

# FHEx: Transforming Generic Compute Chips into Secure FHE Engines via a Hardware-software Co-designed Framework

Yibo Du<sup>1,2</sup>, Ying Wang<sup>1</sup>✉, Mengdi Wang<sup>1</sup>, Cangyuan Li<sup>1</sup>, Lian Liu<sup>1,2</sup>, Hui Li<sup>3</sup>, Kai Zhang<sup>3</sup>, Yinhe Han<sup>1</sup>✉

<sup>1</sup> Research Center for Intelligent Computing Systems, State Key Lab of Processors, Institute of Computing Technology, Chinese Academy of Sciences, Beijing, China

<sup>2</sup> University of Chinese Academy of Sciences, Beijing, China

<sup>3</sup> Jinan Inspur Data Technology Co., Ltd.

{duyibo21s, wangying2009, wangmengdi, licangyuan, liulian21s, yinhes}@ict.ac.cn

**Abstract**—Fully Homomorphic Encryption (FHE) is a powerful privacy-preserving technology enabling secure computation on encrypted data, but it suffers from substantial performance overheads. Running FHE efficiently typically requires developing dedicated FHE accelerators, which can be costly and inflexible. Instead of pursuing entirely new accelerators, this paper explores an alternative paradigm: augmenting generic computing devices with a modular FHE-specific hardware extension unit (HEU) to create an efficient FHE engine. To make this paradigm viable, we propose three key innovations: (1) recognizing that some FHE operators are data-intensive and involve a massive volume of ciphertexts, we design the HEU with a 3D stacked memory-based architecture to handle data-intensive operators.

We also provide software-level support to facilitate deploying FHE tasks on this extension-based architecture. (2) To capitalize on the hardware parallelism, we propose an adaptive offloading algorithm that intelligently distributes FHE operators between the computing device and the HEU. (3) To optimize the data layout and minimize the inter-tile data communications in the novel 3D stack memory, we propose a dedicated ciphertext mapping mechanism. Experimental results demonstrate that our work achieves substantial acceleration in FHE tasks.

## I. INTRODUCTION

Fully Homomorphic Encryption (FHE) enables computation directly on encrypted data, offering strong security guarantees for privacy-preserving applications such as privacy-inference (PI). However, this powerful capability comes with prohibitive computational costs. For instance, performing inference over a 20-layer neural network using TFHE can be up to  $10^5$  slower than executing the same unencrypted model on a CPU [1]. Prior efforts to accelerate FHE often rely on building dedicated FHE accelerators [2]–[7]. These designs integrate massive amounts of customized computation and memory resources to offer the desired acceleration for FHE tasks. Despite delivering impressive speedups, this ASIC-based approach incurs significant development costs and suffers from inflexibility in adapting to the rapid evolution of FHE algorithms.

Instead of pursuing entirely new accelerators, we explore an alternative path: enhancing the existing computing device with a modular hardware extension. Existing computing devices in a system already provide powerful computational

resources, particularly through single-instruction multiple-data (SIMD) cores that are highly effective for accelerating the computation-intensive kernels of FHE. Rather than discarding these powerful resources and designing an accelerator from scratch, we propose to attach a specialized *hardware extension unit* (HEU) to an existing computing device to enhance its ability for FHE tasks. This modular HEU can be updated independently and avoids microarchitectural modification of the computing device.

However, realizing a HEU that effectively augments existing computing devices for FHE is non-trivial, as FHE involves massive ciphertexts, which could incur substantial data transfer between the HEU and the computing device. FHE tasks involve a considerably larger volume of ciphertexts, which grow by  $10^6 \sim 10^7$  times compared with their unencrypted equivalents. The ciphertext transfer penalties could potentially offset the benefits of extension-based approaches.

To make this paradigm viable, we conducted a detailed analysis of TFHE, which reveals that **some operators are computation-intensive, while others are data-intensive**, which are basically element-wise computations but involve enormous amounts of ciphertexts. Motivated by this, we design a 3D-stacked memory-based HEU that vertically bonds DRAM dies to the FHE-custom logic die. This 3D stack memory-based architecture provides high internal bandwidth and abundant parallelism, which is ideally suited to process the data-intensive element-wise operators and contains their massive data traffic entirely within the extension, therefore minimizing most ciphertext movement between the computing device and the HEU.

To fully unleash the potential of our extension-based architecture, two major challenges must be addressed. **Challenge 1:** An efficient operator offloading strategy plays a crucial role in the overall performance. While the 3D stack memory-based HEU is promising for data-intensive operators, the static offloading strategy that always offloads data-intensive operators to HEU and computation-intensive operators to the computing device enforces strict sequential execution due to data dependencies and causes sub-optimal performance. **Challenge 2:** Ciphertext mapping on the novel 3D-stack memory-

✉ Corresponding author

based HEU. 3D-stack memory has asymmetric memory access latency, where each logic tile can only directly access its vertically aligned memory tiles, and accessing data from other tiles requires inter-tile communication. An inefficient mapping will cause excessive inter-tile communication.

