HTCNN: High-Throughput Batch CNN Inference with Homomorphic Encryption for Edge Computing

Zewen Ye, Tianshun Huang, Tianyu Wang, Yonggen Li, Chengxuan Wang, Ray C.C. Cheung, *Senior Member, IEEE*, Kejie Huang, *Senior Member, IEEE*

Abstract—Homomorphic Encryption (HE) technology allows for processing encrypted data, breaking through data isolation barriers and providing a promising solution for privacy-preserving computation. The integration of HE technology into Convolutional Neural Network (CNN) inference shows potential in addressing privacy issues in identity verification, medical imaging diagnosis, and various other applications. The CKKS HE algorithm stands out as a popular option for homomorphic CNN inference due to its capability to handle real number computations. However, challenges such as computational delays and resource overhead present significant obstacles to the practical implementation of homomorphic CNN inference, largely due to the complex nature of HE operations. In addition, current methods for speeding up homomorphic CNN inference primarily address individual images or large batches of input images, lacking a solution for efficiently processing a moderate number of input images with fast homomorphic inference capabilities, which is more suitable for edge computing applications. In response to these challenges, we introduce a novel leveled homomorphic CNN inference scheme aimed at reducing latency and improving throughput using the CKKS scheme. Our proposed inference strategy involves mapping multiple inputs to a set of ciphertext by exploiting the sliding window properties of convolutions to utilize CKKS's inherent Single-Instruction-Multiple-Data (SIMD) capability. To mitigate the delay associated with homomorphic CNN inference, we introduce optimization techniques, including mask-weight merging, rotation multiplexing, stride convolution segmentation, and folding rotations. The efficacy of our homomorphic inference scheme is demonstrated through evaluations carried out on the MNIST and CIFAR-10 datasets. Specifically, results from the MNIST dataset on a single CPU thread show that inference for 163 images can be completed in 10.4 seconds with an accuracy of 98.95%, which is a 6.9$\times$ throughput improvement over state-of-the-art works. Comparative analysis with existing methodologies highlights the superior performance of our proposed inference scheme in terms of latency, throughput, communication overhead, and memory utilization.

Index Terms—Homomorphic encryption, CKKS, Convolutional Neural Network, CNN Inference

Zewen Ye is with the College of Information Science & Electronic Engineering, Zhejiang University and the Department of Electrical Engineering, City University of Hong Kong. E-mail: lucas.zw.ye@zju.edu.cn.
Ray C.C. Cheung is with the Department of Electrical Engineering, City University of Hong Kong, Hong Kong, China. E-mail: r.cheung@cityu.edu.hk. Tianshun Huang, Tianyu Wang, Yonggen Li, Chengxuan Wang, and Kejie Huang are with the College of Information Science & Electronic Engineering, Zhejiang University, Hangzhou, China. E-mail: z1458152445@163.com, {wang_tianyu, li_yonggen, wangchengxuan, huangkejie}@zju.edu.cn.

1 Introduction

Convolutional Neural Networks (CNN) [1] play a crucial role in facial recognition and medical image diagnosis applications, which often involve sensitive personal data. However, the existence of data silos, caused by security concerns among different data subjects and within individual data subjects, obstructs the smooth sharing of data and creates challenges in realizing the full potential of data [2], [3]. As a result, protecting data privacy, preventing unauthorized data exposure, and meeting the requirements of applications have become significant obstacles [4], [5]. Integrating homomorphic encryption (HE) technology [6] into CNN inference can help maintain user data confidentiality through encryption, thereby safeguarding user privacy. The significance of data privacy and security has increased with the rapid expansion of big data and artificial intelligence, as operations like face recognition and the development of large language models heavily rely on extensive training data to improve their performance. Homomorphic encryption is a cryptographic technique that enables operations on encrypted data without decryption. Several homomorphic encryption algorithms have been developed in recent years, including YASHE [7], BGV [8], FV [9], TFHE [10], and CKKS [11], [12]. Among these, the CKKS scheme is notable for its incorporation of approximate computing features based on the complexity of the Ring Learning With Errors (RLWE) problem. This method maintains precision by including a scaling factor and conducting a rescaling operation on the ciphertext after multiplication to reduce noise levels. CKKS's ability to handle real numbers makes it well-suited for artificial intelligence and machine learning applications. Moreover, the CKKS algorithm supports batch encoding calculations for vectors, improving the efficiency of homomorphic operations.

There is a growing demand in academia and industry to enhance the implementation of homomorphic CNN inference technology. A key challenge is the significant computational delay associated with homomorphic CNN inference. CKKS homomorphic encryption transforms a single floating-point number or integer into a set of polynomials with numerous coefficients (2^{12} to 2^{16}), resulting in compu-

tation latency over three orders of magnitude longer than traditional calculations and leading to substantial storage overhead. For example, Cryptonets [13] takes 250 seconds for one homomorphic CNN inference compared to milliseconds in non-encrypted operations. Thus, the pursuit of homomorphic CNN inference with minimal latency, storage overhead, and high throughput is crucial to enhance the practicality and applicability of such technology in real-world scenarios.

In previous studies, several optimizations have been proposed for homomorphic CNN inference. Cryptonets [13] introduced a method for CNN evaluation using homomorphic encryption, where multiple inputs are combined into a single ciphertext. However, this approach resulted in excessive memory usage due to packing too many inputs together. In contrast, LoLa [14] achieved lower latency in CNN evaluations by encrypting only one input per ciphertext, thereby reducing CKKS slot utilization. Falcon [15] improved latency by implementing spectral inference techniques. While LoLa and Falcon stack the same input multiple times in a ciphertext to lower costs and enhance efficiency compared to traditional vectorized methods [16], [17], which are more suitable for single input computations rather than efficient for multi-input CNN inference.

As can be seen, current packing methods for speeding up homomorphic CNN inference primarily address individual images or large batches of input images. However, an efficient solution for processing a moderate and variable number of inputs with fast homomorphic inference capabilities on edge computing platforms [18], [19] is still lacking. Secure edge computing is a critical topic for real-world applications and research [20], [21]. While cloud computing can handle multiple data streams with abundant resources, and Internet-of-Things (IoT) devices process limited data with constrained resources, edge computing bridges this gap by bringing enterprise applications closer to data sources such as IoT devices or local edge servers. These edge servers possess more resources than IoT devices but fewer than cloud computing platforms. On the other hand, prior studies on secure CNN inference [16], [22] have overlooked scenarios involving multiple convolutions, which is a significant limitation. These scenarios require more communication overhead with data provider clients, potentially exposing more security vulnerabilities.

Contributions: To address the requirement of efficient HE CNN inference on edge computing, this paper introduces a novel approach for enhancing efficient multiple-input CNN inference within homomorphic encryption by proposing a batch input image packing scheme that maximizes ciphertext slot utilization. The proposed design allows for homomorphic computing without the extra need for client communication. The key contributions of this work are outlined as follows:

- A novel packing method is introduced to encode and encrypt a moderate number of inputs for CNN, achieving nearly 100% ciphertext slot utilization. Additionally, a novel CNN model weight packing strategy is proposed to align with the pattern of the input packing scheme. This approach leverages homomorphic rotations, additions, and multiplications to facilitate operations in the convolutional and full connection layers.
- Several optimization techniques are presented to enhance the performance of homomorphic convolutional layers for stride values of both 1 and greater than 1. Methods including sliding windows rotated ciphertext sharing, and stride convolution segmentation are introduced to improve performance and reduce memory overhead.
- An auto-compile tool is developed to streamline the generation of the full connection layer packing, ensuring that the data order aligns with the corresponding encrypted outputs.

