incurs higher runtime per execution, it substantially lowers per-sample cost via workload amortization, making it popular for accelerating compute-intensive tasks in fields like e-commerce and healthcare [30], [31].

Among HE-based approaches, MPCNN [21] and CBC [22] represent state-of-the-art (SoTA) solutions providing batch inference but with distinct limitations. MPCNN [21] underutilizes ciphertext slots in HE operations, leading to suboptimal performance. Although it supports batch inference via independent threads, CBC [22] notes this is inefficient for runtime amortization. In CBC [22], it scales ciphertext count with channel number, causing high memory and computation overhead for wide-kernel models, and its performance is insensitive to batch size, resulting in high latency even with small batches.

Besides the limitations mentioned above, both SoTA methods require numerous rotation keys for evaluation. Reference [32] highlights the issue of excessive rotation keys in HE-based inference scenarios (cf. §II-E), including: (1) increased computational demands on the client to generate keys; (2) higher communication costs between the client and server; and (3) substantial memory consumption on the server to store

Please cite the accepted version of this paper published at IEEE CSF'26 [1].

rotation key sets from multiple clients for model inference. These inherent SoTA limitations, coupled with high rotation key demands, restrict practicality and scalability.

Contributions. We propose *channel-interleaved packing* (CHIP) along with a suite of techniques to improve the efficiency of HE-based private inference across various batch sizes. Moreover, our approach significantly reduces the number of rotation keys required. Our main contributions are summarized as follows:

- **Novel Packing Technique:** CHIP embeds the channel values of data elements into ciphertext slots, enabling the encryption of multi-dimensional data—both test samples and model weights—within a two-dimensional (2-D) ciphertext array. We present two representations for organizing channel values, enabling seamless transformation between them via CHIP-based convolution.
- **Rotation Key Reduced Convolution Algorithm:** The CHIP-based convolution is implemented as a 2-D HE convolution combined with channel aggregation via ciphertext rotations. The number of rotation keys required depends solely on the number of distinct channel sizes, and it is significantly reduced compared to SoTA approaches, whose key sizes scale with the number of different data shapes.
- **Multiplication-Efficient 2-D Convolution:** We develop an efficient HE-based 2-D convolution algorithm based on the Fast Convolution Algorithm [33], requiring only half the number of multiplications compared to the HE-based Direct Convolution Algorithm [20], yielding up to a 3× speedup.
- **Circular Evaluations for Computationally Intensive Inference:** To effectively manage ciphertext slots when evaluating wide models and large batches of samples, we propose a *partial-kernel* and a *mini-batch* strategy that iteratively evaluates only partial kernels and subsets of samples. The final output of the original task is obtained by aggregating the intermediate results from these iterative evaluations.

Evaluations. The experimental results can be distilled into the following main points:

- **Enhanced Runtime Performance:** Our solution demonstrates superior performance across various batch sizes, with more significant improvements observed at smaller batch sizes. On ResNet18 with CIFAR10 and a batch size of 64, our method achieves speedups of 1.1× and 2.9× compared to CBC [22] and MPCNN [21], respectively. On VGG11 with CIFAR10 and VGG16 with CIFAR100, the method achieves speedups of up to 2.6× and 4.7× over the baselines, respectively. At a batch size of one, performance improvements increase to 4.4× on ResNet18, 60× on VGG11, and 27× on VGG16 compared to both baselines.
- **Output and Accuracy Equivalence:** The CHIP-based convolution produces identical results to SoTA methods. By employing the same ReLU approximation, all methods achieve equivalent accuracy—91.9% for ResNet18, 89.6% for VGG11, and 67.7% for VGG16—comparable to cleartext model performance.
- **Rotation Key Reduction:** The number of required rotation keys for both models decreases from 258 in MPCNN [21] and 45 in CBC [22] to only 29.
- **Improved Memory Efficiency:** Combined with the rotation key reduction, our method reduces memory usage by 37% and 45% compared to MPCNN [21] and CBC [22], respectively.

II. Preliminaries

This section offers foundational background knowledge pertinent to this paper, with relevant notations enumerated in Table I. We outline the CNN components in §II-A, mention the Fast Convolution Algorithm in §II-B, explain the Kronecker product in §II-C, and introduce the CKKS cryptosystem in §II-D. Finally, we present the HE-based PPML inference workflow and our threat model in §II-E.

TABLE I: A summary of notations.

Notation	Description
x, $x^{(i)}$	Scalar and the i-th sample of scalar x
$\mathbf{x}$, $\mathbf{x}_i$	Vector and the i-th element of vector $\mathbf{x}$
$\mathbf{X}$, $\mathbf{X}_i$, $\mathbf{X}_{i:i+a,j:j+b}$	Multi-dimensional tensor, elements of $\mathbf{X}$ where the first dimension is i, and elements where the first dimension ranges from i to $i+a-1$ and the second dimension ranges from j to $j+b-1$
N, Q, S	Lattice dimension, ciphertext modulus, and number of slots
B	Number of batch samples that can be packed in a CHIP-based private inference
T	Number of ciphertext slots required per sample in a CHIP-based convolution
P, M	Number of iterations of the loops for the partial-kernel and mini-batch strategies
$\hat{x}$, $[x]$	Plaintext scalar and ciphertext scalar
$\boxplus$, $\boxtimes$	HE addition and multiplication
$\odot$, $\boxdot$	Real-number and ciphertext element-wise multiplications

A. CNN

CNN is a class of ML models characterized by capturing spatial hierarchies and patterns through convolutional layers. The key components are outlined below.

Convolutional Layer. The Direct Convolution Algorithm (DCA) is a widely used method for performing convolution operations in CNNs. Let $\mathbf{X} \in \mathbb{R}^{C^i \times W \times H}$ is a 3-D input, $\mathbf{K} \in \mathbb{R}^{C^o \times C^i \times r \times r}$ is a 4-D filter, and $\mathbf{b} \in \mathbb{R}^{C^o}$ is a bias vector. Assuming a stride length of l, the output is denoted as $\mathbf{Y} \in \mathbb{R}^{C^o \times W^o \times H^o}$, where $W^o = \lceil (W-r)/l \rceil$ and $H^o = \lceil (H-r)/l \rceil$. The DCA for the k-th channel of $\mathbf{Y}$ is computed as:

$$\mathbf{Y}_{k,i,j} = \sum_{c \in C^i} \sum_{a \in r} \sum_{b \in r} \mathbf{X}_{c,il+a,jl+b} \times \mathbf{K}_{k,c,a,b} + \mathbf{b}_k. \quad (1)$$

Dense Layer. In a dense layer, each neuron associated with the vector $\mathbf{y}$ is connected to each neuron in its previous layer $\mathbf{x}$ with a weight matrix $\mathbf{W}$ and a bias vector $\mathbf{b}$. That is,

$$\mathbf{y} = \mathbf{x}\mathbf{W} + \mathbf{b}.$$

ReLU Function. ReLU, defined as $ReLU(x) = \max(0, x)$, is a non-linear activation function commonly applied after linear layers, such as convolutional or dense layers, to retain only non-negative outputs.

Batch Normalization Function. Assume the mean and variance of a vector $\mathbf{x}$ are μ and σ, respectively, the batch normalization updates each element by:

$$\mathbf{x}_i^{'} = \frac{\gamma(\mathbf{x}_i - \mu)}{\sqrt{\sigma^2 + \epsilon}} + \beta,$$

where γ and β are two parameters to scale and shift the upcoming results, respectively, and ϵ is a number close to zero.

Mean-pooling Function. Given a matrix $\mathbf{X} \in \mathbb{R}^{W \times H}$, the output of the $m \times m$ mean-pooling function is $\mathbf{X}^{'} \in \mathbb{R}^{W/m \times H/m}$, where

$$\mathbf{X}_{i,j}^{'} = \frac{1}{m^2} \sum_{a=0}^{m-1} \sum_{b=0}^{m-1} \mathbf{X}_{im+a, jm+b}.$$

B. Fast Convolution Algorithm

The Fast Convolution Algorithm (FCA) [33], derived from Winograd's Minimal Filtering Algorithm [34], reduces the number of multiplications required while producing the same convolutional output as the DCA, thereby ensuring both computational efficiency and correctness.

The FCA requires two parameters: the output pile size o and the kernel size r. If we use an r-tap filter to produce o outputs, the minimum input size is $o + r - 1$, and there is a total of $o + r - 1$ multiplications. The 1-D FCA is denoted as F_r^o. Let us examine F_3^2 in detail. Assume the targeting data $\mathbf{h} = [\mathbf{h}_0, \mathbf{h}_1, \mathbf{h}_2, \mathbf{h}_3]$ and the kernel $\mathbf{f} = [\mathbf{f}_0, \mathbf{f}_1, \mathbf{f}_2]$, F_3^2 FCA works as follows:

$$F_3^2(\mathbf{h}, \mathbf{f}) = \begin{bmatrix} \mathbf{h}_0 & \mathbf{h}_1 & \mathbf{h}_2 \\ \mathbf{h}_1 & \mathbf{h}_2 & \mathbf{h}_3 \end{bmatrix} \cdot \begin{bmatrix} \mathbf{f}_0 \\ \mathbf{f}_1 \\ \mathbf{f}_2 \end{bmatrix} = \begin{bmatrix} \mathbf{m}_1 + \mathbf{m}_2 + \mathbf{m}_3 \\ \mathbf{m}_2 - \mathbf{m}_3 - \mathbf{m}_4 \end{bmatrix}, \tag{2}$$

where $\mathbf{m}_1 = (\mathbf{h}_0 - \mathbf{h}_2)\mathbf{f}_0$, $\mathbf{m}_2 = (\mathbf{h}_1 + \mathbf{h}_2)(\mathbf{f}_0 + \mathbf{f}_1 + \mathbf{f}_2)/2$, $\mathbf{m}_3 = (\mathbf{h}_1 - \mathbf{h}_3)\mathbf{f}_2$, and $\mathbf{m}_4 = (\mathbf{h}_2 - \mathbf{h}_1)(\mathbf{f}_0 - \mathbf{f}_1 + \mathbf{f}_2)/2$.

