FxHENN: FPGA-based acceleration framework for homomorphic encrypted CNN inference

Yilan Zhu[†‡], Xinyao Wang[†‡], Lei Ju[‡§*] and Shanqing Guo[†‡]
[†] Key Laboratory of Cryptologic Technology and Information Security, Ministry of Education, Shandong University
[‡] School of Cyber Science and Technology, Shandong University, Qingdao, China
[§] Quancheng Laboratory, Jinan, China
[*] Corresponding author: julei@sdu.edu.cn

2023 IEEE International Symposium on High-Performance Computer Architecture (HPCA) | 978-1-6654-7652-2/23/$31.00 ©2023 IEEE | DOI: 10.1109/HPCA56546.2023.10071133

***Abstract*—Fully homomorphic encryption (FHE) is a promising data privacy solution for machine learning, which allows the inference to be performed with encrypted data. However, it typically leads to 5-6 orders of magnitude higher computation and storage overhead. This paper proposes the first full-fledged FPGA acceleration framework for FHE-based convolution neural network (HE-CNN) inference. We then design parameterized HE operation modules with intra- and inter- HE-CNN layer resource management based on FPGA high-level synthesis (HLS) design flow. With sophisticated resource and performance modeling of the HE operation modules, the proposed FxHENN framework automatically performs design space exploration to determine the optimized resource provisioning and generates the accelerator circuit for a given HE-CNN model on a target FPGA device. Compared with the state-of-the-art CPU-based HE-CNN inference solution, FxHENN achieves up to 13.49X speedup of inference latency, and 1187.12X energy efficiency. Meanwhile, given this is the first attempt in the literature on FPGA acceleration of full-fledged non-interactive HE-CNN inference, our results obtained on low-power FPGA devices demonstrate HE-CNN inference for edge and embedded computing is practical.**

***Index Terms*—fully homomorphic encryption, convolution neural network, FPGA acceleration, high-level synthesis, design space exploration**

I. Introduction

When machine learning techniques are deployed to application domains with sensitive data (e.g., biomedical or financial data), data privacy becomes a crucial design consideration. Fully homomorphic encryption (FHE) [9], [14] is a promising methodology to protect data privacy for machine learning by enabling inference on encrypted data without decryption. Taking machine learning as a service [22] for example, to perform a convolutional neural network (CNN) inference service running on a cloud or edge computing server, users are required to submit their raw input data with privacy information to the third-party server, which may lead to data privacy leakage. With FHE, users will be able to encrypt data locally and submit the encrypted data to the server that supports HE-enabled CNN inference on the encrypted data without decryption. Finally, the encrypted results are sent back to clients for decryption and display. Only the clients who have the private key can decrypt the results, thus deep learning with FHE is an inspiring privacy-preserving method.

However, current deep neural networks with HE suffer from extremely high latency and large ciphertext sizes. For example, CryptoNets [15] first demonstrated a homomorphic encryption-enabled convolutional neural network (HE-CNN) inference on encrypted data, which takes $\sim$205 seconds to perform inference on one single MNIST image. The recent work LoLa [5] reduced the latency of CryptoNets to 2.2 seconds on a CPU server, by batching multiple pixels of an image into one single ciphertext to reduce the HE operations. Although substantial performance improvement has been achieved with algorithm-level and compilation-level optimization techniques on CPU-based platforms, the computational cost of HE-CNN inference is still too expensive for practical applications.

Meanwhile, heterogeneous processing units including GPUs, FPGAs, and ASICs have shown great success in domain-specific acceleration. In particular, tremendous progress has been made in the accelerator design for deep neural networks (DNN) inference tasks on resource-constrained hardware platforms [13]. As a result, hardware acceleration become a natural choice for HE-CNN. A*FV [2] presented the first GPU-accelerated HE-CNN which used low-precision training and optimized FHE parameters to achieve high performance on server GPUs. Cheetah [20] proposed an ASIC accelerator architecture that supports a multiple-layer neural network by interactively performing layer-wise convolution with HE and layer-wise activation with secure multi-party computation (MPC). Between each convolution and activation, the encrypted intermediate output is required to communicate, decrypt, and re-encrypt. This interactive method with the hybrid scheme between FHE and MPC brings a large computational and communication overhead [17]. In many application scenarios, an efficient accelerator for non-interactive privacy-preserving HE-CNN is required.

Compared with GPU and ASIC-based accelerators, power-efficient and re-programmable FPGA devices is a promising solution for CNN inference on edge and embedded computing platforms [19]. FPGA acceleration of elementary HE operations on FHE schemes has been studied in the past few years. In [21], Riazi et al. proposed FPGA-based hardware architecture for CKKS FHE, where the fundamental number-theoretic transform (NTT) blocks and high-level parallel operations can be scaled to different encryption parameters and hardware resources. In [27], multi-level parallelism design has been exploited to achieve better FPGA resource utilization for the HE Multiply-Accumulate operations in matrix-vector multiplication. In [16], an FPGA acceleration framework has been proposed for FHE kernels based on the high-level synthesis (HLS) design flow. A very recent work [28] proposed an efficient FPGA accelerator design to speed up the performance of one single HE-enabled convolution layer. However, FPGA acceleration for full-fledged HE-CNNs with multiple different layers is still an open challenge for the community.

To the best of the authors' knowledge, this paper proposes the first full-fledged FPGA acceleration framework FxHENN for HE-CNN inference. In the proposed end-to-end accelerator design, all HE-CNN layers are deployed on a target FPGA, without any interaction with the client (input data owner) for decryption and re-encryption of intermediate results. Given an HE-CNN model and a target FPGA device, FxHENN automatically determines the optimized resource provisioning for computation and on-chip memory resources. The optimized resource allocation will be provided to the Xilinx Vivado toolchain to generate the accelerator circuit from a set of parameterized HE operation modules built with HLS. The experimental results with low-power FPGA (i.e., Xilinx Zynq UltraScale+ MPSOC with the thermal design power of 10W) show 0.19 seconds and 54.1 seconds inference latency on small- and medium-sized input data (i.e., MNIST and CIFAR10 datasets), respectively. Considering this is the first attempt in the literature on FPGA acceleration of full-fledged non-interactive HE-CNN inference, our results demonstrate HE-CNN inference for edge and embedded computing is practical. The contributions of this paper are as follows:

- We conduct an empirical study to demonstrate the open challenges of FPGA-based accelerator design for full-fledged CNN inference. In particular, we show there is severely unbalanced usage between computation resources (e.g., DSP) and on-chip block RAM (BRAM), which requires sophisticated resource management and automatic design space exploration for FPGA-based HE-CNN accelerator design.
- We propose an intra-layer pipeline execution scheme for HE operations for typical HE-CNN layers, as well as intra- and inter-layer resource reuse schemes for on-chip computation and storage resources. The FPGA HLS-based design provides parameterized HE operations and I/O connections, which allows flexible hardware accelerator generation for different HE-CNN models and FPGA devices.
- Based on the proposed resource-latency model, our FxHENN framework performs automatic design space exploration (DSE) to obtain the optimized resource provisioning for a given HE-CNN model and a target FPGA platform. The DSE process generates parameters which determine the structure, parallelism, and on-chip buffer usage of the prebuilt HE modules in various HE-CNN layers to achieve high resource efficiency and overall performance.
- We evaluate the performance and flexibility of FxHENN with two HE-CNN models on two commercial-off-the-shelf (COTS) Xilinx FPGA devices. Compared with the state-of-the-art CPU-based HE-CNN solution LoLa [5], FxHENN achieves up to 13.49X speedup of inference latency, and 1187.12X energy efficiency.

II. Background

A. Fully Homomorphic Encryption

Fully Homomorphic Encryption (FHE) supports arbitrary polynomial computations, *i.e.*, multiplication and addition, on encrypted data without decryption. To be specific, given two plaintexts $m1$ and $m2$, the party who has a public key can encrypt them into two ciphertexts $ct1$ and $ct2$ respectively. Then homomorphic addition and multiplication are supported by the equation $Dec(ct1+ct2) = m1+m2, Dec(ct1\times ct2) = m1 \times m2$, where $Dec()$ function defines the decryption that requires a secret key so that only the owner of the secret key can decrypt a ciphertext.

RNS and Rescale. The CKKS scheme [9] that deals with any real numbers is one of the most promising FHE schemes. It is especially suitable for machine learning applications that often represent parameters as real numbers. Its advanced version RNS-CKKS [8] with Residue Number System (RNS) enables parallel word-wise operations and reduces the latency more than $10\times$. Given a polynomial ring $R = Z[X]/(X^N + 1)$, the CKKS ciphertext can be defined as polynomials in the residue ring $R_Q = R/QR$, where N is a power-of-two integer and Q is the coefficients modulus. In CKKS, Q is very large, *e.g.*, several hundreds bits [9]. During RNS-CKKS, Q is decomposed into L levels, and each level i has a decomposed factor q_i where $Q = \prod_{i=1}^{L} q_i$, and q_i is a smaller word-wise number. Both CKKS and RNS-CKKS use the Rescale operation after HE multiplication to manage the magnitude of plaintext by truncating a ciphertext into a smaller modulus, which corresponds to rounding of plaintext.