Through these optimizations, we achieve notable performance improvements, high throughput, and reduced memory overhead on the MNIST and CIFAR-10 datasets, surpassing previous works by several orders of magnitude.

Paper Organization: The following paper is organized as follows: Firstly, we provide a brief introduction to the core operations of CKKS and CNN in Section 2. Then, Section 3 details the proposed batch packing scheme for inputs and the CNN model. The detailed computations of homomorphic convolution, activation, and full connection layer operations are presented in Section 4. We present the evaluation results and compare our work in Section 5. Finally, we conclude this paper in Section 6.

2 Preliminary

This section will elaborate on the mathematical aspects of homomorphic operations in the CKKS scheme and convolutional neural network.

2.1 CKKS

The CKKS scheme, introduced by Cheon et al. [11], is a type of homomorphic encryption that facilitates approximate arithmetic and processing of real numbers. In CKKS homomorphic computation, each plaintext encapsulates $N/2$ real numbers before being encrypted into a ciphertext. In the CKKS scheme, a ciphertext is represented as $c = (\mathbf{c_0}, \mathbf{c_1}) \in R_q^2$, where $R_q = \mathbb{Z}_q[X]/(X^N + 1)$. Various homomorphic operations denoted as $\mathbf{f}$ are executed on the ciphertext to carry out specific computations on the data. Subsequently, decryption and decoding processes are employed to retrieve the processed data. This typical homomorphic computation process is illustrated in Figure 1.

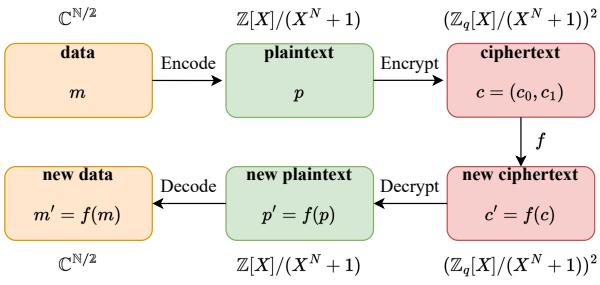


Fig. 1. CKKS homomorphic processing.

2.1.1 CKKS Homomorphic Addition and Multiplication

Additions and multiplications are the two basic homomorphic operations of CKKS. Given two plaintexts a and b, CKKS homomorphic additions and multiplications satisfy the following properties:

$$\begin{aligned} \textbf{CKKS.ENC}(a) + \textbf{CKKS.ENC}(b) &\approx \textbf{CKKS.ENC}(a+b) \\ \textbf{CKKS.ENC}(a) \cdot \textbf{CKKS.ENC}(b) &\approx \textbf{CKKS.ENC}(a \cdot b) \end{aligned} \tag{1}$$

In CKKS, after encryption, the ciphertexts consist of two polynomials. In this case, one homomorphic addition consists of two polynomial additions. However, homomorphic multiplication is much more complex as a **relinearization** operation is required. Both homomorphic addition and multiplication would consume ciphertext level and increase noise. The consumption of homomorphic multiplication is much larger than that of homomorphic addition. Usually, **rescaling** is required after homomorphic multiplication, which would switch the ciphertext to the next ciphertext level. When the ciphertext level decreases to 0, the ciphertext can not be decrypted correctly. Therefore, in real-world applications, one should ensure how many homomorphic multiplications are required to design the CKKS modulus properly.

2.1.2 CKKS Homomorphic Rotation

Rotation is to rotate the vector $z_0, z_1, ..., z_{N/2-1}] \in \mathbb{C}^{N/2}$ homomorphically. When performing CKKS rotation by one step, the resulting data will be $[z_1, z_2, ..., z_{N/2-1}, z_0]$. In homomorphic CNN inference, rotations are widely used since there are many data permutations during the computation. However, the complexity of rotation is much larger than that of homomorphic additions and multiplications.

2.2 Convolutional Neural Network

CNN is a class of artificial neural networks that have become dominant in various computer vision tasks [23]. The CNN architecture includes several building blocks, including convolutional layers, pooling layers, and full connection layers. Advanced features of images are extracted through operations such as the convolutional layer, pooling layer and full connection layer, which have been implemented for tasks such as classification, target detection, face recognition, etc.

The convolutional layer is the most crucial part of CNN, which gets the output by projecting the convolutional kernel onto the input features for sliding window computation. The operation of the convolutional layer is to perform an element-wise product between each element of the convolution kernel, and the input tensor is calculated at each location of the tensor and summed to obtain the output value in the corresponding position of the output tensor. It can be represented as:

$$O(x,y) = \sum_m \sum_n I(i+m, j+n) \cdot K(m,n) \tag{2}$$

where O represents the output features, I represents the input features and K represents the convolution kernel. The role of the step size is to allow the convolution kernel to slide how many columns to the right or how many rows down at a time. A convolution with a step size of 1 is a normal convolution, and a convolution with a step size greater than 1 is called a stride convolution.

The pooling layer, also known as the downsampling layer, reduces the size of the input features and retains important information about the input features. Common pooling operations in neural networks typically consist of maximum pooling and average pooling. Maximum pooling involves selecting the highest value within a specific local area as the result, whereas average pooling computes the mean value of the input characteristics in that region. When the division step is eliminated, average pooling transforms into sum pooling, leading to a reduction in computational complexity. Sum pooling is often preferred in HE CNN inference due to its straightforward nature, while maximum pooling involves the additional overhead of comparing ciphertexts.

The Full Connection (FC) layer, also known as the multilayer perceptron, is an important part of the CNN. The FC layer receives the image features extracted by the convolution or pooling layers and unfolds them into a one-dimensional vector. Matrix multiplications are performed on the flattened one-dimensional vector. The flattened operation would require rotations when performed in encryption mode.

Activation layer are usually non-linear transformation functions that act on the previous layer's output to learn more complex features and patterns. The rectified linear unit (ReLU) is the most common nonlinear activation function. However, it is hard to implement this function homomorphically, as it would be decomposed into large-order polynomials, leading to large computation overhead and noise.

3 Batch CNN Packing Scheme

In this section, we introduce the batch encoding scheme for our high-throughput CNN inference method. Our goal is to optimize the use of ciphertext slots while reducing the required number of homomorphic operations. Our approach removes the need for client-side communication during encrypted CNN inference by analyzing the output data patterns of the convolutional layer and the requirement for multiple convolutional layers.

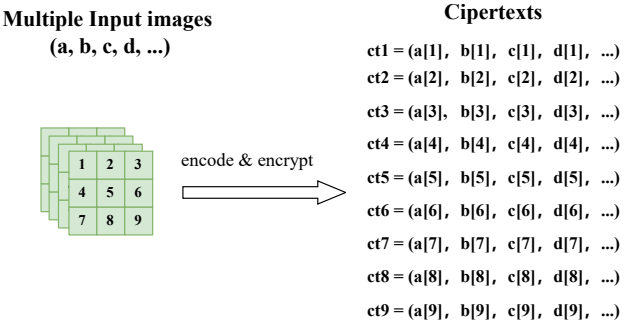


Fig. 2. The details of the full-ciphertext packing scheme.

