SAFE: A Scalable Homomorphic Encryption Accelerator for Vertical Federated Learning

Zhaohui Chen[1,2,3], Zhen Gu[1,2,4], Yanheng Lu[1,2], Xuanle Ren[1], Ruiguang Zhong[1], Wen-jie Lu[5], Jiansong Zhang[1], Yichi Zhang[1], Hanghang Wu[5], Xiaofu Zheng[5], Heng Liu[1], Tingqiang Chu[5], Cheng Hong[5], Changzheng Wei[5], Dimin Niu[1,2] and Yuan Xie[1,2], *Fellow, IEEE,*

[1] DAMO Academy, Alibaba Group, China {chenzhaohui.czh, guzhen.gz, yanheng.lyh, xuanle.rxl, ruiguang.zrg, jiansong.zjs, yichi.zyc, hengliu.lh, dimin.niu, y.xie}@alibaba-inc.com [2] Hupan Lab, China
[3] Peking University, China [4] Tsinghua University, China [5] Ant Group, China {juhou.lwj, hanghang.whh, linke.zxf, chutingqiang.ctq, vince.hc, changzheng.wcz}@antgroup.com

***Abstract*—Privacy preservation has become a critical concern for governments, hospitals, and large corporations. Homomorphic encryption (HE) enables a ciphertext-based computation paradigm with strong security guarantees. In emerging cross-agency data cooperation scenarios like vertical federated learning (VFL), HE protects the data interaction from exposure to counterparts. However, computation on ciphertext has significant performance challenges due to increased data size and substantial overhead. Related work has been proposed to accelerate HE using parallel hardware, such as GPUs, FPGAs, and ASICs. However, many existing hardware accelerators target specific HE operations, such as number theoretic transform and key switching, providing limited performance improvement for end-to-end applications. Others support bootstrapping, which requires quite a large ASIC design. To better support existing VFL training applications, we propose SAFE, an HE accelerator for scalable homomorphic matrix-vector products (HMVP), which is the performance bottleneck. SAFE adopts coefficient-wise encoded HMVP algorithm, despite a vanilla mode, we further explore the compressed and concatenated modes, which can fully utilize the polynomial encoding slots. The proposed hardware architecture, customized for HMVP dataflow, supports spatial and temporal parallelization of function units. The most costly polynomial function, number theoretic transform, is implemented with a low-area constant geometry unit which improve efficiency by 2.43×. SAFE is implemented as a CPU-FPGA heterogeneous acceleration system, unleashing the multithread potential. The evaluation demonstrates an up to 36× speed-up in end-to-end federated logistic regression training.**

***Index Terms*—Homomorphic encryption, hardware acceleration, privacy-preserving computing, number theoretic transform, federated learning.**

I. Introduction

In the big-data era, information is increasingly used as the driving force behind many applications. However, sharing plaintext data can be challenging and prohibited in certain fields, such as medical systems, banks, and governments [1]. Privacy-preserving computing aims to overcome this restriction by sharing only encrypted data, thus avoiding the exposure of sensitive information [2], [3]. In the agency-to-agency vertical federated learning (VFL) scenarios, the two parties have the same user community, while they reserve different features, and only one of them knows the label. They try to update the model with loss functions during the training steps without exposing user data. To hide the plaintext in the intermediate steps, homomorphic encryption (HE) is used for exchanging information between agencies [4]–[6].

HE has the capability of computing on encrypted data, which was first proposed by Rivest *et al.* in 1978, for achieving "privacy homomorphism" [7]. Since then, cryptographers have constructed various HE schemes to improve functionality and efficiency [8]. Among these schemes, Brakerski/Fan-Vercauteren (BFV)-like algorithms, including BFV [9], [10], Brakerski-Gentry-Vaikuntanathan (BGV) [11], and Cheon-Kim-Kim-Song (CKKS) [12]–[14] are state-of-the-art, which have efficiency advantages on encryption, decryption and arithmetic features. These HE schemes are based on the learning-with-errors (LWE) problem and its ring-based variant (RLWE). Compared to other schemes, such as the Paillier [15] and the El-Gamal [16] cryptosystems, BFV-like schemes can execute wide single-instruction multi-data (SIMD) operations, which amortize the overhead by the factor of polynomial dimension N. Regarding arithmetic functions, operations like multiplication, addition, and rotation are mapped to manipulations of LWE or RLWE ciphertexts [17].

Performing arithmetic operations on encrypted data is more complicated than working with unencrypted data. HE typically introduces over $10^4\times$ overhead on the following aspects [18]. Firstly, besides the straightforward arithmetic, HE operations introduce many extra key switching operations to keep data format and modulus switching operations to control error [19], [20]. Secondly, the polynomial ring structure of RLWE introduces constraints, which require compact encoding methods to utilize the coefficient slots and use technologies like number theoretic transform (NTT) to reduce the overhead [21]–[23]. Thirdly, the basic integer arithmetic is transformed into modular operations in the ciphertext domain, which always costs many clock cycles on CPU arithmetic units [24].

The performance of fully HE (FHE) becomes even worse, which supports computation with arbitrary depth. More precisely, upon encryption, a ciphertext is associated with a noise budget that will be consumed during computation. This noise budget should not be exhausted to ensure decryption correctness, meaning that only limited-depth computation is allowed. Bootstrapping overcomes this limitation by refreshing

the noise budget before it is about to exhaust [8]. However, bootstrapping introduces a huge amount of computation overhead, which usually dominates the whole computation [25], [26].

Driven by the acceleration demands, several high-performance hardware implementations, *e.g.*, FPGAs, GPUs, and ASICs, have been proposed in recent years. The FPGA-based work targets HE operators, such as polynomial multiplication and NTT [27]–[34]. The Microsoft HEAX accelerates key switching by implementing a pipelined architecture on FPGAs [27]. However, accelerating key switching solely provides limited end-to-end improvement from Amdahl's law. GPUs demonstrate hundreds of speed-up compared to CPUs [35]–[40]. The bottleneck of GPU, however, resides in limited shared memory that cannot accommodate large polynomials and the intermediate results during HE evaluation. ASICs achieve a tremendous performance improvement, benefiting from the use of the high-frequency clock, high-bandwidth memory, and dense compute logic [41]–[46]. However, hundreds of megabytes of SRAM and massive logic units are required to support the bootstrapping of FHE. Although some FHE accelerators have about $10^4\times$ speedup, the area of these ASICs is also quite large, typically $100\,\mathrm{mm}^2$ to $400\,\mathrm{mm}^2$, which incurs highly high cost.

