Optimized Privacy-Preserving CNN Inference With Fully Homomorphic Encryption

Dongwoo Kim, *Associate Member, IEEE*, and Cyril Guyot

***Abstract*— Inference of machine learning models with data privacy guarantees has been widely studied as privacy concerns are getting growing attention from the community. Among others, secure inference based on Fully Homomorphic Encryption (FHE) has proven its utility by providing stringent data privacy at sometimes affordable cost. Still, previous work was restricted to shallow and narrow neural networks and simple tasks due to the high computational cost incurred from FHE. In this paper, we propose a more efficient way of evaluating convolutions with FHE, where the cost remains constant regardless of the kernel size, resulting in 12–46× timing improvement on various kernel sizes. Combining our methods with FHE bootstrapping, we achieve at least 18.9% (and 48.1%) timing reduction in homomorphic evaluation of 20-layer CNN classifiers (and a part of it) on CIFAR10/100 (and ImageNet, respectively) datasets. Furthermore, in consideration of our methods being effective for evaluating CNNs with intensive convolutional operations and exploring such CNNs, we achieve at least 5× faster inference on CIFAR10/100 with FHE than the prior works having the same or less accuracy.**

***Index Terms*— Privacy-preserving machine learning, fully homomorphic encryption, convolutional neural network.**

I. Introduction

As MACHINE learning models demonstrate their utility in many domains that rely on confidential data (e.g., financial, medical, and other personal data), concerns about data privacy are also gaining significant attention from the community, and various solutions are being studied under the umbrella of Privacy-Preserving Machine Learning (PPML). In particular, Secure Inference — which aims to provide privacy on client's data during an inference phase performed by a service provider — has been widely studied due to its versatility and feasibility in practice. Various realizations of secure inference have been proposed via Multiparty Computation [1], [2], Secure Processors (e.g., Intel SGX), Fully Homomorphic Encryption (FHE) [3], and their combinations.

Among them, FHE-based solutions are seen as promising since they offer the most stringent data privacy properties relying on the hardness of certain mathematical problems (as is usual in cryptography) and do not require additional interactions between a client and a service provider compared to usual inference without privacy guarantees.

Though many works have demonstrated the feasibility of FHE-based secure inference — see Section II for detail — only a few have focused on deep and wide Convolutional Neural Networks (CNNs) that necessitate bootstrapping of FHE.

In this paper, we facilitate the study in this direction by providing more concise representation of convolution and convolutional layers with FHE operations resulting in:

- *Constant*-time convolution evaluation, irrespective of the kernel size, resulting in 12–46× faster evaluation when kernel widths are 3–7.
- Reduction of the evaluation time of a convolutional layer by 13–83% depending on the kernel and batch size when combined with FHE Bootstrapping.
- Reduction by 18.9% and 48.1% of secure inference timing of 20-layer and a part of 18-layer CNN classifiers on CIFAR10/100 and ImageNet, respectively.
- Achieving state-of-the-art accuracy (94.0% and 73.5%) in the lowest latency (398 s) among prior FHE-based inferences on CIFAR10/100 datasets by exploring CNNs — with larger kernels, wider layers, and less depth — more favorable to secure inference in our method.

We achieved those results via several fundamental improvements on the evaluation method of convolutions and convolutional layers in FHE as follows:

1) We provide a concise representation of convolution operations via arithmetic operations in the plaintext space, a ring $\mathcal{R} := \mathbb{Z}[X]/(X^N+1)$, of CKKS FHE scheme [4]. More precisely, we pack each input into coefficients of a polynomial of the ring, so that single convolution can be evaluated with *single* multiplication without any rotation.
2) We further develop this method to a batch convolutions (that usually arises in CNNs) by generalizing the algorithm of [5] then utilizing it to pack as many convolution outputs as possible into each output ciphertext.
3) For evaluation of convolutional layers, we also modify the bootstrapping procedure of FHE so that convolutions are evaluated in the coefficient domain while activation functions are evaluated in the slot domain.

We also mention that our method can be applied to other FHE schemes (such as BGV [6] and B/FV [7], [8]) improving the cost of hybrid approaches in PPML inferences; see Section II-C and Appendix D.

Manuscript received 18 August 2022; revised 10 January 2023; accepted 8 March 2023. Date of publication 31 March 2023; date of current version 13 April 2023. This work was supported by the Dongguk University Research Fund of 2022 under Grant S-2022-G0001-00070. The associate editor coordinating the review of this manuscript and approving it for publication was Dr. George Theodorakopoulos. *(Corresponding author: Dongwoo Kim.)*

Dongwoo Kim is with the College of AI Convergence, Dongguk University, Seoul 04620, South Korea (e-mail: Dongwoo.Kim@dgu.edu).

Cyril Guyot is with Western Digital Research, Milpitas, CA 95035 USA.

Digital Object Identifier 10.1109/TIFS.2023.3263631

II. Related Work

Among several works on PPML, we focus on secure inferences solely based on FHE which require *only one* interaction round between a client and a service provider. Those approaches can be divided into two categories: high-throughput results and low-latency results. While the former aims to achieve high amortization efficiency on many parallel inputs, the latter aims to achieve a low latency for single input. Our result is classified as the latter, aiming low latency for a single inference. Then, we also investigate hybrid approaches requiring multiple interactions between parties and using FHE evaluation as a subroutine where our solution can be utilized.

A. High-Throughput Results

Cryptonets [9] firstly presented Neural Network (NN) evaluation with FHE by showing that its computational cost can be amortized by encrypting *many parallel inputs* into one ciphertext, e.g., 51000 predictions per an hour for MNIST task [10]. Followingly, several works [11], [12], [13], [14] improved accuracy and amortized timing. These approaches, however, show problematic latency when only single or few inference results are needed, and a lot of work developed low-latency results.

B. Low-Latency Results

To resolve this problem, Gazelle [15] proposed a vectorized representation of convolutions to achieve faster evaluation of a convolutional layer with FHE on *single* input. The vectorized representation has been widely used in other works as well, and [16], [17], [18] presented a compiler called EVA that automatically encodes a vectorized representation into FHE, along with an FHE-based CNN evaluation result. LoLa [19] achieved lower latency result for NN evaluations with a novel FHE packing method tailored to it. Recently, Falcon [20] achieved an improved latency applying the idea of spectral inference [21], [22] (see Section III-C for detail). LoLa and Falcon, however, aggressively stack the same input many times in a ciphertext to reduce the cost and are more effective than usual vectorized methods [15], [16], [17] only if target NNs' layer size is much smaller than the message slots available in one ciphertext. On the other hand, [23] (and [24] recently) proposed to evaluate discretized NN using TFHE scheme [25], [26]: while they can evaluate any depth of NN with constant overhead, the latency is not better than [17] due to the higher cost per single arithmetic operation. The best result in this direction is SHE [27] where TFHE scheme is used to process input and weight bits-by-bits allowing faster ReLU evaluation. However, the ciphertext size required for input is quite larger (123MB) than ours (2MB). Moreover, all TFHE-based works require appropriate discretization of the network weights and inputs for efficient evaluation.

All of those works, except TFHE-based one, are limited to low-depth NN with square activation function. Recently, [28] employed CKKS FHE scheme [4] with bootstrapping to evaluate 20-layer CNNs while their result on CIFAR10 recorded 10602 s even with 64-threads due to an inefficient packing.[1] Followingly, [29] achieved 2271 s for 91.3% accuracy adapting the vectorized convolution from [15] (and strided one from [30]). Compared to this result, we significantly reduce the cost of convolution. It allows us to exploit CNNs with more intensive convolution operations and less depth, thereby achieve similar or higher accuracy at much lower latency, e.g., 255 s for 92.0% and 398 s for 94.0%. We refer to Fig. 1 for the latency and accuracy of the prior works and ours on CIFAR10 dataset.

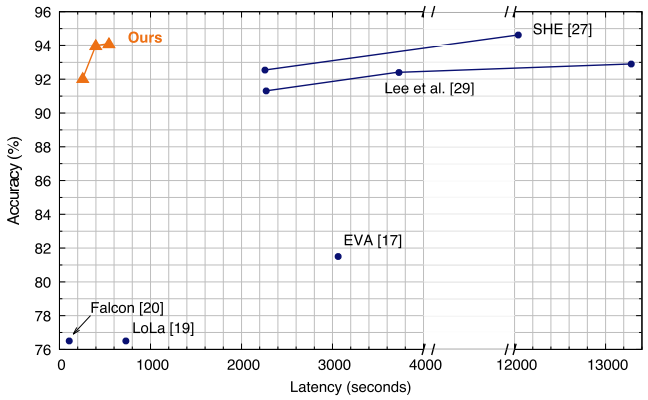

Fig. 1. Low-latency FHE-based inference results on CIFAR10 dataset.

C. Hybrid Approaches for PPML Inference

If we allow more than one interaction between parties, the most promising solutions, in terms of latency, are Hybrid approaches [15], [21], [22], [31], [32], [33], [34] where FHE is used for evaluating linear functions and Garbled Circuit or Oblivious Transfer is for non-linear activation functions. They, however, require higher communication costs and rounds that are proportional to the number of layers, e.g. ≈110MB (the best result [34]) for a 32-layer CNN on CIFAR10; for comparison, ours need only 2MB of ciphertexts and 1 round even for deeper CNNs. Among those work, [33] and [34] also represented convolution with polynomials for efficient evaluation in FHE. Their solution, however, cannot be applied to the case where interaction is not allowed (see Section IV-B). In contrast, our FHE evaluation method for convolution can be utilized in hybrid approaches for fast convolutions, as well as reducing the communication cost by packing more intermediate inputs into one ciphertext than [33] and [34] (see Appendix D).

III. Preliminaries

We first recall some necessary prerequisites on Fully Homomorphic Encryption and convolutions.

A. Notation

$\mathbb{Z}$, $\mathbb{R}$, and $\mathbb{C}$ denote the ring of integers, real, and complex numbers, respectively. We use $\mathcal{R} := \mathbb{Z}[X]/(X^N + 1)$ with N being power-of-two to denote the ring of polynomials modulo $X^N + 1$. Bold letters denote vectors, e.g., $\mathbf{z} \in \mathbb{C}^{N/2}$.

[1] *One* ciphertext has *one* conv output, resulting in 16–64 bootstrappings at *each* layer.

B. CKKS Fully Homomorphic Encryption (FHE)

Fully Homomorphic Encryption (FHE) [3], [35] is an encryption scheme that allows performing operations on underlying messages without a secret key. In this paper, we use CKKS scheme [4] that allows approximate arithmetic and is widely employed in PPML studies.[2] The scheme has encryption/decryption algorithms ($\mathsf{Enc}/\mathsf{Dec}$) with the ring $\mathcal{R}$ as a plaintext space, i.e., for $m_1, m_2 \in \mathcal{R}$,

$$\mathsf{Dec}(\mathsf{Enc}(m_1) \boxplus \mathsf{Enc}(m_2)) \approx m_1 + m_2$$
$$\mathsf{Dec}(\mathsf{Enc}(m_1) \boxdot \mathsf{Enc}(m_2)) \approx m_1 \cdot m_2$$

where $\boxplus$ and $\boxdot$ denote ciphertext addition and multiplication, respectively. Here, $\approx$ implies that the coefficients of difference between the left and right sides are bounded by small values (the unique aspect of CKKS scheme).