3.1 Input Packing Scheme

Previous input packing schemes can be divided into two kinds: one is to pack the same pixel of multiple image into

one ciphertext (**full-ciphertext packing scheme**), which will produce many ciphertexts [13] (shown in Figure 2); the other is to pack one image into one ciphertext (**one-ciphertext packing scheme**), where some pixel are replicated multiple times to meet the computation requirements [14], [16], [22]. Our proposed image packing scheme is different from these two approaches, where we also pack multiple images into some ciphertexts but have fewer ciphertexts than the full-ciphertext packing scheme.

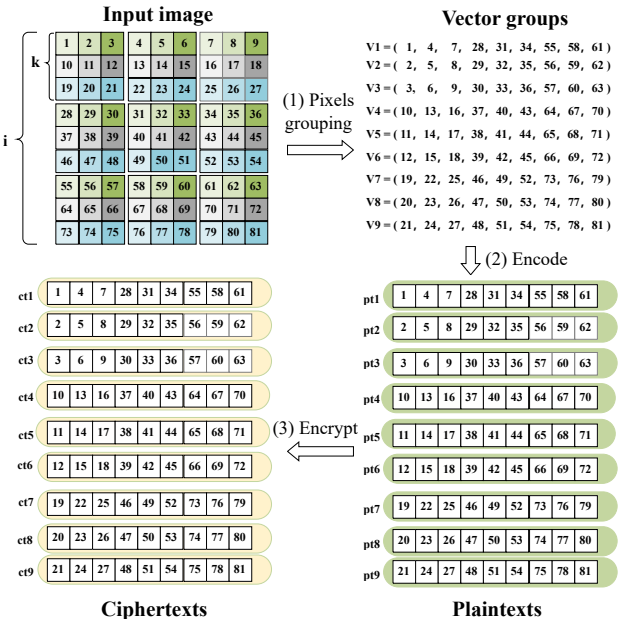


Fig. 3. Proposed input packing scheme.

Our image packing scheme is illustrated in Figure 3. Assuming a convolutional kernel size of 3×3 and a stride of 1, the 9×9 image is divided into groups labeled as $V1$ to $V9$. Pixels of the same color are paired with corresponding pixels from the kernel and consolidated into a single vector.

In general, when dealing with an image of size $i \times i$ and a convolutional kernel of size $k \times k$, the image is segmented into $\lceil i/k \rceil$ sections. Pixels that interact with the same kernel pixel are encoded into distinct vectors, resulting in a total of k^2 vectors. For homomorphic encryption with a polynomial size of N, a total of $N/2$ slots can be used to store data. Therefore, the remaining $N/2 - k^2$ slots are repurposed to accommodate more input images to maximize slot utilization. Overall, our proposed image packaging method can encode approximately $\lfloor (N \cdot k^2)/(2i^2) \rfloor$ images into k^2 vectors. Subsequently, these vectors are encoded and encrypted into k^2 ciphertexts for homomorphic encryption CNN inference.

Our devised image packaging technique achieves close to 100% utilization. In contrast, Lola [14] demonstrates a utilization rate of around 20% due to the use of the one-ciphertext packing scheme. While Cryptonets [13] achieves full utilization with the full-ciphertext packing scheme, it generates i^2 ciphertexts, demanding much more memory space.

3.2 Packing of Convolutional Kernel

When encrypting multiple images, the same processed pixels are combined into a single ciphertext. Therefore, to match the computation, each convolutional kernel pixel is replicated $N/2$ times to form a vector that is then converted into plaintext. This approach maintains a slot utilization of approximately 100%. The packing scheme of the convolutional kernel is illustrated in Figure 4. Consequently, a $k \times k$ convolutional kernel is encoded and packed as k^2 plaintexts for use in homomorphic convolutions. It is important to note that in cases of multi-channel convolutions, each channel is encoded into k^2 plaintexts.

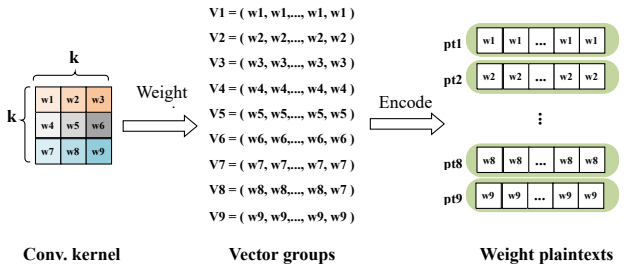


Fig. 4. Convolutional kernel packing.

The first convolutional layer in the complete CNN model is straightforward to pack, whereas the packing process for the second convolutional layer differs slightly. To support the second convolutional layer, it is essential to rotate the results of the first convolution, thereby altering the order of ciphertext pixels. During the execution of the second convolutional layer, certain pixels need to be masked through multiplication with specific zero and one values, introducing additional noise. To circumvent this, the multiplication for masking can be conducted prior to homomorphic encryption, specifically by performing these operations with the convolutional kernels of the second layer (referred to as mask-weight merging). A comprehensive understanding of this packing strategy is shown in Figure 5. As can be seen, compared with Figure 5(a), (b) can reduce one plaintext and ciphertext multiplication, which avoids introducing additional noise. The only overhead is that the weight needs to be masked initially, which is usually performed once as the CNN model is prepared by edge computing devices.

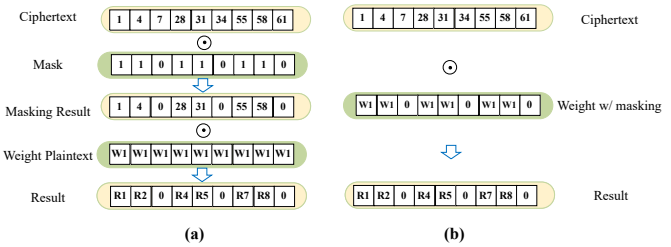


Fig. 5. The details of mask-weight merging. (a) is the conventional approach, and (b) merges the mask with the weight plaintext.

3.3 Packing of Full Connection Layer

The inputs to the full connection layer typically originate from the convolutional and pooling layers. In scenarios