Batching and Rotate. The batching technique in FHE enables encrypting multiple messages into a single ciphertext and performs parallel processing in single instruction multiple data (SIMD) manner. During the batching, each element in the message is encoded into a 'slot'.The maximum number of slots in CKKS is $N/2$. Rotate can change the position of the numbers within the slots.

Homomorphic Operations. Specifically, given a homomorphic application, *e.g.,* CNN, it can be implemented with HE operations, which contain plaintext-ciphertext addition (PCadd), plaintext-ciphertext multiplication (PCmult), ciphertext-ciphertext addition (CCadd), ciphertext-ciphertext multiplication (CCmult), Rescale, Relinearize, and Rotate operations. Because Relinearize and Rotate have similar algorithms, we use KeySwitch to denote a Relinearize or Rotate operation in this paper. During the practical implementation, all these HE operations are converted into basic operations such as NTT/INTT, Barrett Reduction, Modular Multiplication, Modular Subtraction, and Modular Addition operations.

B. Deep Neural Networks with FHE

CryptoNets [15] is the first work to show the feasibility of performing a neural network inference on encrypted data without decryption. Deep neural networks consist of polynomial operations, *i.e.*, convolutions, and non-polynomial operations, *i.e.*, $ReLU$ activation. FHE directly supports polynomial operations, but is incapable of supporting non-polynomial $ReLU$

activation. CryptoNets solved this issue by using polynomial approximation, *i.e.*, replacing $ReLU$ with $square$ function. However, CryptoNets suffers from enormous latency overhead although the ciphertext batching technique was used to improve throughput. To reduce the latency, Fast CryptoNets [10] leverages sparse representations for plaintext to speed up HE multiplication. Falcon [18] reduces the number of rotation operations by using homomorphic DFT for convolutional and fully connected layers. Moreover, improving the data packing method for a single image can also reduce the inference latency [5], [18]. These works encrypt multiple pixels in one image to slots of a single ciphertext, thus substantially reducing the number of HE operations by tens to hundreds of times and the amount of memory required as well. Although these schemes affect the number of HE operations and thus the efficiency, there are still commonalities for neural networks (computations such as convolutional layers) and the same homomorphic operations (homomorphic addition multiplication, etc.). We can exploit these features for HE-enabled CNN speedup on FPGA. What's more, [6], [11], [12], [26] are homomorphic-based compilers that can automate the generation of homomorphic implementation for application, even for CNN. nGraph [3], [4] is a specialized compiler for converting CNN into HE-based CNN. But they are still CPU-based frameworks, making it difficult to achieve large performance improvements.

Meanwhile, accelerating HE-CNN on heterogeneous processors is expected to outperform CPU platforms. The state-of-the-art work CHOCO [25] and Cheetah [20] both accelerated the HE-CNN model based on the MPC scheme. F1 [23] and CraterLake [24] optimized the homomorphic operators and achieved the acceleration of the neural networks. But all these works are based on ASICs and cannot be configured to accelerate specific neural networks.

To accelerate HE-CNN on FPGA, we need to know that converting the neural network into a homomorphic version is not straightforward. After packing CNN data in HE, the operation logic of the ciphertext will be different from the original CNN logic. And exclusive HE operations such as Rescale, Relinearize or Rotate will be added. As Listing 1 shows, LoLa [5] reorganized the layout of the CNN data placed in the ciphertext slots. The original convolution procedure is reduced to a single loop. Moreover, the multiplication is transformed into PCmult and Rescale. To make an accurate evaluation, we must extract the HE operations and data relations at this level.

```
Ciphertext in_data[25], output;
Plaintext in_filter[25];
void convLayer1(Ciphertext in_data[25],
  Plaintext in_filter[25],Ciphertext output){
    Buffer temp_c1[25];
    Buffer temp_c2[25];
    for(int i=0; i<25; i++){
        temp_c1[i]=HmultPlain(in_data[i],
                    in_filter[i]);
        temp_c2[i]=Rescale(temp_c1[i]);
        output=Hadd(temp_c2[i],output);}}
```

Listing 1. The first convolution layer of LoLa-MNIST [5] in CKKS.

III. Motivation

TABLE I
IMPLEMENTATION OF HE OPERATION MODULES ON ALINX ACU9EG.

	HE operations	nc_{NTT}	DSP (%)	BRAM blocks(%)	Latency (ms)
OP1	CCadd	-	0.00	10.53	0.25
OP2	PCmult	-	3.97	10.53	0.25
OP3	CCmult	-	3.97	15.79	0.25
OP4	Rescale	2	4.44	10.53	1.19
		4	7.30	10.53	0.68
		8	13.01	21.05	0.34
OP5	KeySwitch	2	10.08	35.09	3.17
		4	19.01	35.09	1.60
		8	28.61	70.18	0.81

The ultimate goal of the FxHENN framework is to automatically generate an FPGA-based accelerator for a given HE-CNN inference task on a target FPGA device. In this section, we demonstrate the key design issues and challenges for building the HE-CNN inference accelerator and the FxHENN framework.

Recent studies have proposed sophisticated design and optimization techniques for FPGA acceleration of the HE operations of CKKS scheme (e.g., [21] and [16]). With the reference of these literature works, we implement our high-level synthesis (HLS) based parameterized HE operation modules on an ALINX ACU9EG FPGA platform with the Xilinx Zynq UltraScale+ MPSoC XCZU9EG device (consisting of 2,520 DSP slices and 32.1Mbit on-chip BRAM blocks). Table I shows the five HE operations (labeled from "OP1" to "OP5"), possible design choices (nc_{NTT}), as well as corresponding resource usage percentage and latency.

The first observation is that the NTT module is the fundamental building block of both Rescale and KeySwitch (including Relinearize and Rotate) operations, and the performance bottleneck among all HE operations. In Table I, we show the impact of the internal NTT parallelism nc_{NTT} (i.e., the number of NTT cores in a basic-level NTT module) on the resource utilization and performance of the high-level HE modules. While nc_{NTT} for KeySwitch increases from 2 to 4, the BRAM block usage remains unchanged due to the dual-port BRAM design of the commercial off-the-shelf FPGA devices. Two basic NTT cores can read/write the same BRAM block in parallel within the same clock cycle. However, if 8 NTT cores are used in an NTT module for better performance, the same amount of the data has to be further partitioned into more different BRAM blocks to ensure the single-cycle access time by the doubled NTT cores.

TABLE II
A PRELIMINARY ACCELERATOR DESIGN FOR LOLA-MNIST [5] WITH PER-LAYER RESOURCE USAGE ON ACU9EG ($nc_{NTT} = 2$).

Layer	HE Operations	DSP (%)	BRAM (%)
Cnv1	OP1,OP2,OP4	10	25
Act1	OP3,OP4,OP5	18	57
Fc1	OP1,OP2,OP4,OP5	15	53
Act2	OP3,OP4,OP5	12	39
Fc2	OP1,OP2,OP4,OP5	10	32
Sum		65	206

说理的时候如何描述BRAM和IO
的balance

The second observation is that efficient utilization of on-chip BRAM resources is of vital importance for the HE-CNN accelerator design. Table II shows a preliminary implementation of the 5-layer LoLa-MNIST HE-CNN [5], with the per-layer HE operations and corresponding DSP/BRAM usage with the $nc_{NTT} = 2$ on ACU9EG. It can be observed that the hardware resource usage is severely unbalanced. While the demanding BRAM block usage has exceeded the total available BRAM on the FPGA chip (i.e., 206% of the available 32.1Mbit BRAM with 912 blocks), the most scarce computation resources, i.e., DSP, are under-utilized (e.g., only 65% of the total available DSP on ACU9EG). The high on-chip memory demand is due to the huge volume of HE ciphertext (typically 5~6 orders-of-magnitude larger than the plaintext) being accessed by the basic NTT operations in a non-burst manner. Therefore, accessing the ciphertext from off-chip main memory leads to substantial performance degradation. For example, Table III shows that accessing data from on-chip BRAM significantly improves the performance of HE-CNN inference. Taking the fully connected layer (Fc1) as an example, sufficient BRAM blocks (i.e., 773 blocks) may lead to 139.58X speedup for the inference latency of this layer, compared with accessing all data from the off-chip DRAM. As a result, efficient management of the scarce BRAM resource across different HE-CNN layers for an overall performance optimization is the key design challenge in FPGA acceleration of HE-based CNN inference.

TABLE III
THE IMPACT OF BRAM USAGE ON THE HE-CNN INFERENCE LATENCY.