1) Encoding/Decoding & Rotation: To utilize the plaintext space $\mathcal{R}$ for messages of complex numbers, CKKS scheme employs encoding ($\mathsf{Ecd}_\Delta : \mathbb{C}^{N/2} \to \mathcal{R}$) and decoding ($\mathsf{Dcd}_\Delta : \mathcal{R} \to \mathbb{C}^{N/2}$) algorithms parameterized by a scaling factor Δ (which controls the precision of following identity), satisfying, for $\mathbf{z}_1, \mathbf{z}_2 \in \mathbb{C}^{N/2}$,

$$\mathsf{Dcd}_\Delta(\mathsf{Ecd}_\Delta(\mathbf{z}_1) + \mathsf{Ecd}_\Delta(\mathbf{z}_2)) \approx \mathbf{z}_1 \oplus \mathbf{z}_2$$
$$\mathsf{Dcd}_\Delta(\mathsf{Ecd}_\Delta(\mathbf{z}_1) \cdot \mathsf{Ecd}_\Delta(\mathbf{z}_2)) \approx \mathbf{z}_1 \odot \mathbf{z}_2$$

where $\oplus$ and $\odot$ denote the component-wise addition and Hadamard product of vectors, respectively. With those algorithms, the CKKS scheme encrypts a *vector* of complex numbers into a ciphertext and operates additions/multiplications on those vectors via ciphertext operations. The scheme also has a rotation operation denoted by Rot. On input ciphertext ct, it outputs a ciphertext $\mathsf{ct}_{rot} := \mathsf{Rot}(\mathsf{ct}, j)$ having a message vector rotated j-steps left from the input vector, i.e.,

$$\mathsf{Dcd}(\mathsf{Dec}(\mathsf{ct}_{rot})) \approx (z_j, z_{j+1}, \ldots, z_{N/2-1}, z_0, \ldots, z_{j-1})$$

where

$$(z_0, z_1, \ldots, z_{N/2-1}) = \mathsf{Dcd}(\mathsf{Dec}(\mathsf{ct})).$$

2) Computational Cost of FHE: The cost of FHE for evaluating a given function mainly depends on the number of ciphertext multiplications and rotations required. Moreover, in the CKKS scheme, each ciphertext has a level (usually denoted by $l \in \{0, 1, \ldots, L\}$), and one can truncate some least significant digits of the messages by decreasing the level. Hence, performing computations of multiplicative depth L usually requires a ciphertext of level L as input. Ciphertexts of higher levels have a larger size and higher operating costs.

3) Bootstrapping: To continue operations on a ciphertext of level 0, bootstrapping [36] — which outputs a new ciphertext of level L having similar messages — must be performed on the ciphertext. It requires, however, heavy computations and has been mostly abandoned in prior work on PPML (by targeting inferences with low multiplicative depth). Luckily, recent improvements [37], [38], [39] on bootstrapping reduced its cost significantly, and we utilize it for deep CNN inferences.

[2]While we present our solution focusing on CKKS FHE, ours can also be utilized for other FHE schemes such as BGV [6] or B/FV [7], [8], especially when they are used as a subroutine for hybrid approaches, see Appendix D for a detail.

C. Homomorphic Convolution

We first recall the 2D convolution widely employed in Convolutional Neural Networks (CNN), then review the previous methods for evaluating it with FHE. From now on, the term *homomorphic convolution* stands for a method evaluating convolutions on messages encrypted with FHE.

1) 2D-Convolution: Let $I \in \mathbb{R}^{w \times w}$ be an input and $K \in \mathbb{R}^{k \times k}$ be a kernel. The single 2D convolution $\mathsf{Conv}(I, K) \in \mathbb{R}^{d \times d}$ ('VALID' padding) is defined by

$$\mathsf{Conv}(I, K)_{i,j} := \sum_{0 \le i', j' < k} K_{i',j'} \cdot I_{i+i', j+j'} \tag{1}$$

where the subscript i, j denotes the i-th row, j-th column component of a matrix, and $0 \le i, j < d := w - k + 1$.

We will focus on the *batch convolution* with BB' kernels $K^{(B,B')} := (K^{0,0}, \ldots, K^{i,i'} \in \mathbb{R}^{k \times k}, \ldots, K^{B-1,B'-1})$, which takes B-batch input $I^{(B)} := (I^0, \ldots, I^i \in \mathbb{R}^{w \times w}, \ldots, I^{B-1})$ then outputs B'-batch output as follows: for $0 \le b' < B'$,

$$\mathsf{Conv}(I^{(B)}, K^{(B,B')})^{b'} := \sum_{0 \le i < B} \mathsf{Conv}(I^i, K^{i,b'}) \tag{2}$$

where the superscript b' and i denote the b'-th and i-th batch, respectively.

2) Homomorphic Convolution in Previous Works: In prior works [15], [16], and [17], given encrypted input vector as one ciphertext, single convolution was done by generating $k \times k$ (the size of a single kernel) rotated input ciphertexts, then multiplying each with the corresponding kernel element, and summing them up; see Fig. 2 left. As a result, it requires $k^2 - 1$ rotations and multiplications. The cost for batch convolution with B^2 kernels is given at Vector Encoding in Table I: $k^2 B$ mults for generating B summands of Eq. (2), and B rotations for the summation (more detail is in [15]).

References [22] and [21] proposed to evaluate convolutions in the frequency domain to reduce the number of FHE rotations required. They observed that $\mathsf{Conv}(I, K)$ could be extracted from the product of polynomials $I(t)$ and $K(t)$ whose coefficients are related to I and K, respectively. Then, Discrete Fourier Transform (DFT) mapping a polynomial to a vector gives that $\mathsf{DFT}(I(t) \cdot K(t)) = \mathsf{DFT}(I(t)) \odot \mathsf{DFT}(K(t))$ where $\odot$ denotes the Hadamard product. From this, they proposed to encrypt the vectors $\mathsf{DFT}(I(t))$ and $\mathsf{DFT}(K(t))$ so that $\mathsf{DFT}(I(t)) \odot \mathsf{DFT}(K(t))$ can be computed with one ciphertext multiplication; see Fig. 2 right. Their solution, however, requires an interaction with the secret key holder for each convolution in CNN. To remove interactions, [20] evaluated DFT and DFT^{-1} also with FHE. Still, the cost of DFT and DFT^{-1} are problematic for CNNs having a large size of intermediate layers; see Spectral Encoding in Table I.

IV. ConvFHE

We present our method ConvFHE evaluating convolution with FHE at minimal cost, which leads to an efficient CNN evaluation. We start from the same observation of [22] and [21] described above but *remove* DFT and DFT^{-1} via novel packing methods of input and kernel into *coefficients* of plaintext polynomials in FHE.

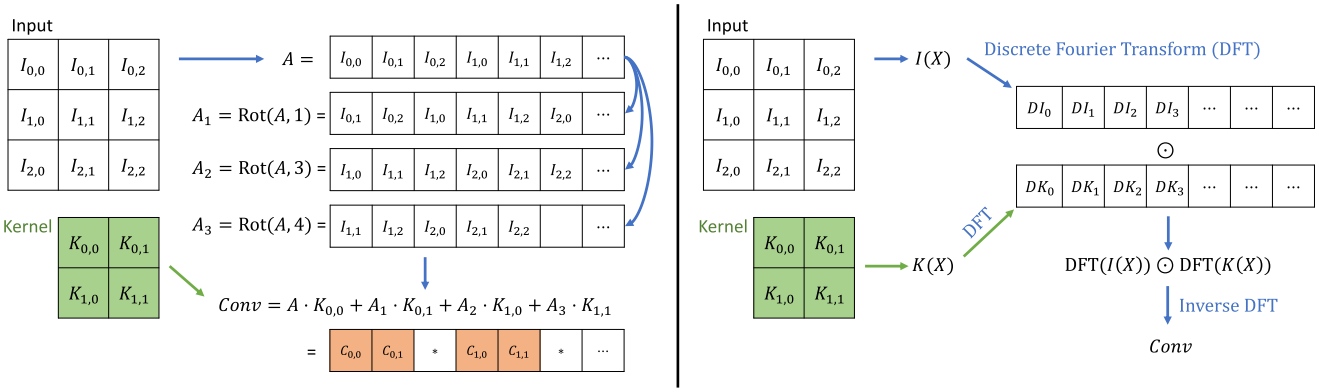

Fig. 2. Prior homomorphic convolution methods: Vector Ecd [15], [16], [17] in the left side & Spectral Ecd [20], [21], [22] in the right side.

TABLE I

COST OF HOMOMORPHIC CONVOLUTION IN VARIOUS METHODS. (B: INPUT/OUTPUT BATCH NUMBERS), (k: WIDTH OF KERNEL), (w: WIDTH OF INPUT)

METHOD	MULTS	ROTATIONS	# OF MESSAGES IN A CTXT	MULT DEPTH	# ROT KEYS($\approx$)
VECTOR ENCODING [15]	k^2B	k^2+B-2	$N/2$	1	k^2+B
SPECTRAL ENCODING [20]	$B+2$	$B-1+4\log_2 w$	$N/2w^2$	3	$B+\log_2 w^2$
OURS	$\mathbf{3B-2}$	$\boldsymbol{B-1}$	$\boldsymbol{N}$	**1**	$\mathbf{\log_2 B}$

A. Coefficient Encoding of Messages

Recall that the plaintext space of CKKS scheme is $\mathcal{R} := \mathbb{Z}[X]/(X^N+1)$. Instead of using the usual encoding/decoding (Ecd/Dcd in Section III-B) of CKKS scheme, we propose to encode/decode the message vector directly into/from coefficients of a plaintext polynomial as follows:

- $\mathsf{Cf\text{-}Ecd}_\Delta((r_0, r_1, \ldots, r_{N-1}) \in \mathbb{R}^N)$
  $\rightarrow \lfloor \Delta \cdot (r_0 + r_1 X + \cdots + r_{N-1}X^{N-1}) \rceil \in \mathcal{R}$
- $\mathsf{Cf\text{-}Dcd}_\Delta((m_0 + m_1 X + \cdots + m_{N-1}X^{N-1}) \in \mathcal{R})$
  $\rightarrow (m_0/\Delta, m_1/\Delta, \ldots, m_{N-1}/\Delta) \in \mathbb{R}^N$

where $\lfloor \cdot \rceil$ means that each coefficient is rounded to the nearest integer. Note that our encoding allows homomorphically computing polynomial addition/multiplication containing $\boldsymbol{N}$ real numbers as coefficients, while the usual CKKS encoding allows operations between vectors containing $N/2$ real (or complex) numbers only.