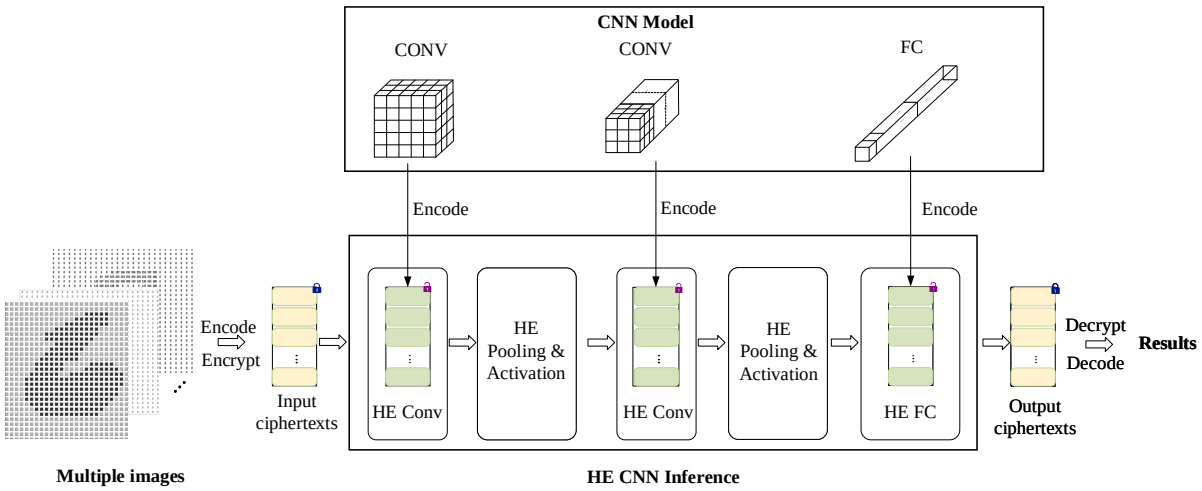


Fig. 6. HE CNN computing flow. Inputs are packed and encrypted as several ciphertexts, and the CNN weights are packed and encoded as plaintexts. All the inference operations are performed homomorphically to generate some output ciphertexts, which are further decrypted and unpacked to some result vectors.

where data is unencrypted, these inputs need to be flattened into a matrix for FC operations, necessitating rotations of the input data. To streamline operations and improve efficiency, a decision is made to execute FC operations directly on the input ciphertext. As a result, special attention is paid to encoding the weights of the FC Layer to align with the multiplicand ciphertext. For optimization purposes, an automatic processing tool is developed to assess computational patterns and produce corresponding FC packing plaintexts for improved efficiency.

4 High Throughput Batch CNN Inference Scheme

This section outlines the operational procedures of our proposed homomorphic CNN inference, as illustrated in Figure 6. Multiple images undergo mapping and encryption into various ciphertexts based on Subsection 3.1. Different layers of the CNN model are translated into multiple plaintexts, as expounded in Subsection 3.2 and Subsection 3.3. The primary procedures of HE CNN inference involve HE convolutions, HE poolings, activations, and HE full connections. Specifically, two types of convolutions are employed: standard convolution with a stride of 1 (ordinary convolution) and convolution with a stride greater than 1 (stride convolution). Subsequently, a comprehensive overview of these HE operations will be provided.

4.1 Ordinary Convolution

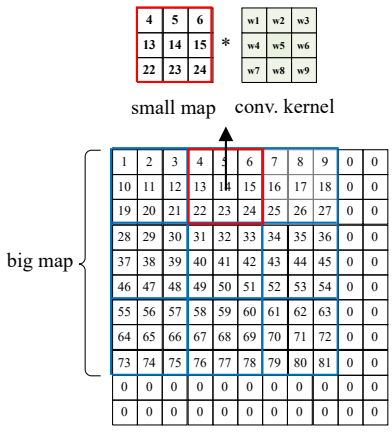
In ordinary convolution, the convolutional kernel moves from the top-left to the bottom-right corner of the input image, multiplying with the respective pixels along the way and summing up the results to generate a new output pixel. With our proposed packing approach, the computations of ordinary convolution is divided into three distinct types, including Type-1, Type-2 and Type-3 convolution, which are based on the rotation amounts and direction.

4.1.1 *Type-1 convolution*

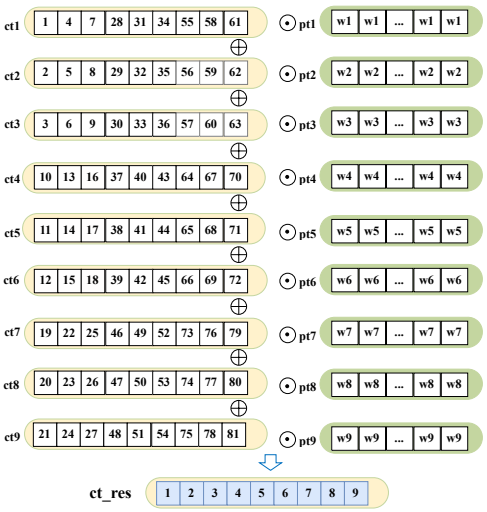
Taking $k = 3, i = 9$ as an example, the diagram illustrating Type-1 convolution in Figure 7 demonstrates that this convolution can be executed without rotating the ciphertexts of the input image. In Type-1 convolution, the image is divided into $(\lceil i/k \rceil)^2$ small maps, each containing k^2 pixels, aligning with our input packing approach. As depicted in Figure 7 (b), the image ciphertexts undergo multiplication with the convolutional kernel and are then aggregated into a single ciphertext. In homomorphic encryption, a rescaling step is necessary for homomorphic multiplication. In our method, rescaling occurs subsequent to accumulation, reducing the need for $k^2 - 1$ rescaling operations and thereby boosting performance. It is important to note that Type-1 convolution constitutes just a segment of the complete ordinary convolution process, with Type-2 and Type-3 convolutions being applied subsequently.

4.1.2 *Type-2 convolution*

The specifics of Type-2 convolution are depicted in Figure 8, where the big map is shifted to the right by 1 pixel. Rotations are necessary during convolution to align pixel positions correctly. In this instance, *ct1*, *ct4*, and *ct7* are rotated by 1. Additionally, the plaintexts of the corresponding convolutional kernels are adjusted and some slots are masked with zeros to reflect the actual operations. To mitigate the impact of irrelevant pixels, some plaintexts are masked. Rescalings are also conducted following accumulations to reduce the number of operations. The big map will be shifted by more pixels to the right during the whole ordinary convolution. When the big map is shifted by 2 pixels to the right, the ciphertexts *ct1*, *ct4*, *ct7*, *ct2*, *ct5*, and *ct8* are each rotated by 1. Notably, ciphertexts rotated by 1 (*ct1*, *ct4*, *ct7*) are utilized in both scenarios, enabling their reuse to avoid redundant calculations (termed **rotation multiplexing**). The situation where the big map is shifted by 3 pixels to the right is already addressed in Type-1 convolution.



(a) Conventional computation.



(b) HE convolution.

Fig. 7. Type-1 convolution (multiplication and addition operations are performed without any rotations). The big map is the whole input feature map.

Typically, the count of ciphertexts rotated by 1 is $k \cdot t$ when the big map is shifted by t pixels ($t < k$). The rotated ciphertexts in a right shift of t are given by:

$$\text{Rotation}(ct_{k \cdot m+n}, 1), 0 \leq m \leq k-1, 1 \leq n \leq t \qquad (3)$$

Here, $\text{Rotation}(ct, j)$ denotes the rotation of ct by j. Furthermore, the positions of slots masked as zeros are $p \cdot \lceil i/k \rceil$, where $1 \leq p \leq k$.

4.1.3 *Type-3 convolution*

The figure presented in Figure 9 illustrates a typical Type-3 convolution, involving the downward movement of the big map by 1 unit. Just like in Type-2 convolution, rotations and masking operations are necessary before conducting homomorphic multiplications. Specifically, in this instance, *ct1*, *ct2*, and *ct3* undergo a rotation of 3 units. When the big map is shifted downward by 2 units, *ct1*, *ct2*, *ct3*, *ct4*, *ct5*, and *ct6* are rotated by 3 units. Notably, to streamline operations, the ciphertexts rotated by 3 units (*ct1*, *ct2*, and *ct3*) are recycled to minimize complexity, which is the case of rotation multiplexing.