In this paper, we propose FHEx, a comprehensive hardware-software co-design framework that transforms existing computing devices into secure and efficient FHE engines, as shown in Figure 1. We propose a 3D stack memory-based HEU to enhance generic compute chips. In addition, we include FHEx with software support: First, we lower the FHE task to primitive operators and construct a computation graph. Then, we propose an adaptive offloading algorithm that intelligently distributes operators in the computation graph between the computing device and the HEU to optimize the overall execution (Challenge 1). Third, we propose a ciphertext mapping mechanism tailored for the 3D stack memory that determines the ciphertext layout for operators offloaded on the 3D stack memory (Challenge 2) to optimize the inter-tile ciphertext communication. Our contributions are as follows:

- We propose FHEx, a hardware-software co-design framework that transforms a generic compute chip into an efficient FHE engine by integrating a modular hardware extension unit.
- Based on the observation that some FHE operators are data-intensive and involve substantial ciphertext movement, we design a 3D-stacked memory-based hardware extension unit and propose a Parallelism-Aware Ciphertext Mapping mechanism tailored for the 3D-stack memory architecture to exploit its high parallelism.
- Beyond simply utilizing the high bandwidth of 3D stacked memory. We propose an adaptive offloading algorithm guided by an FHE-specific offloading model to achieve an efficient operator offloading to the computing device and the introduced HEU, minimizing the overall execution latency.
- Evaluation results show that FHEx achieves substantial performance improvements, delivering 3240.4 $\times$ , 191.3 $\times$ , 11.7 $\times$ , 3.8 $\times$ , and 3.0 $\times$  speedups over CPU, GPU, and state-of-the-art TFHE accelerators on TFHE tasks.

## II. BACKGROUND AND MOTIVATION

### A. Fully Homomorphic Encryption (FHE)

FHE is one of the most promising privacy-preserving techniques with superior security to counteract even quantum attacks [8], [9]. In this work, fast fully homomorphic encryption over the torus (TFHE) [10] is referenced because it has the most efficient *bootstrapping* (BSP) among FHE schemes [2]. *Bootstrapping* is a core mechanism in FHE, which can homomorphically refresh noises accumulated in the ciphertext during computations, enabling an unlimited number of computations on encrypted data [11]. In addition, TFHE supports programmable bootstrapping, which facilitates arbitrary univariate functions on ciphertexts [1]. This capability makes TFHE efficient in supporting a variety of operations,

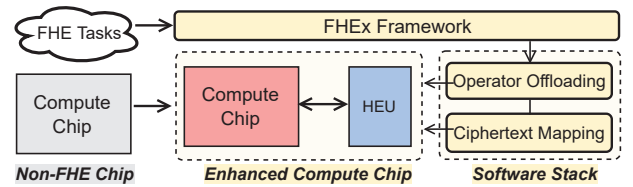


Fig. 1. The proposed extension-based design approach enhances a generic compute chip as an efficient FHE engine via a hardware extension unit (HEU).

especially the activation/non-linear function, and increasingly popular in privacy-preserving inference (PI) [1], [12], [13].

FHE ciphertexts are represented as high-degree polynomials, and the core FHE computations work on ciphertext polynomials. For instance, bootstrapping involves the external product (EP), a matrix-vector multiplication, where each element is an  $N$ -degree polynomial. To optimize the polynomial multiplication, Number Theoretic Transform (NTT) is adopted to reduce the complexity from  $O(N^2)$  to  $O(N \log N)$ . After applying NTT, the polynomial multiplication is transformed into *element-wise multiplication*. In addition, it also requires other element-wise operators such as polynomial scale (PS) that multiplies the polynomial by a scalar, and decomposition, which decomposes a polynomial into many smaller polynomials. **In summary, TFHE fundamentally involves two types of primitive operators: NTT/INTT and element-wise type operators such as polynomial scale (PS), external product (EP), and Decomposition (Decomp).**

### B. Extension-based approach vs. ASIC-based approach

Running FHE tasks efficiently on a non-FHE-tailored system typically requires developing dedicated FHE accelerators. Prior studies have explored dedicated architectures to accelerate TFHE [2]–[4], [14], [15]. These dedicated accelerators integrate abundant computing and memory resources to offer the desired acceleration. Although this ASIC-based approach demonstrates remarkable speedup, it suffers from long development cycles and inflexibility to adapt to the rapid evolution of FHE algorithms. These limitations raise a critical question: Can we achieve efficient FHE acceleration without relying on fully customized ASICs?

In this paper, we explore an alternative paradigm. As shown in Figure 1, we propose to augment a computing device with a modular, FHE-specific hardware extension to make it efficient for FHE tasks. We distribute the FHE workload across the introduced hardware extension and the computing device to utilize the powerful SIMD cores of the host device. This extension-based solution requires no microarchitectural modifications to the host device, and preserves flexibility to adapt to the algorithmic advancements, as the hardware extension unit can be updated independently.

### C. Why a 3D Stacked Memory-Based Extension?

To answer this, we conducted an operator profiling of TFHE, as shown in Figure 2. We measure the operation and memory breakdown of each FHE operator in the bootstrapping, which reveals that some operators are **computation-intensive** while some are **data-intensive**. For instance, EP

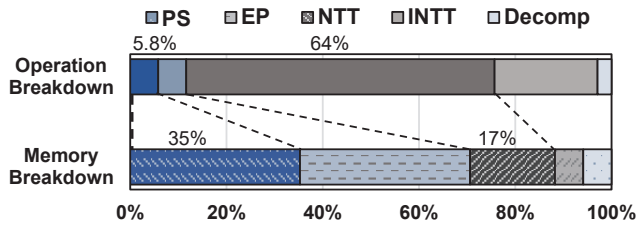


Fig. 2. Operation and Memory breakdown of TFHE bootstrapping, which includes polynomial scale (PS), external product (EP), NTT, INTT, and decomposition (Decomp) operators.

accounts for a minor portion of the arithmetic operations, only 5.8%, but imposes significant memory overhead, consuming approximately 35% of the total memory (The operation is measured by multiplications and memory is measured by bytes). In contrast, operators such as NTT/INTT are computation-intensive, account for a large amount of arithmetic computations, and only consume a small proportion of the memory. These computation-intensive operators are well-suited to the computing device with abundant computing resources, such as AMD MI50/MI100 [16]. However, data-intensive operators such as EP have a low arithmetic intensity, only 2 Ops/Byte. According to the Roofline model [17] on the NVIDIA A100 GPU [18], EP operators are severely bottlenecked by the memory bandwidth. This observation suggests that the data-intensive operators involve massive ciphertext volumes, which could incur significant ciphertext transfer overhead.