The logistic regression (LR) model is widely used for the VFL training task thanks to its accuracy and strong interpretability. Compared to implementing the full process with FHE, a more practical solution is to use HE in an interactive manner [4] or combine HE with other techniques such as secret sharing [5]. In both the above two solutions, the major workload is the homomorphic matrix-vector product (HMVP), where the matrix is the local plaintext feature, while the vector is the interaction ciphertext. Since the feature size varies in different training data settings, vanilla encoding mode can not efficiently utilize the encoding space. This paper starts from the work published at DAC 2023 [47] by incorporating a scalable HMVP algorithm and datapath. We further evaluate its efficiency in various settings through ablation studies.

In this work, we propose SAFE, a scalable homomorphic encryption accelerator for vertical federated learning. We observe both algorithm optimization and hardware acceleration are crucial in end-to-end applications. Regarding the algorithm, we designed a novel algorithm of HMVP, SAFE supports different types of ciphertexts (*i.e.*, RLWE and LWE) and the conversion between them. With this optimization, the complexity of homomorphic MVP is reduced from $\mathcal{O}(m\log_2 n)$ to $\mathcal{O}(m)$ where m and n denote the number of rows and columns, respectively. SAFE also supports scalable data size and encoding configurations for different n ranges. To exploit the parallelism capability provided by FPGA, SAFE employs a fully pipelined architecture, where each stage is specifically designed for a maximized throughput. Further, we design dedicated modules for NTT, polynomial processing units, and various buffers. SAFE is implemented on the Xilinx VU9P FPGAs and works at $300\,\mathrm{MHz}$.

The contributions of this work are summarized as follows.

- SAFE provides a comprehensive solution that involves algorithm-hardware co-design. On one hand, SAFE is designed for a secure and flexible HMVP algorithm. On the other hand, we optimize the algorithm to align seamlessly with the dataflow hardware architecture.
- We provide a scalable HMVP algorithm to adopt different training data sizes. The proposed compressed and concatenated modes compact the raw matrix to densely packed plaintext. After a k fold compression or concatenating, the total key switching complexity is reduced by approximately k times, significantly reducing the overhead.
- We design customized hardware architecture to achieve high-performance HMVP. In particular, SAFE employs a customized, fully-pipelined architecture that can effectively utilize the parallelism of FPGA. For NTT, we propose a compute-efficient architecture that enables pipelined data flow without bubbles. Compared to state-of-the-art implementations [48], our NTT logic efficiency has improved by $2.43\times$.
- SAFE has been deployed as a part of the CPU-FPGA heterogeneous engines within the privacy-preserving infrastructure for accelerating real-world applications. We evaluate the end-to-end latency breakdown and achieve up to $36\times$ speed-up for LR training.

II. Background

This section introduces the data structure and computation process of HE ciphertexts. We also demonstrate existing techniques and security requirements associated with HE.

A. Homomorphic Encryption

In HE schemes, arithmetic operations over plaintext are mapped to LWE or RLWE ciphertext manipulations. LWE ciphertexts are lightweight and suitable for scalar arithmetic, while RLWE ciphertexts offer enhanced efficiency for batch processing that involves vectors and matrices.

1) The LWE Ciphertext: An LWE ciphertext consists of a random vector $\vec{a}$ and a scalar b that are encrypted using a secret key $\vec{s}$ as

$$b =< \vec{a}, \vec{s} > +\Delta \cdot \mathtt{pt} + e \tag{1}$$

where e is a small noise and Δ is a lifting scalar. Only the data owner who has $\vec{s}$ can recover the scalar $\mathtt{pt} \approx (b - < \vec{a}, \vec{s} >) \cdot \Delta^{-1}$.

2) The RLWE Ciphertext: In the context of RLWE, all data structures are over polynomial rings modulo $X^N + 1$ where N is a power of two. A plaintext coefficient modulus is a prime number t such that $t \equiv 1 \mod 2N$. An RLWE plaintext $\mathtt{pt}(x) \in \mathbb{Z}_t[X]/(X^N + 1)$ is encrypted as two polynomials $\mathbf{a}(x)$ and $\mathbf{b}(x)$ using a secret key $\mathbf{s}(x)$ as

$$\mathbf{b}(x) = \mathbf{a}(x) \cdot \mathbf{s}(x) + \Delta \cdot \mathtt{pt}(x) + \mathbf{e}(x) \tag{2}$$

where $\mathbf{a}(x)$ is random, $\mathbf{e}(x)$ is a small noise polynomial.

3) The Basic Homomorphic Computations: Given an RLWE ciphertext $E(\mathtt{pt}_1(x)) = \{\mathbf{b}_1(x), \mathbf{a}_1(x)\}$, the result of addition with a plaintext $\mathtt{pt}_2(x)$ is denoted as

$$E(\mathtt{pt}_1(x) + \mathtt{pt}_2(x)) = \mathbf{b}_1(x) + \mathtt{pt}_2(x),\ \mathbf{a}_1(x). \quad (3)$$

Adding $E(\mathtt{pt}_1(x))$ with another ciphertext $E(\mathtt{pt}_2(x)) = \{\mathbf{b}_2(x), \mathbf{a}_2(x)\}$ is calculated as

$$E(\mathtt{pt}_1(x) + \mathtt{pt}_2(x)) = \mathbf{b}_1(x) + \mathbf{b}_2(x), \mathbf{a}_1(x) + \mathbf{a}_2(x). \quad (4)$$

Similarly, the product of $E(\mathtt{pt}_1(x))$ with plaintext $\mathtt{pt}_2(x)$ is represented as

$$E(\mathtt{pt}_1(x) \cdot \mathtt{pt}_2(x)) = \mathbf{b}_1(x) \cdot \mathtt{pt}_2(x),\ \mathbf{a}_1(x) \cdot \mathtt{pt}_2(x). \quad (5)$$

In this cross-agency scenario, we avoid the costly ciphertext multiplication operation, necessitating extra procedures like relinearization [10], [12]. After the homomorphic computation, the data owner who manipulates the secret key $\mathbf{s}(x)$ can decrypt the ciphertext to recover the corresponding plaintext.

B. Batched Data Encoding

RLWE plaintexts and ciphertexts consist of N coefficients that can be used to represent a vector of data elements. Data encoding strategies are developed to arrange the original data with different formats and sizes. Scalars, vectors, matrices, and tensors can be packed or tiled to fit the HE data structure. When executing an HE operation, all the batched elements in the ciphertext can amortize the latency to improve computation efficiency. A batch encoder can be typically constructed in slot-wise or coefficient-wise style.

