

FLASH-FHE ⚡: A Heterogeneous Architecture for Fully Homomorphic Encryption Acceleration

Junxue Zhang¹, Xiaodian Cheng^{2*}, Gang Cao³, Meng Dai³, Yijun Sun¹, Han Tian⁴,
Dian Shen⁵, Yong Wang¹, Kai Chen¹

¹*iSINGLab @ Hong Kong University of Science and Technology* ²*University of Waterloo* ³*Clustar*
⁴*University of Science and Technology of China* ⁵*Southeast University*

Abstract

While many hardware accelerators have recently been proposed to address the inefficiency problem of fully homomorphic encryption (FHE) schemes, none of them is able to deliver optimal performance when facing real-world FHE workloads consisting of a mixture of shallow and deep computations, due primarily to their homogeneous design principle.

This paper presents FLASH-FHE, the first FHE accelerator with a heterogeneous architecture for mixed workloads. At its heart, FLASH-FHE designs two types of computation clusters, *i.e.*, bootstrappable and swift, to optimize for deep and shallow workloads respectively in terms of cryptographic parameters and hardware pipelines. We organize one bootstrappable and two swift clusters into one cluster affiliation, and present a scheduling scheme that provides sufficient acceleration for deep FHE workloads by utilizing all the affiliations, while improving parallelism for shallow FHE workloads by assigning one shallow workload per affiliation and dynamically decomposing the bootstrappable cluster into multiple swift pipelines to accelerate the assigned workload. We further show that these two types of clusters can share valuable on-chip memory, improving performance without significant resource consumption. We implement FLASH-FHE with RTL and synthesize it using both 7nm and 14/12nm technology nodes, and our experiment results demonstrate that FLASH-FHE achieves an average performance improvement of 1.4× and 11.2× compared to state-of-the-art FHE accelerators CraterLake and F1 for deep workloads, while delivering up to 8.0× speedup for shallow workloads due to its heterogeneous architecture.

1 Introduction

Fully Homomorphic Encryption (FHE) has emerged as a promising technology for enabling privacy-preserving computation. It allows rich computation directly over encrypted data without the need for decryption. However, FHE suffers from a significant drawback—inefficiency. Compared to cleartext computation, FHE is approximately 5–6 orders of magnitude slower. To address this challenge and make FHE a practical solution, researchers have proposed FHE accelerators.

*Work done while at iSINGLab @ Hong Kong University of Science and Technology.

FHE accelerators employ increasingly advanced hardware platforms, ranging from CPUs [9] to GPUs [20, 27], FPGAs [3, 7, 39], and ASICs [28–30, 40, 41], in pursuit of high acceleration performance. However, in this paper, we observe that existing FHE accelerators all ignore one crucial feature of real-world FHE workloads: they are highly mixed, thus failing to achieve optimal overall performance for mixed workloads.

The mixed FHE workloads pose different design goals and choices for FHE accelerators. Specifically, some FHE workloads, referred to as shallow workloads, do not require bootstrapping, such as matrix computation and database queries [1]. They require a relatively small multiplication level L and polynomial degree N . Therefore, a potential acceleration method for shallow workloads is to provide adequate parallelism for them. On the other hand, deep workloads, including neural network inference [33, 36] and big data analytics [26], necessitate bootstrapping, along with requiring a larger N and L . Moreover, the overall performance of deep workloads is largely decided by the performance of bootstrapping, which usually requires one dedicated accelerator to be fully accelerated.

Consequently, using FHE accelerators designed for shallow workloads, such as F1, to support mixed FHE workloads results in either the inability to handle deep workloads or over a 10× performance degradation due to naïve extension (*e.g.*, F1+ in [41]). Conversely, utilizing FHE accelerators built for deep workloads, such as CraterLake, results in inadequate parallelism for shallow workloads. Increasing parallelism by allocating more computation clusters leads to overwhelming chip area and poses commercial drawbacks for FHE accelerators. Furthermore, the non-continuous nature of cryptographic parameters for shallow and deep workloads eliminates the possibility of achieving good average performance through parameter averaging.

A potential solution to handling both shallow and deep workloads is to use two distinct architectures. However, Using two separate architectures for shallow and deep workloads nearly doubles production costs, while relying on CPU/GPU/FPGA for shallow workloads results in poor performance and additional costs. This also introduces deployment challenges, such as increased PCIe usage and failure rates.

To this end, we ask: *can we design a FHE accelerator that achieves consistently high performance for mixed work-*

loads? To answer the question, we observe that existing FHE accelerators adopt a *homogeneous* design, where the cryptographic parameter, hardware pipeline design and scheduling mechanism are purely optimized for either shallow or deep workloads only, precluding simultaneous optimization for both shallow and deep workloads.

Based on this observation, we propose FLASH-FHE, the FHE accelerator with a *heterogeneous* architecture for mixed FHE workloads. At its core, FLASH-FHE employs two types of computation clusters: bootstrappable and swift computation clusters, optimized for deep and shallow workloads, respectively. FLASH-FHE organizes one bootstrappable and two swift clusters as one cluster affiliation and has eight affiliations in total. Besides conventional scheduling policy for deep workloads by using all affiliations, FLASH-FHE further incorporates a scheduling mechanism that targets at improving parallelism by assigning one shallow workload to only one cluster affiliation, where the bootstrappable cluster is dynamically decomposed into multiple separate swift pipelines to adequately accelerate the assigned workload. Last but not least, FLASH-FHE integrates a hierarchical data cache, which allows different clusters to share valuable on-chip memory resources. Therefore, we can improve the overall performance of FLASH-FHE without dramatically increasing the chip area. To the best of our knowledge, FLASH-FHE is among the first to introduce a heterogeneous architecture for FHE accelerators.

We implement FLASH-FHE using RTL and synthesize it with two technology nodes, *i.e.*, 7nm and 14/12nm, respectively. We further evaluate FLASH-FHE with seven real-world FHE workloads, comprising three shallow and four deep workloads. Evaluation results demonstrate that compared to CraterLake and F1, which are two representative FHE accelerators with 14/12nm technology node, FLASH-FHE can achieve $1.4\times$ and $11.2\times$ average performance improvement for deep workloads, respectively. For shallow workloads, FLASH-FHE can achieve up to $8.0\times$ speedup since it can accelerate multiple shallow workloads in parallel. Moreover, the significant performance improvement for shallow workloads is achieved at the cost of $< 7\%$ extra hardware resources.

Finally, we summarize the contributions of this paper as follows:

- We thoroughly analyze representative shallow and deep FHE workloads and identify their optimal parameter settings, highlighting the polarization between these workloads.
- Based on our observations, we are among the first to introduce a heterogeneous acceleration architecture for FHE, inspired by the big.LITTLE design philosophy. This includes heterogeneous (i)NTT pipelines, a multi-level transpose module, and a hierarchical caching structure to fully implement the big.LITTLE concept.
- Leveraging our heterogeneous architecture, we propose the first multi-job scheduler for FHE workloads, optimizing

Notation	Definition
N	Degree of a polynomial.
L	Multiplicative depth of a fresh ciphertext.
l	Current multiplicative depth of a ciphertext.
q	Coefficient modulus of cleartext polynomial.
Q	Coefficient modulus of ciphertext polynomial.
G	The number of computation groups in the group architecture.
R	The number of rows in the NTT implementation.
C	The number of columns in the NTT implementation.
l_{sub}	The number of parallel modular multiplication pipelines in the basis converter of sequential BConv unit.
$\{q_i, i \in [0, L]\}$	A set of moduli. $Q = \prod_{i=0}^L q_i$.
P	Special modulus for the keys.
$\{p_i, i \in [0, \alpha - 1]\}$	A set of special moduli. $P = \prod_{i=0}^{\alpha-1} p_i$.
dnum	Decomposition number in key-switching.
α	# of special moduli p_i . $\alpha = \lfloor (L+1)/\text{dnum} \rfloor$.

Table 1: Notations used in this paper.

both parallelization and cache hit ratios.

We believe our contributions provide valuable insights for the FHE community by demonstrating how the concept of heterogeneity can be seamlessly integrated into existing FHE accelerators, such as CraterLake [41], ARK [29], SHARP [28], *etc.* While these accelerators are primarily optimized for deep FHE workloads, our approach enables them to retain their current deep computation engines while incorporating key components such as multi-exit (i)NTT pipelines, multi-level transpose, and hierarchical caching. With the addition of these hardware modules and our multi-job scheduler, these accelerators can significantly improve their support for shallow FHE workloads by allowing parallel execution, making them more versatile and capable of handling mixed FHE workloads with minimal modifications.

2 Background

2.1 Fully Homomorphic Encryption

FHE allows performing specific homomorphic operations directly over ciphertexts without decrypting them [10, 14, 15, 19]. In the following sections, we will take CKKS [14] as an example to demonstrate the operations used in FHE, and other FHE schemes, such as BFV [19], BGV [10], *etc.*, should share similar operations. Table 1 summarizes the notations used in this paper.

Homomorphic Addition: The ciphertext in CKKS can be represented as $\text{ct} = (c_0, c_1)$, where c is a polynomial. The result of the homomorphic addition between one ciphertext ct and one cleartext m , which is also a polynomial, is $(c_0 + m, c_1)$ while the result of two ciphertexts $\text{ct}_0 = (c_{0,0}, c_{0,1})$ and $\text{ct}_1 = (c_{1,0}, c_{1,1})$ is $(c_{0,0} + c_{1,0}, c_{0,1} + c_{1,1})$. The addition operator $+$ here denotes an element-wise addition of two polynomials.