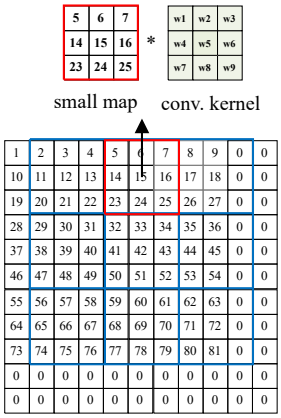
Generally, for a big map down-slided by t units, the number of ciphertexts rotated by k units is $k \cdot t$. This can be expressed as follows:

$$\text{Rotation}(\text{ct}_{k \cdot n+m}, k), 1 \leq m \leq k, 1 \leq n \leq t \qquad (4)$$

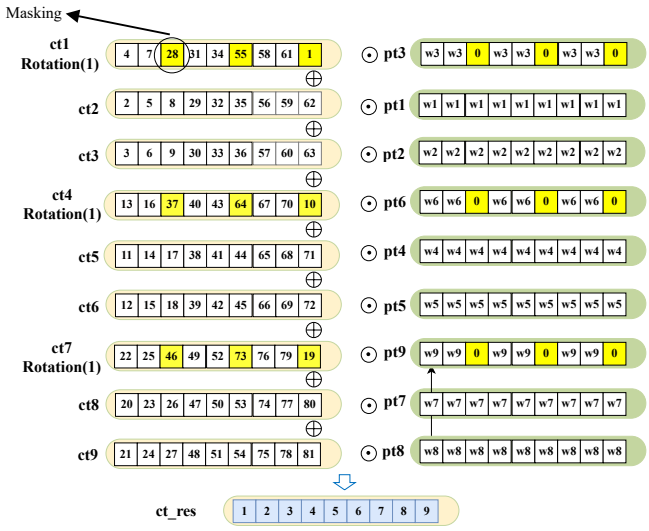
In this scenario, the slots masked as zeros amount to $k^2 - p$, where $0 \leq p \leq k-1$.

TABLE 1
Comparison of convolutions with Lola.

	# of rotations	parallel inputs	slot utilization
Lola [14]	$k^2 - 1$	1	$i^2/2N$
Ours	$(3k-1)(k-1)$	$N \cdot k^2/(2i^2)$	$\lfloor N/(2i^2) \rfloor$



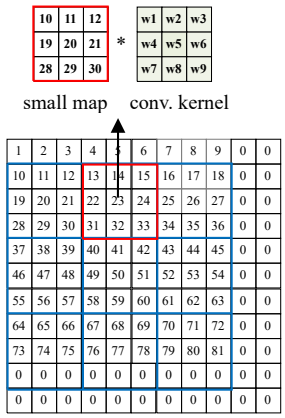
(a) Conventional computation.



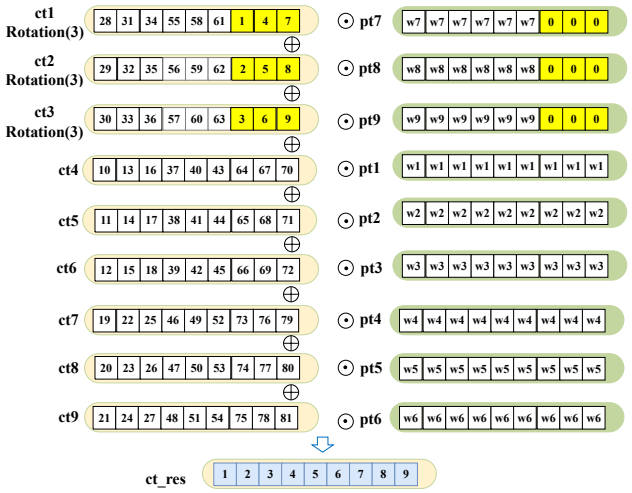
(b) HE convolution with right-slided by 1.

Fig. 8. Type-2 convolution (require rotations by 1). The big map is right-slided by 1.

When the big map is shifted both to the right and down, Type-2 and Type-3 convolutions are simultaneously applied. It is important to note that there exist identical rotated ciphertexts, which means rotation multiplexing can be uti-



(a) Conventional computation.



(b) HE convolution with down-slided by 1.

Fig. 9. Type-3 convolution (require rotations by 3). The big map is down-slided by 1.

lized. The rotated ciphertexts during ordinary convolution satisfy the following:

$$\text{Rotation}(\text{ct}_{k\cdot m+n}, k+1), 0 \le m \le k-2, 1 \le n \le k-1 \quad (5)$$

Throughout the entire ordinary convolution process, Type-1, Type-2, and Type-3 convolutions are employed together. By referencing Equation 3, Equation 4, and Equation 5, the total count of rotations during the whole ordinary convolution process amounts to $(3k-1)(k-1)$ utilizing our rotation multiplexing method. The comparison between our approach and Lola in terms of rotation count, parallel level, and slot utilization is depicted in Table 1. Our proposed packing and convolution method exhibits a reduced number of rotation operations per input image and a higher rate of slot utilization compared to Lola.

4.2 Stride Convolution

A convolution operation with a stride greater than 1 is referred to as a stride convolution. Figure 10 depicts an illustration of a stride-2 convolution, where only specific parts of the convolutional kernel interact with certain image pixels. By analyzing the computational patterns, this type

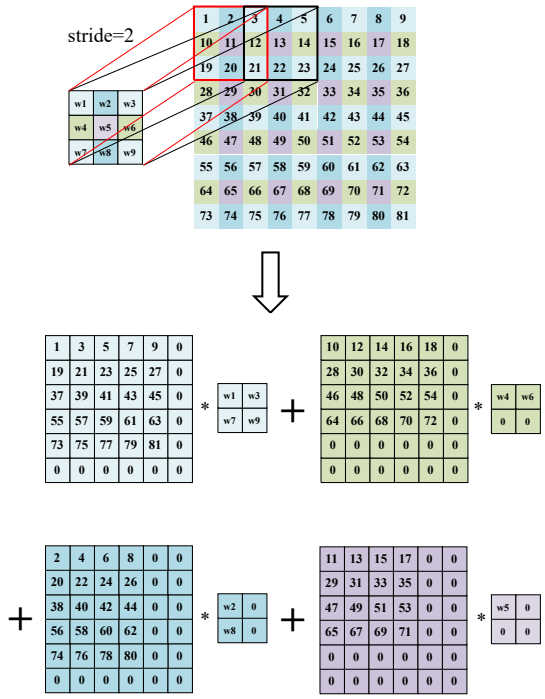


Fig. 10. HE stride convolution with $stride = 2$. The whole operation is split into four ordinary convolutions of smaller convolutional kernels.

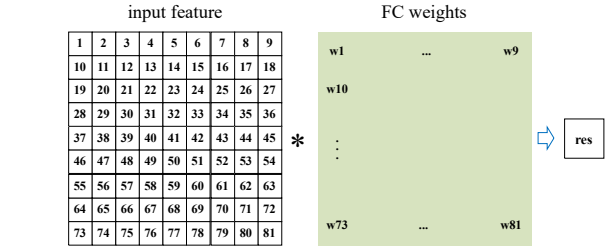
of convolution can be broken down into four ordinary convolutions, as shown in the same figure. Therefore, stride convolution can be converted to several ordinary convolution so that the optimization technique of Type-1, 2, and 3 convolution can be utilized. When a stride convolution serves as the initial layer of a CNN model, adjustments in the image mapping scheme are necessary to align with the decomposition outcomes and minimize additional rotations.