- **Slot-wise**: According to the Chinese Remainder Theorem (CRT), the plaintext space $\mathbb{Z}_t[X]/(X^N+1)$ is isomorphic to the N-fold product of fields $\mathbb{Z}_t[X]/(X-\zeta) \times (X-\zeta^3) \times \cdots \times (X-\zeta^{2N-1}) \mod t$ where the constants $\zeta, \zeta^3, ..., \zeta^{2N-1}$ are all the distinct primitive $2N$-th roots of unity in integers modulo t. Each decomposed field represents a distinct slot that accommodates an integer message. This arrangement enables SIMD operations, simultaneously allowing identical computations across all plaintext slots.
- **Coefficient-wise**: The elements in a vector $\vec{v}$ are placed into the coefficients of a polynomial with the specific order as

$$\vec{v} = (v_0, v_1, ..., v_N) \rightarrow \mathtt{pt}(x) = \sum_{i=0}^{N} v_{\phi(i)} X^i \quad (6)$$

  where $\phi(\cdot)$ specifies a permutation over the vector elements.

The layout of data encoding plays a crucial role in determining the computation process. For example, when calculating the vector inner product $\sum_{i=0}^{N-1} u_i v_i$ in a slot-wise fashion, the homomorphic multiplication provides individual outputs $u_0v_0, u_1v_1, \cdots, u_{N-1}v_{N-1}$. These products are then collectively summed into a single slot using homomorphic rotation and addition. Conversely, encoding vector $\vec{u}$ as $u_0, u_1, \cdots, u_{N-1}$ and vector $\vec{v}$ as $v_0, -v_{N-1}, -v_{N-2}, \cdots, -v_1$ allows for directly get the aggregation $\sum_{i=0}^{N-1} u_i v_i$ at the constant coefficient.

Algorithm 1 PACKLWES

Input: RLWE ciphertexts $\{\mathtt{ct}_i\}_{i=0}^{2^l-1}$ in which only sparse coefficients are valid.
Output: An RLWE ciphertext $\mathtt{ct}$

if $l = 0$ **then**
  return $\mathtt{ct}_0$
else if $l = 1$ **then**
  return PACKTWOLWES$(1, \{\mathtt{ct}_i\}_{i=0}^{1})$
else
  $\mathtt{ct}_{even} =$ PACKLWES$(\{\mathtt{ct}_{2i}\}_{i=0}^{2^{l-1}-1})$
  $\mathtt{ct}_{odd} =$ PACKLWES$(\{\mathtt{ct}_{2i+1}\}_{i=0}^{2^{l-1}-1})$
  return PACKTWOLWES$(l, \{\mathtt{ct}_{even}, \mathtt{ct}_{odd}\})$
end if

Algorithm 2 PACKTWOLWES

Input: An automorphism index l, two sparse RLWE ciphertexts $\{\mathtt{ct}_i\}_{i=0}^{1}$
Output: An RLWE ciphertext $\mathtt{ct}$

$\mathtt{ct}_{mono} = x^{N/2} \cdot \mathtt{ct}_1$ ▷ Multiply by mononomial
$\mathtt{ct}_+ = \mathtt{ct}_0 + \mathtt{ct}_{mono}$
$\mathtt{ct}_- = \mathtt{ct}_0 - \mathtt{ct}_{mono}$
$\mathtt{ct}_{auto} =$ EVALAUTO$(\mathtt{ct}_-, l)$ ▷ Automorphism
return $\mathtt{ct}_+ +$ KEYSWITCH$(\mathtt{ct}_{auto}, l)$ ▷ KeySwitch

C. Packing Encrypted Scalars

The combined utilization of both LWE and RLWE ciphertexts significantly improves the efficiency of evaluating homomorphic linear functions. For example, the result of the vector inner product has only one valid coefficient as Section II-B, hence, extracting the scalar coefficients from multiple ciphertexts and repacking them together into a single ciphertext is required. To unpack and repack, we apply the method proposed by Hao Chen *et al.* [49] while preserving the LWE ciphertext as a sparse RLWE structure. Generally speaking, we extract and cast all the valid scalar coefficients as LWE ciphertexts, which eventually would be packed in an RLWE ciphertext as the final result. Algorithm 1 recursively packs two sparse RLWE ciphertexts into a packed one using Algorithm 2. A PACKTWOLWES procedure calls two critical functions, namely EVALAUTO, KEYSWITCH in Equations 7 and 8.

For an RLWE ciphertext $\mathtt{ct} = (\mathbf{b}(x), \mathbf{a}(x))$, an automorphism with index l is defined as follows:

$$\text{EVALAUTO}(\mathtt{ct}, l) = (\mathbf{b}(x^{2l+1}), \mathbf{a}(x^{2l+1})) \quad (7)$$

Moreover, key switching with automorphism index l is defined as follows:

$$\text{KEYSWITCH}(\mathtt{ct}, l) = (\mathbf{b}(x), 0) + \mathbf{a}(x) \cdot \text{KSK}_l \quad (8)$$

where KSK_l is pre-generated, referring to the key switching key [12] for the automorphism index l.

D. Federated Learning and Security Model

In this work, we adopt a two-party coordinator-assisted FL training model [4], [6] as Fig. 1, where agency A and B share aligned record rows. The two parties owns different features, *i.e.* agency A stores the X_A and the label y, while agency B stores X_B. The FL-variant logistic regration training protocol

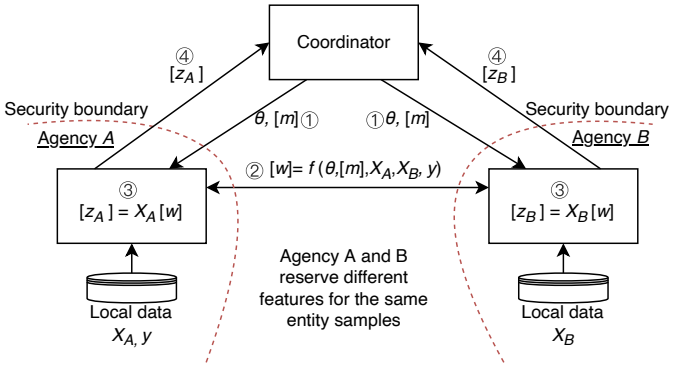


Fig. 1. The overall workflow of cross-agency VFL.

is executed around all the records, in which each iteration works with one mini-batches and updates the weight parameters with a HE-based gradient aggregation accrodingly [4]. The coordinator broadcasts the initial weight θ and an encrypted mask $[m]$ to the participants in each iteration. Then step ② negotiates an intermediate ciphertext $[w]$. Step ③ multiplies the local plaintext feature matrix with the ciphertext $[w]$ to calculate the gradient $[z_A]$ and $[z_B]$, which are fed back to the coordinator in step ④. In this scenario, we evaluate the HMVP in step ③ taking over the major overhead (evaluated in Section VII-C), which is a strong motivation to design accelerators.

We assume the participants are honest-but-curious. In other words, while A and B try to learn as much as possible from the received data, they honestly follow the prescribed protocol.