Cnv1 layer		Fc1 layer	
BRAM36K	Latency(s)	BRAM36K	Latency(s)
292	0.021	773	0.162
0	0.334	0	22.612

TABLE IV
THE MACS COMPARISON BETWEEN CNN AND HE-CNN INFERENCE.

	MACs (10^4)	HOPs	MACs of HOPs (10^4)
Cnv1 layer	2.11	75	11980.7
Fc1 layer	8.45	325	155105.28

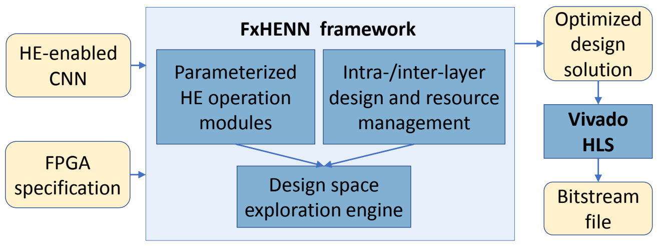
Finally, our observation indicates that an automatic design space exploration (DSE) framework is imperative for the HE-CNN inference accelerator to achieve high resource efficiency and overall performance. Besides the number of internal cores for the basic NTT module design (i.e., nc_{NTT} shown in Table I), there are several other design factors that have significant impacts on the FPGA resource provisioning as well as the overall system performance. For example, the design framework needs to decide the intra-parallelism of KeySwitch (how many parallel NTT modules are deployed in the high-level KeySwitch module), and the inter-parallelism of KeySwitch (how many parallel KeySwitch modules are deployed for a HE-CNN layer). Consider the Cnv1 and Fc1 layers of our motivating example LoLa-MNIST on ACU9EG, Table IV shows the multiply accumulate (MAC) operation count on the original CNN network (i.e., "MACs" without FHE), the HE operation count ("HOPs") and corresponding HE-MACs ("MACs of HOPs"). As shown in the result, while in the original CNN model there is 4X MAC difference between Fc1 and Cnv1, the difference grows to 12.95X in the HE-CNN model. Therefore, the inter-layer computation workload must be captured in the resource provisioning.

Table V demonstrates two potential resource allocation schemes between Cnv1 and Fc1, where only the intra-parallelism ("intra") of the KeySwitch operation is considered for DSE. Compared with configuration B, configuration A provides higher intra-parallelism for KeySwitch operations in Fc1 layer with a higher computation workload. As a result, configuration A achieves 2.07X speedup w.r.t. configuration B, while using even a smaller BRAM blocks. Therefore, an automatic DSE framework that provides near-optimal accelerator design solutions will be of paramount importance for the given HE-CNN inference on a target FPGA device.

TABLE V
DSE FOR CNV1 AND FC1 OF LOLA-MNIST [5] ON ACU9EG.

	Cnv1		Fc1		Sum		
	intra	Lat. (s)	intra	Lat. (s)	DSP (%)	BRAM (%)	Lat. (s)
A	1	0.062	3	0.29	18.1	43.9	0.352
B	4	0.021	1	0.709	27.9	49.1	0.73

IV. Overall Design Framework



HE-enabled CNN
FPGA specification
FxHENN framework
Parameterized HE operation modules
Intra-/inter-layer design and resource management
Design space exploration engine
Optimized design solution
Vivado HLS
Bitstream file

Fig. 1. Design flow of FxHENN.

FxHENN is a flexible design framework to generate an FPGA accelerator for a given HE-CNN inference task on a target FPGA device. The overall design flow is shown in Fig. 1. Similarly as most of the HE-CNN proposed in the literature (e.g., [5], [15], [18]), we assume the HE-CNN inference task is performed on ciphertext-input and plaintext-weight to protect the privacy of users' data. Moreover, we adopt the same FHE parameter selection scheme as in the literature works to support the multiplication depth and security level for the entire HE-CNN inference. The FPGA specification includes the capacity of the key FPGA resources (DSPs, LUTs, BRAM/URAM, etc), which will be used as design constraints in the accelerator generation.

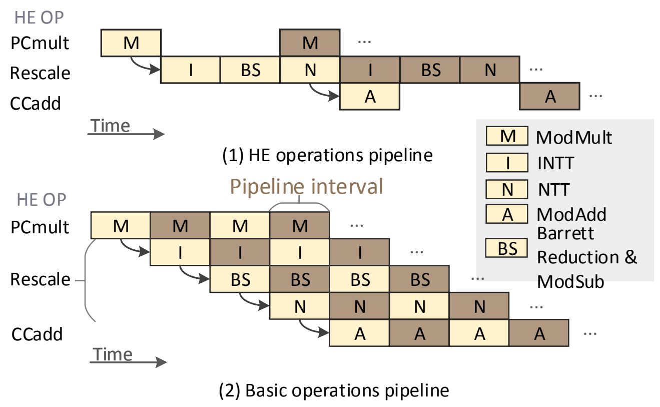
In the proposed FxHENN framework, we first implement a set of parameterized HE operation modules in C++, following the Xilinx Vivado HLS style. The implementation of FPGA-based basic HE operation modules references the recent literature work of [21] and [16]. In this work, we propose an intra-layer pipeline execution scheme for HE operations for typical HE-CNN layers, as well as intra-/ inter-layer resource reuse scheme for on-chip computation and BRAM resources. Finally, FxHENN performs automatic design space exploration (DSE) to obtain the optimized resource provisioning for a given HE-CNN model and a target FPGA platform. The output of the FxHENN framework is a dedicated accelerator design solution, which contains the structure information and HLS pragmas and directives for the parameterized HE operation modules. Then we use the commercial Vivado HLS toolchain to generate the bitstream files of the generated accelerator, which is executable on the COTS FPGA platforms.

V. Configurable HE-CNN design and optimization

Several recent studies have proposed FPGA-based acceleration of basic HE operations. We implement a set of parameterized HE operation modules with HLS technique. The FxHENN framework focuses on the application-specific design and optimization for combinations of HE operation modules in the context of HE-CNN inference. In this section, we will introduce the proposed intra-layer pipeline execution and inter-layer HE module reuse schemes to improve the FPGA resource efficiency and overall system performance.

A. *Optimization within HE-enabled CNN*

In this section, we discuss the common types of HE-CNN layers and corresponding pipeline-based performance optimization between HE operations within a single layer. We first classify the HE-CNN layers into two categories based on whether or not a particular layer contains the most time-consuming KeySwitch operations (including Relinearize and Rotate). In particular, "KS" type layers contain KeySwitch operations, while "NKS" layers contain no KeySwitch operation. The classification is based on the fact that the data dependency within KeySwitch is different from other HE operations, which leads to different pipeline design schemes.



HE OP
PCmult
Rescale
CCadd
M
I
BS
N
A
Time
(1) HE operations pipeline
Pipeline interval
(2) Basic operations pipeline
M ModMult
I INTT
N NTT
A ModAdd
BS Barrett Reduction & ModSub

Fig. 2. Pipeline execution design for the NKS type.

"NKS" type layers are usually resulted from convolutional and fully connected layers, where multiplications (PCmult in HE) are performed between feature maps and weights. The input of such layers contains multiple sets of independent ciphertexts, which enables pipeline design to reduce the execution latency.

There are two ways of pipelining an "NKS" layer as shown in Fig. 2. A straightforward pipeline design between high-level HE operations (i.e., "PCmult", "Rescale", and "CCadd") results in a coarse-grained scheme, where the time-consuming "Rescale" operation stage leads to an unbalanced pipeline stage and low throughput. We further propose a fine-grained basic operation level (i.e., "ModMult", "INTT", "NTT", "ModAdd", "Barrett Reduction" and "ModSub") pipeline scheme in FxHENN. The latency of an "NKS" layer is modeled as follows:

$$LAT_{NKS} = \frac{N_{in} \cdot PI}{P_{NKS}^{inter}} \tag{1}$$

where N_{in} indicates the number of input ciphertexts (determined by the given HE-CNN). PI denotes the time interval of a pipeline stage, which is determined by the internal parallelism of basic HE operations (refer to Section V-B). In addition, with sufficient resources, the latency can be reduced by deployment of P_{NKS}^{inter} such pipeline structures.

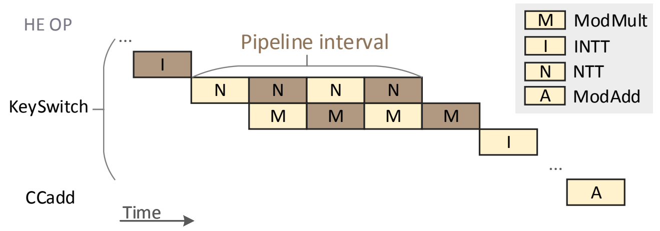


Fig. 3. Pipeline of the KS type.