B. Concise Representation of Convolutions

We will show that (batch) convolution $\mathsf{Conv}(I, K)$ can be represented by a product of two plaintext polynomials in $\mathcal{R}$. We assume, for simplicity, that the size of input I and kernel K does not exceed the degree bound N, so that each input and kernel can be encoded into a single plaintext (contained in one ciphertext). We can regard input and kernel as having integer values with our encoding/decoding (Section IV-A). We use the notation for convolution introduced in Section III-C.

1) Single Convolution: A single convolution (Eq. (1)) can be represented by a multiplication of two polynomials in $\mathcal{R} := \mathbb{Z}[X]/(X^N+1)$ as follows.

Proposition 1: For input $I \in \mathbb{Z}^{w\times w}$ and kernel $K \in \mathbb{Z}^{k\times k}$, assume that $\max(w^2, k^2) \le N$. Let $I(X)$ and $K(X) \in \mathcal{R}$ be defined as follows:

$$I(X) := \sum_{0\le i,j<w} I_{i,j} \cdot X^{(i-k)w+j}$$
$$K(X) := \sum_{0\le i,j<k} K_{i,j} \cdot X^{wk-(iw+j)}$$

where X^t means $-X^{N+t}$ when $t < 0$. Then, $(iw+j)$-th coefficient of $I(X)\cdot K(X)$ (computed in $\mathcal{R}$) is $\mathsf{Conv}(I, K)_{i,j}$.

Proof: The $(iw+j)$-th coefficient of $I(X)\cdot K(X)$ is the summation (over $0 \le i', j' < k$) of $K_{i',j'}$ multiplied by $(iw+j-wk+i'w+j') = ((i+i'-k)w+j+j')$-th coefficient of $I(X)$ which is $I_{i+i',j+j'}$. Note that $I(X)$, $K(X)$ and their product is defined over $\mathcal{R}$ where $X^N = -1$, and X^t for $t<0$ is defined so that the sign of each summand is consistent with that of Eq. (1) in Section III-C. ■

Recently, [33] and [34] observed the same proposition and extended it to batch convolution. In their method, however, the output ciphertext must be decrypted after each convolution to provide the correct result [33],[3] or it contains less convolution output [34] than ours. Therefore, their method can only provide an efficient CNN evaluation *with interaction*, contrary to ours. Before introducing our batch convolution, we remark that Theorem 1 easily extends to the case with *sparsely-packed* polynomials as well:

Corollary 1: For input $I \in \mathbb{Z}^{w\times w}$, kernel $K \in \mathbb{Z}^{k\times k}$, and a positive integer s, assume $\max(sw^2, sk^2) = N$. Let $I_{sp}(X) := I(X^s)$ and $K_{sp}(X) := K(X^s)$ where $I(X)$ and $K(X)$ are from Theorem 1. Then, $s(iw+j)$-th coefficient of $I_{sp}(X)\cdot K_{sp}(X)$ (computed in $\mathcal{R}$) is $\mathsf{Conv}(I, K)_{i,j}$.

[3] In [33], the form of output ciphertext is quite different from the usual FHE ciphertext, and we cannot apply usual ciphertext operation nor bootstrapping on it.

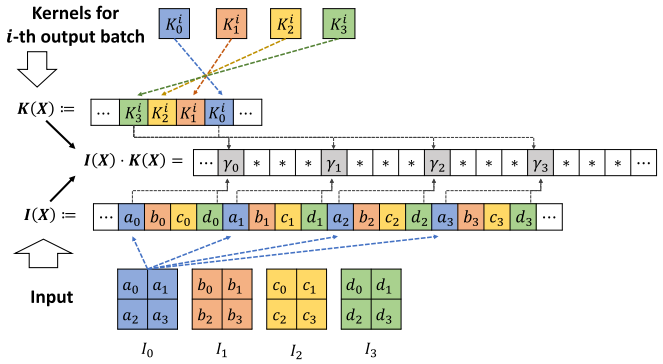

Fig. 3. An example description of our packing method for batch convolution with $B = 4, w = 2, k = 1$. Coefficients of each polynomial are depicted according to its order, and $\gamma_j = \mathsf{Conv}(I, K)^i_j$.

Note that the l-th coefficient of $I_s(X) \cdot K_s(X)$ is nonzero only if $l = 0 \bmod s$, i.e., l is divisible by s. We will call such polynomials as *sparsely-packed*.

2) Batch Convolution: Recall that the batch convolution is a summation of single convolutions (Section III-C, Eq. (2)). Our method for batch convolution encodes batch inputs and kernels into one polynomial each; then the convolutions and their summations are computed via one multiplication between two polynomials. The crux is to use the *sparsely-packed* $I_{sp}(X)$ and $K_{sp}(X)$ from Theorem 1 for every single input and kernel so that convolutions between each batch are computed separately and then summed into one. See Fig. 3 and Algorithm 1.

Theorem 1: [Correctness of Algorithm 1] For batch input $I^{(B)} \in \mathbb{Z}^{w\times w\times B}$ *and kernels* $K^{(B,B)} \in \mathbb{Z}^{k\times k\times B\times B}$, *assume that* $\max(w^2 B, k^2 B) = N$. *For* $b \in \{0, 1, \ldots, B-1\}$, *let* $r_b(X)$ *be the output of Algorithm 1. Then, the* $B(iw + j)$*-th coefficient of* $r_b(X)$ *is* $\mathsf{Conv}(I^{(B)}, K^{(B,B)})^b_{i,j}$.

Proof: Note $r_b(X) = \sum_{0\le i,j<B} I_i(X) \cdot K_j(X)$, and from Theorem 1, the product $I_i(X) \cdot K_i(X)$ has $\mathsf{Conv}(I^i, K^{i,b})_{i',j'}$ as its $B(i'w + j')$-th coefficient where $0 \le i', j' < d := w - k + 1$. On the other hand, if $i \ne j$, the l-th coefficient of $I_i(X) \cdot K_j(X)$ is nonzero only if $l = i - j \bmod B$, i.e., it is zero if l is divisible by B. Thus, among summands of $r_b(X)$, only $\sum_{0\le i<B} I_i(X) \cdot K_i(X)$ contributes to the $B(i'w + j')$-th coefficients of $r_b(X)$, and the claim follows from Eq. (2). ■

Note that Algorithm 1 outputs *one* output among B batches constituting the result $\mathsf{Conv}(I^{(B)}, K^{(B,B)})$, and we can run it B-times for each $b \in \{0, 1, \ldots, B-1\}$ to get the full result. The problem, however, is that the output is composed of B polynomials of $\mathcal{R}$, separately contained in B ciphertexts. We resolve this problem by generalizing Chen et al.'s recent algorithm [5] and then using it for packing desired coefficients of these polynomials into one polynomial. Here, we presented the consequence of it and left the complete description to Appendix A:

Theorem 2 (Generalization of [5]): Let $\mathsf{PackLWEs}$ *be the packing algorithm described as Algorithm 2 in Appendix A. Let* $\{\mathsf{ct}_b\}_{0\le b<B}$ *be the ciphertexts each of which having* $r_b(X)$ *(output of Algorithm 1) as a plaintext polynomial. Then,* $\mathsf{ct} \leftarrow \mathsf{PackLWEs}(\{\mathsf{ct}_b\}_{0\le b<B})$ *has a plaintext polynomial* $r(X)$ *which satisfies that*

$$(iB+b)\text{-th coefficient of } r(X) = i\text{-th coefficient of } r_b(X).$$

Hence, $(B(iw + j) + b)$*-th coefficient of* $r(X)$ *is equal to* $\mathsf{Conv}(I^{(B)}, K^{(B,B)})^b_{i,j}$.

Algorithm 1 Batch Convolution (BatchConv)

Input:
- input $I^{(B)} := (I^0, I^1, \ldots, I^{B-1}) \in \mathbb{Z}^{w\times w\times B}$
- kernel $K^{(B,B)} := (K^{(i,j)})_{0\le i,j<B} \in \mathbb{Z}^{k\times k\times B\times B}$
- out batch index $0 \le b < B$

Output:
$r_b(X) \in \mathcal{R}$ having $\mathsf{Conv}(I^{(B)}, K^{(B,B)})^b$ as coefficients

Procedure: *(All polynomials and operations are in $\mathcal{R}$)
Set $s \leftarrow B$
for $i = 0$ **to** $B - 1$ **do**
  $I_s(X), K_s(X) \leftarrow$ *Theorem 1* with s, I^i, and $K^{i,b}$
  $I_i(X) \leftarrow I_s(X) \cdot X^i$
  $K_i(X) \leftarrow K_s(X) \cdot X^{-i}$
end for
$I(X) \leftarrow \sum_{i=0}^{B-1} I_i(X)$
$K(X) \leftarrow \sum_{i=0}^{B-1} K_i(X)$
Output $r_b(X) \leftarrow I(X) \cdot K(X)$

In retrospect, we designed Algorithm 1 so that each output is located in the B-strided position among coefficients and Algorithm 2 (in Appendix A) can be applied to pack those ciphertexts efficiently into one ciphertext. Now, we can summarize the cost of our method for batch convolution as follows (see **Ours** in Table I):

- It cost B multiplications and one multiplicative depth to get $\{r_b(X)\}_{0\le b<B}$ (one mult for each $r_b(X)$).
- Packing B ciphertexts into one via $\mathsf{PackLWEs}$ costs $2(B\text{-}1)$ multiplications and B-1 rotations without consuming depth (see Appendix A).

Remark 1 (Toward CNN): The output polynomial $r(X)$ has the same packing structure as the input $I(X)$ of Algorithm 1, and it can contain up to N-convolutional outputs utilizing all the coefficients. Therefore, our algorithm can also evaluate convolutions having $r(X)$ as input, which implies that it is appropriate for evaluating CNNs having many consecutive convolutions. More detail will be described in Section IV-C.

Remark 2 (Convolution With Padding/Strides): Ours can evaluate padded convolution by encoding the input properly with zeros: e.g., for k-width kernels, adding $\frac{k+1}{2}$-rows and columns of zeros to input is sufficient to evaluate 'SAME' padding. For consecutive (strided) convolution operations, we must extract the desired output after evaluating each convolution. It can be done at little cost when we evaluate convolutional layers with bootstrapping; see Section IV-C.

Remark 3 (Scalability): Our method scales easily to the case with more batches since the summation for batch convolution (Eq. (2)) can be divided into that with less number of batches. For example, if $\max(w^2 B, k^2 B) = cN$ for an integer

c, we can divide them into $\lfloor B/c \rfloor$ batches and then apply our method c^2 times (then sum up the result). The previous method (e.g., Vector Encoding) scales similarly but requires using smaller batches satisfying $\max(w^2B', k^2B') \leq N/2$.