### III. FHEx FRAMEWORK

We propose a 3D stacked memory-based hardware extension, which, by accommodating these data-intensive operators, eliminates most of the ciphertext transfers with the computing device. 3D stacked DRAM is an industry-leading solution. By stacking multiple layers of LPDDR4/4X dies using hybrid bonding and mini-TSVs, it provides higher bandwidth.

#### A. Framework Overview

We propose FHEx, a framework that transforms a generic compute chip into an efficient FHE engine by integrating a *hardware extension unit (HEU)*. The proposed HEU consists of a custom logic die stacked with DRAM dies. In addition, FHEx also provides software-level support to facilitate deploying FHE tasks on this extension-based architecture. As shown in Figure 3, FHEx proposes an adaptive operator offloading algorithm to assign operators across the HEU and the computing device. Then, it provides a ciphertext mapping mechanism to manage the ciphertext mapping on the 3D stack memory architecture for operators offloaded on it.

**Sec.III-B** introduces the overall architecture. The following **Sec.III-C** presents the ciphertext mapping mechanism on HEU architecture. **Sec.III-D** introduces the adaptive operator offloading. **Sec.III-E** presents FHEx’s overall workflow.

#### B. Architecture Overview

1) *System Architecture*: FHEx enhances an existing, non-FHE-tailored computing device with a modular, FHE-specific hardware extension unit for efficient FHE computation. As

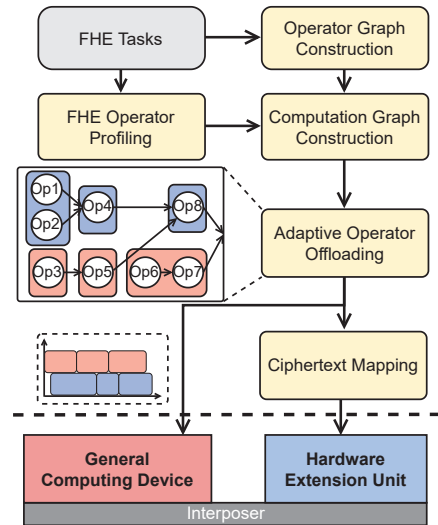


Fig. 3. The overview framework of FHEx.

shown in Figure 4(a), this approach leverages existing computing resources, such as the commonly used and easily accessible SIMD cores in the computing device. For example, AMD MI50/MI100 contains ample SIMD cores. Meanwhile, the hardware extension unit, HEU, is integrated via a silicon interposer using universal chiplet interconnect express with 385 GB/s [19]. This requires no microarchitectural modification to the baseline device. The feasibility of this integration model has been demonstrated in practice [20].

2) *Hardware Extension Unit Architecture*: HEU is based on the 3D-stack memory, as shown in Figure 4(b). It integrates a dedicated logic die beneath a 2-layer DRAM stack using hybrid bonding and mini-through-silicon vias (TSVs). Following the practical configuration in [21], [22], each DRAM die is divided into  $4 \times 4$  tiles, providing 135 Bandwidth per Gbit (GBps/Gbit). It has an overall memory capacity of 4 GB. The logic die is partitioned into a corresponding  $4 \times 4$  grid of logic tiles, each aligned with a DRAM tile. The memory controllers and PHYs of each DRAM tile are located at the corresponding position of the logic die. Each logic tile has direct access to its vertically aligned DRAM tile.

We add the processing unit in the logic die, which does not modify the microarchitecture of the DRAM dies and maintains compatibility with the DRAM die manufacturing. As shown in Figure 4(c), each logic tile contains a **ciphertext processing unit**. The ciphertext processing unit has five modules: (1) a vector multiplication unit, (2) an accumulator, (3) a ciphertext buffer, (4) a local router, and (5) a controller. It is designed to support the element-wise type operators in TFHE, which have a simple compute pattern but are data-intensive. During the preparation phase, ciphertexts are mapped to the memory tile before computation. Then, each logic tile independently loads ciphertexts from the vertically stacked DRAM tile. All logic tiles work in parallel. The results of each logic are stored in the corresponding DRAM tile. The logic tile can also access other DRAM tiles via the mesh network-on-chip in logic die.

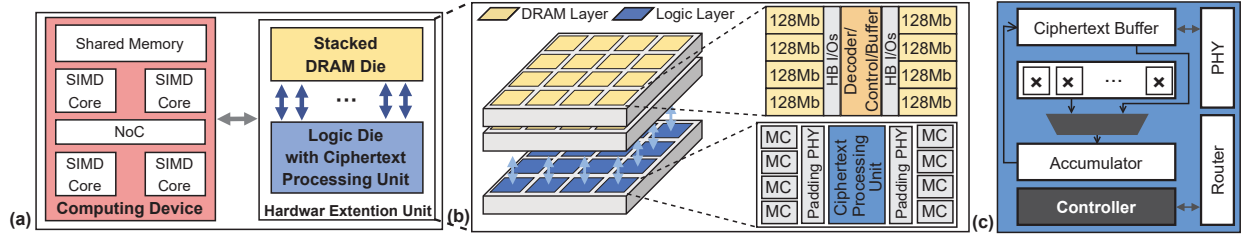


Fig. 4. (a) Augmenting the computing device with a hardware extension unit. (b) 3D stack memory-based hardware extension unit. (c) Ciphertext processing unit in a logic tile.