As shown in (2), F_3^2 FCA uses four multiplications for every $\mathbf{m}_i$, whereas the DCA uses six multiplications: $\mathbf{h}_0\mathbf{f}_0$, $\mathbf{h}_1\mathbf{f}_1$, $\mathbf{h}_2\mathbf{f}_2$, $\mathbf{h}_1\mathbf{f}_0$, $\mathbf{h}_2\mathbf{f}_1$, and $\mathbf{h}_3\mathbf{f}_2$, respectively.

FCA can be extended to 2-D data, denoted as $F_{r\times r}^{o\times o}$. The computation process of the 2-D FCA can be formalized in matrix form as follows:

$$\mathbf{Y} = F_{r\times r}^{o\times o}(\mathbf{H}, \mathbf{F}) = \mathbf{A}^\top \big((\mathbf{G}\mathbf{F}\mathbf{G}^\top) \odot (\mathbf{B}^\top \mathbf{H} \mathbf{B})\big)\mathbf{A}, \tag{3}$$

where the targeting data $\mathbf{H} \in \mathbb{R}^{(o+r-1)\times(o+r-1)}$, the kernel $\mathbf{F} \in \mathbb{R}^{r\times r}$ and the output $\mathbf{Y} \in \mathbb{R}^{o\times o}$. The matrices $\mathbf{A}$, $\mathbf{B}$, and $\mathbf{G}$ are determined by the output pile of size o and the kernel of size r. For instance, when $o = 2$ and $r = 3$,

$$\mathbf{A}^\top = \begin{bmatrix} 1 & 1 & 1 & 0 \\ 0 & 1 & -1 & -1 \end{bmatrix}, \mathbf{B}^\top = \begin{bmatrix} 1 & 0 & -1 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & -1 & 1 & 0 \\ 0 & 1 & 0 & -1 \end{bmatrix}, \text{ and}$$

$$\mathbf{G} = \begin{bmatrix} 1 & 0 & 0 \\ 1/2 & 1/2 & 1/2 \\ 1/2 & -1/2 & 1/2 \\ 0 & 0 & 1 \end{bmatrix}.$$

The 2-D FCA requires $(o+r-1)^2$ multiplications, whereas the DCA requires o^2r^2 multiplications. Therefore, the $F_{3\times3}^{2\times2}$ FCA reduces the number of multiplications by 55%.

C. Kronecker Product

Given two matrices $\mathbf{U} \in \mathbb{R}^{a\times b}$ and $\mathbf{V} \in \mathbb{R}^{c\times d}$, the Kronecker product [35] of these two matrices is

$$\mathbf{U} \otimes \mathbf{V} = \begin{bmatrix} \mathbf{U}_{0,0}\mathbf{V} & \cdots & \mathbf{U}_{0,b-1}\mathbf{V} \\ \vdots & \ddots & \vdots \\ \mathbf{U}_{a-1,0}\mathbf{V} & \cdots & \mathbf{U}_{a-1,b-1}\mathbf{V} \end{bmatrix},$$

where $\mathbf{U} \otimes \mathbf{V} \in \mathbb{R}^{ac\times bd}$.

The "vector trick" is a useful property of the Kronecker product. Given a matrix $\mathbf{C} = \mathbf{UTV}$, we can compute the matrix $\mathbf{C}$ by multiplying the matrices $\mathbf{U}$, $\mathbf{T}$, and $\mathbf{V}$ in sequence, or we can use this trick by computing:

$$vec(\mathbf{C}) = vec(\mathbf{UTV}) = \mathbf{U} \otimes \mathbf{V} vec(\mathbf{T}), \tag{4}$$

where $vec(\cdot)$ is the vectorize function.

D. The CKKS HE Scheme

The CKKS cryptosystem [36] enables HE operations on floating-point numbers, making it particularly suitable for PPML applications. Below, we briefly present the key functions of the CKKS scheme.

- **Encode and Decode:** The encoding function, $Ecd(\mathbf{m})$, encodes $\mathbf{m} \in \mathbb{R}^N$ into a ring element $\hat{m} \in R_N$, which is also referred to as a plaintext. The decoding function, $Dcd(\hat{m})$, converts $\hat{m}$ back to $\mathbf{m}$.
- **Encryption and Decryption:** The encryption function encrypts a plaintext $\hat{m}$ into a ciphertext, using the public key pk and given parameters N and Q. Specifically, $[x] = Enc_{pk}^{N,Q}(\hat{m}) \in R_{N,Q}^2$. The decryption function recovers $[x]$ approximately with the secret key sk, such that $Dec_{sk}^{N,Q}([x]) \approx \hat{m}$.
- **Batching:** When N is a power of two, the ring $R_{N,Q}$ is isomorphic to $S = N/2$ copies of $\mathbb{Z}_Q$. Therefore, each ciphertext contains S slots for holding real numbers. HE operations on a ciphertext are thus applied independently to each slot. Moreover, since plaintext scalars are encoded within a cyclotomic ring, plaintext operations also benefit from this batching capability.
- **Homomorphic Addition:** Assume that $[x^{(1)}]$ and $[x^{(2)}]$ encrypt the plaintexts $\hat{m}^{(1)} = Ecd(\mathbf{m}^{(1)})$ and $\hat{m}^{(2)} =$

$Ecd(\mathbf{m}^{(2)})$, respectively, their HE addition can be obtained as:

$$Dec([x^{(1)}] \boxplus [x^{(2)}]) = Ecd(\mathbf{m}^{(1)} + \mathbf{m}^{(2)}).$$

- **Slot-wise Homomorphic Multiplication:** The slot-wise multiplication can be performed on two ciphertexts or on one ciphertext and one plaintext. In both cases, we have:

$$Dec([x^{(1)}] \boxtimes [x^{(2)}]) = Dec([x^{(1)}] \boxtimes \hat{m}^{(2)}) = Ecd(\mathbf{m}^{(1)} \odot \mathbf{m}^{(2)}).$$

  Note that both homomorphic addition and multiplication introduce noise into ciphertexts, with multiplication causing a greater noise increase. The noise accumulates with each operation, and if it surpasses a critical threshold, decryption may fail to yield correct results.
- **Leveled Ciphertext Modulus:** CKKS scheme constructs an L-level Q as a product of several small co-primes p_i, such that $Q_L = \prod_{i \in L} p_i$. After several HE additions or a multiplication, ciphertexts require rescaling, which lowers the level and correspondingly reduces noise, thereby controlling its growth during computation. Once the level reaches zero, the computationally intensive bootstrapping procedure [37] is required to restore the ciphertext level for continued computation.
- **Rotation Functions:** The left rotation function, denoted as $RotL_{rotk}([x], i)$, cyclically shifts a given vector $\mathbf{m}$ by i positions to the left with a rotation key $rotk$. This can be expressed as:

$$RotL_{rotk}([x], i) = Enc_{pk}^{N,Q}(Ecd(\mathbf{m} \ll i)),$$

  where $\ll$ represents the left-shifting function to the vector. Conversely, the right rotation function, $RotR_{rotk}([x], i)$, performs a cyclic shift of the ciphertext to the right. Because different rotation keys are generated depending on the different shift lengths, they are generated in advance, requiring memory spaces to store them.
- **Rotate-and-Sum Functions:** The rotate-and-sum-left function, $RSL([x], s, t)$, applies the $RotL$ function for t iterations. In each loop, the operation involves adding the original ciphertext $[x]$ to itself rotated $\alpha \times s$ steps to the left, where α is the counter starting from zero. In contrast, RSR is similar in structure but applies the $RotR$ function, decrementing α from $t-1$ to zero.

E. HE-based Private Inference Workflow & Threat Model

An HE-based inference workflow involves two parties, the client and the server. The client possesses encrypted data that needs to be inferred, while the server offers a PPML inference service utilizing a pre-trained model. The process begins with the client encrypting the data and sending it to the server. The server independently performs the private inference on encrypted data and returns the prediction result to the client. Finally, the client can decrypt the encrypted result and obtain the human-readable prediction outcomes.

In this work, we adopt the same threat model as prior studies [11]–[22] under HE. The server is assumed to be semi-honest: it follows the prescribed protocol but may attempt to infer sensitive information during inference. However, as the server operates solely on encrypted data without access to decryption, the confidentiality of the client's data is preserved. Due to the fact that our solution is entirely based on the CKKS scheme, its security fundamentally relies on the hardness of the ring learning with errors problem [38], which is believed to be secure against quantum attacks.

III. Related Work

This section reviews prior work on novel data packing techniques, categorizing them into solutions based solely on HE and hybrid methods that combine HE with secure multi-party computation (MPC) for private inference.

A. Pure HE-based Approaches

Early HE research, including CryptoNets [17], CryptoDL [18], Faster CryptoNets [19], and ngraph-he [20], utilized element-wise packing for batch inference and applied the DCA for convolutions. Each HE-based function was implemented by replacing plaintext operations with their HE equivalents. In contrast, CHET [11], LoLa [12], and Falcon [13] introduced novel packing methods and processed one sample at a time to improve end-to-end runtime. However, the aforementioned approaches suffer from high computational costs and rely on low-degree polynomial approximations of ReLU, which may reduce accuracy. Consequently, they are limited to small models and datasets.