3) Comparison to Previous Work: In Table I, we compare the cost of our method with previous homomorphic convolutions. In Spectral Encoding [20], while the cost of convolution itself is almost the same as ours, it requires additional DFT and IDFT evaluations. They not only consume one multiplicative depth each but also require encoding each input stacked w^2 times to reduce the rotations to be $\log_2 w^2$ each. Therefore, one can pack only $N/2w^2$ valid number of elements in one ciphertext, while ours can pack N many messages. On the other hand, Vector Encoding [15] — which can pack $\times 2$ *fewer* messages than ours — requires $\times k^2$ *more* multiplications than ours. Our method (with Algorithm 2) requires much fewer rotation keys than previous ones; $B \in [2^4, 2^9]$ in usual CNNs.

Remark 4 (Cost in Practice): We use the full-RNS CKKS scheme [40], where the rotation is more costly than multiplication. Rotations in our method, however, are applied on the ciphertext of *level 0* after multiplications are performed on the input ciphertext of *level 1*, and the multiplications dominate the actual cost. In the Vector Encoding method, rotations (w/ hoisting [15]) and multiplications are on the input ciphertext of *level 1*. Therefore, the cost of convolution is roughly expected to be $\times 1/k^2$ or less in our case compared to the Vector Encoding method. In Section V-A, we validate this estimation with a micro-benchmark.

Remark 5 (Sparse Packing for Smaller Inputs): In our method, when the total size Bw^2 of input is relatively small and is equal to $N/2^s$ for an integer $s > 0$, we can pack the input more sparsely using $I(X^{2^s})$ instead of $I(X)$ in Algorithm 1. Then, the output polynomials become $r_b(X^{2^s})$ instead of $r_b(X)$. It not only reduces the number of rotations in PackLWEs (Algorithm 2, Appendix A) but also makes the final output polynomial to be $r(X^{2^s})$ instead of $r(X)$. Note, in this case, that the cost of bootstrapping (Section III-B) — which is necessary for the CNN evaluation in the following section — on $r(X^{2^s})$ is considerably lower than that on $r(X)$.

C. Evaluation of Convolutional Layers

We introduce the way of evaluating convolutional layers (of CNN) with our homomorphic convolution (described previously). The problem is that our method based on coefficient encoding is not appropriate for evaluating an activation function following a convolution in each convolutional layer. Luckily, we can resolve this problem when we want to evaluate the activation function with high precision, which requires bootstrapping (Section III-B) on each layer. In this case, we can modify and combine the bootstrapping procedure with our coefficient encoding method to efficiently evaluate both the convolution and activation functions.

1) Architecture Overview: Our homomorphic convolutional layer proceeds as follows, performing a bootstrapping at each layer. Fig. 4 describes our method and the Vector Encoding method when combined with bootstrapping (the latter was exploited in recent work [29]). At first, we apply our convolution on the input ciphertext of level 1 to get a ciphertext having the result as the coefficients of its plaintext. Then we perform CtoS and EvalSine step of bootstrapping on the ciphertext, which outputs two ciphertexts having the CKKS encoding polynomials that encode the coefficients of the plaintext of the input ciphertext as *vectors*. Now, we can homomorphically evaluate an approximate polynomial of the activation function (e.g., ReLU) on these ciphertexts since the inputs are encoded as vectors (via CKKS encoding), allowing component-wise evaluation. Here, we can also extract (Ext step in Fig. 4) valid values among the output to represent strided or usual convolutions. Finally, via the StoC step of bootstrapping, the result is encoded back to the coefficients of the plaintext (of an output ciphertext) so that the subsequent convolution can be applied to it.

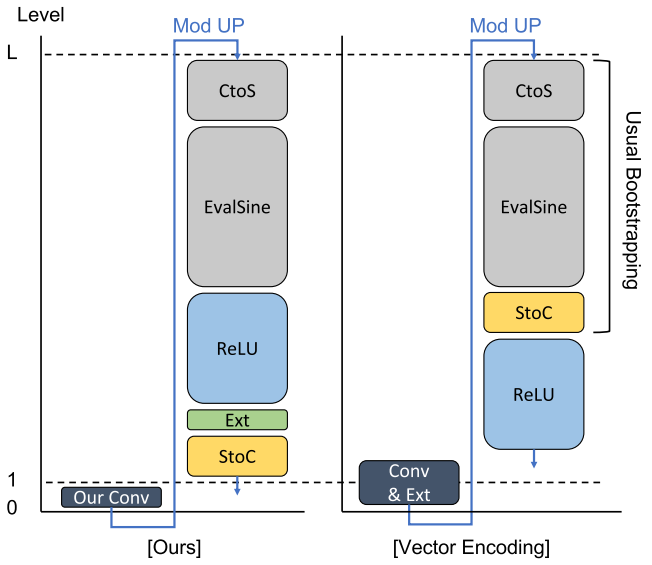

Fig. 4. Evaluation of a convolutional layer with ReLU activation via Bootstrapping. Left axis: the level of input/output ciphertext at each step.

Remark 6 (Comparison to Prior Work): [29] presented that bootstrapping is required at each layer to evaluate deep CNN with FHE in higher precision (so that the accuracy is almost identical to that of the plain CNN). Therefore, if high precision is required, our method necessitating bootstrapping at each layer does not impose any more restrictions than the prior work based on Vector Encoding. Ours — having significant improvement on convolutions — also shows slightly more efficient bootstrapping since the final StoC step which is more costly than EvalSine and ReLU is applied to the ciphertext with a lower level (Fig. 4). See Section V-A for the actual timing comparison.

2) Extracting Valid Values for (Strided) Convolution: Recall that the output $r(X)$ of our homomorphic convolution has the same packing structure as the input $I(X)$ (Remark 1). Still, $r(X)$ also contains invalid values (note, in general, that the size of Conv output is smaller than input, e.g., Eq. (1)), and we must extract the valid ones. It can be done efficiently by multiplying, at the Ext step where the messages are encoded as

a vector (instead of as coefficients), a plaintext vector having 1 at valid positions and 0 otherwise. Ext step is a bit more complicated in strided convolution since we need to modify the packing structure of $r(X)$. Still, it can be done efficiently at an insignificant cost; the details are given in Appendix C.

3) Approximating Activation Functions With High Precision: For the approximation of ReLU activation, we use a higher degree polynomial instead of the square function x^2 from many previous works. This is necessary to get precise results in deep CNNs with FHE. Recently, [41] showed that ReLU could be approximated by *compositing* low-degree polynomials, which results in a significantly faster evaluation of ReLU in the CKKS FHE with high precision. We use one of such approximate polynomials for ReLU found by [42], optimized at level consumption achieving 10-bit precision for the input contained in $[-1, 1]$. By inspecting the maximum of intermediate layer output values (which is not very large due to batch normalization), we can scale all output values into this interval, sacrificing some precision. Further details are given in Appendix B.

D. Applications and Security Model

As an example use-case, assume a client outsourcing the CNN inference on his or her input data to a service provider with a CNN model. Our solution can be used to implement the secure inference as in other previous works: the client encrypts the input data with FHE, and the service provider performs the CNN evaluation on the ciphertexts. In this case, the semantic security of FHE guarantees that no information on the client's data (except its size) is leaked to the service provider. Our solution can also be used when both the CNN model and input data are encrypted and a computing party performs the evaluation. Again, the party cannot get any information on the input data or the weights of the CNN model. Here, we need to assume that the party knows the CNN model structure, such as the number of layers and batches, as in other previous works presented in II. Note, in our method, that the computing party does not need to know the size of the kernel since the FHE evaluation is independent of it. In contrast, previous works require the party to know that information.

V. Experimental Evaluation

To quantify the effectiveness of our methods, we set the Vector Encoding method [15] (Section III-C) as a Baseline and then compared its timing with ours. Vector Encoding was utilized in both the famous work [15] and the recent work [29] to evaluate CNN models with FHE. For a fair comparison, we (re-)implemented both the Vector Encoding and our method in the Lattigo library [43] which supports CKKS scheme with bootstrapping.[4] Source code is available.[5]

We compared the evaluation of (i) convolution, (ii) convolutional layer with ReLU activation, and (iii) CNN classifiers on CIFAR10/100 and ImageNet dataset. All experiments were conducted by running a single thread on AMD EPYC 7402P CPU with a 2.8GHz processor and 512GB of memory ($\leq$100GB was utilized for all experiments).

[4]The implementation of [29] nor the bootstrapping they used is not publicly available.

[5]https://anonymous.4open.science/r/optimal_conv-BD07

TABLE II

FHE Parameters: $N = 2^{16}$, $\Delta = 2^{30}$, Key-Switching Modulus ($\log p_j$'s) $= 61 \cdot 5$, Secret-Key Sparsity $h = 192$

Set	$\{\log q_i\}_{0 \leq i < 28}$
Baseline	Conv(&Ext) + ReLU + StoC + Sine + CtoS $(55 + 60) + (30 \cdot 11 + 60) + 60 \cdot 2 + 55 \cdot 8 + 53 \cdot 4$
Ours	Conv + StoC + Ext + ReLU + Sine + CtoS $(55 + 49) + 60 \cdot 2 + 42 + 30 \cdot 11 + 55 \cdot 8 + 53 \cdot 4$

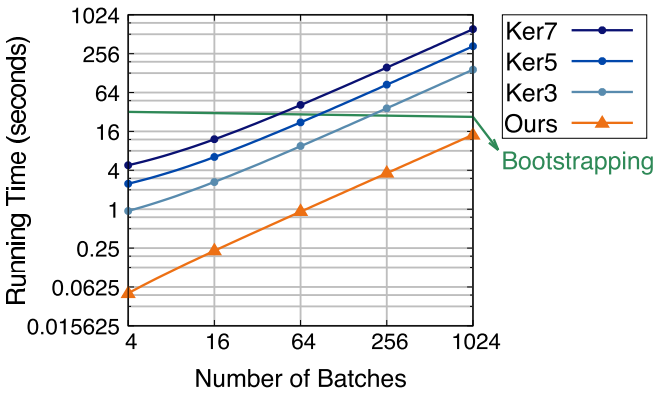

Fig. 5. Homomorphic convolution timing: Ker3, Ker5, Ker7 denote Baseline with corresponding kernel width; Ours is constant on kernel width. Both axes are log scaled.

A. Micro-Benchmark

1) FHE Parameters and Precision: We adapted an FHE parameter set [39] with 128-bit security into two FHE parameter sets, each of which was used for the baseline (Vector Encoding [15], [29]) and our methods, respectively, modifying the order of ciphertext modulus q_i; see Table II. We set the scaling factors for input and plaintexts to $\Delta = 2^{30}$ (except 2^{21} for extracting valid values for (strided) convolution).