E. Encryption Parameters

The selection of encryption parameters, *i.e.*, polynomial degree N and CRT decomposed moduli q_i, is based on the required security level and plaintext space. In particular, complex operation flows and large plaintext spaces require large encryption parameters. However, increasing HE parameters significantly elevates complexity. In this work, we select $N = 2^{12}$ for the lowest computation overhead satisfying the security requirements. This corresponds to a space of $109\,\text{bit}$ where $70\,\text{bit}$, corresponding to two moduli, is used for representing plaintext and ciphertext, while the other $39\,\text{bit}$ is used as a special modulus for key switching.

III. Scalable HMVP Algorithms

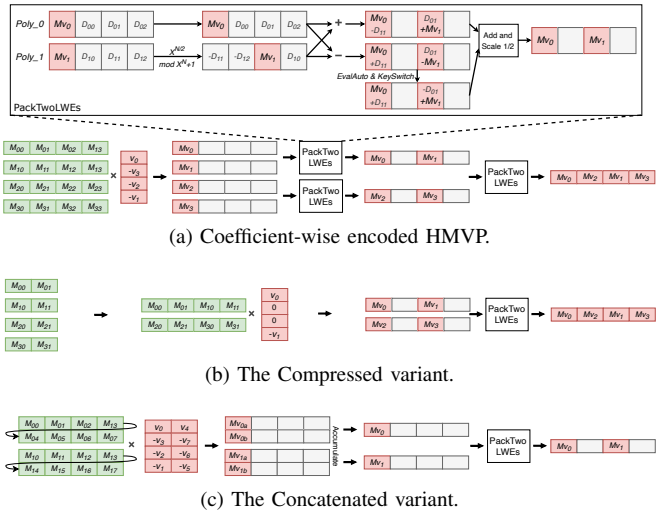
In HMVP tasks, the feature size varies in exact applications, which may not fit exactly with the polynomial encoding size. In this section, we present our efficient and flexible algorithm for HMVP. This technique enables SAFE to accommodate varying sample counts and feature sizes across diverse configurations.

A. Homomorphic Matrix-Vector Product

We observe that the naive homomorphic inner product incurs a cost of $\mathcal{O}(\log_2 n)$ KeySwitch, as each homomorphic rotation requires one KeySwitch. In contrast, the coefficient-wise encoded inner product has two advantages, including free

Algorithm 3 Coefficient-wise encoded HMVP

Input: A matrix $A^{m\times n}$ coefficient-wise encoded as $\texttt{pt}_1 \sim \texttt{pt}_m$, and a ciphertext $\texttt{v}$ corresponding to a coefficient-wise encoded vector $\vec{v}$
Output: A ciphertext $\texttt{u}$ corresponding to vector $\vec{u} = A \cdot \vec{v}$
1: **for** $i \leftarrow 1$ to m **do**
2: $\quad \texttt{c}_i \leftarrow \texttt{pt}_i \times \texttt{v}$ ▷ *S1*: Inner product
3: $\quad \texttt{l}_i \leftarrow \text{ExtScalar}(\texttt{c}_i)$ ▷ *S2*: Extract scalar
4: **end for**
5: $\texttt{u} \leftarrow \text{PackLWEs}(\texttt{l}_1, ..., \texttt{l}_m)$ ▷ *S3*: Rec. Pack2LWEs



(a) Coefficient-wise encoded HMVP.

(b) The Compressed variant.

(c) The Concatenated variant.

Fig. 2. The schematic HMVP examples where the polynomial coefficient number is set to 4. We analyze the workflow for different data sizes when (a) $n = N$, (b) $n = N/2$ and (c) $n = 2N$.

of the costly encoding which requires FFT, and the complexity is only $\mathcal{O}(1)$. When performing an HMVP algorithm, which multiplies a local plaintext matrix by an encrypted vector, as shown in Alg. 3, the complexity can be reduced to $\mathcal{O}(m)$. When compared to the diagonal encoding in [50] whose complexity is also $\mathcal{O}(m)$, Alg. 3 is still faster because it employs a much simpler encoding method.

The computation process consists of three steps, namely *i.e. S1* $\sim$ *S3*. The *S1* step involves performing a homomorphic vector inner product between a row of a matrix and the ciphertext. In the *S2* step, a valid LWE ciphertext result is extracted. Finally, in the *S3* step, the multiple LWEs are repacked into a single RLWE ciphertext. This computing process guarantees only the key owner can recover the result vector from the ciphertext.

Fig. 2(a) shows a $N = 4$ case study where the PackLWEs consist of 3 PackTwoLWEs steps. The scalar results are arranged in different coefficients within each PackTwoLWEs block to avoid data overwriting. To achieve this, polynomial $Poly\_1$ is shifted and rounded by $N/2$. Furthermore, the dirty data (*i.e.* D_{01}, D_{11}) obstruct the final result so that $Poly\_0$ and $Poly\_1$ can not be accumulated directly. The automorphism flips the coefficient of $D_{01} - Mv_1$ to eliminate invalid data. As the key changes, a KeySwitch must be introduced to ensure correctness after decryption.

Moreover, Algorithm 3 can be applied to more linear functions, such as 2-D and 3-D convolutions, through encoding

the original tensors in similar ways [51]. In the cross-agency model training scenario, the column is the number of features, likely not matched with the polynomial size. A naive idea is to expand the original vector by zero-padding. *i.e.* packing each row vector in plaintext.

B. The Compressed HMVP

The polynomial degree is at least 4096, as smaller parameters cannot support high-precision KEYSWITCH. However, zero-padding results in efficiency loss when $n \ll N$. SAFE proposes a compressed HMVP variant that encodes multiple rows in a single plaintext.

In Fig. 2(b), where $n = N/2$, we first compress every two rows into a single long vector. In this way, the total row number can be half of the original matrix. The encrypted vector is arranged as $\vec{v} = v_0, 0, 0, -v_1$. The intermediate value of the inner product has two valid coefficients, namely $u_0X^0 = M_{00}X^0 \cdot v_0X^0 + M_{01}X^1 \cdot -v_1X^3 \bmod (X^4+1)$ and $u_1X^2 = M_{10}X^2 \cdot v_0X^0 + M_{11}X^3 \cdot -v_1X^3 \bmod (X^4+1)$. During the PACKLWES stage, the overall complexity of KEYSWITCH reduces from $m-1$ in vanilla mode to $m/2-1$. With this technique, a smaller feature size will enable higher-fold compression, further reducing the overhead.