[14]–[16] applied HE-based inference to standard CNN architectures, such as ResNet20 [39]. [14] enhanced Gazelle [40] by supporting strided convolution; [15] proposed a convolution method with performance independent of kernel size; and [16] reduced memory usage for high-resolution images. To mitigate accuracy loss, [14], [15] employed minimax polynomial approximations [41] of up to degree twenty-seven to emulate ReLU. On the contrary, [16] utilized AESPA [42] by replacing ReLU with a sequence of Hermite polynomials and, through model retraining, approximated it with a single quadratic polynomial without compromising accuracy. Despite efforts to support standard models, [14]–[16] can process only one sample at a time.

Subsequently, MPCNN [21] proposed multiplexed parallel convolution, a packing method embedding multiple channels into a single ciphertext, and enabled multi-sample parallelization via multithreading, yielding a 4.67× speedup over [14] on ResNet20. Nevertheless, MPCNN does not fully exploit ciphertext slot capacity, leading to inefficient latency amortization. Moreover, CBC [22] demonstrated its single-thread-per-sample parallelism is less efficient.

CBC [22] proposed a channel-by-channel packing scheme in which different channels are encrypted into separate ciphertexts, with each ciphertext embedding as many samples as possible. This approach parallelizes computations across channels, achieving a 5× performance improvement over MPCNN [21] on ResNet models [39]. Additionally, CBC [22] significantly reduces the required rotation keys, decreasing them from 258 in MPCNN [21] to 45. However, the ciphertext

size scales with the channel dimension of each CNN layer. As a result, evaluations on wide models incur significant increases in both computational overhead and memory consumption.

B. Hybrid Approaches

Due to the complexity of evaluating non-linear functions on HE ciphertexts, hybrid methods—including Gazelle [40], Delphi [43], CrypTFlow2 [44], and Cheetah [45]—combine HE with MPC [46] to enable efficient private inference. HE manage linear operations in this workflow, while MPC processes non-linear computations like ReLU. However, these solutions require multiple parties to remain online and incur communication overhead throughout the inference process, meaning that clients must be actively involved during inference. In contrast, pure HE-based solutions require client involvement only before and after inference, since servers perform all ciphertext computations independently.

IV. Proposed Methods

This section presents our proposed methods for efficient private model inference. We first describe the channel-interleaved packing (CHIP) in §IV-A and its associated convolution algorithm in §IV-B. Then, we introduce the HE-based FCA (HE-FCA) in §IV-C, which enhances the performance of CHIP-based convolution on 2-D data.

To enable CHIP packing for consecutive convolutions, we introduce the CHIP-based convolution block in §IV-D. For intensive inference tasks involving wide models or large batch sizes, we propose partial-kernel and mini-batch strategies in §IV-E to enhance CHIP-based convolution performance. Subsequently, additional essential functions are adapted for CHIP compatibility to support full CNN inference, as detailed in §IV-F. Finally, these methods are integrated into a unified evaluation architecture presented in §IV-G.

A. CHIP

CHIP embeds each data element and its corresponding channel values into a single ciphertext. When $C^i < S$, any input $\mathbf{X} \in \mathbb{R}^{C^i \times W \times H}$ can be encrypted as a 2-D $W \times H$ ciphertext array $[\mathbf{X}]$. CHIP utilizes two data representations—dense and sparse—to structure channel values, which can be interconverted via CHIP-based convolution, thereby preventing channel reorganization during inference.

As shown in (1), a convolution over C^i input channels that produces C^o output channels results in $T = C^i C^o$ intermediate convolved values. Leveraging this, CHIP organizes each data element along with its C^i channel values within a segment of T consecutive slots, as illustrated below:

- **Dense Representation:** The C^i channel values of each element are placed in the first C^i slots of its segment, while the remaining slots are padded with zeros. For illustration, the encrypted $[\mathbf{X}_{i,j}]$ can be represented as:

$$[\mathbf{X}_{i,j}] = Enc([\underbrace{\overbrace{\mathbf{X}_{0,i,j}, \mathbf{X}_{1,i,j}, \cdots, \mathbf{X}_{C^i-1,i,j}}^{\text{first } C^i \text{ slots}}, \cdots}_{\text{segment 0 } (T \text{ slots})}]). \quad (5)$$

- **Sparse Representation:** Every C^i channel values of an element is placed every C^o slots. That is:

$$[\mathbf{X}_{i,j}] = Enc([\underbrace{\overbrace{\mathbf{X}_{0,i,j}, \cdots}^{C^o \text{ slots}}, \overbrace{\mathbf{X}_{1,i,j}, \cdots}^{C^o \text{ slots}}, \cdots, \overbrace{\mathbf{X}_{C^i-1,i,j}, \cdots}^{C^o \text{ slots}}}_{\text{segment 0 } (T \text{ slots})}]). \quad (6)$$

B. CHIP-based Convolution

Convolution on either CHIP representation involves identical procedures: data duplication, 2-D HE convolution, and channel aggregation, with only minor variations in the function parameters applied.

1) Data Duplication: This step replicates C^i input channel values C^o times using the RSR function, where $t = \log_2 C^o$ applies to both representations. However, the parameter s differs: $s = C^i$ for the dense format and $s = 1$ for the sparse-style data. Therefore, the duplicated data can be expressed as:

- **Dense Representation:**

$$[\mathbf{X}_{i,j}] = Enc([\underbrace{\mathbf{D}_{0,i,j}, \cdots, \mathbf{D}_{C^i-1,i,j}}_{1^{\text{st}} \text{ copy } (C^i \text{ slots})}, \mathbf{D}_{0,i,j}, \cdots, \mathbf{D}_{C^i-1,i,j}, \cdots, \underbrace{\mathbf{D}_{0,i,j}, \cdots, \mathbf{D}_{C^i-1,i,j}}_{C^o\text{-th copy } (C^i \text{ slots})}]).$$

- **Sparse Representation:**

$$[\mathbf{X}_{i,j}] = Enc([\underbrace{\overbrace{\mathbf{D}_{0,i,j}}^{1^{\text{st}} \text{ copy}}, \overbrace{\mathbf{D}_{0,i,j}}^{2^{\text{st}} \text{ copy}}, \cdots, \overbrace{\mathbf{D}_{0,i,j}}^{C^o\text{-th copy}}}_{C^o \text{ slots}}, \cdots, \underbrace{\overbrace{\mathbf{D}_{C^i-1,i,j}}^{1^{\text{st}} \text{ copy}}, \overbrace{\mathbf{D}_{C^i-1,i,j}}^{2^{\text{st}} \text{ copy}}, \cdots, \overbrace{\mathbf{D}_{C^i-1,i,j}}^{C^o\text{-th copy}}}_{C^o \text{ slots}}]).$$

Accordingly, the 4-D kernel $\mathbf{K} \in \mathbb{R}^{C^o \times C^i \times r \times r}$ is reshaped into a 2-D $r \times r$ plaintext array $\hat{\mathbf{K}}$. Each kernel element contains $C^i C^o$ channel values, which are placed within T slots. Specifically, $\mathbf{K}_{j,i}$ is convolved with the j-th duplicated input channel i to produce the j-th output channel.

2) 2-D HE Convolution: A 2-D HE convolution is performed between the encrypted data $[\mathbf{X}]$ and the 2-D kernel array $\hat{\mathbf{K}}$. Unlike prior works [17]–[20] that employ the DCA with high HE multiplication costs, we adopt the FCA to improve efficiency, which will be detailed in §IV-C. Leveraging CKKS batching, the convolution between $[\mathbf{X}]$ and $\hat{\mathbf{K}}$ is executed slot-wise, enabling T convolutions to run in parallel.

3) Channel Aggregation: This step utilizes the RSL function to gather T intermediate results to produce the final convolution outcome with C^o channels. In both representations, the parameter $t = log_2 C^i$, while the parameter $s = 1$ for the dense representation and $s = C^o$ for the sparse representation. The convolved result for both representations can be expressed as:

- **Dense Representation:**

$$[\mathbf{Y}] = Enc(\underbrace{[\overbrace{\mathbf{Y}_0, \cdots}^{C^i \text{ slots}}, \overbrace{\mathbf{Y}_1, \cdots}^{C^i \text{ slots}}, \cdots, \overbrace{\mathbf{Y}_{C^o-1}, \cdots}^{C^i \text{ slots}}]}_{\text{segmenet } 0}). \quad (7)$$

- **Sparse Representation:**

$$[\mathbf{Y}] = Enc(\underbrace{[\overbrace{\mathbf{Y}_0, \mathbf{Y}_1, \cdots, \mathbf{Y}_{C^o-1}, \cdots}^{C^o \text{ output}}]}_{\text{segment } 0}). \quad (8)$$

As shown in (7) and (8), the convolution output of the dense representation is converted to the sparse format, and vice versa. Consequently, CHIP is suitable for consecutive convolutions, as no additional slot rearrangement is needed before subsequent operations.

To prevent non-output channel values from interfering with the following computations, a filter $\hat{f}$ is utilized to clean unnecessary values. It is a plaintext where its slot values having the same position as the output values are set to one. To conclude, Algorithm 1 details the CHIP-based convolution algorithm based on both representations.