2) Convolution With FHE: Since every convolution of CNN takes input ciphertext of level 1 according to our architecture (Section IV-C), we set input ciphertext as such level, i.e., containing the $\log q_i$'s for Conv ($55 + 60$ or $55 + 49$) only among the q_i's of parameters in Table II. We measured the evaluation timing of full-batched convolutions (Eq. (2)) on it with varying input batches B and the same number of output batches. For example, in our method, the full message slot is $N = 2^{16}$, and we set the batches $B = 4, 16, 64, \ldots$ with the input of width $w = 2^7, 2^6, 2^5, \ldots$. In the baseline (BL), the full message slot is $N/2$, and we iterated four times over two input ciphertexts to get the same result as ours (Remark 3). In Fig. 5, we present the timing result of both methods varying the width of kernels as $3, 5, 7$. Our method shows roughly $\times 12, 25, 46$ faster evaluation than the baseline when the kernel width is $3, 5, 7$, respectively, which follows our rough estimation in Table I and Remark 4. Our method shows *constant* timing for varying kernel sizes, as expected. We also mention that when the batch or kernel

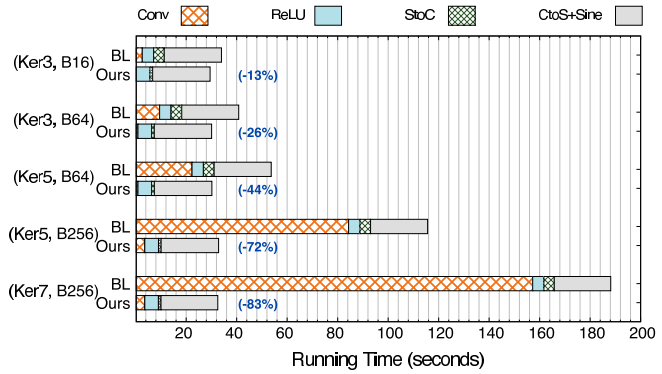

Fig. 6. Homomorphic convolutional layer evaluation timing: numbers following Ker and B denote the kernel width and input/output batch numbers, respectively.

size is large, the convolution in the baseline method costs even more than a bootstrapping. In such cases, our method is preferable to homomorphic encryption *without* bootstrapping, even if precise activation is not required.

3) Convolutional Layer With FHE Bootstrapping: The timing for a convolutional layer — a batch convolution as above followed by (approximate) ReLU activation — is given in Fig. 6. Recall that the evaluation accompanies a bootstrapping on a ciphertext. We optimized the baseline (BL) so that the messages of two input ciphertexts are packed into real and imaginary part of one ciphertext, and only one bootstrapping was performed. Ours reduces the total time by at least 13% to 83% as the convolution timing becomes significant with a larger size of the kernel and/or number of batches.

B. Evaluation of CNN Classifiers

We now present the impact of our improved homomorphic convolution for evaluating deep CNN classifiers.

1) CNN Classifiers on CIFAR10/100 and ImageNet Dataset: With our FHE-based secure inference method, we evaluated the plain-20 CNN [44] classifier — composed of 19 convolutional layers with ReLU activation and 1 fully connected layer — which is the deepest model tested in the previous work [27], [28], [29].[6] For comparison, we chose the most recent method [29] as a baseline and compared it with ours. The detailed CNN architecture and dataset description are given in Appendix E.

On CIFAR10 and CIFAR100 datasets, we measured the evaluation timing of the whole CNN in FHE. On ImageNet dataset, we only evaluated the final 8 layers with FHE assuming Deep Representation method [19] where early layers are evaluated by a client without FHE; note that the final 8 layers contain more than 91% (96% in Ker5 variant) of the entire parameters of the CNN which can be kept private for a service provider. For the baseline, we *favorably* estimated its timing by summing up the timing for each convolutional layer.[7] On our method, we fully implemented the CNNs and measured the end-to-end timing. We optimized our method by employing sparse packing (final Remark in Section IV-B) and faster bootstrapping on it when the size of intermediate outputs is less than $N = 2^{16}$.

TABLE III

RUNNING TIME (IN SECONDS) OF THE VARIANTS OF CNN CLASSIFIERS WITH FHE IN OURS AND BASELINE. KER3 CORRESPONDS TO THE ORIGINAL PLAIN-20 AND PLAIN-18 OF [44], RESP

	Plain-20 on [CIFAR10, CIFAR100]				Final 8 layers of Plain-18 on [ImageNet]		
	Ours	Baseline [29]			**Ours**	Baseline [29]	
		Ker3	Ker5	Ker7		Ker3	Ker5
Conv	**10**	51	118	220	**53**	255	583
ReLU	51	41			35	22	
Boot	303	359			186	269	
FC	4	4			17	17	
Total	**368**	454	522	623	**292**	563	891
Imp. (%)		**18.9**	**29.5**	**40.9**		**48.1**	**67.2**

2) Running Time Reduction in CNN Classifiers: Table III summarizes the comparison results: column 'Ker3' denotes the original CNN, and our method reduces 18.9% and 48.1% of the total timing on CIFAR10/100 and ImageNet classification, respectively, from the baseline. Note that ours effectively reduces the cost of convolutions (**Conv** in the table), and its impact on the overall timing increases as the convolutions take up more portion; e.g., we refer to the evaluation timing of the same network with larger kernel width 5 or 7 — denoted by Ker5 or Ker7 in the table. In our method, even if the kernel width changes, the running time remains the same. Moreover, our result achieves almost the same accuracy as the original CNN without FHE; see Table IV, column 'k3-d20-w1' and 'Ker3'.

3) Exploring Variants of CNN: The CNN evaluation timing in FHE increases linearly with the depth of the network (i.e., the number of total layers) since each layer requires a bootstrapping step which takes up a major portion of the overall running time. Interestingly, [45] observed that wider layers (having a larger number of batches) could considerably increase the accuracy of the classifier without increasing the depth. Motivated by this observation, we explored CNNs having lower depth, wider layers, and a larger kernel widths, which are more favorable to our FHE evaluation — having constant running time for kernels with a larger width.

We trained all possible variants of CNNs having different depths (8, 14, 20), wideness factors (1,2,3), and kernel widths (3,5,7), then measured their accuracy; see Appendix E for details. Fig. 7 summarizes the results. In general, a wider and deeper CNN shows better accuracy. Notably, CNNs with lower depths shows better accuracy when combined with larger kernel widths. This phenomenon can be partially explained by the fact that a convolution with kernel width 5 (or 7) takes the same range of input as two (or three) consecutive ones

[6]Reference [29] also tested deeper variant having more layers, which required higher precision and more timing in FHE. Their result, however, achieved only 1.6% higher accuracy than the original CNN. While ours can also evaluate deeper variant setting higher precision in FHE, we instead investigate more interesting variants of CNNs that achieves higher accuracy without increasing the depth.

[7]I.e., we assume that the strided convolutions don't require additional timing than usual one via [30].

TABLE IV

CLASSIFICATION ACCURACY ON CIFAR10/100 (AND TOP-5 ACC. ON IMAGENET) AND RUNNING TIME OF CNN CLASSIFIERS WITH FHE IN OURS. FOR EXAMPLE, A MODEL 'K5-D8-W3' DENOTES THE VARIANT WITH KERNEL WIDTH 5, DEPTH 8, AND WIDENESS FACTOR 3. RESULTS WITH † MARK ARE TESTED OVER 2000 SAMPLES ONLY; OTHER RESULTS ARE OVER THE FULL TEST SET

Model		k3-d20-w1	k5-d8-w3	k3-d14-w3	k3-d20-w3	Ker3	Ker5
Dataset		CIFAR10 (CIFAR100)				ImageNet	
Accuracy (%)	enc	90.32 (64.08)	92.04 (69.05)	93.99 (**73.47**)	**94.12** (72.65)	87.65†	**88.05**†
	plain	90.39 (64.13)	92.37 (69.57)	94.04 (73.65)	94.3 (72.95)	87.92	88.22
Running time (sec)		368	**255**	398	544	292	

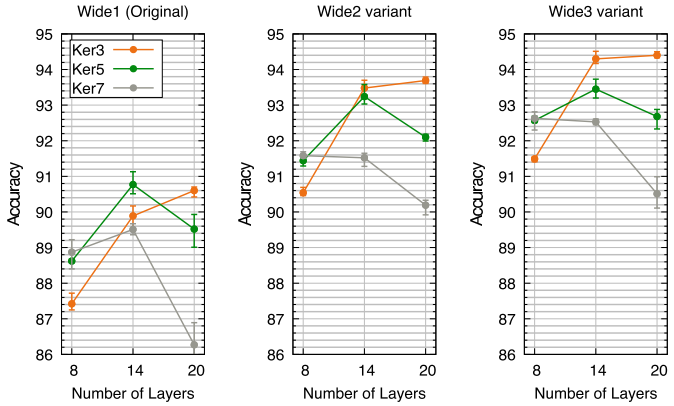

Fig. 7. Accuracy of the variants of CNNs on CIFAR10: averaged over 3 training runs.

with width 3, thereby negating the effect of reduced depth. The same tendency was observed in CIFAR100 task as well. On ImageNet, we additionally tested Ker5 variant without changing the depth or wideness.

Table IV describes some of the best results (in terms of accuracy & timing) of CNN variants when evaluated with our method in FHE. Compared to the original CNN — 'k3-d20-w1' in the table, the variants show better timing and accuracy trade-offs, and some of them even achieves higher accuracy in *less* running times on both CIFAR10 and CIFAR100 datasets. On ImageNet, the variant (Ker5) with higher accuracy can be evaluated in the same running time, thanks to ours. Some of our best results on CIFAR10 dataset are also depicted in Fig. 1 in Section II for comparison to the related work. Our result shows the best timing and accuracy trade-offs.

VI. CONCLUSION

We presented improved methods for evaluating convolutions and CNNs with FHE. Our method — having constant and less evaluation timing irrespective of the kernel size — can boost secure and precise evaluation of many other deep CNNs. Moreover, we open up a new possibility in FHE-based inference by showing that CNNs with larger kernels and lower depth can have significantly better timing without accuracy degradation, especially when implemented with our FHE methods. On the other hand, our solution can also be employed in various other applications of FHE, e.g., an hybrid PPML inferences (Appendix D). We also mention that our result can be more practical along with the recent studies showing that FHE (and its bootstrapping) can be accelerated, roughly to $\times 100$, with GPU implementation [46], [47].

APPENDIX

A. Algorithm PackLWEs

The explicit description of PackLWEs is given in Algorithm 2. We generalize the algorithm of [5] by adding an initial log step s that corresponds to the log2 of output batch numbers; original algorithm considers $s = \log_2 N$ only. In the description, we substituted the $\mathsf{EvalAuto}(\cdot, k)$ of [5] by $\mathsf{Rot}(\cdot, \rho_5(k))$ to emphasize that they are equivalent. The algorithm satisfies the following.