Homomorphic Multiplication: The multiplication result between one ciphertext ct and one cleartext m is $(c_0 * m, c_1 * m)$, where $*$ represents polynomial multiplications. However, the homomorphic multiplication operation between two cipher-

texts is more complicated since multiplication between the above ct_0 and ct_1 results in a ciphertext ct' with three elements:

$$ct' = (c_{0,0} * c_{1,0}, c_{0,0} * c_{1,1} + c_{0,1} * c_{1,0}, c_{0,1} * c_{1,1}) \quad (1)$$

To turn the ciphertext back into two elements for future ciphertext computation with consistent form, we have to perform ciphertext maintenance operation – key-switching – to relinearize the ciphertext. We will introduce the key-switching operation later.

Rotation: CKKS exploits the idea of SIMD (*i.e.*, batching) by packing multiple cleartexts in a single vector, which is further encrypted as a single ciphertext. Although it is efficient for most operations, it requires homomorphic ciphertext rotation to perform intra-batch calculations. The ciphertext rotation consist of two operations. First, it performs automorphisms operation, which is a permutation over coefficients of polynomials for both ciphertext ct and secret key s . Second, key-switching is performed to convert the permuted ciphertext into the final result.

Besides the aforementioned homomorphic evaluation operations, FHE schemes have to perform some ciphertext maintenance operations to preserve the correctness of these operations. Two of the most important operations are key-switching and bootstrapping.

Key-switching: As its name implies, the key-switching operation homomorphically switches the secret key of a ciphertext while keeping the corresponding cleartext unchanged. Therefore, the key-switching operation is extensively used in homomorphic operations, such as multiplication, rotation, *etc.*, to keep the results correct. The key-switching operation accounts for a significant portion of the overall computation time due to the inherent complexity involved in its internal computations.

Bootstrapping: After performing homomorphic operations, especially the multiplication operation, the noise of a ciphertext increase. When too many operations are performed, the noise will overflow, leading to incorrect results after decryption. We call the maximum number of consecutive homomorphic multiplications allowed as the multiplicative level/budget. Therefore, before the number of operations exceeds the multiplicative level, the bootstrapping operation is needed to "refresh" the noise of ciphertexts by homomorphically re-encrypting the old ciphertexts to generate a fresh one.

Bootstrapping is the most complicated operation in FHE, which involves many other operations including key-switching. If used, bootstrapping will considerably determine the performance of the overall performance of a FHE scheme.

Finally, we discuss the cryptographic parameter choices of FHE schemes. There are several important parameters to determine: (1) the polynomial degree N ; (2) the range of the coefficients of a the polynomial. Since CKKS is built on a polynomial ring, the range of the coefficients is decided by the modulus Q . 3) the modulus P used in keys. To ensure security, $N/\log PQ$ must be over a certain threshold, for

example, 128 bits [12]. Moreover, Q determines the multiplicative level/budget, and a larger Q generally leads to more multiplicative levels. For example, multiplicative level of 16 requires Q to be around 512. Therefore, to satisfy both high security and high multiplicative levels, a larger N is also needed, which inevitably causes inflated ciphertexts.

2.2 General Optimizations for FHE Implementations

This section introduces three general optimization techniques used by modern FHE implementations.

NTT/iNTT: To accelerate polynomial multiplications, the Number Theoretic Transform (NTT) and the inverse Number Theoretic Transform (iNTT) operations are used. The NTT/iNTT lowers the time complexity of polynomial multiplications from $O(n^2)$ to $O(n \log n)$, where n is the degree of a polynomial. Readers can regard NTT/iNTT as Fast Fourier Transform over finite field. The basic operation of NTT/iNTT is the butterfly operation [17, 22].

RNS Decomposition: To reduce computation complexity in CKKS, large modulus Q and P are decomposed into the product of several smaller co-prime moduli: $Q = q_0 q_1 \dots q_L$ and $P = p_0 p_1 \dots p_\alpha$. A set containing multiple co-prime moduli $\{q_0, q_1, \dots\}$ is called an RNS basis. Following the Chinese Remainder Theorem [21], a large integer can be represented by taking its modulus results with respect to all moduli in the RNS basis. This representation is referred to as the RNS representation of the integer.

Fast Basis Conversion (BConv) : Some FHE operations involve the conversion (switching) of modulus, which requires special operations under RNS representation. For example, the Fast Basis Conversion (BConv) approximates the modulus switching by converting an integer from the original RNS basis to a new basis. BConv mainly consists of multiple modular multiplications and independent summations.

2.3 Prior FHE Accelerators

One key problem restricting FHE from practical deployment is its inefficiency issue. Even highly optimized FHE schemes suffer from five orders of magnitudes slower performance than cleartexts computation [47]. To solve the problem, various FHE accelerators are proposed [3, 7, 20, 27, 29, 30, 39–41]. We categorize them based on the employed hardware platform.

FPGA: Some FHE accelerators leverage Field Programmable Gate Array (FPGA) as their hardware platform, such as Intel HEXL-FPGA [3], HEAX [39], FAB [7]. However, due to the limitations of FPGA itself, *e.g.*, limited programmable resources, low operational frequency and low memory bandwidth, FPGA-based FHE accelerators cannot satisfy the requirements of FHE schemes, which are both computation and memory intensive. Therefore, their performance

is far from optimal.

GPGPU: General Purpose GPU (GPGPU) has been extensively adopted to offload AI-related applications. Due to their relatively good performance to accelerate data-parallel applications, researchers also utilize GPGPUs to accelerate FHE applications, such as $100\times$ [27], TensorFHE [20], *etc.* However, as revealed in [46], GPGPU suffers from architectural deficiency when handling inflated data, causing impractical performance. Moreover, large power consumption of GPGPU is another drawback of these solutions.

ASIC: Recently, people have begun to explore Application Specific Integrated Circuit (ASIC) for superb performance. F1 is the first programmable FHE accelerator implemented with ASIC [40], which can outperform software FHE implementations by $5400\times$ on average. However, due to the lack of efficient bootstrapping, F1 can only deliver ideal acceleration for shallow FHE workloads. Later, CraterLake [41] and BTS [30] were proposed to support full bootstrapping and both employ massive computation units, such as NTT/iNTT, and considerable amounts of on-chip memory. ARK [29] is the successor of them and it further improves the bootstrapping performance by an algorithm-architecture co-design.

Existing development trend of FHE accelerators is to support bootstrapping, thus mainly targeting improving the performance of deep FHE workloads. However, in this paper, we argue that real-world FHE workloads are highly mixed of shallow and deep ones, thus significant opportunities exist to enhance the performance of existing FHE accelerators by considering how to simultaneously accelerate both workloads.

3 Motivation

In this section, we first introduce the nature of FHE workloads that they are highly mixed — some of the workloads require bootstrapping while some do not (§3.1). Second, we will demonstrate how different cryptographic parameters impact the end-to-end performance, and present the appropriate parameter choices for both workloads (§3.2). Third, we show that both hardware pipeline design and scheduling policies are completely different for both workloads (§3.3).

3.1 Mixed FHE Workloads

FHE workloads exhibit a high degree of variability in real-world scenarios, encompassing both bootstrapping and non-bootstrapping requirements. In this paper, similar to [41], we call workloads that require bootstrapping as deep workloads and those without bootstrapping as shallow workloads.

Shallow FHE workloads are common in real-world. First, some FHE workloads do not require very deep multiplications by themselves, such as database query [1], private information retrieval [8], Beaver’s multiplication triples generation for secret sharing [38], federated learning [42], shallow neural

	2^{12}	2^{13}	2^{14}	2^{15}	2^{16}	2^{17}
Matrix Multiplication	101	192	192	192	192	192
DBTable Lookup	-	-	399	575	653	653
Logistic Regression	-	-	-	816	1550	3125
LSTM	-	-	-	821	1536	2901

Table 2: The log PQ setting used in the paper.

network inference [11], *etc.* Second, people also try to avoid bootstrapping by introducing some stochastic designs for even more complicated applications to preserve acceptable performance. For example, Sphinx synthesizes FHE and differential privacy to achieve a uniformed framework for neural network training and inference without bootstrapping (but requires decryption and re-ryption at client side) [45].

With the increasing requirements of applying privacy-preserving technologies to more complicated applications, deep FHE workloads also become widely adopted. One typical example is machine learning training based on pure FHE, such as FHE-protected logistic regression training [24], FHE-protected deep neural networking training [34], *etc.*

FHE accelerators, as an essential infrastructure, should deliver optimal acceleration for both workloads simultaneously. However, in the following sections, we will demonstrate that these workloads require different choices of cryptographic parameters (§3.2), pipeline designs and scheduling policies (§3.3), thus causing sub-optimal performance for existing FHE accelerators which utilize a homogeneous architecture.

3.2 Impact of Cryptographic Parameters

First, we qualitatively introduce the relationship of some important parameters. As discussed in §2.1, for deep FHE workloads, to support bootstrapping, a large multiplication depth L is needed since (1) frequent bootstrapping should be avoided and (2) bootstrapping itself consumes massive multiplication depths. Moreover, when a large L is chosen, a large polynomial degree N is also required to ensure security [12]. In contrast, for shallow workloads, the N and L are usually much smaller. Next, we present how these parameters impact the performance of both shallow and deep workloads through testbed experiments. Especially, we implement the following four real-world workloads with Lattigo [4], a widely-adopted FHE library.