Algorithm 1 CHIP-based convolution algorithm. (CHIP-CONV)

Require: An encrypted $W \times H$ matrix $[\mathbf{X}]$ and the $r \times r$ kernel $\hat{\mathbf{K}}$. Both the encrypted data and plaintext kernel employ a specified $type$ representation to embed information. The filter $\hat{f}$ is used to remove unnecessary values.
Ensure: An encrypted $W^o \times H^o$ convolved result $[\mathbf{Y}]$.
1: **for** $0 \le i < W$ and $0 \le j < H$ **do**
2: $s = (type == \text{“}dense\text{”})?C^i : 1$
3: $[\mathbf{X}_{i,j}] = RSR([\mathbf{X}_{i,j}], s, log_2 C^o)$ ▷ Data duplication
4: **end for**
5: $[\mathbf{Y}] = \text{HE-FCA}([\mathbf{X}], \hat{\mathbf{K}})$ ▷ 2-D HE Convolution
6: **for** $0 \le i < W^o$ and $0 \le j < H^o$ **do**
7: $s = (type == \text{“}dense\text{”})?1 : C^o$
8: $[\mathbf{Y}_{i,j}] = RSL([\mathbf{Y}_{i,j}], s, log_2 C^i)$ ▷ Channel aggregation
9: $[\mathbf{Y}_{i,j}] = [\mathbf{Y}_{i,j}] \boxtimes \hat{f}$ ▷ Remove unnecessary values
10: **end for**

C. HE-FCA

Although we want to leverage the multiplication efficiency of FCA to accelerate private inference, adapting FCA for private computation is challenging due to its high multiplicative depth and extensive HE operation demands. When data is encrypted, (3) can be reformulated as:

$$[\mathbf{Y}] = \mathbf{A}^\top \left((\mathbf{G}\hat{\mathbf{K}}\mathbf{G}^\top) \boxdot (\mathbf{B}^\top [\mathbf{X}] \mathbf{B}) \right) \mathbf{A}. \quad (9)$$

Naively, computing $[\mathbf{Y}]$ requires five HE multiplications, which are:

$$\begin{aligned} [\mathbf{T}^{(1)}] &= \mathbf{B}^\top [\mathbf{X}], \\ [\mathbf{T}^{(2)}] &= [\mathbf{T}^{(1)}] \mathbf{B}, \\ [\mathbf{T}^{(3)}] &= (\mathbf{G}\hat{\mathbf{K}}\mathbf{G}^\top) \boxdot [\mathbf{T}^{(2)}], \\ [\mathbf{T}^{(4)}] &= \mathbf{A}^\top [\mathbf{T}^{(3)}], \text{and} \\ [\mathbf{Y}] &= [\mathbf{T}^{(4)}] \mathbf{A}. \end{aligned}$$

Five consecutive HE multiplications imply a depth of five levels, resulting in inefficiency. To mitigate this, we utilize the Kronecker product (KP) and intermediate result caching (IRC) for optimization. These techniques reduce the HE-FCA multiplicative depth to one and eliminate numerous redundant HE operations. Table II summarizes the number of HE operations required by the HE-based $F_{3\times3}^{2\times2}$ FCA under various optimizations.

TABLE II: Number of operations required by various HE-based convolution algorithms under different optimization strategies.

Methods	Level	#HE Multiplications	#HE Additions
HE-based DCA [20]	1	36	32
Naive HE-FCA	5	116	56
HE-FCA w/ KP	1	16	80
HE-FCA w/ KP+IRC	1	16	56

1) One-level HE-FCA with KP: Because the encrypted matrix $[\mathbf{X}]$ lies between $\mathbf{B}^\top$ and $\mathbf{B}$, computing $\mathbf{B}^\top[\mathbf{X}]\mathbf{B}$ in the sequential matter requires two levels. However, with the vector trick in (4), we obtain:

$$\mathbf{B}^\top [\mathbf{X}] \mathbf{B} = (\mathbf{B}^\top \otimes \mathbf{B}) vec([\mathbf{X}]),$$

which means we can calculate $\mathbf{B}^\top \otimes \mathbf{B}$ in the cleartext domain in advance, and the computation of $\mathbf{B}^\top[\mathbf{X}]\mathbf{B}$ results in only one HE level.

The vector trick can be applied to combine $\mathbf{A}^\top$ and $\mathbf{A}$ in advance, so (9) can be rewritten as:

$$[\mathbf{Y}] = \mathbf{A}^\top \otimes \mathbf{A} \left(\left(\mathbf{G}\hat{\mathbf{K}}\mathbf{G}^\top \right) \boxdot \left(\mathbf{B}^\top \otimes \mathbf{B} vec([\mathbf{X}]) \right) \right), \quad (10)$$

and the step-by-step process of the HE-FCA is as follows:

$$[\mathbf{u}] = (\mathbf{B}^\top \otimes \mathbf{B}) vec([\mathbf{X}]), \quad (11a)$$

$$[\mathbf{v}] = vec(\mathbf{G}\hat{\mathbf{K}}\mathbf{G}^\top) \boxdot [\mathbf{u}], \text{and} \quad (11b)$$

$$[\mathbf{Y}] = (\mathbf{A}^\top \otimes \mathbf{A})[\mathbf{v}]. \quad (11c)$$

Since the entries of $\mathbf{A}^\top$ and $\mathbf{B}^\top$ are limited to ±1 and 0, the resulting Kronecker products $\mathbf{A}^\top \otimes \mathbf{A}$ and $\mathbf{B}^\top \otimes \mathbf{B}$ also contain only ±1 and 0. Consequently, computing $[\mathbf{v}]$ is the sole operation requiring HE multiplication in (11). To conclude, Algorithm 2 summarizes the HE-FCA procedure for 2-D data.

Algorithm 2 HE-FCA algorithm. (HE-FCA)

Require: An encrypted data $[\mathbf{X}]$ of size $W \times H$ and a plaintext kernel $\hat{\mathbf{K}}$ of size $r \times r$.

Ensure: An encrypted convolved result $[\mathbf{Y}] \in W^o \times H^o$.

1: **Initialize** FCA matrices $\mathbf{A}$, $\mathbf{B}$, and $\mathbf{G}$ with $o = 2$ and r
2: $\alpha = o + r - 1$
3: **for** $0 \leq i < \lceil W^o/o \rceil$ and $0 \leq j < \lceil H^o/o \rceil$ **do**
4: $\quad \beta = o \times i$
5: $\quad \gamma = o \times j$
6: $\quad [\mathbf{u}] = (\mathbf{B}^\top \otimes \mathbf{B}) vec([\mathbf{X}_{\beta:\beta+\alpha,\gamma:\gamma+\alpha}])$ ▷ Only HE additions involved
7: $\quad [\mathbf{v}] = vec(\mathbf{G}\hat{\mathbf{K}}\mathbf{G}^\top) \boxdot [\mathbf{u}]$ ▷ HE multiplications
8: $\quad [\mathbf{Y}_{i:i+\beta,j:j+\gamma}] = (\mathbf{A}^\top \otimes \mathbf{A})[\mathbf{v}]$ ▷ Only HE additions involved
9: **end for**

2) IRC for Saving HE Additions: IRC reduces the number of HE additions required when computing with $\mathbf{A}^\top \otimes \mathbf{A}$ and $\mathbf{B}^\top \otimes \mathbf{B}$. Figure 1 illustrates the use of IRC in computing $[\mathbf{Y}]$ for the $F_{3\times3}^{2\times2}$ FCA, corresponding to (11c).

As Figure 1.a illustrates, the standard computation of $[\mathbf{Y}]$ requires thirty-two HE additions. By precomputing four intermediate results, $[\mathbf{t}^{(0)}]$ to $[\mathbf{t}^{(3)}]$, the number of additions is reduced to twenty-four, as shown in Figure 1.b. Similarly, IRC can also be applied to the computation of $[\mathbf{u}]$ in (11a), resulting in IRC saves in a total of twenty-four HE additions in the HE-based $F_{3\times3}^{2\times2}$ FCA.

(a)

$$[\mathbf{Y}_0] = [\mathbf{v}_0] \boxplus [\mathbf{v}_1] \boxplus [\mathbf{v}_2] \boxplus [\mathbf{v}_4] \boxplus [\mathbf{v}_5] \boxplus [\mathbf{v}_6] \boxplus [\mathbf{v}_8] \boxplus [\mathbf{v}_9] \boxplus [\mathbf{v}_{10}]$$
$$[\mathbf{Y}_1] = [\mathbf{v}_1] \boxminus [\mathbf{v}_2] \boxplus [\mathbf{v}_3] \boxplus [\mathbf{v}_5] \boxminus [\mathbf{v}_6] \boxplus [\mathbf{v}_7] \boxplus [\mathbf{v}_9] \boxminus [\mathbf{v}_{10}] \boxplus [\mathbf{v}_{11}]$$
$$[\mathbf{Y}_2] = [\mathbf{v}_4] \boxplus [\mathbf{v}_5] \boxplus [\mathbf{v}_6] \boxplus [\mathbf{v}_8] \boxplus [\mathbf{v}_9] \boxplus [\mathbf{v}_{10}] \boxplus [\mathbf{v}_{12}] \boxplus [\mathbf{v}_{13}] \boxplus [\mathbf{v}_{14}]$$
$$[\mathbf{Y}_3] = [\mathbf{v}_5] \boxminus [\mathbf{v}_6] \boxplus [\mathbf{v}_7] \boxminus [\mathbf{v}_9] \boxplus [\mathbf{v}_{10}] \boxminus [\mathbf{v}_{11}] \boxplus [\mathbf{v}_{13}] \boxminus [\mathbf{v}_{14}] \boxplus [\mathbf{v}_{15}]$$