Algorithm 2 Homomorphic Packing of LWE Ciphertexts (PackLWEs [5])

*($[n]$ denotes the set $\{0, 1, \ldots, n-1\}$)
*($\rho_5(k)$ denotes the discrete log of k with base 5 in $\mathbb{Z}_{2N}^{\times}$, i.e., the number such that $5^{\rho_5(k)} = k \bmod 2N$)

Input: ciphertexts $\mathsf{ct}_i \in \mathcal{R}_q^2$ for $i \in [2^l]$, an initial log step $s \in \mathbb{Z}$ satisfying $s \geq l$.
if $l = 0$ **then**
  return $\mathsf{ct} \leftarrow \mathsf{ct}_0$
else
  $\mathsf{ct}_{even} \leftarrow \mathsf{PackLWEs}(\{\mathsf{ct}_{2j}\}_{j \in [2^{l-1}]})$
  $\mathsf{ct}_{odd} \leftarrow \mathsf{PackLWEs}(\{\mathsf{ct}_{2j+1}\}_{j \in [2^{l-1}]})$
  $\mathsf{ct} \leftarrow (\mathsf{ct}_{even} + X^{2^{s-l}} \cdot \mathsf{ct}_{odd}) + \mathsf{Rot}_{\mathsf{evk}}(\mathsf{ct}_{even} - X^{2^{s-l}} \cdot \mathsf{ct}_{odd}, \rho_5(N/2^{s-l}+1))$
  return ct
end if
Output: ct

Proposition 2 (Correctness and Cost of PackLWEs, Algorithm 2 [5]): Assume that each ciphertext $\mathsf{ct}_i (0 \leq i < 2^l)$ has a plaintext $m_i(X) := m_{i,0} + m_{i,1}X + \cdots + m_{i,N-1}X^{N-1} \in \mathcal{R}$ satisfying that

$$m_{i,j \cdot 2^s} = \mu_{i+j \cdot 2^s} \quad \text{for } 0 \leq j < N/2^s.$$

Let $n := 2^l$. Then, the output ciphertext ct *of Algorithm 2 has a plaintext $\mu(X) := n(\mu_0 + \mu_1 X + \cdots + \mu_{N-1}X^{N-1}) \in \mathcal{R}$. In other words, the plaintext of output ciphertext collects $j \cdot 2^s$-th coefficients only from each plaintext polynomial $m_i(X)$ of input ct_i. The algorithm requires $n-1$ rotations and $2(n-1)$ plaintext multiplications without consuming any multiplicative depth.*

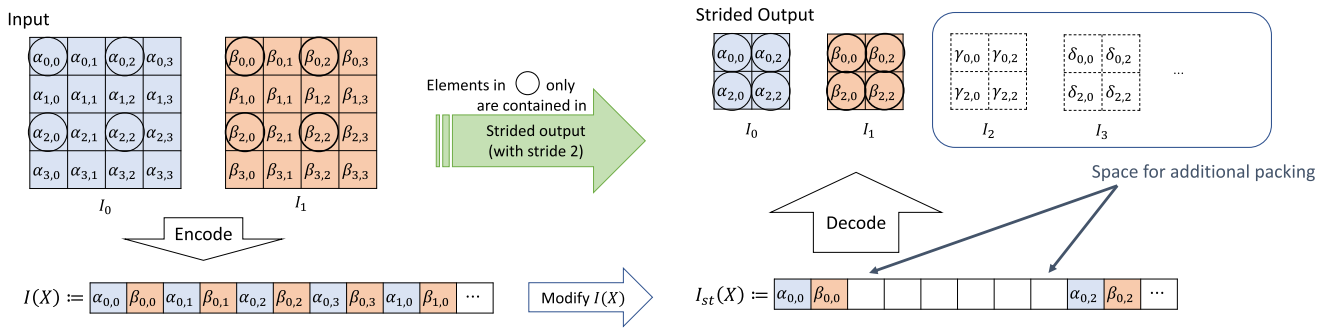

Fig. 8. Extracting strided output from the input encoded with our method (Section IV-B). Example with Input width 4 and batch 2.

Proof: It follows from the fact that $\mathsf{Rot}_{\mathsf{evk}}(\mathsf{ct}, \rho_5(N/2^\ell + 1))$ preserves the i-th coefficient of the underlying plaintext polynomial if i is divisible by $2^{\ell+1}$ and changes only the sign of it if i is divisible by 2^ℓ but not by $2^{\ell+1}$. This can be interpreted more naturally with automorphisms on $\mathcal{R}$ whose detail can be found in [5]. The cost follows directly from the description. We only mention that the multiplication of $X^{N/2^l}$ does not consume multiplicative depth, since it does not increase the scaling factor of the ciphertext (and of its plaintext). ∎

The leading term n can be removed without additional cost by regarding the scaling factor Δ of the input or output ciphertexts (of CKKS scheme) to be $n\Delta$ (hence messages are scaled to $1/n$); or multiply the constant n^{-1} as suggested by [17]. When we adapt $\mathsf{PackLWEs}$ in our batch convolution (Theorem 2, Section IV-B), we set s such that $2^s \geq B$. When the input ciphertexts are *sparsely-packed* (Remark 5), we can set the log initial step to be $s + \log_2 B$ with s of Remark 5 so that the output ciphertext is sparsely-packed as well.

B. Composite Polynomial Approximation for ReLU

An explicit description of polynomials required for approximating $\mathsf{sign}(x)$ from [42] is as follows:

- $f_1(x) = 10.8541842577442x - 62.2833925211098x^3 + 114.369227820443x^5 - 62.8023496973074x^7$
- $f_2(x) = 4.13976170985111x - 5.84997640211679x^3 + 2.94376255659280x^5 - 0.454530437460152x^7$
- $f_3(x) = 3.29956739043733x - 7.84227260291355x^3 + 12.8907764115564x^5 - 12.4917112584486x^7 + 6.94167991428074x^9 - 2.04298067399942x^{11} + 0.246407138926031x^{13}$

$\mathsf{sign}(x)$ can approximated with $f_3(f_2(f_1(x)))$, then with $\mathrm{ReLU}(x) = \frac{x + x\cdot\mathsf{sign}(x)}{2}$, it holds that

$$\mathrm{ReLU}(x) \approx \frac{x + x \cdot f_3(f_2(f_1(x)))}{2}$$

where for $x \in [-1, 1]$, the maximum error is bounded by 2^{-10}, i.e., it provides an approximation with 10-bit precision. Evaluation of $f_1(x)$, $f_2(x)$, and $f_3(x)$ require 3,3, and 4 multiplicative depth, respectively, hence evaluation of approximate ReLU requires 10 multiplicative depth total. Note that we can also evaluate Leaky ReLU similarly with above approximation, or any piece-wise polynomial functions in general.

1) Scaling Into $[-1, 1]$: Let $P_{\mathrm{ReLU}}(x)$ be a polynomial approximation of ReLU on the interval $[-1, 1]$ whose maximum error is bounded by $2^{-\alpha}$. Then, $B \cdot P_{\mathrm{ReLU}}(x/B)$ is a polynomial approximation of ReLU on the interval $[-B, B]$ with error bound $B \cdot 2^{-\alpha}$. Therefore, we can increase the approximation range by sacrificing some precision. When we evaluate ReLU activation on CNN with FHE, we can take this strategy to evaluate ReLU on inputs contained in a bounded interval $[-B, B]$. Since batch normalization is preceded by ReLU activation, the bound B is not very large in CNN inferences, e.g., it was 32 or 64 in our experiments. Moreover, most inputs are distributed closely to the zero and average precision was substantially higher ($\geq$ 8-bit) than $B \cdot 2^{-\alpha}$ in our experiments.

C. Strided Convolution in Our ConvFHE

Recall our packing method for batch convolution described at Fig. 3 and Algorithm 1 in Section IV-B. We present how strided convolutions can be represented with our method. At first, note that the strided convolution can be represented by evaluating usual convolution then extracting appropriate entries (among output) according to the stride, e.g., the circled elements in Fig. 8. Therefore, we will focus on how to extract those values in our method.

Recall the architecture of our convolutional layer evaluation at Fig. 4 in Section IV-C. The extraction is performed on Ext phase where the output values of convolution are located in the slots (instead of the coefficients) of a plaintext polynomial. Here, we remark that the order of entries in slots are *bit-reversed* from that at the coefficients (due to the CtoS step): we will describe more concretely with the following example. In Fig. 8, let $(\mathbf{i}, \mathbf{j}, \mathbf{b}) \in \mathbb{Z}_2^5 := \{0, 1\}^5$ denotes the binary coordinate of input entry at coefficients of $I(X)$, where $\mathbf{i}$, $\mathbf{j}$, and $\mathbf{b}$ denote the row, column coordinate of entries, and batches, respectively: e.g., $\beta_{1,2}$ has the coordinate $(\mathbf{i} = (0, 1), \mathbf{j} = (1, 0), \mathbf{b} = 1) \in \mathbb{Z}_2^5$ at the coefficients of $I(X)$, but it moves to the position with *bit-reversed* coordinate $(\mathbf{b}_{rev} = 1, \mathbf{j}_{rev} = (0, 1), \mathbf{i}_{rev} = (1, 0)) \in \mathbb{Z}_2^5$ at the slots in Ext phase.

Our goal for Ext step is to extract appropriate entries from the slots considering the bit-reversed order. We remark the followings with an example in Fig. 8:

1) Among the entries of input $I(X)$, we need to extract ones located at columns and rows of multiples of 2,

i.e., with coordinates $(\mathbf{i} = (\mathbf{i}', 0), \mathbf{j} = (\mathbf{j}', 0), \mathbf{b})$ at coefficients.

2) It corresponds to $(\mathbf{b}_{rev}, \mathbf{j}_{rev} = (0, \mathbf{j}'_{rev}), \mathbf{i}_{rev} = (0, \mathbf{i}'_{rev}))$, the bit-reversed coordinates at the slots.
3) The extracted output, $I_{st}(X)$ in Fig. 8, should have coordinates $(\mathbf{i}', \mathbf{j}', (0, 0, \mathbf{b}))$ at coefficients; note that we reserve a space for $\times 4$ many batch outputs.
4) It corresponds to $((\mathbf{b}_{rev}, 0, 0), \mathbf{j}'_{rev}, \mathbf{i}'_{rev})$ of the bit-reversed coordinates at the slots.

From 2 and 4 above, it suffices to extract and move only the entries at $(\mathbf{b}_{rev}, 0, \mathbf{j}'_{rev}, 0, \mathbf{i}'_{rev})$ to $(\mathbf{b}_{rev}, 0, 0, \mathbf{j}'_{rev}, \mathbf{i}'_{rev})$ in slot position, which can be done by usual multiply one-then-rotate operation in FHE schemes as follows. Let w be the width of each input, then for each $\mathbf{j}'_{rev} \in \mathbb{Z}_2^{(\log_2 w)-1}$, it suffices to move all elements at $(\mathbf{b}_{rev}, 0, \mathbf{j}'_{rev}, 0, \mathbf{i}'_{rev})$ to the left $(\mathbf{j}'_{rev}, \mathbf{0} \in \mathbb{Z}_2^{(\log_2 w)-1})$ positions. This can be done by (i) multiplying a plaintext vector having one at desired positions (and zeros otherwise), (ii) rotating the multiplied output corresponding to moves, (iii) then summing up all outputs from each rotation together.