For "KS" type layers, it is commonly used to compute homomorphic matrix multiplication in the fully connected layer. The vector is encrypted as ciphertexts, and then each row of the matrix is encoded as plaintexts. After PCmult operation of them, then summing up all the slots of the resulting ciphertext, which is equivalent to iterations of Rotate and CCadd operations, described in [5]. However, for the ciphertext result of the one matrix row, these iterations cannot be pipelined due to computational dependencies. But the calculations between matrix rows of ciphertext are independent. Specifically, the first round of Rotate and CCadd are performed on the first ciphertext, then they are performed on the second ciphertext until the last one, after which the second round of Rotate and CCadd are performed. This process of two adjacent Rotate is independent in computation, so the pipeline can be implemented.

We still perform a basic operation level pipeline for "KS" layers. As shown in Fig. 3, the pipeline stage is L times slower than that of the NKS type due to the KeySwitch algorithm, where L is the level of the ciphertext. The latency of a "KS" layer can be modeled as follows:

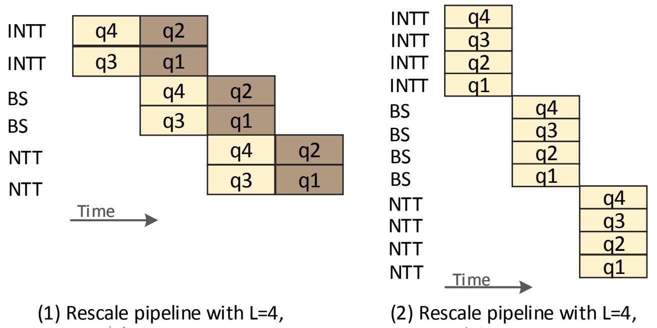
$$LAT_{KS} = \frac{N_{in} \cdot L \cdot PI}{P_{KS}^{inter}} \tag{2}$$

where P_{KS}^{inter} is the number of parallel deployed KS pipeline structures.

B. *Optimization within HE modules*

HE operation modules:

In this section, we describe the design of the parallelism within HE operation modules. The HE operations handle ciphertext with level L, which is composed of L different RNS polynomials. Each of the RNS polynomials can be denoted as $poly_{q_i}$, where $1 \leq i \leq L$. These RNS polynomials



INTT
INTT
BS
BS
NTT
NTT
q4
q2
q3
q1
Time
(1) Rescale pipeline with L=4, P^{intra} = 2.
(2) Rescale pipeline with L=4, P^{intra}= 4.

Fig. 4. Pipeline of rescale for the ciphertext with $L = 4$, where $P^{intra} = 2, 4$. "q1, q2, q3, q4" denotes $poly_{q_i}$, where $1 \leq i \leq 4$.

usually perform independent basis operations, so they can be processed in parallel.

Fig. 4 shows the effect of intra-operation parallelism on the pipeline interval of Rescale when $L = 4$. When $P^{intra} = 2$, we have 2 separated copies for the basic operation modules such as INTT, BS, and NTT, where two RNS polynomials computation can be processed in parallel. Therefore, the pipeline interval is twice the latency for the most time-consuming basic operation. In Fig. 4(b), where $P^{intra} = 4$ (4 copies for each basic operation), the pipeline interval is reduced to half, while the resource usage is also doubled compared to $P^{intra} = 2$. There is also an option of $P^{intra} = 3$, which leads to insufficient utilization of the parallel copies of modules. It can be observed that $P^{intra} = 4$ achieves the best possible performance for a level 4 ciphertext. We will further investigate how to reuse the HE modules across various layers with different levels L in Section V-C. The above-mentioned impact of intra-operation parallelism also applied to other HE operations including KeySwitch.

For the various HE-CNN layers, its pipeline interval PI is equal to the longest latency of its HE operations. We can derive the formula for PI as follows, where LAT_b indicates the maximum latency of the basic modules.

$$PI = \lceil \frac{L}{P^{intra}} \rceil \cdot LAT_b \tag{3}$$

Basic operation modules: To improve pipeline efficiency, the parallelism of each basic module can be adjusted internally, in order for each basic module to have a similar latency and improve the HE-module level pipeline efficiency.

There are two types of basic modules. The first is NTT/INTT, which is similar to FFT and requires $log_2 N$ rounds computation for N numbers. We use the HEAX [21] implementation and set multiple NTT cores to compute in parallel. The number of the NTT cores is denoted as nc_{NTT}, and the latency of the NTT module is as follows:

$$LAT_{NTT} = \log_2 N \cdot \frac{N}{2nc_{NTT}} \tag{4}$$

For the N coefficients of the RNS polynomial, the other type includes modular addition, modular multiplication, Barrett Reduction, or combination of these operations. Since the N numbers are computed independently, multiple parallel computation units can be set up, and latency is expressed as

$$LAT_{basic} = \frac{N}{p} \tag{5}$$

Thus, the latency formula of the basic module in pipeline is as follows:

$$LAT_b = \max\{LAT_{basic}, LAT_{NTT}\} \tag{6}$$

DSP usage: According to the parallelism of HE operations and parallelism within homomorphic operations. We can derive the DSP usage for each HE operation as the following equation:

$$DSP_{op} = P^{inter} \cdot P^{intra} \cdot Const_{op}^{DSP} \tag{7}$$

where $Const_{op}^{DSP}$ denotes the DSP usage when there is no parallelism in HE operations, and they are the constants.

C. Inter-layer HE module reuse

The level (which means the number of RNS polynomials) of the ciphertexts decreases when Rescale operation is performed after the multiplication of each neural network layer. Considering this, a straightforward approach is to set up separate HE operation modules for ciphertexts at different levels, but it leads to low utilization of the computational resources. We propose a scheme for HE modules reuse, which reuses the basic module instances of the HE operation to handle different levels of ciphertext. The basic modules of RNS polynomials that have no computational dependencies with each other can be reused, such as INTT in Rescale. Furthermore, when reusing an HE operation module across different layers with different levels L, the intra-parallelism for the HE operation module remains the same. As a result, it potentially results in a slight under-utilization of resources. Therefore, the intra-parallelism of each HE module must be carefully selected with the proposed automatic DSE.

For example, when there are 4 copies of parallel INTT in a Rescale module, it is capable to compute the results of a level-4 ciphertext in one go. For the Rescale operation with level-3 ciphertext at a different HE-CNN layer, it may reuse the level-4 Rescale module, which provides the same latency at the cost of keeping one of the 4 parallel INTT cores idle. Furthermore, two concurrent Rescale operations with level-2 ciphertext can be executed in parallel with reusing the level-4 Rescale module.

VI. Buffer management and DSE

A. On-chip Buffer Management

We propose an on-chip memory reuse strategy to mitigate the performance degradation due to off-chip memory transactions for memory-intensive FHE computation. First, we specify the reuse unit of the buffer. The proposed buffer reuse strategy reuse data in the granularity of the buffer of RNS polynomials, as the pipeline design is based on basic operation modules with RNS polynomials.

Buffer for HE operations: Before introducing the intra-layer reuse method, we need to figure out the on-chip buffer required

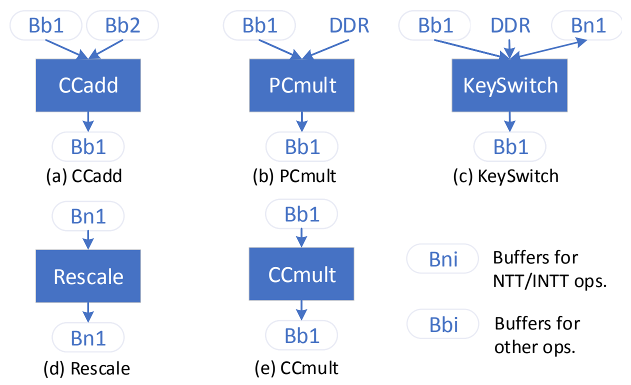


Fig. 5. Buffer reuse strategies for HE operations.

for each HE operation. Based on the characteristics of data access patterns and tiling factors of the basic HE operations, we divide the reusable on-chip BRAM buffers into two types, where "Bn" buffers are dedicated for NTT/INTT operations, and "Bb" buffers for all other basic HE operations. Our ultimate goal is to reuse the same set of buffers to serve basic HE operations as much as possible to reduce overall BRAM usage. For instance, as illustrated in Fig. 5, buffer Bb1 is utilized for both the input and output of CCadd, while buffer Bb2 refers to another buffer with the same buffer type. Similarly, Rescale and CCmult utilize the same buffer for input and output as in Fig. 5. CCmult does the square operation in HE-CNN, so no additional buffer is needed. The plaintext of PCmult is more suitable to be placed to off-chip memory. Because the plaintext is obtained by packing and encoding the weight of the CNN, and it is read only once in the implementation. The KeySwitch requires additional buffers to store intermediate data. In addition, the KeySwitch keys, which are read-only and in large data volume, are also stored in off-chip memory. What's more, the burst mode is used for reading data from off-chip, which does not increase latency during the pipeline.