(b)

$$[t^{(0)}] = [\mathbf{v}_4] \boxplus [\mathbf{v}_5] \boxplus [\mathbf{v}_6] \qquad [\mathbf{Y}_0] = [\mathbf{v}_0] \boxplus [\mathbf{v}_1] \boxplus [\mathbf{v}_2] \boxplus [t^{(0)}] \boxplus [t^{(1)}]$$
$$[t^{(1)}] = [\mathbf{v}_8] \boxplus [\mathbf{v}_9] \boxplus [\mathbf{v}_{10}] \qquad [\mathbf{Y}_1] = [\mathbf{v}_1] \boxminus [\mathbf{v}_2] \boxplus [\mathbf{v}_3] \boxplus [t^{(2)}] \boxplus [t^{(3)}]$$
$$[t^{(2)}] = [\mathbf{v}_5] \boxminus [\mathbf{v}_6] \boxplus [\mathbf{v}_7] \qquad [\mathbf{Y}_2] = [\mathbf{v}_{12}] \boxplus [\mathbf{v}_{13}] \boxplus [\mathbf{v}_{14}] \boxplus [t^{(0)}] \boxminus [t^{(1)}]$$
$$[t^{(3)}] = [\mathbf{v}_9] \boxminus [\mathbf{v}_{10}] \boxplus [\mathbf{v}_{11}] \qquad [\mathbf{Y}_3] = [\mathbf{v}_{13}] \boxminus [\mathbf{v}_{14}] \boxplus [\mathbf{v}_{15}] \boxplus [t^{(2)}] \boxminus [t^{(3)}]$$

Fig. 1: The computation of $[\mathbf{Y}]$ in (11c): (a) without IRC, and (b) with IRC.

D. CHIP-based Convolution Block

Here, we introduce the convolution block, a component that is used for evaluations with consecutive convolutions. The convolution block consists of two convolutional layers, with both input and output represented in dense format. Let the kernels of the two convolutions be $\mathbf{K}^{(1)} \in \mathbb{R}^{C^{o_1} \times C^{i_1} \times r \times r}$ and $\mathbf{K}^{(2)} \in \mathbb{R}^{C^{o_2} \times C^{i_2} \times r \times r}$. To prevent channel value overlap and avoid unnecessary slot movements, three modifications are introduced to adapt the convolution block accordingly.

1) The parameter "s" in the RSR function during the data duplication step of the first convolution is set to $C^d = \max(C^{o_1}, C^{o_2})$, ensuring sufficient space for the second convolution, particularly when $C^{o_2} > C^{o_1}$.
2) Similarly, the parameter "s" of the RSL function in the channel aggregation stage of the second convolution is also set to C^d.
3) Since $C^{o_1} = C^{i_2}$, T is set as $T = C^d C^{o_1} = C^d C^{i_2}$ for both convolutions.

The performance of CHIP-based convolution depends on the number of available slots S and the number of intermediate results T generated during computation. When $T \leq S$, the two convolutions within a convolution block can be executed without modification.

However, when $T > S$—due to wide kernels—embedding all T values into a single ciphertext becomes infeasible, thereby preventing convolution in both CHIP representations. Additionally, handling multiple input samples concurrently presents another challenge, as each sample requires T slots. As the number of samples increases, the total slot demand may exceed the available capacity S. To address these challenges, we propose two techniques: the *partial-kernel* strategy and the *mini-batch* strategy, as detailed in the following section.

E. Circular Evaluations with the Partial-Kernel and Mini-Batch Strategies for Intensive Inference

The *partial-kernel* and *mini-batch* strategies respectively address slot insufficiency caused by wide kernels and multiple samples. Both approaches decompose the original task into multiple sub-tasks, computed iteratively using subsets of kernels or samples to satisfy slot constraints. Notably, the two strategies' loops are nested, with the mini-batch loop enclosing the partial-kernel loop. The final output is obtained by aggregating outputs of all sub-tasks.

1) The Partial-Kernel Strategy: When $T > S$, both $\mathbf{K}^{(1)}$ and $\mathbf{K}^{(2)}$ are partitioned into $P = \lceil T/S \rceil$ sub-kernels to accommodate channel values within a ciphertext. In other words, the partial-kernel strategy decomposes the original process into P sub-tasks, each handling two convolutions as follows:

1) The input data $[\mathbf{X}]$ is convolved with a sub-kernel of $\mathbf{K}^{(1)}$ to produce $C^p = C^{i_1}/P$ output channels, where the output $[\mathbf{X}']$ is represented in the sparse format.
2) The resulting $[\mathbf{X}']$ is served as input to the second convolution, which convolves it with a sub-kernel of $\mathbf{K}^{(2)}$ to generate C^{o_2} output channels by utilizing C^p of the C^{i_2} input channels. Since the output is in dense representation, the results occupy the first C^{o_2} slots.

To obtain the correct output, the results from all P sub-tasks are aggregated through element-wise summation upon completion. In summary, Algorithm 3 illustrates the convolution block workflow employing the partial-kernel strategy.

2) The Mini-batch Strategy: To reduce inference time per testing sample, multiple samples should be packed into the encrypted data $[\mathbf{X}]$ to amortize runtime costs. Because the kernel channel dimensions vary during inference, each sample

Algorithm 3 The partial-kernel strategy algorithm. (PKS)

Require: A dense encrypted data $[\mathbf{X}]$ and two real-number kernels $\mathbf{K}^{(1)}$ and $\mathbf{K}^{(2)}$.
Ensure: An encrypted convolved result $[\mathbf{Y}]$.

$P = max(C^{o_1}, C^{o_2})$
$C^p = C^{i_1}/P$
for $0 \leq i < P$ **do**
  $\hat{\mathbf{K}}^{(1)} = Ecd(\mathbf{K}^{(1)}_{i \times C^p:(i+1)\times C^p,:,:,:})$
  $[\mathbf{X}^{'}] =$ CHIP-CONV$([\mathbf{X}], \hat{\mathbf{K}}^{(1)}, \text{“}dense\text{”})$ ▷ The first convolution
  $\hat{\mathbf{K}}^{(2)} = Ecd(\mathbf{K}^{(2)}_{:,i\times C^p:(i+1)\times C^p,:,:})$
  $[\mathbf{Y}] = [\mathbf{Y}] \boxplus$ CHIP-CONV$([\mathbf{X}^{'}], \hat{\mathbf{K}}^{(2)}, \text{“}sparse\text{”})$ ▷ The second convolution
end for

is assigned C^m slots—where C^m denotes the maximum channel count across all layers—prior to the convolution block, preventing slot rearrangement. Consequently, each ciphertext of $[\mathbf{X}]$ can pack channel values of up to $B = S/C^m$ samples, with the channel values of each sample concatenated sequentially.

Since CHIP-based convolution requires T slots per sample, the complete set of B samples may be divided into multiple mini-batches to meet the slot constraints. The convolution block sequentially processes each mini-batch, so the relationship among T, C^m, and S determines the number of mini-batch iterations M required to process all B samples:

- If $T < C^m$, all B samples can be processed simultaneously by the convolution block without overlapping slots. Hence, $M = 1$.
- If $S \geq T \geq C^m$, only S/T samples can be processed concurrently. Consequently, the convolution block must be executed $M = \lceil B \times T/S \rceil$ rounds.
- If $T > S$, a single ciphertext cannot accommodate all T slots, necessitating the partial-kernel strategy. Thus, each sample is processed independently, resulting in $M = B$.

Furthermore, an extraction step precedes the convolution block, and a merge step follows it:

- **Extraction:** This step prepares the input data for the convolution block by applying a filter $\hat{f}^E$ to the ciphertext, extracting input channel values at every T slots. In the i-th round, the filter is shifted right by $i \times C^m$ slots.
- **Merge:** This step aggregates the convolved results from multiple rounds into a single ciphertext through element-wise summation.

In summary, Algorithm 4 summarizes the complete batch inference procedure utilizing CHIP packing, and Table III presents a quantization analysis of the convolution process in terms of required HE ciphertexts and operations.

F. Rest of CNN Functions

This section adapts the remaining essential layers and functions involved in constructing CNN models to ensure CHIP compatibility.

Algorithm 4 The mini-batch strategy algorithm.

Require: A dense representation $W \times H$ encrypted data $[\mathbf{X}]$ that packs B samples and two kernels $\mathbf{K}^{(1)} \in \mathbb{R}^{C^{o_1} \times C^{i_1} \times r \times r}$ and $\mathbf{K}^{(2)} \in \mathbb{R}^{C^{o_2} \times C^{i_2} \times r \times r}$. The number of batch round is M, and the number of channel reserved for each sample is C^m.
Ensure: An encrypted convolved result $[\mathbf{Y}]$ of size $W^o \times H^o$.

Initialize: The filter $\hat{f}^E$
for $0 \leq k < M$ **do**
  for $0 \leq i < W$ and $0 \leq j < H$ **do**
    $[\mathbf{X}^{'}_{i,j}] = [\mathbf{X}_{i,j}] \boxtimes (\hat{f}^E \gg kC^m)$ ▷ The extraction step
  end for
  $[\mathbf{X}''] = PKS([\mathbf{X}'], \mathbf{K}^{(1)}, \mathbf{K}^{(2)})$
  for $0 \leq i < W^o$ and $0 \leq j < H^o$ **do**
    $[\mathbf{Y}_{i,j}] = [\mathbf{Y}_{i,j}] \boxplus [\mathbf{X}''_{i,j}]$ ▷ The merge step
  end for
end for

1) Secure Dense Layer: To compute the dense layer using dense representation data $[\mathbf{x}]$, each column of $\mathbf{W} \in \mathbb{R}^{m \times n}$ is embedded into separate ciphertexts, with each ciphertext multiplied by $[\mathbf{x}]$ individually. Subsequently, each ciphertext undergoes the RSL function with parameters $s = 1$ and $t = \log_2 m$. The $i \times C^m$-th slot of the j-th ciphertext corresponds to the j-th predicted class for the i-th sample. To conclude, Algorithm 5 details the secure dense layer algorithm.