For a given stride value, denoted as s, the number of ciphertexts aligning with the decomposition results is approximately $\lceil k/s \rceil^2 \cdot s^2$, achieving a slot utilization rate close to 100%. Additionally, exploiting the presence of zeros in the decomposed convolutional kernel permits a further reduction in computations by eliminating homomorphic multiplications involving these zeros. This optimization not only lowers the number of rotations but also reduces the overall number of multiplications required.

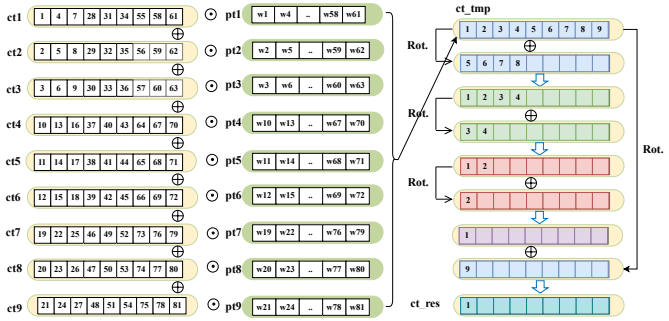
A comparison in Table 2 between ordinary convolution and stride-2 convolution reveals the operational overhead differences. Specifically, ordinary convolution necessitates approximately four times more homomorphic multiplications and additions than stride-2 convolution, highlighting the benefits of the proposed optimizations for stride convolutions.

TABLE 2
Comparison of ordinary convolution and stride convolution.

stride	1	2
# of rotations	$(3k-1)(k-1)$	$3k(k-2)$
# of multiplications	k^4	$k^2(k+1)^2/4$
# of additions	$k^2(k^2-1)$	$(k^2-4)(k+1)^2/4$



(a) Conventional FC.



(b) HE FC.

Fig. 11. Full connection layer. Ciphertexts are multiple with plaintexts, and folding rotation is performed to reduce the number of rotations.

4.3 Pooling Layer

In our proposed HE CNN inference scheme, sum pooling, as in [24], [25], is utilized. The complexity of encrypted comparisons makes max pooling and other pooling methods more challenging to implement. Just like convolution operations, the pooling layer involves sliding the feature maps. Hence, the execution of sum pooling operations necessitates comparable homomorphic rotations and additions.

4.4 Activation Layer

[15], [17], [26] have suggested the utilization of the square activation function $f(x) = x^2$ to mimic non-linear operations. The square function necessitates a ciphertext that undergoes homomorphic multiplication, thereby utilizing a substantial multiplication level of the ciphertext. Despite this, it is a crucial step as a non-linear layer plays a vital role in obtaining favourable CNN outcomes. In alignment with this notion, our design also adopts the square activation function due to its simplicity and effectiveness in encrypted computations.

4.5 Full Connection Layer

Figure 11 illustrates a segment of operations within the full connection layer, where the input feature is derived from preceding layers in the form of a 2-dimensional matrix. For clarity, only a portion of the full connection operations is depicted, involving the multiplication of the input feature with the FC weights, resembling a convolution process. In homomorphic encryption FC computations, the input feature is typically not continuously maintained as ciphertext due to prior convolutional procedures. As depicted in Figure 11 (b), the FC weights are encoded as plaintexts to align with the pixel sequence using our auto-compile tool, followed by homomorphic multiplications and accumulation. Furthermore, the resulting ciphertext from accumulation is rotated to aggregate all slot values. A **folding rotation** strategy is implemented to rotate and aggregate all slots, necessitating approximately $\log_2(N)$ rotations. This strategy can reduce the number of rotations compared with applying the rotations on ciphertexts directly. It is important to note that only the ultimate result slot is of significance; the remaining slots are deemed insignificant and are disregarded.

5 Implementations and Results

This section will present our implementation of convolution neural networks on MINST [27] and CIFAR-10 [28] datasets. We carried out several experiments to prove the advantage of our proposed methods in terms of throughput, performance, and memory overhead under similar CNN accuracy.

TABLE 3
Convolution tasks setting and evaluation results in performance.

	task#1	task#2	task#3	task#4
stride	1	2	1	2
input size	28×28	28×28	32×32	32×32
kernel size	5×5	5×5	3×3	3×3
additions (ms)	132	44	19	7
multiplications (ms)	357	102	37	19
rotations (ms)	1697	1093	432	247
total (ms)	2186	1239	488	273

5.1 Experimental Settings

We implemented the HE CNN inference with C++ based on the homomorphic encryption library Microsoft SEAL 4.0 [29]. The CNN models are trained with Pytorch [30] and reimplemented with C++ code for HE CNN inference. The experiment running machine is Intel Core i5-10400 CPU@ 2.90 GHz with 192 GB memory, and the operating system is Windows 10 64-bit. It is noted that all the programs ran with a single CPU thread rather than multiple threads. The experimental results are compared with prior works that have similar model accuracy.

5.2 Evaluations of Ordinary Convolution and Stride Convolution

Four convolution tasks are set to evaluate the performance of ordinary convolution and stride convolution, as described in Table 3. Note that 32×32 inputs are padded zeros from 28×28 inputs. The evaluation results of different numbers of operations are also presented in Table 3. The running time of homomorphic addition and multiplication of stride-2 convolution (**task#2, #4**) are about 2 to $3\times$ less than ordinary convolution (**task#1, #3**). While the homomorphic rotation of stride-2 convolution is about 60% to 100% of ordinary convolution. Therefore, our proposed scheme has a better performance in stride convolution. In this case, the CNN model can be designed with a more stride convolutional layer to enhance performance for the HE CNN inference application.

TABLE 4
Evaluation results of two MNIST models.

MNIST model A		MNIST model B	
layer	**latency (ms)**	**layer**	**latency (ms)**
CONV1	1921	CONV1	1956
ACT1	413	ACT1	412
FC1	7248	CONV2	4004
ACT2	516	ACT2	923
FC2	305	FC1	794
Total	10403	**Total**	8090
accuracy	98.95%	**accuracy**	98.83%

5.3 Evaluation on MNIST

To prove the advantage of our proposed scheme, we trained and implemented two CNN models on MNIST. The details of the models are as follows:

- MNIST model A:
  - **CONV1**: $5 \times 5 \times 5$ stride-2 convolutional layer
  - **ACT1**: square activation layer
  - **FC1**: 980×100 full connection layer
  - **ACT2**: square activation layer
  - **FC2**: 100×10 full connection layer
- MNIST model B:
  - **CONV1**: $5 \times 5 \times 5$ stride-2 convolutional layer
  - **ACT1**: square activation layer
  - **CONV2**: $5 \times 5 \times 20$ ordinary convolutional layer
  - **ACT2**: square activation layer
  - **FC1**: 3920×10 full connection layer

Model A is designed with one convolutional layer, and model B is designed with two convolutional layers to prove the flexibility for supporting continuous convolutions of our scheme. Stride-2 convolution is used for models A and B and split into 4 ordinary convolutions. 5×5 convolutional kernel are spilt into four 3×3 convolution kernels. On the other hand, the 28×28 input feature (MNIST image size) is split into four 14×14 new features. Therefore, each image requires $\lceil 14/3 \rceil^2 = 25$ slots of one ciphertext. There are a total of $4 \times 3 \times 3 = 36$ ciphertexts and 36 convolution kernel plaintexts. In MNIST CNN homomorphic inference, the polynomial size is set to $N = 8192$, which provides 4096 ciphertext slots. The security level λ is set to 128, with moduli bits $\{40, 21, 21, 21, 21, 21, 21, 40\}$ to ensure 6 homomorphic multiplications. Since the number of slots is 4096, the maximum number of images encoded and encrypted in one ciphertext is $\lfloor 4096/25 \rfloor = 163$ for both two models, which is the parallel level of HE CNN inference.