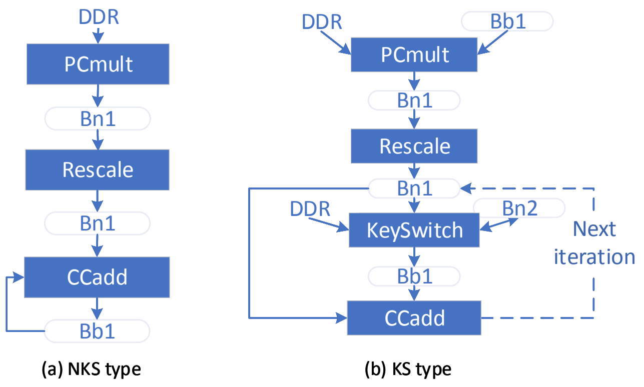


Fig. 6. Buffer reuse strategies for the different type of layer of CNN.

Intra-layer reuse: We present the intra-layer buffer reuse graphs for the two types of the HE-CNN layer in Sec. V-A. We summarize the reuse strategy in two points shown in Fig. 6. First, the output and input of two adjacent HE operations use the same buffers to transmit data. Considering that the output buffers and input buffers were different types originally. We set a buffer with a large tilling factor to reuse for one with a small tilling factor, which does not degrade the effect of basic operation parallelism. For example, in the NKS layer, the output of PCmult is stored in buffers of type bn as the input for the next Rescale operation. Second, buffers of the same type can be reused for non-adjacent operations. For example, buffer Bb1 in the KS layer is used for the input of PCmult and the output of KeySwitch respectively. Reusing buffers increases the utilization of BRAM, saving the total buffer usage of the layer, and avoiding performance loss.

On the other hand, the size of the buffer for each HE operation also depends on the intra and inter parallelism of the layer. In order to derive the buffer usage of each layer more clearly and avoid involving more details of HE operations, we use some constants to represent values that are not relevant to the design, which are denoted as $Const_t^B$. The buffer of each layer lr is represented by the following formula:

$$BRAM_{lr} = Bn_{lr} + Bb_{lr} \tag{8}$$

where $lr \in \{\text{NKS, KS}\}$. Then we have the formulas:

$$\begin{aligned} Bn_{NKS} &= (Const_{NKS}^{Bn} \cdot P_{NKS}^{intra} \cdot P_{NKS}^{inter}) \cdot Bn \\ Bb_{NKS} &= (Const_{NKS}^{Bb} \cdot P_{NKS}^{inter}) \cdot Bb \end{aligned} \tag{9}$$

$$\begin{aligned} Bn_{KS} &= ((Const_{KS}^{Bn} \cdot P_{KS}^{intra} + Const_{KS}^{Bn}) \cdot P_{KS}^{inter}) \cdot Bn \\ Bb_{KS} &= (Const_{KS}^{Bb} \cdot P_{KS}^{inter}) \cdot Bb \end{aligned} \tag{10}$$

where P_{lr}^{intra} and P_{lr}^{inter} denotes the degree of parallelism in Sec.V-A. Bn denotes the buffer of NTT/INTT, and Bb is the buffer of other basic operations.

Inter-layer reuse: To implement inter-layer reuse, it should be clear that the layer can be calculated after the previous layer has been completed for HE-CNN. Thus the buffers can be reused between layers. In order to map the buffers to each layer, the same type of buffers can be reused. Since the amount of buffers of each type is always different at each layer, we take its maximum usage as the total BRAM consumption.

URAM Utilization Conversion: For some COTS FPGA boards, on-chip memory resources include Ultra-RAM (URAM), which has a higher capacity compared to BRAM. However, the URAM block has a larger block size but the same read/write bandwidth as BRAM. Since the buffer unit in our scheme is partitioned for parallel accesses, which results in a relatively low utilization for URAM. Nevertheless, URAM is still non-negligible on-chip memory to alleviate the BRAM deficiency. We can get the ratio from URAM to BRAM based on the amounts of numbers in each tile of the buffer, denoted as num. URAM and BRAM each have 4K and 1K addresses. When num is smaller than 1K, the ratio is 1, while num is bigger than 4K, and the ratio is 4. When num is between 1K and 4K, the ratio is $\frac{num}{1K}$.

B. Design Space Exploration

Given the HE-CNN structure and performance modeling for the HE operation modules and resource management, we perform design space exploration to obtain the optimal design solution. The design solution consists of the value for the

TABLE VI
INFORMATION ABOUT THE BENCHMARK HE-CNN NETWORKS.

Networks	Layers	HOPs (10^3)	Acc (%)	Mod.Size (MB)
FxHENN-MNIST	Cnv1, Act1, Fc1, Act2, Fc2	0.83	98.9	15.57
FxHENN-CIFAR10	Cnv1, Act1, Cnv2, Act2, Fc2	82.73	74.1	2471.25

decision variables including intra- and inter-layer parallelism and on-chip buffer partitions, which can be used as the HLS pragmas and directives to generate the final accelerator circuit.

The objective function for the DSE is to minimize the aggregated latency for all HE-CNN layers, subject to the given FPGA's bottleneck resource capacity (DSP_{max} and $BRAM_{max}$), i.e.,

$$\begin{aligned}\text{Minimize} \quad & \sum_{\forall lr \in Layer} LAT_{lr} \\ \text{s.t.} \quad & \sum_{\forall op \in OP} DSP_{op} \leq DSP_{max} \\ & \max\{BRAM_{lr} | \forall lr \in Layer\} \leq BRAM_{max}\end{aligned} \tag{11}$$

where OP is the set of all HE operation modules, and $Layer$ is the set of all layers in the given HE-CNN.

Given the non-linear modeling of the problem, we currently solve the DSE by an exhaustive search of the entire design space. Due to the complexity and high resource usage of HE-CNN operations (e.g., KeySwitch as shown in Table I), the design space consisting of a few thousand design points that can be solved within a few seconds (negligible compared with the FPGA synthesis which takes up to a few hours).

VII. EXPERIMENTAL EVALUATION

A. Experimental setup

In order to generate an accelerator for a given HE-CNN inference task on a particular FPGA device, the proposed FxHENN framework automatically performs design space exploration to determine the deployment, memory allocation, and parallelism of the HLS-based parameterized HE operations modules. In our evaluation, FxHENN is integrated with the Xilinx Vivado toolchain v2019.1 to generate the FPGA bitstream files. In this section, we present the detailed experimental setup in the evaluation.

Benchmarks. A dozen of HE-CNN models have been proposed in the literature (refer to Table VII). These models may use different HE schemes, encryption parameters, data packing schemes, or even underlying HE operations. Among these models that have been open-sourced, LoLa [5] achieves the lowest inference latency per image frame (instead of throughput) with ingenious input/weight packing schemes. In this work that targets HE-CNN inference, we use two HE-CNN models and data packing schemes proposed in LoLa [5], i.e., FxHENN-MNIST and FxHENN-CIFAR10.

In this work, we use HEAX [21] and coxHE [16] for reference to implement the basic HE operations on FPGA. As a result, our two benchmark HE-CNN models are implemented with the recently proposed CKKS-RNS FHE scheme, the same as the above-mentioned references.

Table VI shows the per-layer information of the two HE-CNN models. Although both networks have the same multiplication depth of 5, CIFAR10 has a larger input image size, which results in 2 orders of magnitude in the number of HE operations (HOPs) compared with the FxHENN-MNIST with MNIST data set. This also poses a challenge for full-fledged deployment of the entire network on an FPGA device, especially with limited on-chip memory.

HE Parameters selection. The HE parameter selection influences the ciphertext and plaintext data sizes as well as the computational complexity. We choose level $L = 7$ to support the multiplication depth of the two 5-layer networks. We choose $N = 8192$ and 30 bit-width q_i for FxHENN-MNIST; $N = 16384$ and 36 bit-width q_i for FxHENN-CIFAR10 based on the parameter selection in [5]. The parameter selection yields 210 and 252 bit-width for Q in the two networks, which satisfies the security requirements of 128 and 192 bits, respectively [1], [8].

Platforms. We evaluate FxHENN on two low-power MPSoC FPGA platforms to demonstrate the potential of performing HE-CNN inference tasks for edge/embedded computing. In particular, ALINX ACU9EG (with Xilinx Zynq UltraScale+ MPSoC XCZU9EG device) is a mid-end embedded FPGA platform with 2,520 DSP slice and 32.1Mbit on-chip BRAM. While ALINX ACU15EG (with Xilinx Zynq UltraScale+ MPSoC XCZU15EG device) is a high-end embedded FPGA platform with 3,528 DSP slices, 26.2 Mbit on-chip BRAM and 31.5 Mbit on-chip URAM. We show that the proposed HLS-based framework automatically generates dedicated designs on different platforms for a given HE-CNN model, which fully utilizes the available hardware resources.

B. Performance Comparison

To the best of the authors' knowledge, FxHENN is the first full-fledged FPGA accelerator design framework for HE-CNN inference. To evaluate the performance of the accelerators generated by FxHENN, we compare the end-to-end inference latency w.r.t. the publicly reported data in the literature work.