Algorithm 5 Secure dense layer algorithm.

Require: A dense format input $[\mathbf{x}]$ encrypts B samples. Each sample has m channels and is occupied with C^m slots. The model weight $\mathbf{W} \in \mathbb{R}^{m \times n}$.
Ensure: An encrypted result $[\mathbf{y}]$.

for $0 \leq i < n$ **do**
  for $0 \leq b < B$ and $0 \leq j < m$ **do**
    $\mathbf{w}_{bC^m+j} = \mathbf{W}_{j,i}$
  end for
  $\hat{w} = Ecd(\mathbf{w})$
  $[\mathbf{y}_i] = RSL([\mathbf{x}] \boxtimes \hat{w}, 1, log_2(m))$
end for

2) Secure Activation Function: We apply AESPA [42] to minimize the complexity associated with activation functions. AESPA [42] substitutes the ReLU function with a three-base Hermite expansion and incorporates basis-wise normalization concatenated to each basis to prevent gradient explosion. Therefore, their proposed activation function is defined as follows:

$$f(x) = \gamma \sum_{i=0}^{d=2} \mathbf{f}_i \frac{h_i(x) - \boldsymbol{\mu}_i}{\sqrt{\boldsymbol{\sigma}_i^2 + \epsilon}} + \beta,$$

where

$$h_0(x) = 1,\ h_1(x) = x, \text{ and } h_2(x) = \frac{x^2 - 1}{\sqrt{2}},$$

TABLE III: Computational complexity of convolution under different methods.

Metrics	MPCNN [21]	CBC [22]	Ours
#Input Ciphertext	$r^2\lceil\frac{WHC^i}{S}\rceil B$	$r^2C^i\lceil\frac{WHB}{S}\rceil$	WH
#Model Weight	r^2C^o	$r^2C^iC^o$	r^2
#HE Multiplication	$r^2C^o\lceil\frac{WH}{S}\rceil B$	$C^iC^or^2\lceil\frac{WHB}{S}\rceil$	$16\times PMW^oH^o$
#HE Addition	$(r^2+2\times log_2C^i+1)C^o\lceil\frac{WH}{S}\rceil B$	$C^iC^o(1+r^2)\lceil\frac{WHB}{S}\rceil$	$56\times PMW^oH^o$
#HE Rotation	$(r^2+(2\times log_2C^i+1)C^o)\lceil\frac{WH}{S}\rceil B$	$r^2C^i\lceil\frac{WHB}{S}\rceil$	$(log_2C^i+log_2C^o)PMW^oH^o$

and

$$\mathbf{f}_0=\frac{1}{\sqrt{2\pi}},\ \mathbf{f}_1=\frac{1}{2},\ \text{and}\ \mathbf{f}_2=\frac{1}{\sqrt{4\pi}}.$$

$f(x)$ can be expressed as a second-degree polynomial, so the secure activation function can be represented as:

$$f([x])=a[x]^2\boxplus b[x]\boxplus c,$$

where

$$a=\frac{\gamma}{\sqrt{8\pi(\boldsymbol{\sigma}_2^2+\epsilon)}},b=\frac{\gamma}{2\sqrt{\boldsymbol{\sigma}_1^2+\epsilon}},\text{and}$$
$$c=\beta+\gamma\left(\frac{1-\boldsymbol{\mu}_0}{\sqrt{2\pi(\boldsymbol{\sigma}_0^2+\epsilon)}}-\frac{\boldsymbol{\mu}_1}{2\sqrt{\boldsymbol{\sigma}_1^2+\epsilon}}-\frac{1+\sqrt{2}\boldsymbol{\mu}_2}{\sqrt{8\pi(\boldsymbol{\sigma}_2^2+\epsilon)}}\right).$$

The leading coefficient can be combined with the model weights, so the secure activation function would only cost one level.

3) Secure Mean-pooling and Batch Normalization Functions: Both secure functions can be obtained directly by changing arithmetic operations into HE-based counterparts. Therefore, the calculation of mean-pooling function is shown as:

$$[\mathbf{X}_{i,j}^{'}]=\frac{1}{m^2}\boxplus_{a,b=0}^{a,b<m-1}[\mathbf{X}_{im+a,jm+b}].$$

As for the secure batch normalization, it is defined as:

$$[\mathbf{x}_i^{'}]=a[\mathbf{x}_i]\boxplus b,$$

where

$$a=\frac{\gamma}{\sqrt{\sigma^2+\epsilon}},\ \text{and}\ b=\frac{-\gamma\mu}{\sqrt{\sigma^2+\epsilon}}+\beta.$$

The leading coefficients of both functions can be absorbed into the model weights, ensuring that neither function incurs any level consumption. Notably, the activation, mean-pooling, and batch normalization functions do not alter the channel packing format, allowing them to be applied after CHIP-based convolutions without changing the data representation.

G. Evaluation Architecture

Figure 2 illustrates the evaluation architecture, which integrates all techniques and functionalities for private batch inference. To ensure correctness and control noise growth in HE computations, rescaling is performed after each CNN layer or function. Since all subtasks are executed in parallel and reach the same ciphertext level, correctness is preserved after aggregation. Although the partial-kernel and mini-batch strategies replicate encrypted input data—resulting in ciphertext expansion—the merge step consolidates all channel values of a data element across the B samples into a single ciphertext. Consequently, the final two operations—bootstrapping and activation—can be efficiently performed using only $W\times H$ ciphertexts.

V. Evaluations

A. Evaluation Setup

Datasets and Models. We evaluate our private inference solution using ResNet18 [39] and VGG11 [47] on the CIFAR10 dataset, and VGG16 [47] on the CIFAR100 dataset. Both datasets comprise 50,000 training images and 10,000 testing images, each with dimensions of $3\times32\times32$. The CIFAR10 dataset classifies images into ten distinct categories, whereas the CIFAR100 dataset presents a more intricate classification with one hundred categories.

Cryptosystem Configuration. We implement our solution utilizing the HEAAN library [48] with the "FGb" parameter set, which offers a 128-bit security level. In this configuration, the lattice dimension N is 65536, and the ciphertext modulus Q comprises twenty-four prime numbers, enabling nine levels of multiplicative depth.

Experimental Setup. All evaluations were performed on a system with an AMD 7975WX CPU and 512GB of RAM, with parallelization enabled via OpenMP [49].

Training Hyperparameters. Models were pre-trained in the cleartext domain utilizing the stochastic gradient descent optimizer with a batch size of 128. The training was conducted over 200 epochs, with the learning rate fixed at 0.1. Knowledge Distillation [50] was employed to enhance the accuracy of quadratic approximation models. The teacher model corresponds to each original model, whereas the student model represents its modified version. The distillation loss is defined as the mean squared error between the two models.

Baselines. We compare our solution with MPCNN[1] [21] and CBC[2] [22], two HE-based approaches for PPML batch inference. The former utilizes a multiplexed parallel convolution, while the latter employs a channel-by-channel convolution algorithm. Both baselines were modified to approximate the ReLU function using the identical quadratic polynomial approximation [42] applied in our method, thereby ensuring a fair comparison in convolution operations.

[1]https://github.com/snu-ccl/FHE-MP-CNN

[2]https://github.com/altaria1/HEaaN-CBC-ResNet

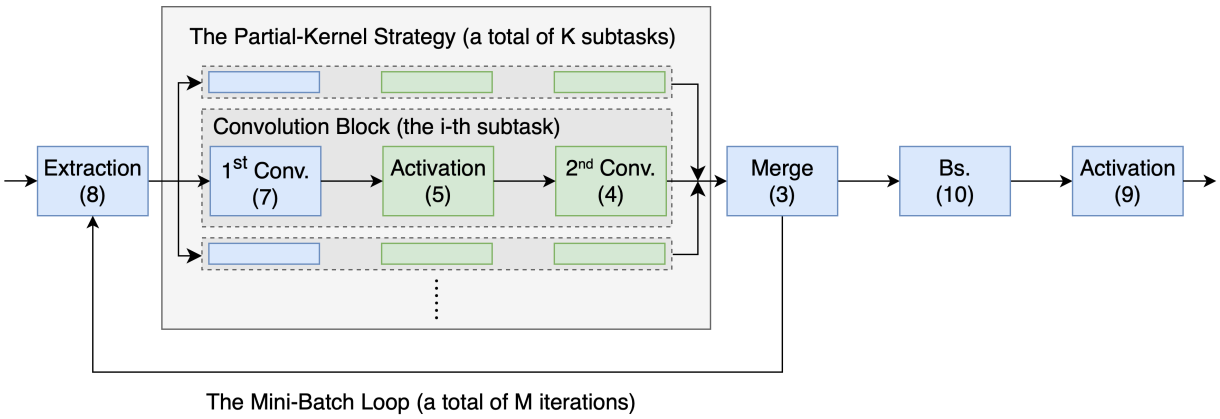

Fig. 2: The evaluation architecture of the CNN models. Numbers in brackets indicate the ciphertext levels of function outputs. Blue blocks represent inputs in dense representation, while green blocks denote inputs in sparse representation. "Conv." denotes the convolution and "Bs." refers to the bootstrapping.

B. The Details of Constructing CNN Models

Table IV presents the model architectures of the ResNet18, VGG11, and VGG16 models, including the detailed parameters utilized in our solution. As an independent convolution is executed before the first convolution block in both models, the input encrypted data is represented in sparse format. Given that all three models have a maximum of 512 data channels, the maximum batch size for each model is 64.