- **Matrix Multiplication:** It’s a shallow FHE workloads. We multiply two matrix of 100×1000 and 1000×10 elements encrypted via CKKS, respectively.
- **DBTable Lookup:** It’s a shallow FHE workloads. We refer the implementation used in [1] and use BGV to encrypt the data. We have modified the algorithm to use binary encoding to encode the key to reduce the required multiplication levels.
- **Logistic Regression:** It’s a deep FHE workload and the implementation is based on [24]. We measure the training time of a single batch with up to 197 features and 50

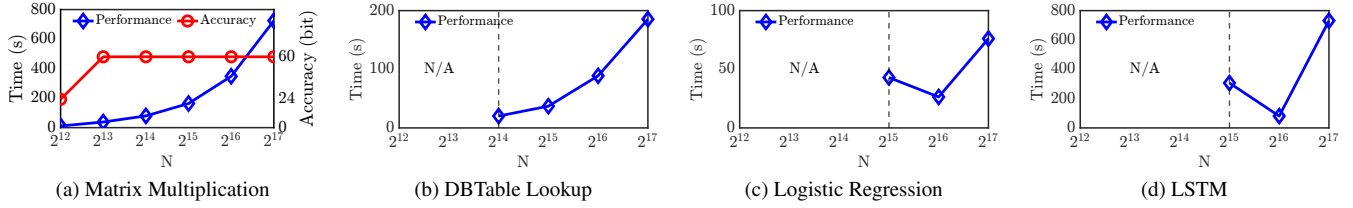


Figure 1: Performance of FHE Workloads.

samples per batch. We perform bootstrapping once the multiplication level is exhausted.

- LSTM:** Long Short-Term Memory (LSTM) is a recurrent neural network (RNN), which is widely adopted to learn the long-term dependencies, especially in sequence prediction problems. In this paper, we follow the implementation in [36] to use CKKS to protect the parameters of a LSTM network. It's also a deep FHE workload.

In every workload, we fix the security level to be 128bit and vary the N from 2^{12} to 2^{17} . Therefore, to ensure the 128bit security level, the maximum $\log PQ$ should vary from 101 to 3125 accordingly. We will choose the most suitable multiplication depth L for each workload while not violating the constraints of maximum $\log PQ$. Another cryptographic parameter that affects the performance of bootstrapping is the decomposition number, dnum , which is used in the key-switching operations. As suggested in some papers [27, 30], to support efficient bootstrapping for deep workloads, we usually choose a small dnum . However, a small dnum may violate the security guarantee. Thus, we try to choose a small dnum that will not violate the security guarantee when N and L are fixed. The detailed settings of each workload are shown in Table 2.

In the conducted experiments, we focused on measuring the execution time of the FHE workloads. Specifically, for the Logistic Regression workload, we calculated the average execution time of one iteration. As for the LSTM workload, we reported the execution time of one LSTM unit. The obtained results are illustrated in Figure 1. In summary, we have the following observations:

- Shallow workloads, such as Matrix Multiplication (Figure 1a) and DBTable Lookup (Figure 1b), can achieve ideal performance with small N and L . A large N and L in turn leads to performance degradation. For example, for Matrix Multiplication, when the N equals 2^{13} , the application can simultaneously achieve adequate accuracy (60bit) while keeping efficient. For DBTable Lookup, when the N equals 2^{14} , it can achieve highly efficient performance. With analysis on more shallow workloads (which are not shown due to limited space), we find $N \leq 2^{14}$ is a reasonable value for shallow FHE workloads.
- Deep workloads, such as Logistic Regression (Figure 1c) and LSTM (Figure 1d), require a large N and L to work properly. For example, if the $N < 2^{15}$, the Logistic Regres-

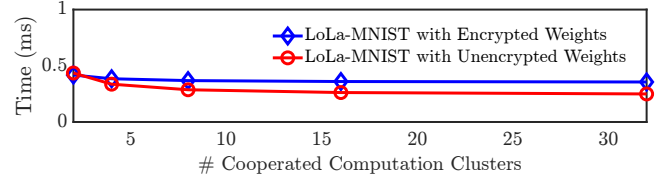


Figure 2: The performance of shallow FHE workloads.

sion cannot even finish one iteration, leading to task failure. Moreover, when the N (also $\log PQ$) becomes larger, the performance of the FHE application first improves, then drops. The reason is that if N is small, the multiplicative depth is low. As a result, the bootstrapping is too frequent, degrading the overall performance. In contrast, if N is too large, the performance also degrades due to the increased complexity of cryptographic system. In summary, we observe that $2^{15} \leq N \leq 2^{16}$ is a sweet range for most deep FHE workloads.

Consequently, FHE accelerators designed for shallow workloads, such as F1 [40], either fail to run deep workloads or suffer from enormous performance reduction. The reason is that, since they are optimized for shallow workloads, they usually choose to support a small N and L . For example, F1 supports $N \leq 2^{14}$. Thus, deep workloads, such as Logistic Regression, cannot even run when the $N \leq 2^{14}$. Furthermore, as pointed out in the previous work [41], if we arbitrarily increase the N of F1 to 2^{16} (denoted as F1+ in Table 3 of [41]), its performance is still more than $10\times$ slower than expected. The reason is that it lacks an efficient bootstrapping implementation (details in §3.3).

3.3 Impact of Pipeline Design & Scheduling

As discussed in previous literature [47], bootstrapping occupies more than 80% of the total computation time. Therefore, efficiently accelerating bootstrapping is a key to improving the performance of deep FHE workloads.

Recent works have explored to build a dedicated $\text{iNTT} \rightarrow \text{BConv} \rightarrow \text{NTT}$ pipeline to enable efficient bootstrapping computation for deep workloads [28–30, 41]. Moreover, the pipeline execution involves the cooperation of all computation clusters due to the heavy workload. Therefore, the scheduler of recent FHE accelerators tends to schedule the whole accelerator to expedite one dedicated deep FHE workload, and a development trend is to utilize more hardware resources for better performance [28–30, 40, 41].

On the contrary, in this paper, we find that for shallow workloads, blindly allocating more hardware resources, *e.g.*, computation clusters, does not lead to effective acceleration. To demonstrate this, we use experiments to investigate how the performance of two shallow FHE workloads changes when we only increase the number of computation clusters without changing the workloads. To align with most results in previous works, we use LoLa-MNIST with and without Unencrypted Weights as workloads here (extensive details in §6.1). The results are presented in Figure 2 and we can observe that when we allocate more than four 128-bit NTT computation clusters to cooperatively accelerate one shallow FHE workload, the performance improves very little. Instead, we argue that for shallow FHE workloads, the optimal scheduling policy is to distribute multiple shallow FHE workloads on one accelerator in parallel while concurrently implementing sufficient computation clusters.

As a result, if we use existing FHE accelerators that are designed for deep FHE tasks to accelerate the shallow FHE workloads, they can only handle one job simultaneously, leading to low data parallelism. Meanwhile, naively increasing the number of clusters causes overwhelming chip area consumption, which is impractical.

Conclusion: Existing FHE accelerators cannot consistently achieve high performance facing real-world mixed FHE workloads. The crux lies in the fact that they all adopt a *homogeneous* design: their target cryptographic parameters, pipeline design and scheduling policy are optimized for either shallow or deep FHE workload.

3.4 Two Different Architectures?

A straightforward approach would be to utilize two distinct architectures for shallow and deep workloads, respectively. However, this solution either incurs significantly higher production costs or results in degraded performance, along with practical deployment challenges.

- *Using two ASICs:* One option is to produce two separate ASICs, each dedicated to either shallow or deep workloads. However, this approach nearly doubles production costs due to the non-recurring engineering (NRE) costs—such as design, verification, and mask set creation—that are required for each chip.
- *Using CPU/GPU/FPGA to handle shallow workloads:* Another option is to use a dedicated ASIC for deep FHE workloads and rely on CPU/GPU/FPGA for shallow workloads. However, as demonstrated in §6, CPU/GPU/FPGA platforms fail to deliver competitive performance. Additionally, GPUs and FPGAs introduce further costs.

Furthermore, both of these solutions present deployment challenges, including increased PCIe slot usage, higher failure rates, *etc.*

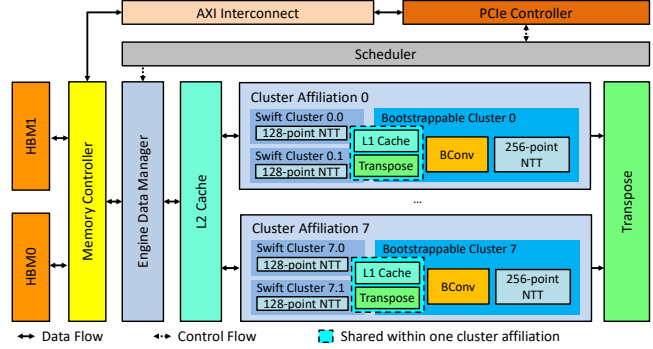


Figure 3: FLASH-FHE’s architectural overview.

4 FLASH-FHE

To solve this problem, we propose FLASH-FHE, a practical FHE accelerator with a *heterogeneous* architecture. Instead of relying on homogeneous computation clusters, FLASH-FHE incorporates two types of clusters: bootstrappable and swift. In terms of cryptographic parameters and pipeline design, these clusters are optimized for deep and shallow workloads, respectively. We organize one bootstrappable and two swift clusters as one cluster affiliation and FLASH-FHE has eight cluster affiliations in total (§4.1). Then, FLASH-FHE’s scheduler tries to maximize the parallelism for shallow FHE workloads by assigning one shallow workload to only one cluster affiliation, which can deliver effective acceleration by decomposing one bootstrappable cluster into multiple suitable pipelines for shallow workloads. In this way, FLASH-FHE can use eight cluster affiliations to achieve a parallelism of eight (§4.2). Furthermore, we introduce the design of a hierarchical data cache that is shared between the bootstrappable and swift clusters. This approach allows us to enhance performance by allocating additional computation clusters without significantly increasing the demand for on-chip memory resources (§4.3).