Note that the depth consumption is 1 and total rotations and plaintext multiplications required for this task is only $w/2 - 1$ and $w/2$, respectively, since the moves depend on $\mathbf{j}'_{rev} \in \mathbb{Z}_2^{(\log_2 w)-1}$ only.

Remark 7 (Sparsely-Packed Case): When the size of input is much less than N, we should sparsely-pack the input to reduce the timing for bootstrapping; we refer to the Remark in IV-B. The strided convolution, in this case, can also be handled similarly as the above description. There are only minor differences from the above description:

1) The binary coordinate of input entry is $(\mathbf{i}, \mathbf{j}, \mathbf{b}', \mathbf{0})$ instead of $(\mathbf{i}, \mathbf{j}, \mathbf{b})$, where the $\mathbf{0}$ is composed of s zeros when the input is sparsely packed with $I(X^{2^s})$
2) The binary coordinate of output entry should be $(\mathbf{i}', \mathbf{j}', \mathbf{b}', 0, 0, \mathbf{0})$ instead of $(\mathbf{i}', \mathbf{j}', 0, 0, \mathbf{b})$.

The procedure for Ext step, in this case, is almost the same as the above description with full packing. The difference is that we need $B \cdot w/2$ rotations and $B \cdot w/2 - 1$ plaintext multiplications (where B is the number of batches) to generate the desired output since we now need to move both $\mathbf{j}$ and $\mathbf{b}'$. Note that we can reduce the cost effectively by using more depth and separating the required moves: e.g., with 2 depths, we only need $B + w/2$ rotations and $B + w/2 - 2$ plaintext multiplications; in general, with L depths, we can reduce the cost to $L \cdot \sqrt[L]{Bw/2}$ rotations and similar number of plaintext multiplications.

In Table V, we presented some actual timing of the above procedures (Ext) for the strided convolution when implemented with optimizations employing 2 depth. Note that the timing accounts for an insignificant portion of the total timing of the CNN evaluation in V-B.

D. Application to Hybrid Approach for PPML Inference

Our concise FHE evaluation method for convolutions can also be utilized in the hybrid approaches [15], [21], [22], [31], [32], [33], [34] where FHE is used for evaluating convolutions, resulting in much less communication cost for convolutions.

TABLE V
COST OF EXT FOR STRIDED CONVOLUTIONS. (S: TOTAL SIZE, w: INPUT WIDTH, B: NUMBER OF BATCHES)

Parameters (S, w, B)	# rotations	Timing (sec)
$(2^{15}, 32, 32)$	$48 = 16 + 32$	2.7
$(2^{14}, 16, 64)$	$72 = 8 + 64$	4.0
$(2^{17}, 32, 128)$	$4 \cdot 80 = 4 \cdot (16 + 64)$	18.0
$(2^{16}, 16, 256)$	$2 \cdot 96 = 2 \cdot (64 + 32)$	10.8
$(2^{17}, 16, 512)$	$4 \cdot 96 = 4 \cdot (64 + 32)$	19.4

At first, recall that both BGV [6] and B/FV [7], [8] FHE schemes — utilized in those hybrid approaches — have encryption/decryption algorithms (Enc/Dec) with the ring $\mathcal{R}_p := \mathcal{R}/p\mathcal{R}$ as a plaintext space, i.e., for $m_1, m_2 \in \mathcal{R}_p$,

$$\mathsf{Dec}(\mathsf{Enc}(m_1) \boxplus \mathsf{Enc}(m_2)) \approx m_1 + m_2$$
$$\mathsf{Dec}(\mathsf{Enc}(m_1) \boxdot \mathsf{Enc}(m_2)) \approx m_1 \cdot m_2$$

where $\boxplus$ and $\boxdot$ denote ciphertext addition and multiplication, respectively. Since our FHE-evaluation of convolution (in Section IV-B) can be implemented with polynomial addition and multiplication over the ring $\mathcal{R}$, it can be also utilized over the plaintext space $\mathcal{R}_p := \mathcal{R}/p\mathcal{R}$ of those FHE schemes given that the modulus p does not occur during the computation (of convolutions), which always holds in all of the hybrid approaches by setting large enough p.

In fact, the recent work [34] presented polynomial representation for more efficient FHE evaluation of convolutions. While their solution is almost the same as ours for single convolution, our solution for batch convolution is considerably more efficient in communication cost, due to our elaborated packing structure and novel application of PackLWEs algorithm (Appendix A). The main difference is that the previous work can pack only *one* convolution output in one ciphertext, whereas our solution enables to pack as *many* convolution output as possible in one ciphertext.

To see the advantage of our scheme in practice, we can estimate and compare the communication cost as follows. At first, recall that [34] used the FHE parameter with $N = 2^{12}$, $\log q = 105$, $\log p = 37$ while we can use the parameter with $N' = 2^{13}$, $\log q' = \log q + 63$, $\log p = 37$. Though we need bigger ciphertext degree N' for auxiliary modulus bits $(\log q' - \log q = 63)$ for rotations (in PackLWEs), we have the same modulus size $\log q$ of each ciphertext. Therefore, the size of each ciphertext is $2\times$ bigger in our case while it can also pack $2\times$ many messages compared to that of [34].

For ease of comparison, we assume that the size w^2 of single input is less than or equal to N. Then, for input of size $w^2 B_{in}$, [34] and ours require $\lceil \frac{w^2 B_{in}}{N} \rceil$ and $\lceil \frac{w^2 B_{in}}{N'} \rceil$ number of input ciphertexts, respectively, and it does not cause any difference in communication cost given that $w^2 B_{in} > N$. For output of size $w^2 B_{out}$, however, [34] requires B_{out} number of output ciphertexts (since it cannot pack multiple batches in one output ciphertext) while ours require only $\lceil \frac{w^2 B_{out}}{N'} \rceil$ (thanks to PackLWEs algorithm). This difference results in considerable reduction of communication cost in our case when $w^2 \ll N$.

For concrete comparison, we assume that the bit size of input and output ciphertext is $2N \log q$ and $N(2 \log p + \epsilon)$,

TABLE VI

COMMUNICATION COST OF HOMOMORPHIC CONVOLUTION. (w: WIDTH OF INPUT), (k: WIDTH OF KERNEL), (B_{in}, B_{out}: INPUT/OUTPUT BATCH NUMBERS) (c_{in}, c_{out}: SIZE OF INPUT/OUTPUT CIPHERTEXT)

Parameters ($k^2 = 3^2$)	[34]	**Ours**
$w^2,\ B_{in},\ B_{out}$	$\lceil \frac{w^2 B_{in}}{N} \rceil \cdot c_{in}$ $+B_{out} \cdot c_{out}$	$\lceil \frac{w^2 B_{in}}{N'} \rceil \cdot c'_{in}$ $+\lceil \frac{w^2 B_{out}}{N'} \rceil \cdot c'_{out}$
$7^2,\ 256,\ 256$	11.41 MB	**0.58** MB
$15^2,\ 128,\ 128$	6.32 MB	**1.16** MB
$31^2,\ 64,\ 64$	4.39 MB	**2.33** MB
$63^2,\ 32,\ 32$	4.66 MB	4.66 MB

TABLE VII

PLAIN-20 AND PLAIN-18 CNN FOR CIFAR10 AND IMAGENET [44]. KER: $[k \times k \times B] \times L$ DENOTES THAT EACH KERNEL IS OF SIZE $k \times k$, THE OUTPUT BATCH NUMBER IS B, AND LAYERS ARE STACKED L TIMES. EACH LAYER HAS RELU PRECEDED BY BATCH NORMALIZATION

BLOCK	PLAIN-20	PLAIN-18
INPUT	$32 \times 32 \times 3$ (CIFAR10)	$224 \times 224 \times 3$ (IMAGENET, CROPPED)
CONV0	KER: $[3 \times 3, 16]$ OUT: $32 \times 32 \times 16$	KER: $[7 \times 7, 64]$ (STRIDE 2) OUT: $112 \times 112 \times 64$
CONV1	KER: $[3 \times 3, 16 \times \omega] \times 6$ OUT: $32 \times 32 \times 16 \times \omega$	MAX POOL: 3×3 (STRIDE 2) KER: $[3 \times 3, 64] \times 4$ OUT: $56 \times 56 \times 64$
CONV2	KER: $[3 \times 3, 32 \times \omega] \times 6$ OUT: $16 \times 16 \times 32 \times \omega$	KER: $[3 \times 3, 128] \times 4$ OUT: $28 \times 28 \times 128$
CONV3	KER: $[3 \times 3, 64 \times \omega] \times 6$ OUT: $8 \times 8 \times 64 \times \omega$	KER: $[3 \times 3, 256] \times 4$ OUT: $14 \times 14 \times 256$
CONV4		KER: $[3 \times 3, 512] \times 4$ OUT: $7 \times 7 \times 512$
FINAL	AVERAGE POOL: 8×8 FC: 64×10 OUT: 10-DIM VECTOR	AVERAGE POOL: 7×7 FC: 512×1000 OUT: 1000-DIM VECTOR

respectively, where N is substituted by N' in our case.[8] Then, with 1MB $\approx 8 \times 2^{20}$bits, we can estimate the concrete communication cost for convolutions in various parameters (the convolutions from Plain-20 CNN, Table VII with $w = 3$ in Appendix E); see Table VI.

In Table VI, ours has roughly 2–20$\times$ less communication cost compared to [34]. The reduction of cost is more significant when the size w^2 of single input is smaller (hence much less than N) as expected. When the size of single output is large enough (e.g., 63^2 or more), however, ours has the same communication cost since we cannot pack more than one convolution output in single ciphertext. We remark that convolutions with small input width w^2 arises often in many deep CNN architectures to capture abstract features. On the other hand, our scheme requires more computational cost than [34] due to the algorithm PackLWEs.

E. CNN Model Architecture and Dataset

We describe the dataset and (variants of) CNN architectures in our experiment.

[8]Here, we assume the optimization of [34] that utilizes the fact that only the higher bits of output ciphertext are sufficient for decryption, where $\epsilon = \lceil \log 6.6\sqrt{N} \rceil + 1$. The optimization is also applicable to our case, since we also pack messages in the coefficients of plaintext polynomial.

1) Dataset: CIFAR10/100 [48]: 50K/10K training/testing images ($32 \times 32 \times 3$) in 10 or 100 classes; ImageNet(2012) [49]: 1.28M/50K/100K ($224 \times 224 \times 3$ after cropping) training/validation/test images in 1000 classes.

2) CNN Architecture: Plain-20 and Plain-18 CNN from [44] are used for CIFAR10 and ImageNet classification task, resp., which are described in Table VII with $w = 1$. The Plain-20 CNN is composed of one initial convolutional layer (Conv0) and three consecutive blocks (Conv1, Conv2, Conv3) each of which are composed of 6 convolutional layers. The first layer of Conv2 and Conv3 block is strided convolution (with stride 2) so that the output width is an half of input width. The Plain-20 CNN is similar but with different number of blocks and layers.