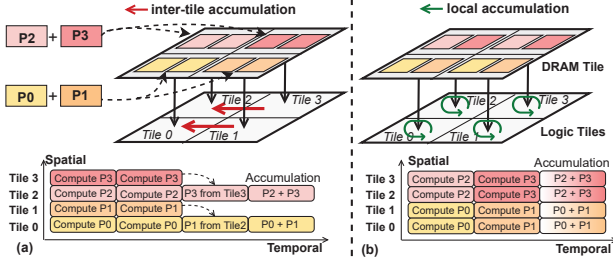


Fig. 5. P0 and P1 are accumulated, P2 and P3 are accumulated. (a) Example of mapping an entire polynomial in a memory tile. (b) The proposed Parallelism-Aware Ciphertext Mapping mechanism. P0~P3 represents polynomials.

### C. Ciphertext Mapping on the 3D stack memory-based HEU

This section presents how ciphertexts are mapped onto the novel 3D stack memory architecture. The 3D stacked memory architecture provides high parallelism but has asymmetric access latency: each logic tile can directly access its vertically aligned memory tile, while accessing data from other tiles must rely on the on-chip network (NoC) on the logic die, which incurs significant latency and traffic overhead. Usually, each ciphertext polynomial is stored entirely within a single memory tile, allowing multiple logic tiles to operate on different polynomials in parallel, as shown in Figure 5 (a). However, this would cause a significant inter-tile communication traffic when accumulating results in different tiles. In the example in Figure 5 (a), four polynomials,  $P_0$ – $P_3$ , are mapped to different tiles respectively ( $P_i$  is the element-wise multiplication result of two polynomials, and represents the mapping of input polynomials and the result polynomial). When  $P_0$  is accumulated with  $P_1$ , and  $P_2$  with  $P_3$ , the extra inter-tile communication will be incurred, which brings more execution time.

To overcome this limitation, we propose **Parallelism-Aware Ciphertext Mapping (PACM)**. The key idea is to partition a single polynomial across multiple tiles rather than binding it to a single tile, while co-locating polynomials that require accumulation on the same group of tiles. As illustrated in Figure 5 (b), PACM partitions polynomial  $P_0$  across tiles 0 and 1, and also partitions polynomial  $P_1$  in exactly the same way across tiles 0 and 1, as  $P_1$  must be accumulated with  $P_0$ . Polynomials  $P_2$  and  $P_3$  are partitioned across tiles 2 and 3. This ensures accumulation happens locally within each tile group, minimizing inter-tile communication.

The PACM mechanism is implemented as a deterministic, three-step process to generate the optimal data layout: First, PACM partitions an  $N$ -degree polynomial on  $n$  tiles, where  $n = \frac{N}{\text{vector\_width}}$  and  $\text{vector\_width}$  is the per-tile

vector processing width. Second, PACM maps polynomials that require accumulation on the same tiles. Third, independent polynomials that do not participate in the same accumulation are assigned to different tiles. Thus, the mapping of polynomial partitions to a specific tile is described by:  $\text{Tile\_ID} = \text{offset} + (\text{partition\_index} \bmod \text{tiles\_num})$ , where  $\text{partition\_index} = \text{range}(\frac{N}{\text{vector\_width}})$ .  $\text{offset}$  represents the base tile index assigned to an accumulation group. For  $P_0$  and  $P_1$ :  $\text{Tile\_ID} = 0 + \{0, 1\} \bmod 4$ , which is  $\{0, 1\}$ . For  $P_2$  and  $P_3$ :  $\text{Tile\_ID} = 2 + \{0, 1\} \bmod 4$ , which is  $\{2, 3\}$ .

### D. Adaptive Offloading between HEU and Computing Device

The task partition between the HEU and the existing computing device is crucial for performance. While the 3D-stacked memory-based HEU is highly effective for accelerating data-bound operators, a static offloading strategy that always assigns data-intensive operators to the HEU and computation-intensive operators to the computing device enforces most operators to be executed sequentially and causes severe resource underutilization and idleness. Given that TFHE computations significantly differ from other domains, such as deep learning, and exhibit a massive ciphertext movement, offloading strategies for other domains are not applicable to FHE.

To address this challenge, we propose an adaptive offloading strategy to intelligently offload operators to minimize the overall execution time. We model the workload as a directed acyclic graph (DAG), where each of the nodes represents an FHE operator  $op_i$  and edges represent dependencies. We formally model the problem as finding an offloading decision OD:

$$\text{OD} : \{op_0, op_1, \dots, op_v\} \rightarrow \{\text{Dev}, \text{HEU}\} \quad (1)$$

that maps each operator to either the computing device (Dev) or the HEU in an order, to minimize the total execution time, as shown in Figure 6. To solve this, we construct an offloading model and develop a heuristic search algorithm:

1) *Offloading Model*: For an offloading decision, each operator  $op_i$  has an execution latency modeled as:

$$t_{op_i} = t_{op_i}^{\text{core}} + t_{\text{transfer}}, \quad \text{core} \in \{\text{Ext}, \text{Dev}\}, \quad (2)$$

where  $t_{op_i}^{\text{core}}$  is execution time on either the device or HEU, and  $t_{\text{transfer}}$  is the ciphertext transfer latency between the computing device and the HEU.  $t_{op_i}^{\text{core}}$  can be profiled offline.  $t_{\text{comm}}$  can be estimated according to the data transfer volume and the memory bandwidth.