Architectural Overview: Figure 3 illustrates the architectural overview of FLASH-FHE. It comprises three key components: computation clusters, control subsystem, and memory subsystem. As previously mentioned, the computation clusters consist of eight cluster affiliations, each housing one bootstrappable and two swift clusters. The clusters are connected using a multi-leveled transpose module. The core module of the control subsystem is the scheduler, and FLASH-FHE also leverages PCIe and memory controller for auxiliary functions. In terms of the memory subsystem, FLASH-FHE employs two high-bandwidth memory (HBM) [2] units for off-chip data storage, and incorporates a hierarchical data cache (L1/L2) for on-chip storage.

4.1 Computation Clusters

In this section, we first introduce the design of the two types of computation clusters used in FLASH-FHE, and then present how we use cluster affiliation to organize them.

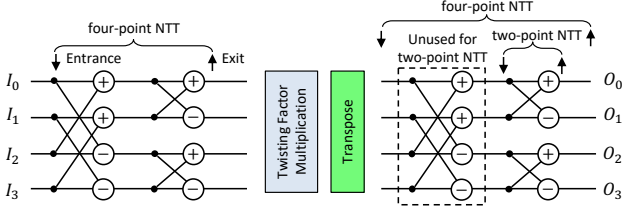


Figure 4: (i)NTT workflow. The (i)NTT pipeline in FLASH-FHE supports multiple entrances and multiple exits to enable efficient execution of varying-point NTTs.

Bootstrappable Clusters: To ensure efficient bootstrapping, we implement a complete $iNTT \rightarrow BConv \rightarrow NTT$ pipeline for bootstrappable clusters, following previous works [28–30, 41]. Within a bootstrappable cluster, we incorporate one (i)NTT pipeline and one BConv module.

The (i)NTT pipeline consists of a four-step (i)NTT¹ module with $N = 2^{16}$, encompassing a \sqrt{N} -point (i.e., 2^8 -point) standard (i)NTT circuit, a multiplication circuit, and a transpose circuit (L1, more details in the following sections). A simplified example of the NTT workflow for $N = 2^4$ is depicted in Figure 4. The input data undergoes the standard (i)NTT circuit in the first step, followed by processing through the multiplication and transpose circuits. Finally, the transposed data is fed back into the (i)NTT circuit for a second round, resulting in the final results. To accommodate dynamic changes in the value of N , we extend the (i)NTT circuit to support multiple entry and exit points. This allows data to be injected or fetched at any position within the circuit, enabling the pipeline to support (i)NTT with $N \leq 2^{16}$. One example is shown in the right part of Figure 4, we can use the 4-point NTT pipeline to support two parallel 2-point NTT computations. Therefore, our design enables the decomposition of the large (i)NTT circuit into multiple parallel smaller (i)NTT circuits, boosting the acceleration of workloads requiring a smaller NTT, e.g., shallow workload. As a result, FLASH-FHE’s bootstrappable computation clusters can support both deep FHE workloads and some parallel shallow workloads (further details of scheduling policy will be discussed in §4.2).

Figure 5 presents the physical design of the bootstrappable computation cluster. The left part demonstrates the above-introduced $iNTT$ pipeline. The right part is the BConv module, which performs l_{sub} parallel modular multiplications and sums their results to obtain the converted RNS basis. We choose $l_{sub} = 60$ to maximize the performance of BConv in the key-switching pipeline. The two parts are connected via a L1-transpose module, which will be introduced later in this section. To accommodate various modes, all modules within the bootstrappable cluster can be bypassed. For instance, when accelerating shallow workloads, the BConv module can be entirely bypassed since shallow FHE workloads do not require the key-switching operation which involves the $iNTT$

¹The four-step NTT algorithm has been comprehensively introduced in previous works [40, 41, 47], thus we omit the detailed algorithm in this paper.

$\rightarrow BConv \rightarrow NTT$ pipeline.

Swift Clusters: The swift computation clusters are dedicated to shallow workloads. Therefore, the BConv module is not allocated within the swift clusters. Instead, we design one (i)NTT pipeline with $N = 2^{14}$ in each swift cluster, as $N = 2^{14}$ is sufficient for shallow workloads at the 128-bit security level. Similar to the bootstrappable clusters, the (i)NTT pipeline within the swift clusters comprises a 2^7 -point standard (i)NTT circuit, a multiplication circuit, and a transpose circuit.

Cluster Affiliation: In the design of FLASH-FHE, one bootstrappable cluster and two swift clusters are organized as a cluster affiliation, and we allocate eight cluster affiliations in total. The choice of the one-to-two ratio is guided by both the *parameter considerations for the clusters* and *hardware design efficiency*. As previously mentioned, based on the analysis of representative workloads, the bootstrappable cluster is equipped with a 2^8 -point (i)NTT circuit, while the swift cluster contains a 2^7 -point (i)NTT circuit. This configuration allows for a straightforward and efficient connection of one 2^8 -point (i)NTT circuit with two 2^7 -point (i)NTT circuits using a transpose module (further details are provided in **Multi-level Transpose**). Additionally, the four 2^7 -point (i)NTT circuits within a cluster affiliation can effectively accelerate a single shallow FHE workload without requiring the involvement of other cluster affiliations.

It’s important to note that this ratio is not dictated by the proportion of deep to shallow FHE workloads in mixed workloads. Instead, FLASH-FHE utilizes the scheduler to handle dynamic workloads (more details in §4.2).

As a result, different from previous FHE accelerators that use all computation clusters to accelerate one FHE workloads [28–30, 40, 41], either shallow or deep, FLASH-FHE utilizes eight cluster affiliations to provide sufficient parallelism for shallow FHE workloads. Worth noting, the computation clusters within a single cluster affiliation share the valuable on-chip L1 cache. As a result, the increased parallelism achieved by adding more swift computation clusters does not come at the expense of significant resource consumption. We will cover more details of this part in §4.3.

On the contrary, when it comes to accelerating deep FHE workloads, FLASH-FHE does not employ the swift clusters. Instead, it utilizes all bootstrappable clusters across cluster affiliations to achieve optimal acceleration for bootstrapping operations. Due to the relatively minor computation resource consumption of the swift clusters, only $< 7\%$ hardware resources are not being utilized, as demonstrated in §6.2.

Other Modules: Similar to previous works [41], we have implemented real-time key generation and automorphism modules to respectively mitigate off-chip memory bandwidth bottlenecks and accelerate the rotation operations.

Multi-level Transpose: In this section, we present our approach of utilizing a multi-level transpose module to establish connectivity between computation clusters both within and

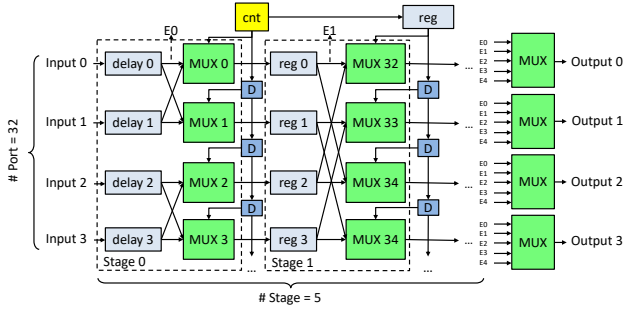


Figure 7: Unified building block of L1 transpose module.

compound operations, such as key-switching, which contains multiple basic operations, FLASH-FHE’s scheduler optimizes their workflows to increase the cache hit ratio. This optimization is achieved purely through hardware implementation. In the following section, we will delve into these details in depth.

The software driver provides standard APIs to integrate FLASH-FHE with upper-layer FHE libraries like Lattigo and SEAL. It takes key cryptographic parameters, such as N and L , as inputs. Similar to previous works [28–30, 40, 41], the FLASH-FHE driver generates static control instructions as output. Since FHE workloads are oblivious, meaning that the workflow of an FHE program is independent of its input data, all operations and their dependencies are known in advance. As a result, the FLASH-FHE software driver can generate instructions that pipeline computation modules in FLASH-FHE, mitigating most overhead caused by data read/write operations.

However, unlike previous approaches, FLASH-FHE’s instructions explicitly mark the entrances and exits of the corresponding computation cluster’s (i)NTT pipeline. The hardware controller follows these instructions to move input data to specific caches, perform computations, and output results to caches or external memory.

The goal of FLASH-FHE is to provide optimal scheduling policy for both shallow and deep FHE workloads. To achieve it, the scheduler takes the following steps:

1. FLASH-FHE analyzes the cryptographic parameters to determine whether the FHE workloads are deep or shallow.
2. For shallow workloads, the goal of the scheduler is to improve parallelism. Therefore, FLASH-FHE driver mainly generates instructions to execute parallel small-point (i)NTTs. No data exchange between cluster affiliations is scheduled.
3. For deep workloads, FLASH-FHE driver generates instructions of large-point (i)NTTs with BConv operations, which will be executed on all bootstrappable clusters to accelerate the bootstrapping operation as much as possible. We do not use swift computation clusters in deep workloads due to redundant data movement among multiple (i)NTT pipelines. Currently, we do not use swift computation clusters to execute these instructions, As we will show in §6.2, since the swift clusters only consume less than 7% chip

area, it does not cause a considerable hardware resource underutilization. How to efficient combine multiple swift clusters to execute large-point (i)NTTs is one of our future works.

Preemptive Scheduling: Our scheduler supports preemptive scheduling to efficiently manage mixed workloads. For example, if a deep FHE task is running on FLASH-FHE and shallow FHE tasks arrive, the deep task can be preempted, allowing the shallow tasks to run first and avoiding the convoy effect. This results in improved average task completion time. At the low level, we provide a priority-based preemptive mechanism, enabling users to assign different priorities to FHE tasks and simulate various scheduling policies, such as shortest-job-first. Preemption is implemented by inserting instructions to temporarily transfer data from SRAM to HBM, loading it back on-demand between tasks.