Table VII summarizes the state-of-the-art works on HE-CNN inference with HE-CNN model, dataset, FHE schemes and parameters, hardware platform (with thermal design power, TDP), and inference latency, where any missing information is marked with "-". Note that we consider only end-to-end HE-CNN inference solutions, without intermediate communication, decrypt, and re-encrypt processes required as in an MPC-based HE-CNN inference.

Among the state-of-the-art solutions listed in Table VII, CryptoNets [15] was the first HE-enabled neural network on CPU proposed by Microsoft, where the initial design requires 205s to complete the inference of a single image. EVA [11] and Intel's nGraph-HE [4] utilized multiple compiler optimization techniques to reduce inference latency. LoLa [5] further optimized input packing strategies to reduce the number of homomorphic encryption operations. Falcon [18] made the HE

TABLE VII
PERFORMANCE OF HE-CNN INFERENCE ON MNIST AND CIFAR10.

	MNIST						Cifar						Platform		Scheme
	HOP	KS	λ	N	Q	Lat. (s)	HOP	KS	λ	N	Q	Lat. (s)	Architecture	TDP (W)	
CryptoNets [15]	215K	945	-	-	-	205	-	-	-	-	-	-	Intel Xeon E5-1620L	140	BFV
nGraph-HE [4]	-	-	128	13	210	16.7	-	-	192	14	300	1324	Xeon Platinum 8180 with 112 CPUs	205	CKKS
EVA [11]	10K	2K	128	14	480	121.5	150K	16K	128	16	1225	3062	4 socket machine with Intel Xeon Gold 5120	4*105	CKKS
LoLa [5]	798	227	128	14	440	2.2	123K	61K	128	14	440	730	Azure standard B8ms virtual machine with 8 vCPUs	8*110	BFV
Falcon [18]	626	122	128	14	440	1.2	21K	7.9K	128	14	440	107	Azure standard B8ms virtual machine with 8 vCPUs	8*110	BFV
AHEC [7]	215K	945	128	13	-	29.17	-	-	-	-	-	-	Xeon Platinum 8180 with 112 CPUs	250	CKKS
A*FV [2]	47K	0	82	13	330	5.2	7M	0	91	13	300	553.89	3*P100 and 1*V100 Nvidia GPUs	4*250	BFV
FxHENN	826	280	128	13	210	0.19	82K	57K	192	14	252	54.1	ALINX ACU15EG	10	CKKS
FxHENN	826	280	128	13	210	0.24	82K	57K	192	14	252	254	ALINX ACU9EG	10	CKKS

inference on frequency domain and achieve 1.2s inference latency on Azure standard B8ms virtual machine with 8 vCPUs. A*FV [2] utilized the massively parallel GPU architecture to accelerate HE-based CNN inference.

The emerging research on HE-enabled machine learning focuses on various aspects including the FHE scheme, network structure, data organization, etc. However, there is a lack of well-recognized benchmark HE-CNN models for the purpose of performance evaluation and accelerator design. As we can observe from Table VII, various literature studies use different HE-CNN models, datasets, FHE schemes, encryption parameters, and hardware platforms. Therefore, it is generally difficult to provide a completely fair performance comparison w.r.t. the existing CPU/GPU-based implementations of HE-CNN inference.

As we have discussed in Section VII-A, the HE-CNN models in this work follow the same input/weight packing schemes as proposed by LoLa [5]. Although we adopt the CKKS-RNS FHE scheme for the FPGA implementation of the basic HE operations, it can be observed from Table VII that FxHENN-MNIST and FxHENN-CIFAR10 have a similar computation workload as LoLa-MNIST and LoLa-CIFAR10, in terms of total HOP count and the bottleneck KeySwitch operation count (the “HOP” and “KS” columns in Table VII). Therefore, we compare the performance of the FxHENN-MNIST and FxHENN-CIFAR10 accelerators w.r.t. the CPU-based performance of LoLa [5]. Moreover, as our accelerator design framework is built on top of the parameterized HLS-based CKKS HE operations, it can be used to generate FPGA accelerators for other HE-CNN models (with different network structures, encryption parameters, data packing schemes, etc) without loss of generality.

Compared with the best-available CPU-based HE-CNN inference solution LoLa [5], the generated FPGA accelerator FxHENN-MNIST achieves inference speedup of 9.17X and 11.58X on ACU9EG and ACU15EG, respectively. Meanwhile, since LoLa [5] is executed on an Azure standard B8ms virtual machine with 8 vCPUs(with TDP of 8$\times$110W), On ACU9EG and ACU15EG, FxHENN-MNIST achieves up to 806.96X and 1019.04X energy efficiency w.r.t. CPU-based LoLa-MNIST, respectively. As for the FxHENN-CIFAR10 with 100X higher computation workload, compared with LoLa-CIFAR10 on CPU, we achieve a speedup of 2.87X and 13.49X, and energy efficiency of 252.56X and 1187.12X on ACU9EG and ACU15EG, respectively. Alternatively, compared with A*FV [2], which is the best-known GPU implementation of HE-CNN inference with 3 P100 plus 1 V100 GPU device, our accelerator running on ACU15EG achieves 27.37X speedup and 3000X energy efficiency for MNIST, and 10.26X speedup and 563X energy efficiency for CIFAR10, respectively.

TABLE VIII
PERFORMANCE COMPARISON OF CONVOLUTIONAL LAYERS WITH [28].

	Layer	N	q_i	DSP	Lat. (ms)	Speedup
FPL'21 [28]	conv1	2048	54	3584	26.32	-
	conv2_3	2048	54		12.03	-
FxHENN	conv1	2048	54	3072	19.95	1.32X
	conv2_3	2048	54		10.87	1.11X

To the best of the authors' knowledge, [28] proposes the state-of-the-art FPGA acceleration of BFV-based HE-CNN. However, it only considers the acceleration for a single individual convolutional layer. As a result, [28] only involves simple HE operations including PCmult and CCadd, while the most time and resource-consuming Rotate operation (KeySwitch module) is not implemented. Moreover, inter-layer resource utilization for a full-fledged CNN model inference (i.e., one of the key contributions in this work) is not considered in [28]. While DSP is the primary FPGA resource for PCmult and CCadd operations, Table VIII shows the proposed FxHENN achieves 1.11-1.32X performance speedup w.r.t. [28] for different convolutional layers of ResNet-50 while utilizing even fewer DSP resources.

C. FxHENN Optimizations

Due to the absence of other FPGA-based accelerators for end-to-end HE-CNN inference, we implement a “baseline” ac-

TABLE IX
PERFORMANCE AND RESOURCES OF BASELINE AND FXHENN ON FXHENN-MNIST.

	Peak utilization (%)		Aggregated utilization(%)		Latency
	DSP	BRAM	DSP	BRAM	(s)
Baseline	67.78	81.25	67.78	81.25	1.17
FxHENN	63.25	81.36	136.25	170.67	0.24

celerator on the ALINX ACU9EG platform to demonstrate the effectiveness of the optimization techniques proposed in the FxHENN framework. In particular, the "baseline" implementation does NOT attempt to reuse any computation or on-chip storage resources across different HE-CNN layers. Instead, given the network model and available FPGA resources, we perform an intuitive resource allocation so that more resources are assigned to the heavily burdened CNN layers.

Table IX shows the peak and aggregated DSP and BRAM usage (the two most constrained resources for HE-CNN inference) for the baseline approach and FxHENN, respectively. It indicates that the baseline approach is designed without any cross-layer resource reuse, therefore shows the same percentage values for total and aggregated utilization. On the other hand, FxHENN achieves 136.25% and 170.67% aggregated utilization for DSP and BRAM, respectively. The result indicates that both DSP (for computation) and BRAM blocks (for on-chip storage) are effectively reused across different HE-CNN layers, which leads to substantial improvement on the inference latency. Note that it is generally impossible for a complicated application kernel to use 100% of a particular FPGA resource (e.g., BRAM blocks) due to the restrictions in the HLS design flow and the FPGA placement and routing process.

We further conduct a fine-grained investigation on the FPGA resource reuse and its effectiveness on the HE-CNN inference latency. Fig. 7 presents the per-layer breakdowns of BRAM usage and latency of the baseline and FxHENN schemes. Similar to what we have discussed in Section III, high BRAM utilization has a significant impact on the HE-CNN inference latency. In particular, for the most time-consuming layer, "Fc1" which consists of a large number of heavy KeySwitch HE-operations, the heuristic allocation scheme without inter-layer BRAM sharing assigns 25.8% BRAM blocks (highest among all 5 layers). Instead, the proposed FxHENN allows up to 84.8% BRAM usage for the "Fc1" layer with BRAM sharing among all layers. As a result, the generated accelerator by the proposed FxHENN achieves 0.16s inference latency for the "Fc1" layer, which leads to 6.63X speedup compared with the baseline approach.