C. The Concatenated HMVP

When the feature size has more items than 4096, a naive idea is to enlarge the parameter sets, *i.e.* N, to encode one row in a polynomial. Nevertheless, enlarging N causes a drop in efficiency since the complexity of NTT and INTT is superlinear. A HMVP task could also be vertically decomposed into small tasks, *i.e.*, performing $M^{(m\times n)}\vec{v}$ with two $M^{(m\times \frac{n}{2})}\vec{v}$. Nevertheless, the overall computation overhead increases to $2(m-1)$.

SAFE explores the concatenated HMVP mode where two rows are combined to form a long vector. In the case where $n = 2N$, as shown in Fig. 2(c), every two adjacent rows represent a full-length vector. This allows us to accumulate the polynomial multiplication results after the inner product. As a result, the total complexity of KEYSWITCH is only $m-1$, even for larger columns.

IV. PROPOSED MICROARCHITECTURE DESIGN

SAFE uses a customized microarchitecture enhanced with spatial and temporal parallelization to improve the performance. The compute engine uses a fully pipelined architecture to accelerate coefficient-wise encoded HMVP tasks. We also comprehensively analyzed computation intensity to determine the appropriate kernel granularity.

A. The Parallelized Hardware Design

SAFE can divide an HMVP task of size $M^{(m\times n)}\vec{v}$ into l smaller tasks of size $M^{(m/l\times n)}\vec{v}$. The partial products of these smaller tasks can be combined using simple homomorphic addition operations without introducing any KEYSWITCH overhead. We can take advantage of this property to reduce the overall latency by using parallel compute engines. The design of a compute engine contains three modules as depicted in Fig. 3, *i.e.*, INNERPRODUCT, EXTRACTLWES, and PACKTWOLWES. These modules are further divided into smaller functional units (FUs). SAFE instantiates parallel FUs according to the HE data structure. As implied by the encryption parameters in Section II-E, each ciphertext contains two 35-bit moduli and an extra 39-bit modulus for reducing noise. This means that each plaintext and ciphertext contains three and six polynomials respectively. The error introduced by homomorphic operations is eliminated in the MODSWITCH unit, after which the 39-bit modulus is drop off.

INNERPRODUCT is performed using a simple polynomial multiplication. Since the computation of polynomial multiplication involves a convolution, we transform the input matrix and vector to the NTT format such that the convolution becomes a coefficient-wise multiplication [52]. The design of NTT and INTT FUs employs a pipeline microarchitecture that will be elaborated in Section V-A. After the polynomial multiplication, we use a MODSWITCH to reduce the error in the ciphertext. In particular, the accumulation buffer stores temporary data in the concatenated HMVP dataflow.

The output of INNERPRODUCT is then supplied to the EXTRACTLWES module. Since the valid result is located in coefficients as Fig. 2, this step extracts new LWE ciphertexts from the RLWE ciphertext. The LWE ciphertexts are stored as sparse RLWE ciphertexts and sequentially fed into the following PACKLWES step.

As indicated by Algorithm 1, PACKLWES is applied using a reduction tree where each node involves the function of PACKTWOLWES. Hence, the reduction tree is implemented using a single PACKTWOLWES module that takes two ciphertexts as input and reduces them into one. The input ciphertexts come from either the preceding modules, *i.e.*, EXTRACTLWES, or the reduce buffer used for storing intermediate reduction results. In particular, once the intermediate reduction results are ready for the next-level reduction, they preempt the pipeline and stall the execution of INNERPRODUCT and EXTRACTLWES. The extra modulus is introduced before the NTT based KSK multiplication to tolerate the noise. The KSK is preloaded to the on-chip buffer in the NTT format. Thanks to the small parameter settings, the hardware volume is sufficient to accommodate the full KSK.

B. The Fully-pipelined Dataflow

The design of a compute engine employs a fully-pipelined architecture. Here the pipeline refers to the macro-pipeline, meaning that each stage endures thousands of clock cycles. For example, the INNERPRODUCT module is divided into four stages as indicated by Fig. 3. The FUs within the EXTRACTLWES module, however, reside in the same pipeline stage with the MODSWITCH unit because the coefficient-wise datapath is easy to combine. The PACKTWOLWES module, corresponding to Algorithm 2, contains the most number of pipeline stages, where KEYSWITCH takes up five of them (*i.e.*, from P5 to P9). The MULTMONO and AUTOMORPH merge into a single stage because their execution can be easily overlapped as Section V-B.

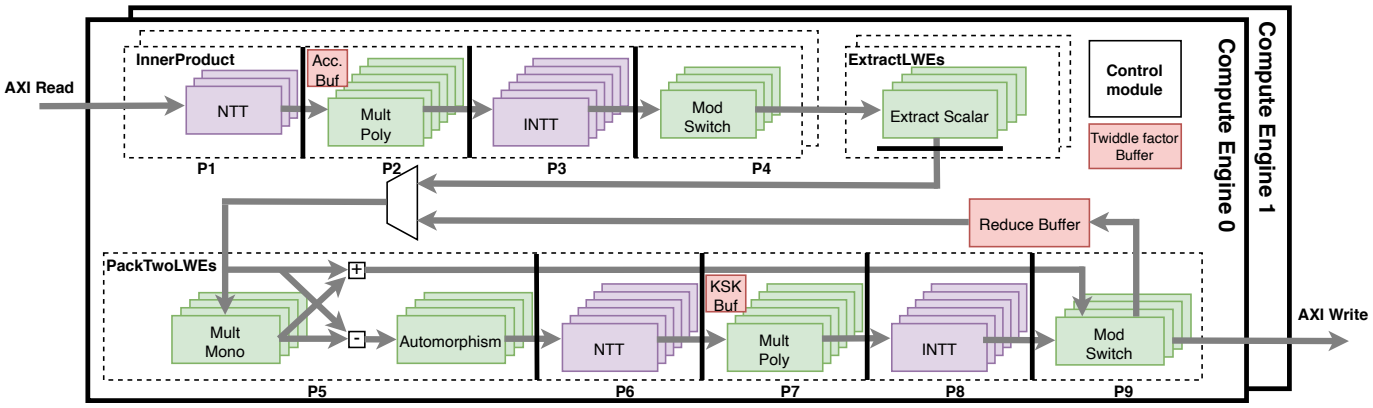


Fig. 3. An overview of the SAFE microarchitecture, which consists of isomorphic compute engines. Each compute engine is composed of spatial and temporal parallelized FUs.

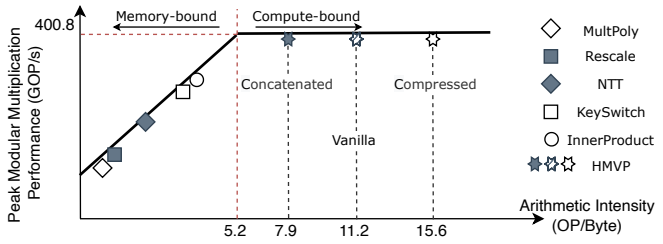


Fig. 4. Roofline analysis for SAFE based on the Xilinx VU9P FPGA.