In MNIST model A, the output of CONV1 has a dimension of $14 \times 14 \times 5$ and is activated by the square function. The results are multiple with FC1 weights and accumulated as a 1×100 vector. ACT2 and FC2 are performed to get the final outputs (10 classes). In MNIST model B, the output of CONV1 is also $14 \times 14 \times 5$. CONV2 produces a $14 \times 14 \times 20$ result and FC1 reduces the dimension to 1×10. The detailed latency of each layer of the two models is presented in Table 4. As can be seen, models A and B have similar accuracy, but B is faster. The reason is that FC1 of MNIST model A has 100 output channels, which requires more homomorphic rotations. Neither the least, our proposed scheme is able to perform more than one convolution without decryption and re-encryption.

5.3.1 Comparisons with Prior Works

To show the supremacy of our proposed scheme on MNIST, we compare our results with state-of-the-art works [14], [15], [17], [24], [31], [32] in terms of ciphertext size, memory overhead, latency, throughput, and accuracy, as depicted in Table 5. As can be seen, the compared works have similar model accuracy with our works. [24] used the full-ciphertext packing scheme with 112 CPUs for acceleration. However, the average throughput of one CPU is less than our work. The other works applied the one-ciphertext packing scheme. Due to their low slot utilization, their throughput is several orders less than our work. Moreover, our works utilize much less memory (less than 10 GB) for storing the encoded CNN model plaintext.

5.4 Evaluation on CIFAR-10

The details of our CIFAR-10 model are as follows:

- **CONV1**: $5 \times 5 \times 64$ stride-2 convolutional layer
- **Dropout1**: drop-out layer with a probability of 0.25
- **ACT1**: square activation layer
- **POOL1**: 2×2 sum pooling layer
- **CONV2**: $3 \times 3 \times 64$ ordinary convolutional layer
- **ACT2**: square activation layer
- **Dropout2**: drop-out layer with a probability of 0.25
- **FC1**: 4096×128 full connection layer
- **ACT3**: square activation layer
- **FC2**: 128×10 full connection layer

In CIFAR-10 model, the output of CONV1 has a dimension of $16 \times 16 \times 64$ and is activated by the square function. During model training, the CONV1 results are dropped out with a probability of 0.25. The dropout layers are only available during model training, and removed during model inference. POOL1 is applied later to reduce the feature map size to $8 \times 8 \times 64$. The output of CONV2 has a dimension of $8 \times 8 \times 64$ and the output of FC1 has a dimension of 128×1. FC2 obtains a dimension of 128×1 input and produces 10×1 output features. In CIFAR-10 CNN homomorphic inference, the polynomial size is set to $N = 16384$, which provides 8192 ciphertext slots with a parallel packing of 227 images. The detailed latency of each layer of the CIFAR-10 model is presented in Table 6.

5.4.1 Comparisons with Prior Works

We compare our CIFAR-10 model results with prior works [14], [15], [17], as shown in Table 7. Our CIFAR-10 model required more memory due to the larger CNN model parameters. However, our throughput is much larger than prior works, with a more than $36\times$ improvement. Nevertheless, it can be seen that our proposed scheme has relatively small memory overhead and high throughput, which would be good for edge computing applications. To prove the advantage of our scheme, we additionally implemented the SqueezeNet model for the CIFAR-10 dataset, which was proposed from [33], the same as the model utilized in [17]. This model comprises one ordinary convolutional layer with

TABLE 5
Comparison with prior works on MNIST.

	message (MB)	**memory (GB)**	**latency (s)**	**throughput ($image/s$)**	**model accuracy (%)**
Lola [14]	51	32	16	0.06	98.95
nGraph-HE [24]	368	376	135 (with 112 CPUs)	60.6	98.95
EVA [17]	63	190	5.6	0.17	99.32
Falcon [15]	51	32	1.2 (with 8 CPUs)	0.83	98.95
Qin [31]	-	16	110	0.01	98.7
SpENCNN [32]	-	256	0.38	2.63	98.95
Our MNIST model A	**36**	**5.9**	**10.4**	**15.6**	**98.95**
Our MNIST model B	**36**	**3.3**	**8.9**	**18.3**	**98.83**

TABLE 6
Evaluation results of CIFAR-10 model.

layer	**latency (s)**
CONV1	109.6
ACT1	16.2
POOL1	36.5
CONV2	423.3
ACT2	10.7
FC1	79.1
ACT3	1.8
FC2	0.8
Total	678.0
accuracy	74.19%

a kernel size of 3 serving as the first layer, followed by four Fire modules, a conventional layer, and an average pooling layer. The Fire modules are made up of two convolutional layers with a kernel size of 1 and one convolutional layer with a kernel size of 3. For more details, readers can refer to [34]. The SqueezeNet model requires more running time but has a smaller memory overhead. When running with the polynomial size of 16384, the number of parallel packing images is $8192/\lceil 32/3 \rceil^2 = 67$, which indicates a smaller number of parallelism. As a result, the throughput of the SqueezeNet model is lower than our proposed CIFAR-10 model but with better accuracy. On the other hand, the throughput of the SqueezeNet model is still several orders better than prior works, which shows the advantage of our proposed method.

6 Conclusion

In this paper, we have proposed HTCNN, a novel batch homomorphic CNN inference scheme designed to support batch computation on edge computing platforms. We introduced an innovative image packing scheme alongside a corresponding CNN model packing scheme. Building on these foundations, we present several optimizations for homomorphic CNN inference aimed at enhancing performance and reducing memory usage. Experimental results on the MNIST and CIFAR-10 datasets demonstrate that our scheme achieves high throughput and relatively low memory usage compared to prior works, offering significant advantages for edge computing applications.

References

[1] J. Wu, "Introduction to convolutional neural networks," *National Key Lab for Novel Software Technology. Nanjing University. China*, vol. 5, no. 23, p. 495, 2017.

[2] A. B. Patel, M. Birla, and U. Nair, "Addressing big data problem using hadoop and map reduce," in *2012 Nirma University International Conference on Engineering (NUiCONE)*. IEEE, 2012, pp. 1–5.

[3] J. Patel, "Bridging data silos using big data integration," *International Journal of Database Management Systems*, vol. 11, no. 3, pp. 01–06, 2019.

[4] Z. Xiao and Y. Xiao, "Security and privacy in cloud computing," *IEEE communications surveys & tutorials*, vol. 15, no. 2, pp. 843–859, 2012.

[5] M. Zhou, R. Zhang, W. Xie, W. Qian, and A. Zhou, "Security and privacy in cloud computing: A survey," in *2010 sixth international conference on semantics, knowledge and grids*. IEEE, 2010, pp. 105–112.