3) Our CNN Variants: For depth 14 or 8, the number of layers in each block is modified from 6 to 4 or 2, resp. The wideness factor ω (see Table VII) implies that each layer has $\times\omega$ many number of output batches compared to the original CNN. The original CNN has convolutions with 3×3 kernels and we substitute it with 5×5 or 7×7 kernels. For training, we used the same hyper parameter setting as [45]. For the variant of Plain-18, we only substituted the 3×3 kernels of Conv3 and Conv4 block by 5×5 kernels. We used cropping when desired.

REFERENCES

[1] A. C. Yao, "Protocols for secure computations," in *Proc. 23rd Annu. Symp. Found. Comput. Sci. (SFCS)*, Nov. 1982, pp. 160–164.

[2] O. Goldreich, S. Micali, and A. Wigderson, "How to play any mental game," in *Proc. 19th Annu. ACM Conf. Theory Comput. (STOC)*, 1987, pp. 218–229.

[3] C. Gentry, "Fully homomorphic encryption using ideal lattices," in *Proc. 45st Annu. ACM Symp. Theory Comput.*, May 2009, pp. 169–178.

[4] J. H. Cheon, A. Kim, M. Kim, and Y. Song, "Homomorphic encryption for arithmetic of approximate numbers," in *Proc. Int. Conf. Theory Appl. Cryptol. Inf. Secur.* Cham, Switzerland: Springer, 2017, pp. 409–437.

[5] H. Chen, W. Dai, M. Kim, and Y. Song, "Efficient homomorphic conversion between (ring) LWE ciphertexts," in *Proc. Int. Conf. Appl. Cryptogr. Netw. Secur.* Cham, Switzerland: Springer, 2021, pp. 460–479.

[6] Z. Brakerski, C. Gentry, and V. Vaikuntanathan, "(Leveled) fully homomorphic encryption without bootstrapping," *ACM Trans. Comput. Theory*, vol. 6, pp. 1–36, Jul. 2014.

[7] Z. Brakerski and V. Vaikuntanathan, "Efficient fully homomorphic encryption from (standard) LWE," *SIAM J. Comput.*, vol. 43, no. 2, pp. 831–871, 2014.

[8] J. Fan and F. Vercauteren, "Somewhat practical fully homomorphic encryption," *Cryptol. ePrint Arch.*, to be published.

[9] R. Gilad-Bachrach, N. Dowlin, K. Laine, K. Lauter, M. Naehrig, and J. Wernsing, "CryptoNets: Applying neural networks to encrypted data with high throughput and accuracy," in *Proc. Int. Conf. Mach. Learn.*, 2016, pp. 201–210.

[10] Y. LeCun. (1998). *The MNIST Database of Handwritten Digits*. [Online]. Available: http://yann.lecun.com/exdb/mnist/

[11] E. Hesamifard, H. Takabi, and M. Ghasemi, "CryptoDL: Towards deep learning over encrypted data," in *Proc. Annu. Comput. Secur. Appl. Conf. (ACSAC)*, vol. 11, Los Angeles, CA, USA, 2016, pp. 1–2.

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

[13] F. Boemer, Y. Lao, R. Cammarota, and C. Wierzynski, "NGraph-HE: A graph compiler for deep learning on homomorphically encrypted data," in *Proc. 16th ACM Int. Conf. Comput. Frontiers*, Apr. 2019, pp. 3–13.

[14] F. Boemer, A. Costache, R. Cammarota, and C. Wierzynski, "NGraph-HE2: A high-throughput framework for neural network inference on encrypted data," in *Proc. 7th ACM Workshop Encrypted Comput. Appl. Homomorphic Cryptogr.*, Nov. 2019, pp. 45–56.

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

[16] R. Dathathri et al., "CHET: An optimizing compiler for fully-homomorphic neural-network inferencing," in *Proc. 40th ACM SIGPLAN Conf. Program. Lang. Design Implement.*, Jun. 2019, pp. 142–156.

[17] 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.*, Jun. 2020, pp. 546–561.

[18] S. Chowdhary, W. Dai, K. Laine, and O. Saarikivi, "EVA improved: Compiler and extension library for CKKS," in *Proc. 9th Workshop Encrypted Comput. Appl. Homomorphic Cryptogr.*, Nov. 2021, pp. 43–55.

[19] A. Brutzkus, R. Gilad-Bachrach, and O. Elisha, "Low latency privacy preserving inference," in *Proc. Int. Conf. Mach. Learn.*, 2019, pp. 812–821.

[20] Q. Lou, W.-J. Lu, C. Hong, and L. Jiang, "Falcon: Fast spectral inference on encrypted data," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 33, 2020, pp. 2364–2374.

[21] S. Bian, T. Wang, M. Hiromoto, Y. Shi, and T. Sato, "ENSEI: Efficient secure inference via frequency-domain homomorphic convolution for privacy-preserving visual recognition," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2020, pp. 9403–9412.

[22] S. Li et al., "FALCON: A Fourier transform based approach for fast and secure convolutional neural network predictions," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2020, pp. 8705–8714.

[23] F. Bourse, M. Minelli, M. Minihold, and P. Paillier, "Fast homomorphic evaluation of deep discretized neural networks," in *Proc. Annu. Int. Cryptol. Conf.* Cham, Switzerland: Springer, 2018, pp. 483–512.

[24] I. Chillotti, M. Joye, and P. Paillier, "Programmable bootstrapping enables efficient homomorphic inference of deep neural networks," *IACR Cryptol. ePrint Arch.*, vol. 2021, p. 91, Jan. 2021.

[25] I. Chillotti, N. Gama, M. Georgieva, and M. Izabachene, "Faster fully homomorphic encryption: Bootstrapping in less than 0.1 seconds," in *Proc. Int. Conf. Theory Appl. Cryptol. Inf. Secur.* Cham, Switzerland: Springer, 2016, pp. 3–33.

[26] I. Chillotti, N. Gama, M. Georgieva, and M. Izabachéne, "Faster packed homomorphic operations and efficient circuit bootstrapping for TFHE," in *Proc. Int. Conf. Theory Appl. Cryptol. Inf. Secur.* Cham, Switzerland: Springer, 2017, pp. 377–408.

[27] Q. Lou and L. Jiang, "SHE: A fast and accurate deep neural network for encrypted data," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 32, 2019, pp. 1–9.

[28] J.-W. Lee et al., "Privacy-preserving machine learning with fully homomorphic encryption for deep neural network," *IEEE Access*, vol. 10, pp. 30039–30054, 2022.

[29] E. Lee et al., "Low-complexity deep convolutional neural networks on fully homomorphic encryption using multiplexed parallel convolutions," in *Proc. Int. Conf. Mach. Learn.*, 2022, pp. 12403–12422.

[30] M. Kim, X. Jiang, K. Lauter, E. Ismayilzada, and S. Shams, "Secure human action recognition by encrypted neural network inference," 2021, *arXiv:2104.09164*.

[31] D. Rathee et al., "CrypTFlow2: Practical 2-party secure inference," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, Oct. 2020, pp. 325–342.

[32] P. Mishra, R. Lehmkuhl, A. Srinivasan, W. Zheng, and R. A. Popa, "Delphi: A cryptographic inference system for neural networks," in *Proc. Workshop Privacy-Preserving Mach. Learn. Pract.*, Nov. 2020, pp. 2505–2522.

[33] S. Bian, D. E. S. Kundi, K. Hirozawa, W. Liu, and T. Sato, "APAS: Application-specific accelerators for RLWE-based homomorphic linear transformations," *IEEE Trans. Inf. Forensics Security*, vol. 16, pp. 4663–4678, 2021.

[34] 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 Secur.)*, 2022, pp. 809–826.

[35] R. L. Rivest, L. Adleman, and M. L. Dertouzos, "On data banks and privacy homomorphisms," *Found. Secure Comput.*, vol. 4, no. 11, pp. 169–180, 1978.

[36] J. H. Cheon, K. Han, A. Kim, M. Kim, and Y. Song, "Bootstrapping for approximate homomorphic encryption," in *Proc. Annu. Int. Conf. Theory Appl. Cryptograph. Techn.* Cham, Switzerland: Springer, 2018, pp. 360–384.

[37] H. Chen, I. Chillotti, and Y. Song, "Improved bootstrapping for approximate homomorphic encryption," in *Proc. Annu. Int. Conf. Theory Appl. Cryptograph. Techn.* Cham, Switzerland: Springer, 2019, pp. 34–54.

[38] K. Han and D. Ki, "Better bootstrapping for approximate homomorphic encryption," in *Proc. Cryptographers' Track RSA Conf.* Cham, Switzerland: Springer, 2020, pp. 364–390.

[39] J.-P. Bossuat, C. Mouchet, J. Troncoso-Pastoriza, and J.-P. Hubaux, "Efficient bootstrapping for approximate homomorphic encryption with non-sparse keys," in *Proc. Annu. Int. Conf. Theory Appl. Cryptograph. Techn.* Cham, Switzerland: Springer, 2021, pp. 587–617.

[40] J. H. Cheon, K. Han, A. Kim, M. Kim, and Y. Song, "A full RNS variant of approximate homomorphic encryption," in *Proc. Int. Conf. Sel. Areas Cryptogr.* Cham, Switzerland: Springer, 2018, pp. 347–368.

[41] J. H. Cheon, D. Kim, and D. Kim, "Efficient homomorphic comparison methods with optimal complexity," in *Proc. Int. Conf. Theory Appl. Cryptol. Inf. Secur.* Cham, Switzerland: Springer, 2020, pp. 221–256.

[42] E. Lee, J.-W. Lee, J.-S. No, and Y.-S. Kim, "Minimax approximation of sign function by composite polynomial for homomorphic comparison," *IEEE Trans. Depend. Secure Comput.*, vol. 19, no. 6, pp. 3711–3727, Nov. 2022.

[43] (Jan. 2022). *Lattigo v2.4.0*. ePFL-LDS. [Online]. Available: https://github.com/ldsec/lattigo

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

[45] S. Zagoruyko and N. Komodakis, "Wide residual networks," 2016, *arXiv:1605.07146*.

[46] A. A. Badawi, L. Hoang, C. F. Mun, K. Laine, and K. M. M. Aung, "PrivFT: Private and fast text classification with homomorphic encryption," *IEEE Access*, vol. 8, pp. 226544–226556, 2020.

[47] 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 Cryptol. ePrint Arch.*, vol. 2021, p. 508, Jan. 2021.

[48] A. Krizhevsky et al., "Learning multiple layers of features from tiny images," Tech. Rep., 2009.

[49] O. Russakovsky et al., "ImageNet large scale visual recognition challenge," *Int. J. Comput. Vis.*, vol. 115, no. 3, pp. 211–252, Dec. 2015.