The start time of each operator depends on both hardware availability and dependency constraints, the maximum

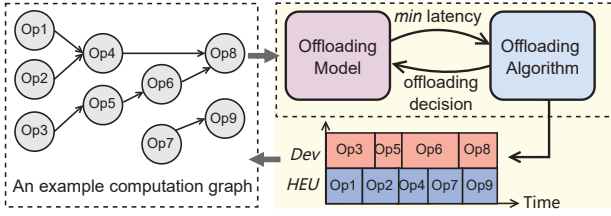


Fig. 6. Illustration of the proposed adaptive offloading.

between the available time of the offloaded hardware and the latest finish time of all its former operators ( $t_{node_j}^{end}$ ):

$$t_{node_i}^{start} = \max(t_{core}^{avail}, \max_{j \in \text{preds}(i)} t_{node_j}^{end}), \quad (3)$$

$$t_{node_i}^{end} = t_{node_i}^{start} + t_{op_i}. \quad (4)$$

$t_{core}^{avail}$  is the available time of the offloaded hardware. The total task latency is the maximum of all  $t_{node_i}^{end}$ .

2) *Genetic-Based Search Algorithm*: We develop a genetic-based search algorithm. Each offloading scheme is encoded as a chromosome, where each gene specifies the target hardware and the execution order of an operator. This search algorithm incorporates our proposed offloading model to evaluate the fitness of each chromosome and evolve a population iteratively [23]. The adaptive offloading optimization is performed offline at compilation time for a given FHE workload and runs about 110 seconds to solve for the optimal offloading, making its overhead acceptable as a single offline compilation process.

#### E. Overall Workflow

1) **First**, FHEX lowers TFHE workload into primitive operators, such as the NTT, external product, polynomial scale, or decomposition, and constructs a computation graph.

2) **Second**, based on the computation graph, FHEX offloads each operator in the computation graph using the proposed adaptive offloading.

3) **Third**, for operators offloaded in HEU, FHEX optimizes the ciphertext layout on the 3D stack memory utilizing the proposed PACM mechanism. For operators offloaded to the computing device with SIMD cores, FHEX compiled them into SIMD code. Mapping NTT to this device can be efficiently done following the method in [24].

## IV. EVALUATION

### A. Experimental Methodology

**Implementation.** We implemented the HEU logic die in RTL and synthesized it using the Synopsys Design Compiler with a TSMC 45nm library, setting the clock rate to 1 GHz. The 3D stacked memory configuration is referenced after a feasible design from recent work [21], [22]. The internal bandwidth is 4320 GB/s, and the external bandwidth is 385 GB/s [19]. To evaluate the micro-architectural behaviors of the FHEX extension, we build a cycle-accurate simulator based on [21], [25]. For the computing device, we use the AMD MI50 architecture with 32 SIMD-64 cores, simulated using [26].

**Baselines.** We compare against a comprehensive set of baselines: We set the CPU baselines Intel(R) Xeon(R) Platinum

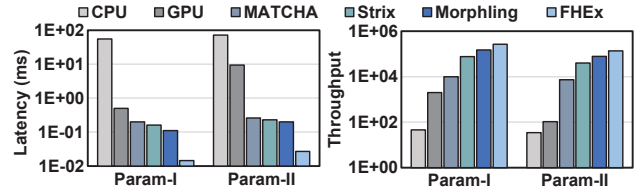


Fig. 7. (a) Latency and (b) throughput comparison of FHEX extension over baselines on micro-benchmark.

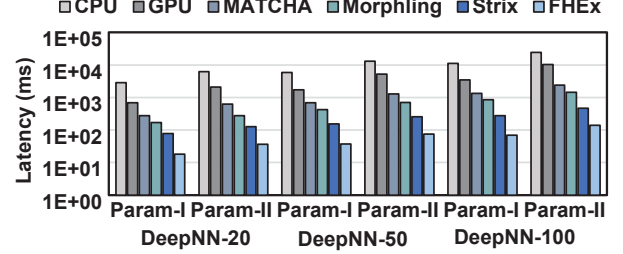


Fig. 8. Performance comparison of FHEX over baselines on DeepCNNs.

8163 CPU @ 2.50GHz running the TFHE library [10]. We use the NVIDIA A100 GPU running the cuFHE library [27]. We choose MATCHA [2], Morphling [4], and Strix [3] as the TFHE accelerator baselines. We also set the processing-near-memory baseline, THEIM [28]. For fair comparisons, we set the same computing resources.

**Benchmarks.** We set both micro-benchmarks and also the application benchmarks under different encryption parameters in Table I recommended by [10]. For micro-benchmarks, we evaluate bootstrapping, the most bottleneck operation. For application benchmarks, we use DeepCNNs [1], TFHE-based models from ZAMA company. We set three configurations of layer number: DeepCNN20, DeepCNN50, and DeepCNN100.

TABLE I  
TFHE ENCRYPTION PARAMETERS.