Fig. 7 also shows that even with the proposed inter-layer resource reuse scheme, the per-layer BRAM utilization ratio is divergent. The automatic design space exploration prefers to assign more on-chip BRAM blocks to HE-operations in the bottleneck layer "Fc1" to achieve high resource efficiency. Meanwhile, HE operations in other layers reuse the BRAM blocks with the best-effort approach, subject to the characteristics of HE operations, the organization of the BRAM buffers, and the HLS coding restrictions. Particularly in the

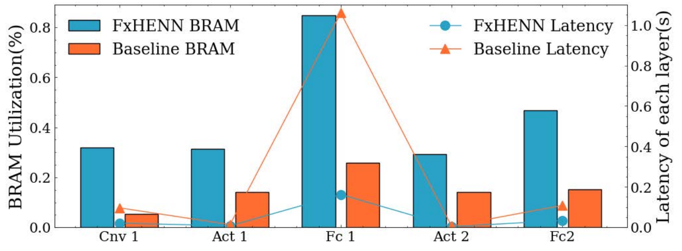


Fig. 7. BRAM usage and latency for each layer of FxHENN-MNIST.

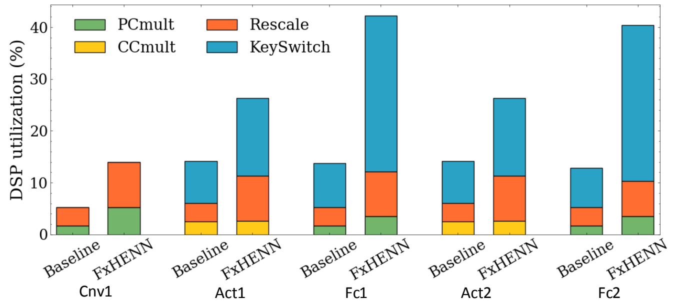


Fig. 8. Comparison of DSP usage of each HE operation for each layer of the MNIST network.

example of FxHENN-MNIST, "Act1" and "Act2" layers have low computation burdens and require fewer BRAM resources. On the other hand, some buffers required by the "Cnv1" and "Fc2" layers do not exactly match the buffers in the "Fc1" layer in terms of size and HLS partition factors.

The use of on-chip BRAM blocks increases monotonically with the parallel level of computation (hence the computation resource utilization). Fig. 8 shows the per-layer usage of DSP slices for each HE operation type in the FxHENN-MNIST accelerators generated by the baseline and FxHENN approaches. FxHENN performs module-level reuse optimization, which tries to reuse a module instance of an HE operation as much as possible across different layers. As a result, we observe higher DSP utilization in all layers w.r.t. the baseline approach without module reuse. In particular, FxHENN automatically identifies the reuse opportunity and deploys two parallel KeySwitch modules that are used by both "Fc1" and "Fc2". Meanwhile, "Act1" and "Act2" invoke the KeySwitch operation only once, so only one of the two KeySwitch modules is used by each of the two layers. On the contrary, without module-level reuse, the baseline approach deploys four separated KeySwitch modules (with lower intra-operation parallelism and higher latency), each invoked by a different layer. Note that as we have discussed in Section V, the design of a reusable HE module needs to consider various factors including HE encryption parameters, network structures (operation frequencies), as well as resource usage, in order to achieve high resource efficiency and overall performance.

D. Design Space Exploration

In this section, we evaluate the effectiveness of design space exploration (DSE) in the proposed FxHENN framework. The HLS-based FxHENN framework enables automatic DSE to generate the optimal accelerator for a given HE-CNN model,

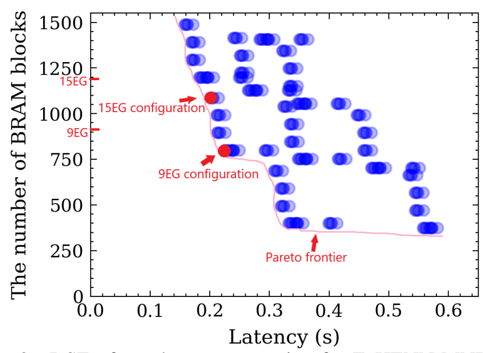
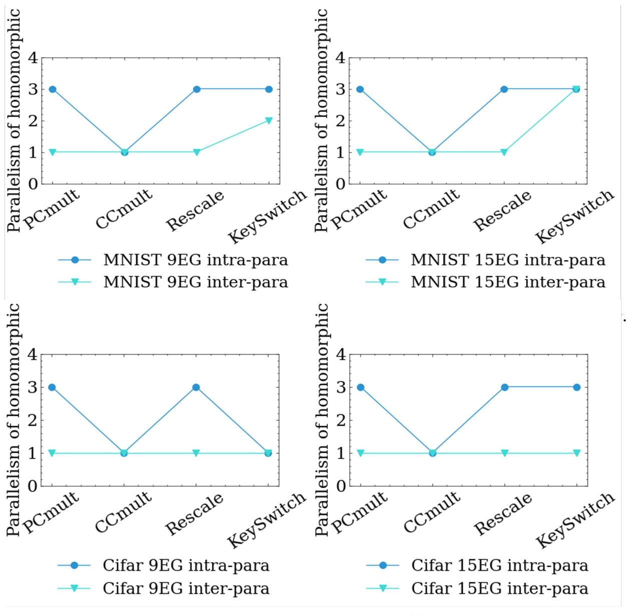


Fig. 9. DSE of accelerator generation for FxHENN-MNIST.



(c) Cifar on ALINX ACU9EG. (d) Cifar on ALINX ACU15EG.

Fig. 10. The intra-/inter-parallelism of HE operation modules for FxHENN-MNIST and FxHENN-CIFAR10 on ACU9EG and ACU15EG.

subject to the resource restrictions of the target FPGA device. As we have discussed in previous sections, on-chip BRAM blocks are the most restricted FPGA resources for HE-CNN inference due to the large volume of data with non-burst access patterns. Fig. 9 shows the possible acceleration design solutions for FxHENN-MNIST HE-CNN inference when the number of BRAM blocks is limited in the range between 350 to 1500. The corresponding inference latency for each design solution and the Pareto frontier with all non-dominated solutions is shown in the Fig. 9 . As we can verify from the result that the automatically generated solutions by the proposed FxHENN on the two target FPGA devices ACU9EG and ACU15EG dominate other solutions under the same BRAM block constraints (URAM are converted to equivalent BRAM blocks according to Sec. VI-A).

In the proposed HLS-based framework for the automatic generation of the HE-CNN accelerator, each solution can be expressed with the intra- and inter-parallelism of the HE operations (and corresponding I/O buffer connections). In Fig. 9, it can be observed that with a low BRAM budget, there are a few possible design solutions, since both intra- and inter-parallelism need to keep at a very low level in order to satisfy the BRAM restriction. On the other hand, more possible design solutions are available when the BRAM budget increases, and an automatic DSE framework becomes imperative to achieve high resource efficiency and overall performance.

In order to further elaborate the effectiveness of the proposed DSE framework, Fig. 10 presents the generated optimal design solutions with intra- and inter-parallelism of HE operation modules for FxHENN-MNIST and FxHENN-CIFAR10 on ACU9EG and ACU15EG, respectively. From the results, it can be observed that the FxHENN framework produces distinct accelerator designs for different HE-CNN models and target FPGA devices. For example, comparing Fig. 10(a) and (c) with the same FPGA device ACU9EG shows that higher intra- and inter-parallelism can be achieved for the KeySwitch operation of FxHENN-MNIST with the encryption parameter $N = 2^{13}$. On the other hand, only the minimum intra- and inter-parallelism (i.e., 1) can be achieved for the KeySwitch operation of FxHENN-CIFAR10 on the same FPGA device, due to $N = 2^{14}$ and therefore a double-sized BRAM buffer is required by the operation. On the other hand, when FxHENN-CIFAR10 is deployed on the ACU15EG with larger BRAM/URAM capacity, The FxHENN framework automatically explores the design space and determines the intra-parallelism of KeySwitch to be 3 for optimal overall performance. Meanwhile, for all design solutions shown in Fig. 10, the parallelism of the CCmult operation is set to be only 1 for high resource efficiency, due to the extremely low frequency of CCmult operations in our context of HE-CNN inference with ciphertext-input and plaintext-weight.

VIII. Conclusion

He-CNN is a promising solution to protect data privacy, but the enormous latency overhead becomes a major obstacle for practical usage in real-world applications. In this work, we present the first FPGA-based acceleration framework, FxHENN, to enable a fast and efficient HE-CNN inference. To reduce the on-chip memory usage, we proposed a memory management strategy and significantly improved the performance of HE-enabled networks on FPGA. We also provide a framework that models the performance and resource usage of HE operations, which enables effective design space exploration to generate optimal accelerators for various HE-CNN models on target FPGA boards.