TABLE IV: The primary structures of the ResNet18 (RN18), VGG11 (V11), and VGG16 (V16) models. The numbers in the description column denote the output channels of the convolutions, while the letter "M" signifies the mean-pooling function.

Model	Layer Name	Input Size	Description	P	M
RN18	conv_1	3×32×32	64	1	1
	conv_block_1	64×32×32	(64, 64)×2	$\lceil B/8 \rceil$	1
	conv_block_2	64×32×32	(128, 128)×2	$\lceil B/2 \rceil$	1
	conv_block_3	128×16×16	(256, 256)×2	B	2
	conv_block_4	256×8×8	(512, 512)×2	B	8
V11	conv_1	3×32×32	64, M	1	1
	conv_block_1	64×16×16	128, M, 256	B	1
	conv_block_2	256×8×8	256, M, 512	B	4
	conv_block_3	512×4×4	512, M, 512	B	8
	conv_2	512×2×2	512, M	B	8
V16	conv_1	3×32×32	64	1	1
	conv_block_1	64×32×32	64, M, 128	$\lceil B/4 \rceil$	1
	conv_block_2	128×16×16	128, M, 256	B	1
	conv_block_3	256×8×8	256, 256, M	B	2
	conv_block_4	256×4×4	512, 512	B	8
	conv_block_5	512×4×4	512, M, 512	B	8
	conv_block_6	512×2×2	512, 512, M	B	8

C. Microbenchmarks

1) HE-FCA on 2-D Convolutions: Table V illustrates that the naive HE-FCA exhibits the slowest performance, while the introduction of KP significantly enhances HE-FCA's performance. When both KP and IC are applied, HE-FCA achieves speed-ups of up to 53× relative to its unoptimized version. Furthermore, HE-FCA+KP+IC is approximately 3× faster than HE-based DCA [20] under the same packing technique.

TABLE V: Runtime (in seconds) for the 2-D convolution using a 3×3 kernel. Best values in bold.

Input Size	HE-based DCA [20]	Naive HE-FCA	HE-FCA +KP	HE-FCA+KP +IRC (Ours)
32×32	5.4	42.3	2.5	**1.9**
16×16	1.4	12.1	0.7	**0.5**
8×8	0.4	4.6	0.2	**0.1**

2) CHIP-based 3-D Convolutions: As demonstrated in Table VI, both dense-style and sparse-style inputs exhibit comparable runtimes. While increasing the channel size, MPCNN [21] and CBC [22] require more time for convolutions. Notably, CBC [22] exhibits exponential growth. In contrast, the performance of our proposed solution is more closely associated with the input size; thus, a smaller input size with a larger channel size still yields quicker performance. Overall, our solution is up to 326× faster than MPCNN [21] and can be over 500× faster than CBC [22].

TABLE VI: Runtime (in seconds) for the CHIP-based convolution with a 3×3 kernel. "D-input" and "S-input" denote dense-style and sparse-style inputs, respectively. Best values in bold.

Input	C^i	C^o	MPCNN [21]	CBC [22]	Ours	
					D-Input	S-Input
32×32	64	64	257.1	19.8	**10.8**	**10.8**
16×16	128	128	301.2	68.2	3.1	**2.9**
8×8	256	256	405.9	261.2	**1.6**	1.7
4×4	512	512	621.1	1,006.5	**1.9**	**1.9**

D. Comparisons of CNN inference with Related Work

1) Runtime: Table VII shows that our method outperforms MPCNN [21] and CBC [22] across various batch sizes, with greater advantages at smaller ones. Using ResNet18 on the

CIFAR10 dataset with a batch size of 64, our method demonstrates speedups of 1.1× and 2.9× compared to CBC [22] and MPCNN [21], respectively. For a single test sample, it achieves up to a 4.4× speedup relative to both baselines. Similarly, for VGG models at a batch size of 64, our approach attains up to 2.6× and 4.7× speedups over CBC [22] and MPCNN [21], respectively. At batch size one, speedups reach up to 60× and 27× over both baselines for VGG11 and VGG16, respectively.

Figure 3 shows that convolution is the primary performance bottleneck across all three methods. However, our CHIP-based convolution is more efficient than the other two approaches, resulting in a lower total runtime on both models. Specifically, the three methods exhibit distinct runtime trends:

- The performance of MPCNN [21] improves with increased thread count, with all runtime components scaling accordingly. However, the runtime increases exponentially when the batch size approaches the CPU's maximum thread capacity.
- CBC [22] exhibits stable performance as the batch size increases, until a batch size of 64, where multiple ciphertexts are required to pack all batch samples for a single channel. Moreover, since the number of ciphertexts scales with the channel dimension, its computational complexity does not decrease with smaller batch sizes, resulting in suboptimal performance compared to the other two methods under small-batch settings.
- Due to the use of the partial-kernel and mini-batch strategies, our method requires more runtime as the batch size increases, particularly for operations within the convolution block, such as convolutions and activations. However, by preventing ciphertext expansion beyond the convolution block, we ensure that only $W \times H$ ciphertexts undergo bootstrapping. As a result, bootstrapping time remains constant regardless of batch size.

2) Accuracy: All methods compared utilize the same polynomial approximation for the ReLU and yield identical convolution outputs, resulting in equivalent accuracy across the board. The teacher models achieve accuracies of 91.7%, 89.4%, and 67.6% on ResNet18, VGG11, and VGG16, respectively. Incorporating knowledge distillation with quadratic polynomial approximation, the models attain final accuracies of 91.9%, 89.6%, and 67.7% on these architectures, respectively. The accuracy results demonstrate that our CHIP-based private inference effectively controls noise from homomorphic encryption operations while maintaining correct inference outcomes.

3) Rotation Keys & Memory Usage: Table VIII shows that our method reduces memory consumption by approximately 37% and 45% compared to MPCNN [21] and CBC [22], respectively. This improvement stems from two key factors. First, our approach uses significantly fewer rotation keys—88% fewer than MPCNN [21] and 35% fewer than CBC [22]. The cyclic shifts in our rotation key set are given by $2^{\{0,\pm1,\ldots,\pm14\}}$. Second, CHIP enhances packing efficiency by reducing both the number of encrypted data and plaintext model weights. As a result, our method offers a more memory-efficient solution for private inference.

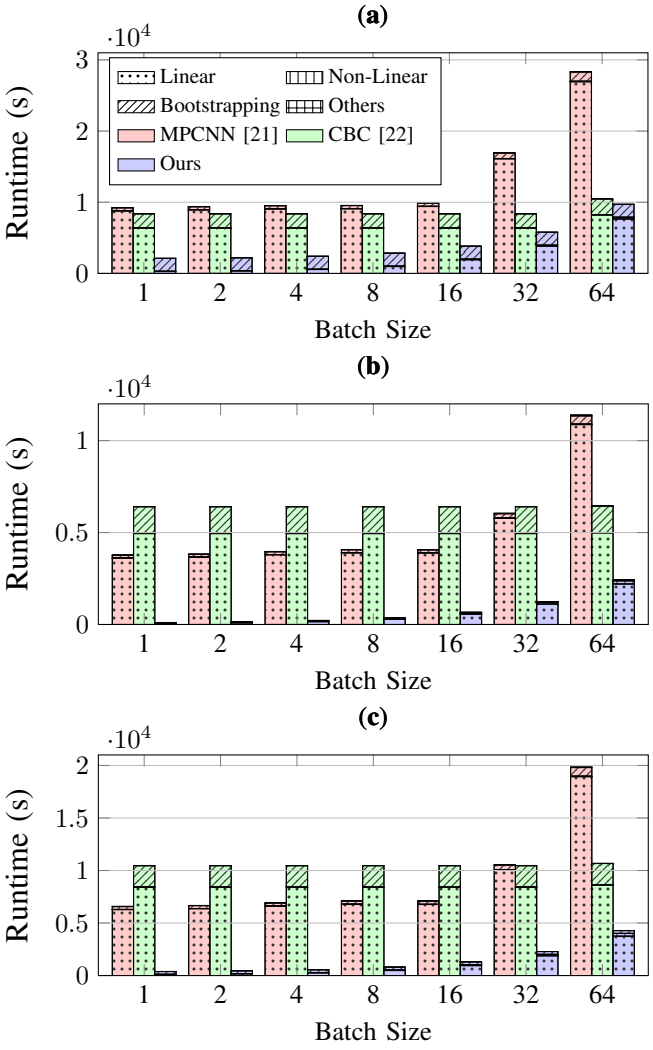

Fig. 3: Runtime breakdown across different batch sizes for (a) ResNet18, (b) VGG11, and (c) VGG16.

VI. Conclusions

This paper presents techniques based on our channel-interleaved packing to enhance efficiency of homomorphic encryption (HE)-based private inference for CNNs. Experiments show our method significantly reduces computational and memory overhead and dependence on rotation keys. By supporting flexible batch sizes, it improves scalability and applicability across diverse HE inference scenarios. We believe these contributions advance efficient HE-based inference for deeper, wider models, promoting practical adoption of privacy-preserving ML.

Acknowledgment

This project received funding from the National Science and Technology Council in Taiwan (111-2221-E-002-134-MY3).

TABLE VII: Amortized runtime (in seconds) across varying batch sizes. Best values in bold.