TFHE	Parameters
Parameter Set I	$n = 500, N = 1024, L = 2, k = 1$
Parameter Set II	$n = 630, N = 2048, L = 3, k = 1$

### B. Results.

1) *Area and Power*: We have the ciphertext processing unit in HEU configured with 64 multipliers and 64 adders in each logic tile. To ensure the vertical alignment, FHEX has 16 logic tiles. The total amount of buffer of the logic tile is 256KB. The total area consumption of the FHEX is 104 mm<sup>2</sup> and the power consumption is 38.3 W in TSMC 45nm technology.

2) *Performance Results*: First, we evaluate FHEX on the micro-benchmark, bootstrapping, which is widely recognized as the performance bottleneck in TFHE. As shown in Figure 7, FHEX achieves 3240.4 $\times$ , 191.3 $\times$ , 11.7 $\times$ , 3.8 $\times$ , and 3.0 $\times$  speedup, and 4893.9 $\times$ , 706.6 $\times$ , 18.0 $\times$ , 3.6 $\times$ , and 1.8 $\times$  throughput improvement compared with CPU, GPU, MATCHA, Strix, and Morphling, respectively. The reasons for performance improvement are threefold. First, FHEX with the proposed 3D stack memory-based HEU efficiently accelerates the data-intensive operators and simultaneously utilizes the ample computing power of the existing computing device. Second, beyond simply utilizing the high bandwidth of 3D stack memory, the proposed ciphertext mapping optimizes ciphertext layout

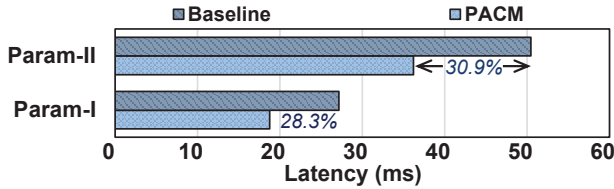


Fig. 9. Analysis of the proposed Parallelism-Aware Ciphertext Mapping.

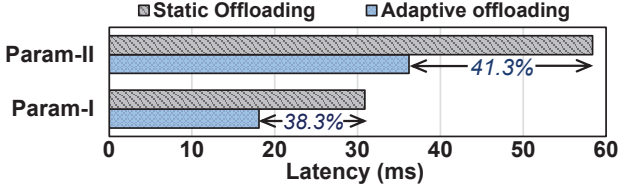


Fig. 10. Analysis of the proposed adaptive offloading algorithm.

on the 3D stack memory architecture and reduces the inter-tile ciphertext communication. Third, the proposed adaptive offloading intelligently offloads FHE operators between the HEU and the computing device, exploiting the parallelism between them and minimizing the overall execution latency.

Then, we also evaluate FHEX on application benchmarks: DeepCNNs, which are practical Private Inference models from the ZAMA company. As shown in Figure 8, FHEX achieves 165.5 $\times$ , 55.9 $\times$ , 17.4 $\times$ , 10.0 $\times$ , and 4.0 $\times$  speedup compared with CPU, GPU, MATCHA, Morphling, and Strix. The evaluation results demonstrate that FHEX effectively transforms an existing computing device into an efficient FHE engine.

### C. Performance Analysis

Here, we analyze the impact of the proposed ciphertext mapping and the adaptive offloading algorithm.

#### 1) Impact of the Parallelism-Aware Ciphertext Mapping:

To provide a direct, quantitative impact of the proposed PACM mechanism, we set a baseline where each polynomial is entirely stored within a memory tile. We measure the latency on DeepCNN-20. As shown in Figure 9, compared with the baseline, our Parallelism-Aware Ciphertext Mapping achieves 29.6% latency reduction. This improvement stems from the PACM mechanism, which maps polynomials to be accumulated with the same tile, reducing inter-tile communications.

2) *Impact of the Adaptive Offloading Algorithm:* Next, we analyze the impact of the proposed adaptive offloading. As shown in Figure 10, compared with a static offloading policy that always sends computation-intensive operators to the computing device and the data-intensive operators to the hardware extension unit, our proposed adaptive offloading yields a 39.8% performance improvement. While the static policy is intuitive, its enforcement of sequential execution leads to significant hardware idle time. In contrast, the proposed adaptive offloading intelligently offloads the operators to exploit parallelism between the HEU and the computing device. For example, the EP operator is bottlenecked by memory bandwidth on the computing device. On the HEU, EP execution has a smaller latency. FHEX’s adaptive offloading wisely offloads FHE operators to minimize the overall latency.

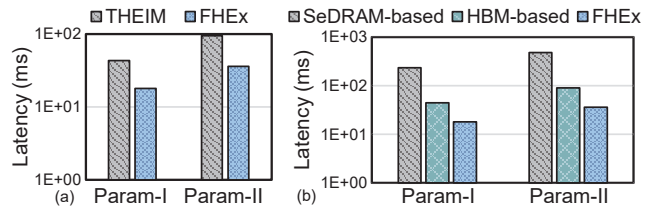


Fig. 11. Comparison with PIM-based designs.

### D. Comparative Analysis with PIM-based Solutions

Next, we compared FHEX against THEIM [28], a state-of-the-art HBM-PIM TFHE accelerator. THEIM consists of two PIM-HBM stacks and places processing unit per bank. THEIM has a total of 1024 multipliers. For fair comparisons, we configure FHEX with the same computational resources and use the same TFHE algorithmic implementation. THEIM has an internal memory bandwidth of 1.23 TB/s, which is higher than that of FHEX. Despite this bandwidth advantage, FHEX achieves a 2.6 $\times$  speedup over THEIM on DeepCNN20, as shown in Figure 11 (a). THEIM employs a coarse-grained, static offloading model, which executes the entire bootstrapping on the PIM, while offloading other operations to an FPGA. In contrast, FHEX utilizes a fine-grained, adaptive offloading. FHEX lowers the bootstrapping into primitive operators and employs the adaptive offloading to find the optimal operator offloading. This adaptive offloading is critical because not all parts of bootstrapping are suitable for PIM. Specifically, the butterfly computations within the NTT are highly irregular, which cannot fully leverage the high bandwidth of PIM. These results demonstrate that the FHE acceleration is beyond simply utilizing the high memory bandwidth.