IX. Acknowledgments

We sincerely thank the anonymous reviewers for their insightful and valuable feedback. This work was supported by the Natural Science Foundation of China grant number 92064008, Taishan Scholars Program, and Qilu Young Scholar Program of Shandong University.

References

[1] M. R. Albrecht, "On dual lattice attacks against small-secret LWE and parameter choices in helib and SEAL," in *Advances in Cryptology - EUROCRYPT 2017 - 36th Annual International Conference on the*

Theory and Applications of Cryptographic Techniques, Paris, France, April 30 - May 4, 2017, Proceedings, Part II, ser. Lecture Notes in Computer Science, J. Coron and J. B. Nielsen, Eds., vol. 10211, 2017, pp. 103–129.

[2] A. A. Badawi, C. Jin, J. Lin, C. F. Mun, S. J. Jie, B. H. M. Tan, X. Nan, K. M. M. Aung, and V. R. Chandrasekhar, "Towards the alexnet moment for homomorphic encryption: Hcnn, the first homomorphic CNN on encrypted data with gpus," *IEEE Trans. Emerg. Top. Comput.*, vol. 9, no. 3, pp. 1330–1343, 2021.

[3] 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, WAHC@CCS 2019, London, UK, November 11-15, 2019*, M. Brenner, T. Lepoint, and K. Rohloff, Eds. ACM, 2019, pp. 45–56.

[4] 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, CF 2019, Alghero, Italy, April 30 - May 2, 2019*, F. Palumbo, M. Becchi, M. Schulz, and K. Sato, Eds. ACM, 2019, pp. 3–13.

[5] A. Brutzkus, O. Elisha, and R. Gilad-Bachrach, "Low latency privacy preserving inference," in *ICML , USA*, K. Chaudhuri and R. Salakhutdinov, Eds., vol. 97. PMLR, 2019, pp. 812–821.

[6] H. Chen, R. Cammarota, F. Valencia, and F. Regazzoni, "Plaidml-he: Acceleration of deep learning kernels to compute on encrypted data," in *37th IEEE International Conference on Computer Design, ICCD 2019, Abu Dhabi, United Arab Emirates, November 17-20, 2019*. IEEE, 2019, pp. 333–336.

[7] H. Chen, R. Cammarota, F. Valencia, F. Regazzoni, and F. Koushanfar, "AHEC: end-to-end compiler framework for privacy-preserving machine learning acceleration," in *57th ACM/IEEE Design Automation Conference, DAC 2020, San Francisco, CA, USA, July 20-24, 2020*. IEEE, 2020, pp. 1–6.

[8] 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*, ser. Lecture Notes in Computer Science, C. Cid and M. J. J. Jr., Eds., vol. 11349. Springer, 2018, pp. 347–368.

[9] J. H. Cheon, A. Kim, M. Kim, and Y. S. Song, "Homomorphic encryption for arithmetic of approximate numbers." in *ASIACRYPT (1)*, ser. Lecture Notes in Computer Science, T. Takagi and T. Peyrin, Eds., vol. 10624. Springer, 2017, pp. 409–437. [Online]. Available: http://dblp.uni-trier.de/db/conf/asiacrypt/asiacrypt2017-1.html#CheonKKS17

[10] E. Chou, J. Beal, D. Levy, S. Yeung, A. Haque, and L. Fei-Fei, "Faster cryptonets: Leveraging sparsity for real-world encrypted inference," *CoRR*, vol. abs/1811.09953, 2018.

[11] 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," *CoRR*, vol. abs/1912.11951, 2019.

[12] R. Dathathri, O. Saarikivi, H. Chen, K. Laine, K. E. Lauter, S. Maleki, M. Musuvathi, and T. Mytkowicz, "CHET: an optimizing compiler for fully-homomorphic neural-network inferencing," in *Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI 2019, Phoenix, AZ, USA, June 22-26, 2019*, K. S. McKinley and K. Fisher, Eds. ACM, 2019, pp. 142–156.

[13] L. Deng, G. Li, S. Han, L. Shi, and Y. Xie, "Model compression and hardware acceleration for neural networks: A comprehensive survey," *Proc. IEEE*, vol. 108, no. 4, pp. 485–532, 2020.

[14] C. Gentry, "Fully homomorphic encryption using ideal lattices," in *Proceedings of the 41st Annual ACM Symposium on Theory of Computing, STOC 2009, Bethesda, MD, USA, May 31 - June 2, 2009*, M. Mitzenmacher, Ed. ACM, 2009, pp. 169–178.

[15] R. Gilad-Bachrach, N. Dowlin, K. Laine, K. E. Lauter, M. Naehrig, and J. Wernsing, "Cryptonets: Applying neural networks to encrypted data with high throughput and accuracy," in *ICML, USA*, vol. 48. JMLR.org, 2016, pp. 201–210.

[16] M. Han, Y. Zhu, Q. Lou, Z. Zhou, S. Guo, and L. Ju, "coxhe: A software-hardware co-design framework for FPGA acceleration of homomorphic computation," in *2022 Design, Automation & Test in Europe Conference & Exhibition, DATE 2022, Antwerp, Belgium, March 14-23, 2022*, C. Bolchini, I. Verbauwhede, and I. Vatajelu, Eds. IEEE, 2022, pp. 1353–1358.

[17] C. Juvekar, V. Vaikuntanathan, and A. Chandrakasan, "Gazelle: A low latency framework for secure neural network inference," in *Proceedings of the 27th USENIX Conference on Security Symposium*, ser. SEC'18. USA: USENIX Association, 2018, p. 1651–1668.

[18] Q. Lou, W. Lu, C. Hong, and L. Jiang, "Falcon: Fast spectral inference on encrypted data," in *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, and H. Lin, Eds., 2020.

[19] S. Mittal, "A survey of fpga-based accelerators for convolutional neural networks," *Neural Comput. Appl.*, vol. 32, no. 4, pp. 1109–1139, 2020.

[20] B. Reagen, W. Choi, Y. Ko, V. T. Lee, H. S. Lee, G. Wei, and D. Brooks, "Cheetah: Optimizing and accelerating homomorphic encryption for private inference," in *IEEE International Symposium on High-Performance Computer Architecture, HPCA 2021, Seoul, South Korea, February 27 - March 3, 2021*. IEEE, 2021, pp. 26–39.

[21] M. S. Riazi, K. Laine, B. Pelton, and W. Dai, "HEAX: an architecture for computing on encrypted data," in *ASPLOS, Switzerland*, J. R. Larus, L. Ceze, and K. Strauss, Eds. ACM, 2020, pp. 1295–1309.

[22] M. Ribeiro, K. Grolinger, and M. A. Capretz, "Mlaas: Machine learning as a service," in *2015 IEEE 14th International Conference on Machine Learning and Applications (ICMLA)*, 2015, pp. 896–902.

[23] N. Samardzic, A. Feldmann, A. Krastev, S. Devadas, R. G. Dreslinski, C. Peikert, and D. Sánchez, "F1: A fast and programmable accelerator for fully homomorphic encryption," in *MICRO '21: 54th Annual IEEE/ACM International Symposium on Microarchitecture, Virtual Event, Greece, October 18-22, 2021*. ACM, 2021, pp. 238–252.

[24] N. Samardzic, A. Feldmann, A. Krastev, N. Manohar, N. Genise, S. Devadas, K. Eldefrawy, C. Peikert, and D. Sánchez, "Craterlake: a hardware accelerator for efficient unbounded computation on encrypted data," in *ISCA '22: The 49th Annual International Symposium on Computer Architecture, New York, New York, USA, June 18 - 22, 2022*, V. Salapura, M. Zahran, F. Chong, and L. Tang, Eds. ACM, 2022, pp. 173–187.

[25] M. van der Hagen and B. Lucia, "Client-optimized algorithms and acceleration for encrypted compute offloading," in *ASPLOS '22: 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Lausanne, Switzerland, 28 February 2022 - 4 March 2022*, B. Falsafi, M. Ferdman, S. Lu, and T. F. Wenisch, Eds. ACM, 2022, pp. 683–696.

[26] A. Viand, P. Jattke, and A. Hithnawi, "Sok: Fully homomorphic encryption compilers," in *42nd IEEE Symposium on Security and Privacy, SP 2021, San Francisco, CA, USA, 24-27 May 2021*. IEEE, 2021, pp. 1092–1108.

[27] G. Xin, Y. Zhao, and J. Han, "A multi-layer parallel hardware architecture for homomorphic computation in machine learning," in *ISCAS, South Korea*. IEEE, 2021, pp. 1–5.

[28] T. Ye, S. R. Kuppannagari, R. Kannan, and V. K. Prasanna, "Performance modeling and FPGA acceleration of homomorphic encrypted convolution," in *31st International Conference on Field-Programmable Logic and Applications, FPL 2021, Dresden, Germany, August 30 - Sept. 3, 2021*. IEEE, 2021, pp. 115–121.