4.3 Shared Data Cache

In this section, we will begin by introducing the hierarchical caching structure implemented in FLASH-FHE. Then, we will delve into how we use algorithm-level knowledge to determine the appropriate cache volume.

Hierarchical Caching Structure: The hierarchical cache contains one shared L1 cache in each cluster affiliation and one global L2 cache shared by all affiliations. The L1 caches are designed to facilitate the pipelined execution of individual basic operations within each affiliation, such as NTT or BConv. In FLASH-FHE, we allocate 8MB SRAM as one L1 cache. In the case of deep workloads, where workloads are distributed among eight affiliations, the L1 cache in each affiliation is capable of supporting polynomial operations with a size of $N \leq 2^{16}/8$. Moreover, for shallow applications where the size of ciphertexts and keys is significantly smaller, the L1 cache remains sufficient to support one complete shallow workload.

The L2 cache is designed to store data when cluster affiliations need to cooperate with each other in the case of deep FHE workloads. The volume of L2 cache has a significant impact on the overall performance of deep FHE workloads. In our paper, we follow an algorithm-guided approach to decide the volume of L2 cache.

After thoroughly analyzing the memory space requirements of operations in deep workflows, we determine that key-switching (the version used in bootstrapping that requires an iNTT \rightarrow BConv \rightarrow NTT pipeline) is the most time-consuming operation, necessitating a significant amount of memory space to cache keys and intermediate data. The performance of key-switching with varying total cache volumes is depicted in Figure 8. We evaluate three common dnum settings, namely $dnum = 1, 2, 3$. Our observations indicate that for $dnum = 1$, a prevalent setting in most deep learning applications, the optimal performance is achieved

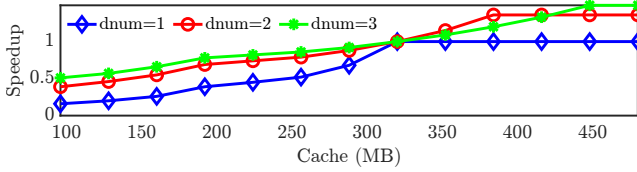


Figure 8: Performance of key-switching over varying cache volume.

when the total cache volume reaches 320MB. Allocating additional L2 cache beyond this point does not yield further performance improvements. Furthermore, 320MB serves as a reasonable setting for $dnum = 2, 3$, as it does not result in a significant degradation from optimal acceleration. In our paper, FLASH-FHE targets technology nodes of both 14/12nm and 7nm. Consequently, we have chosen to set the total cache volume to 320MB SRAM, which is a reasonable value for production. However, we suggest that when utilizing a more advanced technology node, such as 3nm, a larger cache could be allocated to further accelerate the key-switching operation for $dnum = 2, 3$.

The L2 caches are also utilized to store output data that may be reused by subsequent compound operations to eliminate slow off-chip I/O. We use instruction preprocess technology to achieve the goal.

5 Implementation

We implement FLASH-FHE in RTL and synthesize it with both an open-sourced 7nm predictive process design kit – ASAP7 [16] – and a commercial 14/12nm technology library. To achieve a balance between the power consumption and chip area, we choose 1.0 GHz to synthesize most of FLASH-FHE’s components, including FLASH-FHE’s swift and bootstrappable clusters, hardware controller, *etc.* The data cache run double pumped at 2.0 GHz, which enables using a single-ported SRAM while serving up to two accesses per clock cycle.

We use HBM2e [2] as the off-chip memory. We use two HBM2e PHYs and each can reach a bandwidth of 512 GB/s. Meanwhile, we use the reported data in previous works [30, 41] to estimate the area and power of the HBM2e memory.

For the software part, we modify the Lattigo library [4] and SEAL library [5] (for shallow workloads only) to integrate FLASH-FHE’s functionalities into it, including the software driver, *etc.* We also design command-line tools to debug and test FLASH-FHE. We leverage PCIe 5.0 interface for the communication between FLASH-FHE and the host since it can offer sufficient bandwidth [6].

6 Evaluation

In this section, we mainly evaluate FLASH-FHE to demonstrate its effectiveness towards mixed FHE workloads. We highlight the following evaluation results.

Components	7nm (mm ²)	14/12nm (mm ²)
128-point NTT	0.50	1.42
Modular Mul/Add	0.31	0.91
Total. Swift Clusters (16×NTT, 16×Mod. M/A)	12.96	37.28
256-point NTT	0.99	2.81
Modular Mul/Add	0.63	1.81
BConv	0.69	2.01
Total. Bootstrappable Clusters (8×NTT, 8×Mod. M/A, 60×BConv)	55.09	160.56
Key Generation	0.73	3.00
Automorphism	3.21	9.23
Transpose	0.13	0.37
SRAMs in Clusters	19.50	96.6
Hierarchical Cache	58.00	185.5
2×HBM2e	29.80	29.80
Total. FLASH-FHE	178.69	519.34

Table 3: Area analysis with both 7nm and 14/12nm technology nodes (Mod. M/A denotes modular multiplication and addition).

- FLASH-FHE achieves a comparable chip area with other FHE accelerators. The extra area of swift clusters consumes $< 7\%$ of the total area (§6.2).
- For deep FHE workloads, FLASH-FHE achieves an average of $1.4\times$ and $11.2\times$ performance gain than CraterLake and F1+, respectively (§6.3).
- For shallow FHE workloads, due to FLASH-FHE’s parallel scheduling policy, its performance can achieve up to $8.0\times$ of existing FHE accelerators (§6.3).

6.1 Methodology

Evaluation Method: Since we have not yet taped out FLASH-FHE and are unable to use an FPGA to implement the entire FLASH-FHE logic, similar to previous works [28–30, 40, 41], we evaluate FLASH-FHE using a cycle-accurate simulator.

Compared Schemes: We mainly compare FLASH-FHE with the following schemes:

- **F1+** [14/12nm]: F1+ is a upgraded version of F1 [40]. Similar to the scheme used in [41], F1+ is scaled to a 256 MB 32-bank scratchpad, 32 compute clusters with 256 lanes each, and 1 MB register file per cluster. However, F1+ suffers from an unoptimized version of key-switching, thus leading to suboptimal performance for deep FHE workloads. The area of F1+ is 636mm^2 .
- **CraterLake** [14/12nm]: CraterLake is the succeder of F1, which contain 8 256-lane computation group. To compensate the drawback of F1, CraterLake makes all its computing clusters to support bootstrapping by designing an efficient $i\text{NTT} \rightarrow \text{BConv} \rightarrow \text{NTT}$ pipeline for all of its engines. The chip area of CraterLake is 472mm^2 .

- **ARK** [7nm]: ARK leverages a novel algorithm-architecture co-design to accelerate bootstrapping. The chip area of ARK is 418mm².
- **SHARP** [7nm]: SHARP utilizes a short word length to reduce the required chip area. The chip area of SHARP is 179mm².
- **Other GPU/FPGA Solutions:** We primarily compare FLASH-FHE with one GPU-based accelerator, TensorFHE [20], and two FPGA-based accelerators, HEAX [39] and FAB [7].

Due to the unavailability of open-source implementations of the FHE accelerators, it is not possible to reproduce their reported performance results independently. Therefore, for the purpose of a fair comparison, we rely on the data provided in the original papers of CraterLake [41], ARK [29], and SHARP [28], similar to previous research works. By utilizing the data from these papers, we aim to establish a meaningful and consistent basis for comparison between FLASH-FHE and the aforementioned FHE accelerators.

FHE Applications: We mainly evaluate FLASH-FHE with three shallow and four deep FHE applications. Besides the Logistic Regression and LSTM applications mentioned in §3, we further evaluate the following applications:

- **LoLa-CIFAR with Unencrypted Weights:** LoLa is a low latency privacy-preserving neural network [11]. LoLa contains three different variants and all of them are shallow FHE workloads. The first variant is with the CIFAR-10 dataset [31]. The parameters of the neural network are not encrypted. We set $N = 2^{13}$ and $L = 7$.
- **LoLa-MNIST with Unencrypted Weights:** This is the second variant of LoLa with MNIST dataset [32]. The parameters of the neural network are not encrypted. We choose $N = 2^{13}$ and $L = 6$.
- **LoLa-MNIST with Encrypted Weights:** This is the third variant of LoLa with MNIST dataset, which further adopts encrypted weights for the neural network. We set $N = 2^{13}$ and $L = 6$ for this application.
- **Packed Bootstrapping:** Packed bootstrapping is a deep workload. Similar to [41], the packed bootstrapping takes a ciphertext with $N = 2^{16}$ and $L = 3$ as the initial parameters, and then exhausts its multiplicative level by bringing L to 57. Afterwards, it performs bootstrapping to refresh the multiplication level of the ciphertext. In packed bootstrapping, different from unpacked bootstrapping, the ciphertext uses all available slots, *i.e.*, $N/2$. The packed bootstrapping is also the bootstrapping algorithm in other deep workloads. We set $N = 2^{16}$ and $L = 57$.
- **ResNet-20:** It’s a deep FHE workload performing neural network inference with ResNet-20 [25], whose parameters are encrypted via CKKS. We mainly refer the implementation mentioned in [33]. We set $N = 2^{16}$ and $L = 41$.

For LSTM, we set $N = 2^{16}$ and $L = 13$. For the Logistic Regression evaluation, we will utilize batch sizes of both 256 and 1024, with a feature size of 256. We set $N = 2^{13}$ and $L = 33$. These settings are chosen to align with the evaluation parameters used in CraterLake and ARK.

6.2 Area Analysis

In this section, we present the area analysis for FLASH-FHE. Following the approach of previous works such as BTS [30], SHARP [28], and ARK [29], we employ FinCACTI [43] to model the SRAMs and caches utilized in FLASH-FHE. The results of the analysis are summarized in Table 3.