We also compare FHEX with alternative solutions that adopt different types of memory devices with different bandwidth: HBM3 [29] with a bandwidth of 896 GB/s and SeDRAM with a bandwidth of 136 GB/s [30]. In each case, we place the FHE-specific logic into the custom logic die within the memory stack, and each case applies the proposed adaptive offloading. As shown in Figure 11 (b), FHEX achieves 2.4 $\times$  and 13.2 $\times$  performance improvement compared with the HBM3-based and the SeDRAM-based design. These results demonstrate the advantages of FHEX’s architecture.

## V. CONCLUSION

In this paper, we propose FHEX, a framework that transforms the existing compute device into an efficient FHE engine via a 3D stacked memory-based hardware extension (HEU). Beyond simply utilizing the high bandwidth of 3D stacked memory, we propose the software support for this extension-based architecture, which includes adaptive offloading and a parallelism-aware ciphertext mapping. Evaluation shows FHEX achieves efficient acceleration of FHE and offers a flexible alternative to standalone FHE accelerators.

## VI. ACKNOWLEDGEMENT

This paper is supported by the National Key R&D Program of China: 2023YFB4404400, and the National Natural Science Foundation of China (NSFC) under grant No. 62222411. The corresponding authors are Ying Wang and Yinhe Han.

## REFERENCES

- [1] Ilaria Chillotti, Marc Joye, and Pascal Paillier. Programmable bootstrapping enables efficient homomorphic inference of deep neural networks. In *Cyber Security Cryptography and Machine Learning: 5th International Symposium, CSCML 2021, Be'er Sheva, Israel, July 8–9, 2021, Proceedings 5*, pages 1–19. Springer, 2021.
- [2] Lei Jiang, Qian Lou, and Nrushad Joshi. Matcha: A fast and energy-efficient accelerator for fully homomorphic encryption over the torus. In *Proceedings of the 59th ACM/IEEE Design Automation Conference*, pages 235–240, 2022.
- [3] Adiwena Putra, Prasetyo, Yi Chen, John Kim, and Joo-Young Kim. Strix: An end-to-end streaming architecture with two-level ciphertext batching for fully homomorphic encryption with programmable bootstrapping. In *Proceedings of the 56th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 1319–1331, 2023.
- [4] Adiwena Putra, Joo-Young Kim, et al. Morphling: A throughput-maximized tthe-based accelerator using transform-domain reuse. In *2024 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, pages 249–262. IEEE, 2024.
- [5] Axel Feldmann et al. F1: A fast and programmable accelerator for fully homomorphic encryption (extended version). 2021.
- [6] Nikola Samardzic, Axel Feldmann, Aleksandar Krastev, Nathan Manohar, Nicholas Genise, Srinivas Devadas, Karim Eldefrawy, Chris Peikert, and Daniel Sanchez. Craterlake: a hardware accelerator for efficient unbounded computation on encrypted data. In *Proceedings of the 49th Annual International Symposium on Computer Architecture*, pages 173–187, 2022.
- [7] Yibo Du, Ying Wang, Bing Li, Fuping Li, Shengwen Liang, Huawei Li, Xiaowei Li, and Yinhe Han. Chiplever: Towards effortless extension of chiplet-based system for fhe. In *Proceedings of the 61st ACM/IEEE Design Automation Conference*, pages 1–6, 2024.
- [8] Chiara Marcolla, Victor Sucasas, Marc Manzano, Riccardo Bassoli, Frank HP Fitzek, and Najwa Aaraj. Survey on fully homomorphic encryption, theory, and applications. *Proceedings of the IEEE*, 110(10):1572–1609, 2022.
- [9] Mingqin Han, Yilan Zhu, Qian Lou, Zimeng Zhou, Shanqing Guo, and Lei Ju. coxhe: A software-hardware co-design framework for fpga acceleration of homomorphic computation. In *2022 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 1353–1358. IEEE, 2022.
- [10] Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachène. Tthe: fast fully homomorphic encryption over the torus. *Journal of Cryptology*, 33(1):34–91, 2020.
- [11] Ilaria Chillotti, Damien Ligier, Jean-Baptiste Orfila, and Samuel Tap. Improved programmable bootstrapping with larger precision and efficient arithmetic circuits for tthe. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 670–699. Springer, 2021.
- [12] Arthur Meyre, Benoit Chevallier-Mames, Jordan Frery, Andrei Stoian, Roman Bredehoff, Luis Montero, and Celia Kherfallah. Concrete ml: a privacy-preserving machine learning library using fully homomorphic encryption for data scientists (2022).
- [13] Andrei Stoian, Jordan Frery, Roman Bredehoff, Luis Montero, Celia Kherfallah, and Benoit Chevallier-Mames. Deep neural networks for encrypted inference with tthe. In *International Symposium on Cyber Security, Cryptology, and Machine Learning*, pages 493–500. Springer, 2023.