[6] M. Albrecht, M. Chase, H. Chen, J. Ding, S. Goldwasser, S. Gorbunov, S. Halevi, J. Hoffstein, K. Laine, K. Lauter *et al.*, "Homomorphic encryption standard," *Protecting privacy through homomorphic encryption*, pp. 31–62, 2021.

[7] J. W. Bos, K. Lauter, J. Loftus, and M. Naehrig, "Improved security for a ring-based fully homomorphic encryption scheme," in *Cryptography and Coding: 14th IMA International Conference, IMACC 2013, Oxford, UK, December 17-19, 2013. Proceedings 14*. Springer, 2013, pp. 45–64.

[8] Z. Brakerski, C. Gentry, and V. Vaikuntanathan, "(leveled) fully homomorphic encryption without bootstrapping," *ACM Transactions on Computation Theory (TOCT)*, vol. 6, no. 3, pp. 1–36, 2014.

[9] J. Fan and F. Vercauteren, "Somewhat practical fully homomorphic encryption," *Cryptology ePrint Archive*, 2012.

[10] I. Chillotti, N. Gama, M. Georgieva, and M. Izabachène, "TFHE: fast fully homomorphic encryption over the torus," *Journal of Cryptology*, vol. 33, no. 1, pp. 34–91, 2020.

[11] J. H. Cheon, A. Kim, M. Kim, and Y. Song, "Homomorphic encryption for arithmetic of approximate numbers," in *Advances in Cryptology–ASIACRYPT 2017: 23rd International Conference on the Theory and Applications of Cryptology and Information Security, Hong Kong, China, December 3-7, 2017, Proceedings, Part I 23*. Springer, 2017, pp. 409–437.

[12] J. H. Cheon, K. Han, A. Kim, M. Kim, and Y. Song, "A full RNS variant of approximate homomorphic encryption," in *Selected Areas in Cryptography–SAC 2018: 25th International Conference, Calgary, AB, Canada, August 15–17, 2018, Revised Selected Papers 25*. Springer, 2019, pp. 347–368.

[13] 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 *International conference on machine learning*. PMLR, 2016, pp. 201–210.

[14] A. Brutzkus, R. Gilad-Bachrach, and O. Elisha, "Low latency privacy preserving inference," in *International Conference on Machine Learning*. PMLR, 2019, pp. 812–821.

[15] Q. Lou, W.-j. Lu, C. Hong, and L. Jiang, "Falcon: Fast spectral inference on encrypted data," *Advances in Neural Information Processing Systems*, vol. 33, pp. 2364–2374, 2020.

[16] C. Juvekar, V. Vaikuntanathan, and A. Chandrakasan, "{GAZELLE}: A low latency framework for secure neural network inference," in *27th USENIX security symposium (USENIX security 18)*, 2018, pp. 1651–1669.

[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 *Proceedings of the 41st ACM SIGPLAN conference on programming language design and implementation*, 2020, pp. 546–561.

TABLE 7
Comparison with prior works on Cifar10.

	message (MB)	**memory (GB)**	**latency (s)**	**throughput ($10^{-3} \cdot image/s$)**	**model accuracy (%)**
Lola [14]	210	32	730	1.4	74.1
EVA [17]	63	190	72.7	13.9	79.3
Falcon [15]	210	32	107 (with 8 CPUs)	9.3	76.5
Our CIFAR-10 model	**297**	44	**678**	**334.8**	**75.3**
SqueezeNet	**74**	30	**2170**	**30.9**	**79.1**

[18] K. Cao, Y. Liu, G. Meng, and Q. Sun, "An overview on edge computing research," *IEEE access*, vol. 8, pp. 85 714–85 728, 2020.

[19] J. Chen and X. Ran, "Deep learning with edge computing: A review," *Proceedings of the IEEE*, vol. 107, no. 8, pp. 1655–1674, 2019.

[20] H. Zhou, G. Yang, Y. Huang, H. Dai, and Y. Xiang, "Privacy-preserving and verifiable federated learning framework for edge computing," *IEEE Transactions on Information Forensics and Security*, vol. 18, pp. 565–580, 2022.

[21] H. Zhong, L. Wang, J. Cui, J. Zhang, and I. Bolodurina, "Secure edge computing-assisted video reporting service in 5g-enabled vehicular networks," *IEEE Transactions on Information Forensics and Security*, vol. 18, pp. 3774–3786, 2023.

[22] W. Z. Srinivasan, P. Akshayaram, and P. R. Ada, "DELPHI: A cryptographic inference service for neural networks," in *Proc. 29th USENIX Secur. Symp*, 2019, pp. 2505–2522.

[23] R. Yamashita, M. Nishio, R. K. G. Do, and K. Togashi, "Convolutional neural networks: an overview and application in radiology," *Insights into imaging*, vol. 9, pp. 611–629, 2018.

[24] F. Boemer, Y. Lao, R. Cammarota, and C. Wierzynski, "nGraph-HE: a graph compiler for deep learning on homomorphically encrypted data," in *Proceedings of the 16th ACM international conference on computing frontiers*, 2019, pp. 3–13.

[25] F. Boemer, A. Costache, R. Cammarota, and C. Wierzynski, "nGraph-HE2: A high-throughput framework for neural network inference on encrypted data," in *Proceedings of the 7th ACM workshop on encrypted computing & applied homomorphic cryptography*, 2019, pp. 45–56.

[26] 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 *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 9403–9412.

[27] L. Deng, "The mnist database of handwritten digit images for machine learning research [best of the web]," *IEEE signal processing magazine*, vol. 29, no. 6, pp. 141–142, 2012.

[28] A. Krizhevsky, G. Hinton *et al.*, "Learning multiple layers of features from tiny images," 2009.

[29] "Microsoft SEAL (release 4.0)," https://github.com/Microsoft/SEAL, Mar. 2022, microsoft Research, Redmond, WA.

[30] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "PyTorch: An Imperative Style, High-Performance Deep Learning Library," in *Advances in Neural Information Processing Systems 32*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d 'Alché-Buc, E. Fox, and R. Garnett, Eds. Curran Associates, Inc., 2019, pp. 8024–8035.

[31] H. Qin, D. He, Q. Feng, and M. Luo, "Secure cnn training and inference based on multi-key fully homomorphic encryption," in *2023 IEEE 29th International Conference on Parallel and Distributed Systems (ICPADS)*. IEEE, 2023, pp. 370–377.

[32] R. Ran, X. Luo, W. Wang, T. Liu, G. Quan, X. Xu, C. Ding, and W. Wen, "Spencnn: orchestrating encoding and sparsity for fast homomorphically encrypted neural network inference," in *International Conference on Machine Learning*. PMLR, 2023, pp. 28 718–28 728.

[33] David Corvoysier, "SqueezeNet for CIFAR-10," https://github.com/kaizouman/tensorsandbox/tree/master/cifar10/models/squeeze, 2017.

[34] F. N. Iandola, S. Han, M. W. Moskewicz, K. Ashraf, W. J. Dally, and K. Keutzer, "Squeezenet: Alexnet-level accuracy with 50x fewer parameters and¡ 0.5 mb model size," *arXiv preprint arXiv:1602.07360*, 2016.