To maximize module utilization, we carefully determine the number of FUs and the parallelism inside each FU to make all pipeline stages have similar latency. We define the parallelism (P) of a stage as the product of the number of FUs in the stage (n_f) and the parallelism inside the FU (p_f), *i.e.*, $P = n_f \times p_f$. Take the P2 stage (with six MULTPOLY units) and the P3 stage (with six INTT units) in the INNERPRODUCT module as an example. Considering that P2 involves $6\times$ less complexity than P3, the ideal parallelism of P2 should be $6\times$ smaller than the parallelism of P3, *i.e.*, $P_3 = 6P_2$, for a balanced latency.

C. Architectural Roofline Model

We plot the roofline model for SAFE based on the Xilinx VU9P FPGA in Fig. 4. The measured operation refers to 39-bit integer modular multiplication that utilizes 5 DSP slices on FPGA. Fig. 4 reveals that the performance of fine-grained low-level compute kernels is restricted by memory bandwidth since they show poor arithmetic intensity. For HMVP acceleration, utilizing the data locality to reduce memory access and increase compute intensity brings significant benefits. All three work modes discussed in Section III are compute-bound. Based on this observation, the design of SAFE focuses on accelerating HMVP operations instead of smaller HE operations. An additional advantage of our FPGA solution is that it does not require expensive high-bandwidth memory (HBM). Instead, our use of off-the-shelf DDR memory successfully meets the application's requirements.

Algorithm 4 Constant-geometry forward NTT

Input: Polynomial $\mathbf{a}(x)$, and twiddle factors stored in $\omega_{2N}[\log_2 N \cdot N/2]$.
Output: $\bar{\mathbf{a}}(x) = \mathrm{NTT}(\mathbf{a}(x))$ in bit-reversed order

1: **for** $i \leftarrow 0$ to $\log_2 N$ **do** ▷ i is stage index
2: **for** $j \leftarrow 0$ to $N/2$ **do** ▷ j is butterfly index
3: $\omega_{ij} = \omega[i \cdot N/2 + j]$ ▷ Fetch factor
4: $\bar{\mathbf{a}}(x)_{2j} = \mathbf{a}(x)_j + \mathbf{a}(x)_{j+N/2} \cdot \omega_{ij}$ ▷ Butterfly
5: $\bar{\mathbf{a}}(x)_{2j+1} = \mathbf{a}(x)_j - \mathbf{a}(x)_{j+N/2} \cdot \omega_{ij}$
6: **end for**
7: **if** $i \neq logN - 1$ **then**
8: $\mathbf{a}(x) = \bar{\mathbf{a}}(x)$
9: **end if**
10: **end for**

V. OPTIMIZING THE FUNCTIONAL UNITS

In this section, we elaborate on the design of the FUs, including NTT units and polynomial processing units.

A. NTT and INTT Units

The NTT is a generalization of the discrete Fourier transform (DFT) to finite fields. The NTT enables fast convolution on integer sequences without any round-off errors, and therefore it is useful for multiplying large polynomials. In particular, the multiplication of polynomials $\mathbf{a}(x)$ and $\mathbf{b}(x)$ is performed by

$$\mathbf{c}(x) = \mathrm{INTT}(\mathrm{NTT}(\mathbf{a}(x)) \circ \mathrm{NTT}(\mathbf{b}(x))) \tag{9}$$

where "$\circ$" refers to coefficient-wise multiplication. The NTT algorithm works in various forms, *e.g.*, constant-geometry [53], [54] and in-place [27], [55]. Algorithm 4 shows a constant-geometry NTT. The outer loop divides the computation of NTT into $\log_2 N$ stages, while each stage consists of $N/2$ butterfly operations. The memory access pattern remains the same for all stages, as the read and write addresses (*lines* 4 to 5) are independent of the stage index.

As Algorithm 4 implies, the NTT operation can be highly parallelized. The number of butterfly units (BFUs) affects both performance and hardware utilization. First, the $N/2$ butterfly operations in each NTT stage are independent of each other, such that they can be easily parallelized with the number of

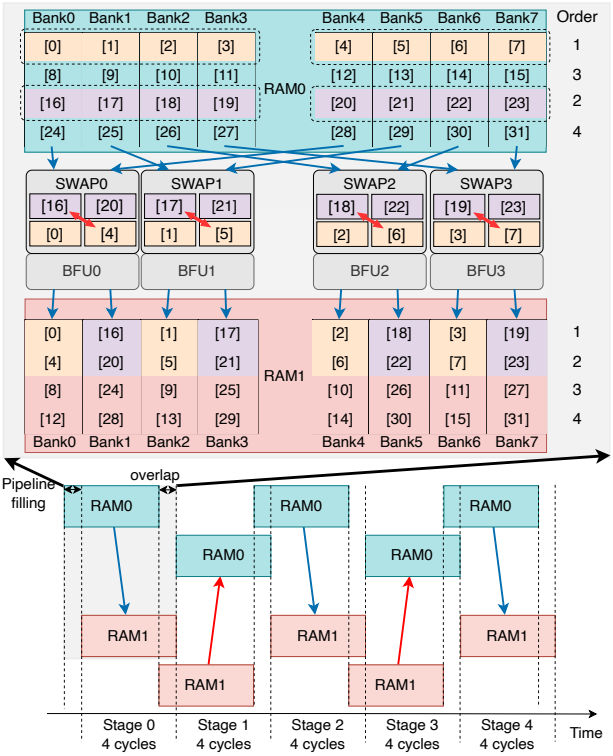


Fig. 5. The NTT datapath with four parallel BFUs when the number of coefficient N=32.

BFUs (n_{bf}) equals 2^s, where s is an integer. To support high parallelism, a polynomial needs to be stored in as many RAM banks as the value of n_{bf}. Therefore, each bank only stores a few data while the rest of the volume is wasted. Although a larger n_{bf} indicates higher parallelism, we find a sweet point when preserving on-chip RAM from under-utilization. We further propose the following three optimizations.

1) Parallel constant-geometry dataflow: This work stores a polynomial in 8 round-robin RAM banks. NTT is executed in a ping-pong fashion, as shown in Fig. 5. More precisely, the polynomial coefficients are read from RAM-0 and written to RAM-1 during the odd stages, while the coefficients are then read from RAM-1 and written to RAM-0 during the even stages. Therefore, the NTT process requires $(N/2 \cdot \log_2 N)/n_{tf}$ clock cycles. In this work, we set $n_{bf} = 4$ to access all RAM banks in parallel.