With the 14/12nm technology node, FLASH-FHE occupies a total area of 519.34mm², which is comparable to that of CraterLake [41]. Notably, FLASH-FHE incorporates 16 swift engines in its design, significantly enhancing the performance of shallow FHE workloads, as demonstrated in §6.3.

Furthermore, the large performance gain is achieved only with the addition of a relatively small portion of the chip area (the logic circuits of swift clusters occupy less than 7% of the total area) since swift clusters share the memory resources of bootstrappable clusters.

As for other components, the logic circuits of FLASH-FHE’s bootstrappable computation clusters account for 31.1% of the area, while the total cache occupies 55.6%. Since FLASH-FHE allots 320MB of cache to enhance the performance of key-switching operation (refer §4.3 for more details), the SRAMs consume slightly more chip area compared to other FHE accelerators. For the 7nm technology node, the total chip area of FLASH-FHE is 178.69mm², which aligns with the estimated scaling from 14/12nm to 7nm area (approximately 2.9×) [35].

6.3 Performance

In this section, we compare the performance of FLASH-FHE with other typical FHE accelerators. First, we compare FLASH-FHE synthesized with the 14/12nm technology node against other FHE accelerators using the same technology, namely CraterLake and F1+. Subsequently, we compare FLASH-FHE implemented with the 7nm technology node with ARK and SHARP, which is also based on the 7nm technology. Additionally, we compare FLASH-FHE with an AMD Ryzen Threadripper PRO 3975WX CPU, featuring 32 cores and 64 threads.

FLASH-FHE and CraterLake employ 128-bit security for all deep FHE workloads and 80-bit security for shallow FHE workloads, which we consider practical for production environments. F1+ employs 80-bit security for all seven workloads. ARK and SHARP use 128-bit security for their two deep FHE workloads.

Single Workload: We first evaluate all accelerators with one single FHE workload at a time and the performance of FLASH-FHE at the 14/12nm technology node is presented

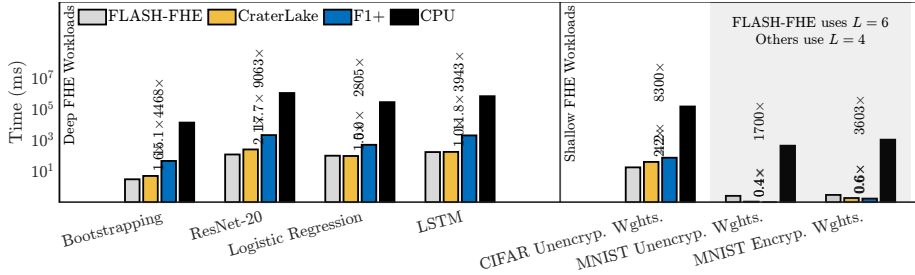


Figure 9: [14/12nm Technology Node] Performance of FLASH-FHE with four deep and three shallow FHE applications.

in Figure 9. Here, we highlight the following results. First, across all four deep workloads, FLASH-FHE can achieve an average (geometric mean) speedup of 1.4× and 11.2× compared to CraterLake and F1+, respectively. The performance improvement over CraterLake is mainly due to the adequate cache (320MB in FLASH-FHE v.s. 256MB in CraterLake), which can boost the performance of key-switching operation (as shown in §4.3).

Second, for the three shallow FHE workloads, FLASH-FHE can achieve a speedup of 0.4–2.2× and 0.4–4.2× compared to CraterLake and F1, respectively. For the specific scenarios of LoLa-MNIST with and without Unencrypted Weights, it is worth noting that FLASH-FHE employs a value of $L = 6$, while F1 and CraterLake utilize $L = 4$ since we encounter difficulties in using $L = 4$ to launch the two applications. Consequently, it is important to acknowledge that FLASH-FHE’s performance in this context is inferior to that of CraterLake and F1+. However, in the subsequent sections, we will show that even with a larger L setting, FLASH-FHE can achieve significantly higher performance when there are multiple shallow FHE workloads.

Next, we compare FLASH-FHE implemented with the 7nm technology node against ARK and SHARP. Since ARK and SHARP do not provide performance results specifically for shallow FHE workloads, we focus our comparison on two typical deep workloads: ResNet-20 and Logistic Regression. Since some accelerators with the 7nm technology utilize dramatic chip area, we also show the metric of performance per area in this part.

The evaluation results are presented in Figure 10, and we highlight the following points. Compared to ARK, FLASH-FHE achieves 42.3% better performance in Logistic Regression but 21.6 worse performance in ResNet-20. However, FLASH-FHE achieves consistency higher performance per area, from 1.49× to 1.78×. The reason is that although ARK adopts an algorithm hardware co-design method to optimize the performance, it results in a dramatically larger chip area, leading to much worse performance per chip area. SHARP achieves better absolute performance and performance per chip area than FLASH-FHE since it adopts a short-word size optimization method, which is not utilized in the current implementation of FLASH-FHE. Worth noting, FLASH-FHE’s idea of leveraging a heterogeneous architecture

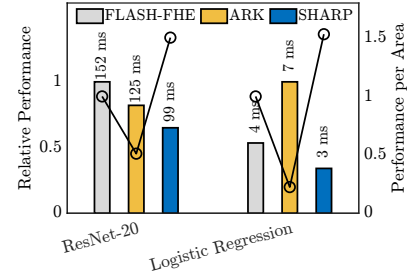


Figure 10: [7nm Technology Node] Performance of FLASH-FHE.

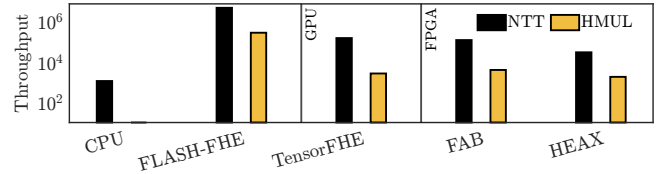


Figure 11: Performance of FLASH-FHE compared to representative GPU and FPGA solutions.

is *orthogonal* to the optimizations used in ARK and SHARP. Therefore, we can combine FLASH-FHE with them to further improve their performance for mixed FHE workloads.

Multiple Shallow Workloads: In this part, we evaluate FLASH-FHE’s performance (implemented with 14/12nm technology node) when there are multiple shallow workloads. We vary the number of shallow workloads from one to ten and measure the average execution time. The results are shown in Figure 12. We observe when there are more than three shallow workloads, FLASH-FHE can achieve a much better average execution time than CraterLake even when FLASH-FHE uses $L = 6$. The reason is that FLASH-FHE can execute multiple shallow workloads in parallel while CraterLake can only use a sequential scheduling policy due to its homogeneous design. FLASH-FHE can achieve up to 8.0× speedup when all of its cluster affiliations are utilized.

Comparison with GPU/FPGA: In this section, we highlight FLASH-FHE’s advantages over GPU/FPGA solutions for shallow workloads. Due to the lack of data on shallow workloads in previous works, we instead compare the performance of NTT and Homomorphic Multiplication (HMUL), which are the primary computations in shallow workloads and account for the majority of execution time. For a fair comparison, the evaluation is conducted using the same shallow parameters as in previous works (*i.e.*, $N = 2^{14}$, $\log PQ = 438$). As shown in Figure 11 (note the Y-axis is in log scale), FLASH-FHE achieves more than 30× throughput (measured in number of executed operations) compared to the state-of-the-art GPU-based solution, TensorFHE [20], and FPGA-based designs, FAB [7] and HEAX [39], for NTT computation. For HMUL operations, FLASH-FHE provides a 60–100× acceleration compared to the FPGA and GPU implementations. These results demonstrate the significant performance improvement of FLASH-FHE over existing GPU/FPGA solutions for shallow

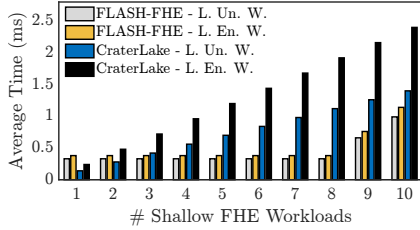


Figure 12: [14/12nm Technology Node] Performance of FLASH-FHE with parallel shallow FHE workloads.

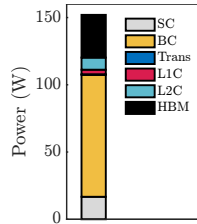


Figure 13: Power consumption breakdown.

workloads.

6.4 Power Consumption

We also conducted an evaluation of the power consumption of FLASH-FHE using the 14/12nm technology node, as we currently lack access to the power budget for certain logic modules in the 7nm technology. The total power consumption of FLASH-FHE amounts to 152.11W, which is superior to CraterLake (317W) and ARK (281.3W), and comparable to BTS (163.2W). Figure 13 provides a breakdown of FLASH-FHE’s power consumption and shows the power consumption of the bootstrappable clusters (BC), swift clusters (SC), transpose unit (Trans), L1 cache (L1C), L2 cache (L2C) and HBM. The results demonstrate that the bootstrappable computation cluster accounts for the majority of power consumption, reaching up to 60.0%, while the swift clusters consume only 11.0% of the total power. Therefore, our results further indicate that the improved performance resulting from adding more swift clusters does not significantly increase power consumption.

7 Discussion

Integration with Other FHE Accelerators: As discussed, it has become evident that several FHE accelerators employ dedicated optimization methods to achieve further performance gain [28, 29]. One such example is SHARP [28], which utilizes a short word length to minimize chip area, albeit with a slight accuracy trade-off. In our paper, we propose FLASH-FHE as an alternative approach to achieve high performance, employing a heterogeneous architecture tailored for mixed FHE workloads. Since FLASH-FHE’s idea is orthogonal to SHARP and ARK, we believe that integrating FLASH-FHE with these existing FHE accelerators can yield superior performance outcomes.