Batch Size	ResNet18-CIFAR10			VGG11-CIFAR10			VGG16-CIFAR100		
	MPCNN [21]	CBC [22]	Ours	MPCNN [21]	CBC [22]	Ours	MPCNN [21]	CBC [22]	Ours
1	9,217.9	8,361.5	**2,104.0**	3,778.0	6,403.4	**106.4**	6,582.8	10,463.4	**386.0**
2	4,687.0	4,180.8	**1,087.9**	1,914.1	3,201.7	**71.6**	3,335.1	5,231.7	**218.8**
4	2,378.4	2,090.4	**598.0**	992.2	1,600.9	**54.2**	1,728.7	2,615.8	**135.3**
8	1,193.3	1,045.2	**353.0**	509.6	800.4	**45.5**	888.0	1,307.9	**98.8**
16	618.2	522.6	**237.8**	254.5	400.2	**41.1**	443.4	654.0	**80.6**
32	529.3	261.3	**180.3**	189.0	200.1	**38.9**	329.4	327.0	**71.5**
64	442.8	163.3	**151.5**	178.0	100.6	**37.8**	310.2	166.8	**66.9**

TABLE VIII: Comparisons of the number of rotation key and the memory usage.

Metrics	MPCNN [21]	CBC [22]	Ours
#Rotation Keys	258	45	**29**
Memory Usage (GB)	442	508	**280**

REFERENCES

[1] H.-C. Wang and J.-L. Wu, "CHIP: Efficient homomorphic encryption-based cnn batch inference using channel-interleaved packing with small rotation key set," in *IEEE CSF*, 2026.

[2] I. Grigoriadis, E. Vrochidou, I. Tsiatsiou, and G. A. Papakostas, "Machine learning as a service (MLaaS)—an enterprise perspective," in *ICDSA*, 2023.

[3] M. Krichen, "Convolutional neural networks: A survey," *Computers*, 2023.

[4] Z. Li, F. Liu, W. Yang, S. Peng, and J. Zhou, "A survey of convolutional neural networks: Analysis, applications, and prospects," *IEEE Transactions on Neural Networks and Learning Systems*, 2021.

[5] Q. Zhang, T. Xiang, Y. Cai, Z. Zhao, N. Wang, and H. Wu, "Privacy-preserving machine learning as a service: Challenges and opportunities," *IEEE Network*, 2022.

[6] R. Xu, N. Baracaldo, and J. Joshi, "Privacy-preserving machine learning: Methods, challenges and directions," *arXiv:2108.04417*, 2021.

[7] D. Parikh, S. Radadia, and R. K. Eranna, "Privacy-preserving machine learning techniques, challenges and research directions," *International Research Journal of Engineering and Technology*, 2024.

[8] D. Micciancio and V. Vaikuntanathan, "SoK: Learning with errors, circular security, and fully homomorphic encryption," in *PKC*, 2024.

[9] Y. Bai, H. Zhao, X. Shi, and L. Chen, "Towards practical and privacy-preserving CNN inference service for cloud-based medical imaging analysis: A homomorphic encryption-based approach," *Computer Methods and Programs in Biomedicine*, 2025.

[10] L. K. Ng and S. S. Chow, "SoK: Cryptographic neural-network computation," in *IEEE S&P*, 2023.

[11] R. Dathathri, O. Saarikivi, H. Chen, K. Laine, K. Lauter, S. Maleki, M. Musuvathi, and T. Mytkowicz, "CHET: An optimizing compiler for fully-homomorphic neural-network inferencing," in *PLDI*, 2019.

[12] A. Brutzkus, R. Gilad-Bachrach, and O. Elisha, "Low latency privacy preserving inference," in *ICML*, 2019.

[13] Q. Lou, W.-j. Lu, C. Hong, and L. Jiang, "Falcon: Fast spectral inference on encrypted data," *NeurIPS*, 2020.

[14] J.-W. Lee, H. Kang, Y. Lee, W. Choi, J. Eom, M. Deryabin, E. Lee, J. Lee, D. Yoo, Y.-S. Kim *et al.*, "Privacy-preserving machine learning with fully homomorphic encryption for deep neural network," *IEEE Access*, 2022.

[15] D. Kim and C. Guyot, "Optimized privacy-preserving CNN inference with fully homomorphic encryption," *IEEE Transactions on Information Forensics and Security*, 2023.

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

[17] R. Gilad-Bachrach, N. presidential Dowlin, K. Laine, K. Lauter, M. Naehrig, and J. Wernsing, "CryptoNets: Applying neural networks to encrypted data with high throughput and accuracy," in *ICML*, 2016.

[18] E. Hesamifard, H. Takabi, and M. Ghasemi, "CryptoDL: Deep neural networks over encrypted data," *arXiv:1711.05189*, 2017.

[19] E. Chou, J. Beal, D. Levy, S. Yeung, A. Haque, and L. Fei-Fei, "Faster CryptoNets: Leveraging sparsity for real-world encrypted inference," *arXiv:1811.09953*, 2018.

[20] F. Boemer, Y. Lao, R. Cammarota, and C. Wierzynski, "nGraph-HE: A graph compiler for deep learning on homomorphically encrypted data," in *CF*, 2019.

[21] 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 *ICML*, 2022.

[22] J. H. Cheon, M. Kang, T. Kim, J. Jung, and Y. Yeo, "Batch inference on deep convolutional neural networks with fully homomorphic encryption using channel-by-channel convolutions," *IEEE Transactions on Dependable and Secure Computing*, 2024.

[23] J. Kosaian, A. Phanishayee, M. Philipose, D. Dey, and R. Vinayak, "Boosting the throughput and accelerator utilization of specialized CNN inference beyond increasing batch size," in *ICML*, 2021.

[24] Y. Kochura, Y. Gordienko, V. Taran, N. Gordienko, A. Rokovyi, O. Alienin, and S. Stirenko, "Batch size influence on performance of graphic and tensor processing units during training and inference phases," in *ICCSEEA*, 2019.

[25] Z. Ye, T. Huang, T. Wang, Y. Li, C. Wang, R. C. Cheung, and K. Huang, "HTCNN: High-throughput batch CNN inference with homomorphic encryption for edge computing," *Cryptology ePrint Archive 2024/1753*, 2024.

[26] R. Jayanth, N. Gupta, and V. Prasanna, "Benchmarking edge AI platforms for high-performance ML inference," in *HPEC*, 2024.

[27] S. M. Nabavinejad, S. Reda, and M. Ebrahimi, "Coordinated batching and DVFS for DNN inference on GPU accelerators," *IEEE Transactions on Parallel and Distributed Systems*, 2022.

[28] T. Yarally, L. Cruz, D. Feitosa, J. Sallou, and A. van Deursen, "Batching for green AI—an exploratory study on inference," in *SEAA*, 2023.

[29] B. Spector and C. Re, "Accelerating LLM inference with staged speculative decoding," *arXiv:2308.04623*, 2023.

[30] Amazon, "Getting recommendations from amazon personalize." [Online]. Available: https://docs.aws.amazon.com/personalize/latest/dg/getting-recommendations.html

[31] Sapien, "Batch inference." [Online]. Available: https://www.sapien.io/glossary/definition/batch-inference

[32] J.-W. Lee, E. Lee, Y.-S. Kim, and J.-S. No, "Rotation key reduction for client-server systems of deep neural network on fully homomorphic encryption," in *ASIACRYPT*, 2023.

[33] A. Lavin and S. Gray, "Fast algorithms for convolutional neural networks," in *CVPR*, 2016.

[34] S. Winograd, *Arithmetic Complexity of Computations*. Siam, 1980.

[35] C. F. Van Loan, "The ubiquitous kronecker product," *Journal of Computational and Applied Mathematics*, 2000.

[36] J. H. Cheon, A. Kim, M. Kim, and Y. Song, "Homomorphic encryption for arithmetic of approximate numbers," in *ASIACRYPT*, 2017.

[37] J. H. Cheon, K. Han, A. Kim, M. Kim, and Y. Song, "Bootstrapping for approximate homomorphic encryption," in *EUROCRYPT*, 2018.

[38] V. Lyubashevsky, C. Peikert, and O. Regev, "On ideal lattices and learning with errors over rings," in *EUROCRYPT*, 2010.

[39] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *CVPR*, 2016.

[40] C. Juvekar, V. Vaikuntanathan, and A. Chandrakasan, "GAZELLE: A low latency framework for secure neural network inference," in *USENIX*, 2018.
[41] 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," *IEEE Access*, 2023.
[42] J. Park, M. J. Kim, W. Jung, and J. H. Ahn, "AESPA: Accuracy preserving low-degree polynomial activation for fast private inference," *arXiv:2201.06699*, 2022.
[43] W. Z. Srinivasan, P. Akshayaram, and P. R. Ada, "Delphi: A cryptographic inference service for neural networks," in *USENIX*, 2019.
[44] D. Rathee, M. Rathee, N. Kumar, N. Chandran, D. Gupta, A. Rastogi, and R. Sharma, "CrypTFlow2: Practical 2-party secure inference," in *CCS*, 2020.
[45] Z. Huang, W.-j. Lu, C. Hong, and J. Ding, "Cheetah: Lean and fast secure Two-Party deep neural network inference," in *USENIX*, 2022.
[46] I. Zhou, F. Tofigh, M. Piccardi, M. Abolhasan, D. Franklin, and J. Lipman, "Secure multi-party computation for machine learning: A survey," *IEEE Access*, 2024.
[47] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv:1409.1556*, 2014.
[48] CryptoLab Inc., "HEAAN." [Online]. Available: https://github.com/snucrypto/HEAAN
[49] OpenMP, "OpenMP." [Online]. Available: https://www.openmp.org
[50] G. Hinton, O. Vinyals, and J. Dean, "Distilling the knowledge in a neural network," *arXiv:1503.02531*, 2015.