The consecutive coefficients of a polynomial are stored across RAM banks, *i.e.* the coefficients $[0] \sim [7]$ are stored in the address 0, such that all 1R1W banks can be read and written simultaneously. The coefficients are read in an interlaced order (*i.e.*, $[0] \sim [7]$, $[N/2] \sim [N/2+7]$, $[8] \sim [15]$, ..., $[N-8] \sim [N-1]$) and written in ascending order (*i.e.*, $[0] \sim [7]$, $[8] \sim [15]$, ..., $[N-8] \sim [N-1]$). This read-write fashion ensures a fixed interconnection between the NTT units and the RAM banks. The SWAP unit reorders the coefficients read by a BFU to reserve the constant geometry feature. For example, the unit of SWAP-0 exchanges the positions of coefficients $[4]$ and $[N/2]$.

2) Storage and utilization of twiddle factors: Fig. 6 unroll all the $\log_2 N \cdot N/2$ twiddle-factors used in the NTT process.

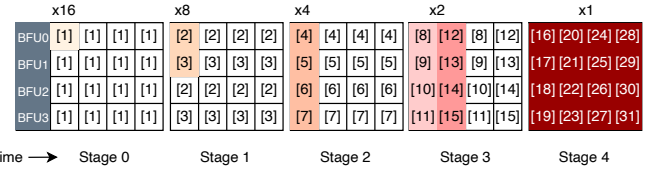


Fig. 6. The arrangement and utilization of twiddle factors for N=32, where $[\cdot]$ represents bit-reversed twiddle-factor indices.

The highlighted squares are essential factors, while others are repetitive. We use a customized data arrangement in which twiddle factors for stages $2 \sim \log_2 N$ can be stored in four $N/4$-depth RAMs. The factor used in stage 1 is hard-coded as a constant. This optimization reduces the total volume requirement from $\log_2 N \cdot N/2$ to N. During the NTT executing process, an address generator fetches the corresponding index. SAFE assigns each BFU a separate ROM bank, eliminating the need for cross-bank data scheduling. Moreover, all NTT and INTT modules in the architecture execute synchronously, thus the twiddle factors can further share across a compute engine. We also noticed a new NTT design that requires only $1.5N$ coefficients memory capacity [56]. However, its data input and output must be distributed across two memory clusters, which is incompatible with our fully pipelined architecture.

3) Customized optimization for modular reduction: Modular multiplication is the most important but complicated operation in HE. We achieve significantly simplified modular arithmetic by pre-defining a finite field over a modulus with low-hamming weight. Each modulus has only three non-zero bits, such that multiplication by them can be simplified as shifts and additions. Following the security model described in Section II-E, we choose the prime numbers

$$q_0, q_1, p = 2^{34} + 2^{27} + 1,\ 2^{34} + 2^{19} + 1,\ 2^{38} + 2^{23} + 1 \quad (10)$$

As described in Section IV, the implementation of SAFE involves separate NTT or INTT units. To save on-chip logic, we design NTT and INTT units respectively, instead of using a unified one. An INTT unit has a similar micro-architecture as the NTT, except that it uses Gentleman-Sande butterfly and includes a step of post-scaling by the factor of N^{-1} [57].

Previous work has proposed several implementations for NTT. For example, F1 proposes an ASIC-based prototype [41], but the $(\sqrt{N} \times \sqrt{N})$-element memory block required by its NTT design is hard to be applied to FPGAs. Another work, HEAX proposes an FPGA-friendly design with careful utilization of the block RAMs [27]. Nevertheless, HEAX handles the stage-variant memory access pattern with many multiplexers.

B. Polynomial Processing Units (PPUs)

Beyond NTT and INTT, the other FUs, including MULTPOLY, RESCALE, MULTMONO, and AUTOMORPH, are based on polynomial arithmetic. Hence, we design PPUs to support these functions. From the implementation perspective, the coefficients of a polynomial are stored in a coefficient-vector data structure, and all polynomial operations are carried out

in a vectorized fashion. For this reason, both LWE ciphertext (composed of a vector and a scalar) and RLWE ciphertext (composed of polynomials) can be well supported by a unified data structure for both polynomials and vectors.

Table I lists the polynomial arithmetic implemented in SAFE. MODADD and MODMUL perform coefficient-wise modular additions and multiplications. REV reverses the order of the polynomial coefficients. SHIFTNEG, serving as an underlying function for MULTMONO, is implemented as a circular shift followed by a negation of the wrapped-around coefficients. AUTOMORPH is implemented as a coefficient permutation without arithmetic overhead.

TABLE I
THE POLYNOMIAL ARITHMETIC SUPPORTED BY SAFE.

Functions	Details
MODADD(A, B)	$[a_0 + b_0, a_1 + b_1, ..., a_{N-1} + b_{N-1}]$
MODMUL(A, B)	$[a_0 \times b_0, a_1 \times b_1, ..., a_{N-1} \times b_{N-1}]$
REV(A)	$[a_{N-1}, ..., a_1, a_0]$
SHIFTNEG(A, s)	$[a_{N-s}, ..., a_{N-1}, -a_1, ..., -a_{N-s-1}]$
AUTOMORPH(A, k)	$a_i \rightarrow (-1)^{\lfloor ik/N \rfloor} a_{ik \mod N}$

Note: $A = [a_0, a_1, ..., a_{N-1}]$ denotes the coefficients of a polynomial.

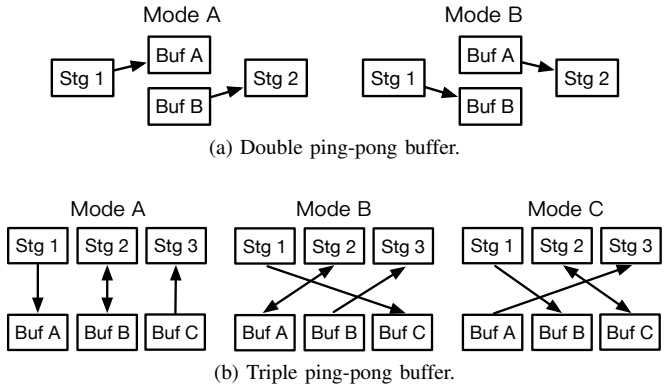
Based on the functions listed in Table I, we design PPUs for the aforementioned pipeline. A PPU consists of an address generator and computation logic. The address generator aims to rearrange the coefficients required by REV, SHIFTNEG and AUTOMORPH, while the computation logic aims to handle modular arithmetic, with modular addition, subtraction and multiplication as its building blocks.