Combining Multiple Swift Clusters for Deep Workloads: In the current design of FLASH-FHE, we have focused on decomposing a bootstrappable computation cluster to support multiple shallow FHE workloads. However, we acknowledge that our system does not currently provide support for combining multiple swift computation clusters to accelerate a

single deep FHE workload. This limitation stems from the challenges associated with complex computation scheduling among multiple (i)NTT pipelines. Such scheduling inherently involves massive data movement, which can potentially degrade performance. Addressing this issue has been left as a future research direction for FLASH-FHE.

8 Related Works

Besides the related works discussed in §2.3, we further cover the following two more directions in this section.

Other Optimizations for FHE: In addition to leveraging accelerators for enhanced performance in fully homomorphic encryption (FHE), there is an alternative approach focused on algorithm-level optimization. While the accelerators discussed in §2.3 primarily target second-generation FHE schemes such as BFV [19], BGV [10], and CKKS [14], third-generation FHE schemes based on the GSW scheme [23] have been proposed, such as TFHE [15]. These schemes use exceptionally fast bootstrapping, often completing in less than 0.1 seconds. However, they come with the drawback of incompatibilities with batching, introducing new trade-offs that limit their usability. Strix introduces the first FHE accelerator for TFHE [37], where it designs a two-level ciphertext batching method with programmable bootstrapping. Another direction of exploration involves optimizing software implementations. Researchers have developed efficient software libraries, including SEAL [5] and Lattigo [4], or compilers such as EVA [18], to improve FHE performance.

Partial Homomorphic Encryption Acceleration: Several existing works focus on enhancing the performance of partial homomorphic encryption (PHE) schemes, especially the Paillier scheme. Notably, FLASH employs FPGA technology to expedite PHE operations specifically for federated learning [46]. Similarly, HAFLO utilizes GPUs to accomplish a comparable objective [13]. Additionally, Shi *et al.* have developed a dedicated 28nm ASIC tailored for accelerating PHE processes [44]. Since the basic building blocks of the Paillier scheme are modular multiplications and exponentiations, the design choices of these PHE accelerators are very different from FHE accelerators, which are built on polynomial computations.

9 Conclusion

This paper proposed FLASH-FHE, the first heterogeneous acceleration architecture for FHE. We have provided a full RTL implementation of FLASH-FHE and synthesized it using two representative technology nodes. Experiments with three shallow and four deep FHE workloads show that FLASH-FHE can consistently achieve high performance for mixed FHE workloads.

References

- [1] Country Lookup Example. https://github.com/homenc/HElib/tree/master/examples/BGV_country_db_lookup. Accessed: 2023-05-23.
- [2] High-Bandwidth Memory (HBM) Delivers Impressive Performance Gains. <https://www.networkworld.com/article/3664088/high-bandwidth-memory-hbm-delivers-impressive-performance-gains.html>. Accessed: 2023-03-26.
- [3] Intel Homomorphic Encryption (HE) Acceleration Library for FPGAs. <https://github.com/intel/hexl-fpga>. Accessed: 2022-07-08.
- [4] Lattigo. <https://github.com/tuneinsight/lattigo>. Accessed: 2023-05-10.
- [5] SEAL. <https://github.com/microsoft/SEAL>. Accessed: 2023-05-13.
- [6] Synopsys IP for PCI Express (PCIe) 5.0. <https://www.synopsys.com/designware-ip/interface-ip/pci-express/pci-express-5.html>. Accessed: 2023-07-11.
- [7] Rashmi Agrawal, Leo de Castro, Guowei Yang, Chirraag Juvekar, Rabia Tugce Yazicigil, Anantha P. Chandrakasan, Vinod Vaikuntanathan, and Ajay Joshi. FAB: an fpga-based accelerator for bootstrappable fully homomorphic encryption. In *The 29th IEEE International Symposium on High-Performance Computer Architecture, HPCA 2023, Montreal, QC, Canada, February 25 - March 1, 2022*. IEEE, 2023.
- [8] Sebastian Angel, Hao Chen, Kim Laine, and Srinath T. V. Setty. PIR with compressed queries and amortized query processing. In *2018 IEEE Symposium on Security and Privacy, SP 2018, Proceedings, 21-23 May 2018, San Francisco, California, USA*, pages 962–979. IEEE Computer Society, 2018.
- [9] Fabian Boemer, Sejun Kim, Gelila Seifu, Fillipe D. M. de Souza, and Vinodh Gopal. Intel HEXL: accelerating homomorphic encryption with intel AVX512-IFMA52. In *WAHC '21: Proceedings of the 9th on Workshop on Encrypted Computing & Applied Homomorphic Cryptography, Virtual Event, Korea, 15 November 2021*, pages 57–62. WAHC@ACM, 2021.
- [10] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. Fully homomorphic encryption without bootstrapping. *Electron. Colloquium Comput. Complex.*, TR11-111, 2011.
- [11] Alon Brutzkus, Ran Gilad-Bachrach, and Oren Elisha. Low latency privacy preserving inference. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*, volume 97 of *Proceedings of Machine Learning Research*, pages 812–821. PMLR, 2019.
- [12] Melissa Chase, Hao Chen, Jintai Ding, Shafi Goldwasser, Sergey Gorbunov, Jeffrey Hoffstein, Kristin Lauter, Satya Lokam, Dustin Moody, Travis Morrison, et al. Security of homomorphic encryption. *HomomorphicEncryption.org, Redmond WA, Tech. Rep*, 2017.
- [13] Xiaodian Cheng, Wanhang Lu, Xinyang Huang, Shuihai Hu, and Kai Chen. HAFLO: gpu-based acceleration for federated logistic regression. *CoRR*, abs/2107.13797, 2021.
- [14] Jung Hee Cheon, Andrey Kim, Miran Kim, and Yong Soo Song. Homomorphic encryption for arithmetic of approximate numbers. In Tsuyoshi Takagi and Thomas Peyrin, editors, *Advances in Cryptology - ASIACRYPT 2017 - 23rd International Conference on the Theory and Applications of Cryptology and Information Security, Hong Kong, China, December 3-7, 2017, Proceedings, Part I*, volume 10624 of *Lecture Notes in Computer Science*, pages 409–437. Springer, 2017.
- [15] Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachène. TFHE: fast fully homomorphic encryption over the torus. *J. Cryptol.*, 33(1):34–91, 2020.
- [16] Lawrence T. Clark, Vinay Vashishtha, Lucian Shifren, Aditya Gujja, Saurabh Sinha, Brian Cline, Chandrasekaran Ramamurthy, and Greg Yeric. ASAP7: A 7-nm finfet predictive process design kit. *Microelectron. J.*, 53:105–115, 2016.
- [17] James W Cooley and John W Tukey. An algorithm for the machine calculation of complex fourier series. *Mathematics of computation*, 19(90):297–301, 1965.
- [18] Roshan Dathathri, Blagovesta Kostova, Olli Saarikivi, Wei Dai, Kim Laine, and Madan Musuvathi. EVA: an encrypted vector arithmetic language and compiler for efficient homomorphic computation. In Alastair F. Donaldson and Emina Torlak, editors, *Proceedings of the 41st ACM SIGPLAN International Conference on Programming Language Design and Implementation, PLDI 2020, London, UK, June 15-20, 2020*, pages 546–561. ACM, 2020.
- [19] Junfeng Fan and Frederik Vercauteren. Somewhat practical fully homomorphic encryption. *IACR Cryptol. ePrint Arch.*, page 144, 2012.