- [14] Tian Ye, Rajgopal Kannan, and Viktor K Prasanna. Fpga acceleration of fully homomorphic encryption over the torus. In *2022 IEEE High Performance Extreme Computing Conference (HPEC)*, pages 1–7. IEEE, 2022.
- [15] Yibo Du, Ying Wang, Mengdi Wang, Xiaowei Li, and Yinhe Han. Chiplever: A hardware-software co-design framework towards extension of chiplet system for fully homomorphic encryption. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2025.
- [16] Amd inc. 2020. introducing cdna architecture, the all-new amd gpu architecture for the modern era of hpc ai. <https://www.amd.com/system/files/documents/amd-cdna-whitepaper.pdf>.
- [17] Samuel Williams, Andrew Waterman, and David Patterson. Roofline: an insightful visual performance model for multicore architectures. *Communications of the ACM*, 52(4):65–76, 2009.
- [18] Mark Field, Takuji Kimura, John Atkinson, Diana Gamzina, Neville C Luhmann, Brad Stockwell, Thomas J Grant, Zachary Griffith, Robert Borwick, Christopher Hillman, et al. Development of a 100-w 200-ghz high bandwidth mm-wave amplifier. *IEEE Transactions on Electron Devices*, 65(6):2122–2128, 2018.
- [19] Debendra Das Sharma, Gerald Pasdast, Zhiguo Qian, and Kemal Aygun. Universal chiplet interconnect express (ucie): An open industry standard for innovations with chiplets at package level. *IEEE Transactions on Components, Packaging and Manufacturing Technology*, 12(9):1423–1431, 2022.
- [20] Jin Hyun Kim, Yuhwan Ro, Jinin So, Sukhan Lee, Shin-haeng Kang, YeonGon Cho, Hyeonsu Kim, Byeongho Kim, Kyungsoo Kim, Sangsoo Park, et al. Samsung pim/pnm for transfrmer based ai: Energy efficiency on pim/pnm cluster. In *2023 IEEE Hot Chips 35 Symposium (HCS)*, pages 1–31. IEEE Computer Society, 2023.
- [21] Dimin Niu, Shuangchen Li, Yuhao Wang, Wei Han, Zhe Zhang, Yijin Guan, Tianchan Guan, Fei Sun, Fei Xue, Lide Duan, et al. 184qps/w 64mb/mm<sup>2</sup> 3d logic-to-dram hybrid bonding with process-near-memory engine for recommendation system. In *2022 IEEE International Solid-State Circuits Conference (ISSCC)*, volume 65, pages 1–3. IEEE, 2022.
- [22] Song Wang, Bing Yu, Wenwu Xiao, Fujun Bai, Xiaodong Long, Liang Bai, Xuerong Jia, Fengguo Zuo, Jie Tan, Yixin Guo, et al. A 135 gbps/gbit 0.66 pj/bit stacked embedded dram with multilayer arrays by fine pitch hybrid bonding and mini-tsv. In *2023 IEEE Symposium on VLSI Technology and Circuits (VLSI Technology and Circuits)*, pages 1–2. IEEE, 2023.
- [23] Deb Kalyanmoy. A fast and elitist multi-objective genetic algorithm: Nsga-ii. *IEEE Trans. on Evolutionary Computation*, 6(2):182–197, 2002.
- [24] Kaustubh Shivdikar, Gilbert Jonatan, Evelio Mora, Neal Livesay, Rashmi Agrawal, Ajay Joshi, José L Abellán, John Kim, and David Kaeli. Accelerating polynomial multiplication for homomorphic encryption on gpus. In *2022 IEEE International Symposium on Secure and Private Execution Environment Design (SEED)*, pages 61–72. IEEE, 2022.
- [25] Mahmoud Khairy, Zhesheng Shen, Tor M Aamodt, and Timothy G Rogers. Accel-sim: An extensible simulation framework for validated gpu modeling. In *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*, pages 473–486. IEEE, 2020.
- [26] Yifan Sun, Trinayan Baruah, Saiful A Mojumder, Shi Dong, Xiang Gong, Shane Treadway, Yuhui Bao, Spencer Hance, Carter McCardwell, Vincent Zhao, et al. Mgpusim: Enabling multi-gpu performance modeling and optimization. In *Proceedings of the 46th International Symposium on Computer Architecture*, pages 197–209, 2019.
- [27] cufhe. [online] Available: . <https://github.com/vernamlab/cuFHE>.
- [28] Kevin Nam, Heonhui Jung, Hyunyoung Oh, and Yunheung Paek. Affinity-based optimizations for tthe on processing-in-dram. In *Proceedings of the 30th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*, pages 16–31, 2025.
- [29] Myeong-Jae Park, Jinyung Lee, Kyungjun Cho, Jihwan Park, Junil Moon, Sung-Hak Lee, Tae-Kyun Kim, Sanghoon Oh, Seokwoo Choi, Yongsuk Choi, et al. A 192-gb 12-high 896-gb/s hbm3 dram with a tsv auto-calibration scheme and machine-learning-based layout optimization. *IEEE Journal of Solid-State Circuits*, 58(1):256–269, 2022.
- [30] Bai Fujun, Jiang Xiping, Wang Song, Yu Bing, Tan Jie, Zuo Fengguo, Wang Chunjuan, Wang Fan, Long Xiaodong, Yu Guoqing, et al. A stacked embedded dram array for lppddr4/4x using hybrid bonding 3d integration with 34gb/s/1gb 0.88 pj/b logic-to-memory interface. In *2020 IEEE International Electron Devices Meeting (IEDM)*, pages 6–6. IEEE, 2020.