- [20] Shengyu Fan, Zhiwei Wang, Weizhi Xu, Rui Hou, Dan Meng, and Mingzhe Zhang. Tensorfhe: Achieving practical computation on encrypted data using GPGPU. *CoRR*, abs/2212.14191, 2022.
- [21] Stephane Foldes. Symmetries of directed graphs and the chinese remainder theorem. *J. Comb. Theory, Ser. B*, 28(1):18–25, 1980.
- [22] W. Morven Gentleman and G. Sande. Fast fourier transforms: for fun and profit. In *American Federation of Information Processing Societies: Proceedings of the AFIPS '66 Fall Joint Computer Conference, November 7-10, 1966, San Francisco, California, USA*, volume 29 of *AFIPS Conference Proceedings*, pages 563–578. AFIPS / ACM / Spartan Books, Washington D.C., 1966.
- [23] Craig Gentry, Amit Sahai, and Brent Waters. Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based. In Ran Canetti and Juan A. Garay, editors, *Advances in Cryptology - CRYPTO 2013 - 33rd Annual Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2013. Proceedings, Part I*, volume 8042 of *Lecture Notes in Computer Science*, pages 75–92. Springer, 2013.
- [24] Kyoohyung Han, Seungwan Hong, Jung Hee Cheon, and Daejun Park. Logistic regression on homomorphic encrypted data at scale. In *The Thirty-Third AAAI Conference on Artificial Intelligence, AAAI 2019, The Thirty-First Innovative Applications of Artificial Intelligence Conference, IAAI 2019, The Ninth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2019, Honolulu, Hawaii, USA, January 27 - February 1, 2019*, pages 9466–9471. AAAI Press, 2019.
- [25] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*, pages 770–778. IEEE Computer Society, 2016.
- [26] Peizhao Hu, Asma Aloufi, Adam Caulfield, Kim Laine, and Kristin Lauter. Sparkfhe: Distributed dataflow framework with fully homomorphic encryption. In *Privacy-preserving Machine Learning Workshop (PPML-PriML), co-located with NeurIPS*, 2020.
- [27] Wonkyung Jung, Sangpyo Kim, Jung Ho Ahn, Jung Hee Cheon, and Younho Lee. Over 100x faster bootstrapping in fully homomorphic encryption through memory-centric optimization with gpus. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2021(4):114–148, 2021.
- [28] Jongmin Kim, Sangpyo Kim, Jaewan Choi, Jaiyoung Park, Donghwan Kim, and Jung Ho Ahn. SHARP: a short-word hierarchical accelerator for robust and practical fully homomorphic encryption. In *ISCA '23: The 50th Annual International Symposium on Computer Architecture, Orlando, FL, USA, June 17 – 21, 2023*. ACM, 2023.
- [29] Jongmin Kim, Gwangho Lee, Sangpyo Kim, Gina Sohn, Minsoo Rhu, John Kim, and Jung Ho Ahn. ARK: fully homomorphic encryption accelerator with runtime data generation and inter-operation key reuse. In *55th IEEE/ACM International Symposium on Microarchitecture, MICRO 2022, Chicago, IL, USA, October 1-5, 2022*, pages 1237–1254. IEEE, 2022.
- [30] Sangpyo Kim, Jongmin Kim, Michael Jaemin Kim, Wonkyung Jung, John Kim, Minsoo Rhu, and Jung Ho Ahn. BTS: an accelerator for bootstrappable fully homomorphic encryption. In Valentina Salapura, Mohamed Zahran, Fred Chong, and Lingjia Tang, editors, *ISCA '22: The 49th Annual International Symposium on Computer Architecture, New York, New York, USA, June 18 - 22, 2022*, pages 711–725. ACM, 2022.
- [31] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009.
- [32] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proc. IEEE*, 86(11):2278–2324, 1998.
- [33] Joon-Woo Lee, HyungChul Kang, Yongwoo Lee, Woosuk Choi, Jieun Eom, Maxim Deryabin, Eunsang Lee, Junghyun Lee, Donghoon Yoo, Young-Sik Kim, and Jong-Seon No. Privacy-preserving machine learning with fully homomorphic encryption for deep neural network. *IEEE Access*, 10:30039–30054, 2022.
- [34] Karthik Nandakumar, Nalini K. Ratha, Sharath Pankanti, and Shai Halevi. Towards deep neural network training on encrypted data. In *IEEE Conference on Computer Vision and Pattern Recognition Workshops, CVPR Workshops 2019, Long Beach, CA, USA, June 16-20, 2019*, pages 40–48. Computer Vision Foundation / IEEE, 2019.
- [35] S. Narasimha, B. Jagannathan, A. Ogino, D. Jaeger, B. Greene, C. Sheraw, K. Zhao, B. Haran, U. Kwon, A. K. M. Mahalingam, B. Kannan, B. Morganfeld, J. Dechene, C. Radens, A. Tessier, A. Hassan, H. Narisetty, I. Ahsan, M. Aminpur, C. An, M. Aquilino, A. Arya, R. Augur, N. Baliga, R. Bhelkar, G. Biery, A. Blauberg, N. Borjemscaia, A. Bryant, L. Cao, V. Chauhan, M. Chen, L. Cheng, J. Choo, C. Christiansen, T. Chu, B. Cohen, R. Coleman, D. Conklin, S. Crown, A. da Silva, D. Dechene, G. Derderian, S. Deshpande, G. Dillway, K. Donegan, M. Eller, Y. Fan, Q. Fang, A. Gassaria, R. Gauthier, S. Ghosh,

- G. Gifford, T. Gordon, M. Gribelyuk, G. Han, J.H. Han, K. Han, M. Hasan, J. Higman, J. Holt, L. Hu, L. Huang, C. Huang, T. Hung, Y. Jin, J. Johnson, S. Johnson, V. Joshi, M. Joshi, P. Justison, S. Kalaga, T. Kim, W. Kim, R. Krishnan, B. Krishnan, K. Anil, M. Kumar, J. Lee, R. Lee, J. Lemon, S.L. Liew, P. Lindo, M. Lingalugari, M. Lipinski, P. Liu, J. Liu, S. Lucarini, W. Ma, E. Maciejewski, S. Madiseti, A. Malinowski, J. Mehta, C. Meng, S. Mitra, C. Montgomery, H. Nayfeh, T. Nigam, G. Northrop, K. Onishi, C. Ordoño, M. Ozbek, R. Pal, S. Parihar, O. Patterson, E. Ramanathan, I. Ramirez, R. Ranjan, J. Sarad, V. Sardesai, S. Saudari, C. Schiller, B. Senapati, C. Serrau, N. Shah, T. Shen, H. Sheng, J. Shepard, Y. Shi, M.C. Silvestre, D. Singh, Z. Song, J. Sporre, P. Srinivasan, Z. Sun, A. Sutton, R. Sweeney, K. Tabakman, M. Tan, X. Wang, E. Woodard, G. Xu, D. Xu, T. Xuan, Y. Yan, J. Yang, K.B. Yeap, M. Yu, A. Zainuddin, J. Zeng, K. Zhang, M. Zhao, Y. Zhong, R. Carter, C.-H. Lin, S. Grunow, C. Child, M. Lagus, R. Fox, E. Kaste, G. Gomba, S. Samavedam, P. Agnello, and D. K. Sohn. A 7nm cmos technology platform for mobile and high performance compute application. In *2017 IEEE International Electron Devices Meeting (IEDM)*, pages 29.5.1–29.5.4, 2017.
- [36] Robert Podschwadt and Daniel Takabi. Classification of encrypted word embeddings using recurrent neural networks. In Oluwaseyi Feyisetan, Sepideh Ghana-vati, Oleg Rokhlenko, and Patricia Thaine, editors, *Proceedings of the PrivateNLP 2020: Workshop on Privacy in Natural Language Processing - Colocated with WSDM 2020, Houston, USA, Feb 7, 2020*, volume 2573 of *CEUR Workshop Proceedings*, pages 27–31. CEUR-WS.org, 2020.
- [37] 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. *CoRR*, abs/2305.11423, 2023.
- [38] Deevashwer Rathee, Thomas Schneider, and K. K. Shukla. Improved multiplication triple generation over rings via rlwe-based AHE. In Yi Mu, Robert H. Deng, and Xinyi Huang, editors, *Cryptology and Network Security - 18th International Conference, CANS 2019, Fuzhou, China, October 25-27, 2019, Proceedings*, volume 11829 of *Lecture Notes in Computer Science*, pages 347–359. Springer, 2019.
- [39] M. Sadegh Riazi, Kim Laine, Blake Pelton, and Wei Dai. HEAX: an architecture for computing on encrypted data. In James R. Larus, Luis Ceze, and Karin Strauss, editors, *ASPLOS '20: Architectural Support for Programming Languages and Operating Systems, Lausanne, Switzerland, March 16-20, 2020*, pages 1295–1309. ACM, 2020.
- [40] Nikola Samardzic, Axel Feldmann, Aleksandar Krastev, Srinivas Devadas, Ronald G. Dreslinski, Christopher Peikert, and Daniel 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*, pages 238–252. ACM, 2021.
- [41] Nikola Samardzic, Axel Feldmann, Aleksandar Krastev, Nathan Manohar, Nicholas Genise, Srinivas Devadas, Karim Eldefrawy, Chris Peikert, and Daniel Sánchez. Craterlake: a hardware accelerator for efficient unbounded computation on encrypted data. In Valentina Salapura, Mohamed Zahran, Fred Chong, and Lingjia Tang, editors, *ISCA '22: The 49th Annual International Symposium on Computer Architecture, New York, New York, USA, June 18 - 22, 2022*, pages 173–187. ACM, 2022.
- [42] Sinem Sav, Apostolos Pyrgelis, Juan Ramón Troncoso-Pastoriza, David Froelicher, Jean-Philippe Bossuat, Joao Sa Sousa, and Jean-Pierre Hubaux. POSEIDON: privacy-preserving federated neural network learning. In *28th Annual Network and Distributed System Security Symposium, NDSS 2021, virtually, February 21-25, 2021*. The Internet Society, 2021.
- [43] Alireza Shafaei, Yanzhi Wang, Xue Lin, and Massoud Pedram. Fincacti: Architectural analysis and modeling of caches with deeply-scaled finfet devices. In *IEEE Computer Society Annual Symposium on VLSI, ISVLSI 2014, Tampa, FL, USA, July 9-11, 2014*, pages 290–295. IEEE Computer Society, 2014.
- [44] Guiming Shi, Zhanhong Tan, Dapeng Cao, Jingwei Cai, Wuke Zhang, Yifu Wu, and Kaisheng Ma. A 28nm 68mops 0.18mJ/Op paillier homomorphic encryption processor with bit-serial sparse ciphertext computing. In *IEEE International Solid- State Circuits Conference, ISSCC 2023, San Francisco, CA, USA, February 19-23, 2023*, pages 242–243. IEEE, 2023.
- [45] Han Tian, Chaoliang Zeng, Zhenghang Ren, Di Chai, Junxue Zhang, Kai Chen, and Qiang Yang. Sphinx: Enabling privacy-preserving online learning over the cloud. In *43rd IEEE Symposium on Security and Privacy, SP 2022, San Francisco, CA, USA, May 22-26, 2022*, pages 2487–2501. IEEE, 2022.
- [46] Junxue Zhang, Xiaodian Cheng, Wei Wang, Liu Yang, Jinbin Hu, and Kai Chen. FLASH: towards a high-performance hardware acceleration architecture for cross-silo federated learning. In *NSDI '23: 20th*

USENIX Symposium on Networked Systems Design and Implementation, BOSTON, MA, USA, April 17–19, 2023.
USENIX.

- [47] Junxue Zhang, Xiaodian Cheng, Liu Yang, Jinbin Hu, Ximeng Liu, and Kai Chen. Sok: Fully homomorphic encryption accelerators. *ACM Comput. Surv.*, 56(12):316:1–316:32, 2024.