As mentioned in Section IV, the pipeline stages contain either PPUs or NTTs/INTTs. The NTT module latency with 4 PEs is larger than the coefficient number N. To minimize area while ensuring at least the same throughput as the NTT, PPUs can be executed serially. We assign one PPU for each scalar-polynomial or polynomial-polynomial operation in each pipeline stage. All PPUs are executed in parallel, while each PPU outputs one coefficient per cycle. On the AUTOMORPH function, compared with the 2D solution with row-wise and column-wise permutation and multi-stage transpose in F1 [41], our low-area solution only consumes a simple reorder logic.

C. On-Chip Memory Architectures

The Xilinx VU9P FPGA, equipping with approximately $9.4\,\mathrm{MB}$ block RAMs and $33.7\,\mathrm{MB}$ ultraRAMs (URAMs), can store at most 404 ciphertexts. These RAMs are properly separated for storing input/output polynomials, twiddle factor, intermediate results (*i.e.*, pipeline buffer and reduce buffer), and key-switch key (*i.e.*, KSK buffer). The twiddle factors buffer and the pipeline buffer, frequently used during the whole computation, are implemented using block RAMs, while the KSK buffer ($4.5\,\mathrm{MB}$) and the reduce buffer ($320\,\mathrm{KB}$), used less frequently, are implemented using URAMs.

SAFE allocates pipeline ping-pong buffers for data communication between neighboring pipeline stages. Fig. 7(a) shows how a double ping-pong buffer works. In particular, during the T-th time slot, Stage 1 writes to Buffer A and Stage 2 reads from Buffer B (corresponding to *Mode A*); then during the (T+1)-th time slot, Stage 1 writes to Buffer B and Stage 2 reads from Buffer A (corresponding to *Mode B*). When a pipeline stage needs to read and write a buffer during the same time slot, a triple ping-pong buffer is used, as shown in Fig. 7(b). By using ping-pong buffers, the pipeline stages can start execution immediately upon arrival of the start signal without any data copy. Besides, using buffers in the ping-pong fashion is also efficient because all ping-pong buffers remain in use when the pipeline stages are busy.



(a) Double ping-pong buffer.

(b) Triple ping-pong buffer.

Fig. 7. The ping-pong buffers used in SAFE cross-stage data transfer.

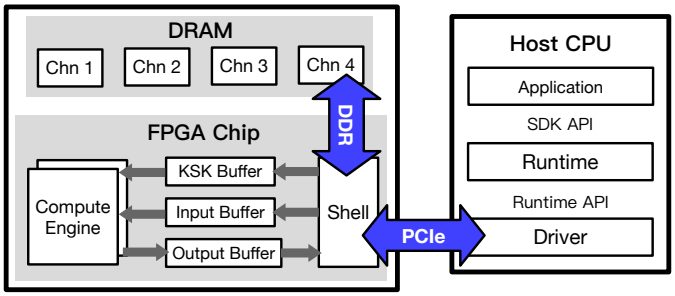


Fig. 8. The heterogeneous system of SAFE.

VI. SYSTEM DESIGN AND IMPLEMENTATION

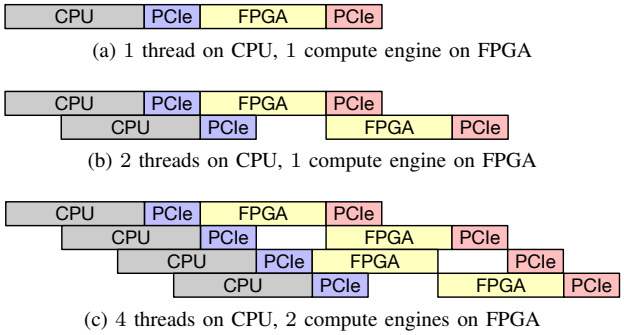
SAFE is a heterogeneous system that consists of a CPU and an FPGA, connected through the Peripheral Component Interconnect Express (PCIe) bus. Our custom-developed FPGA board is designed to offload tasks from the CPU, thereby reducing its overhead and enhancing the overall quality of service. We also developed a software stack comprising runtime and driver to facilitate the use of high-level applications. Then we elaborate on the FPGA implementation with resource utilization and its breakdown.

A. Heterogeneous Computing System

Figure 8 shows that SAFE is instantiated on the FPGA accelerator with 4-channel DDR4. When the CPU program needs to evaluate HMVP, it splits the matrix into supported size and checks if the FPGA is idle. Once the FPGA idles, the host CPU will send data to the DDR4 device memory on the PCIe board, configure the SAFE IP status registers, and trigger the kernel execution. Once the execution is completed, SAFE sends an interrupt to the host CPU.

We interleave computation and data transfer between FPGA and CPU to maximize utilization of the computation units on SAFE. One complete task comprises two parts: the host side and the FPGA side. We pipeline the data transfer and encoding using multiple threads on the host side. On the FPGA side, we use split buffers for the input and output data corresponding to each thread. Figure 9 illustrates how this pipeline mechanism works in the case of different numbers of CPU threads and SAFE compute engines.

An FPGA-based acceleration solution is cost-effective for achieving a high return on investment (ROI). Many cloud service providers already have legacy FPGAs in place, and using FPGAs offers significant performance advantages. In contrast, utilizing CPU multi-threading can potentially compromise service quality, as it may lead to performance degradation due to thread contention and frequency reduction from AVX acceleration.



(a) 1 thread on CPU, 1 compute engine on FPGA

(b) 2 threads on CPU, 1 compute engine on FPGA

(c) 4 threads on CPU, 2 compute engines on FPGA

Fig. 9. An illustration for the pipelined execution of multi-thread CPU and hardware compute engines.

B. Software Stack

For the data-center scenarios, FPGAs are available as PCIe-based accelerator cards. The application will invoke the SAFE IP through the runtime and driver from the host CPU. The runtime handles the memory allocation, task dispatching, and scheduling. We define two runtime Application Programming Interfaces (APIs). One is for loading metadata (*i.e.*, KSK) to the FPGA board, and the other is for starting the execution of homomorphic MVP. The user can bypass the metadata loading API call if multiple MVPs use the same KSK set.

C. Implementation

SAFE is implemented in FPGA mounted on peripheral logic, *i.e.*, shell. To integrate with the shell, SAFE is packaged with two Advanced Extensible Interface (AXI) interfaces. One is AXI master for data transfer, and the other is AXI-lite slave for control. The PCIe Gen3 $\times 16$ connects the FPGA platform to the host CPU. The bandwidth of the PCIe can be up to $16\,\mathrm{GB/s}$ while the attainable peak bandwidth is about $12\,\mathrm{GB/s}$. The Xilinx QDMA IP enables data transfer via the PCIe. In the control path, we define 16 32-bit configuration registers for controlling IP behavior, configuring the HMVP command, and specifying the input/output data addresses, as listed in Table II.

TABLE II
CONFIGURATION REGISTERS.

Type	#	Width	Usage
Control	3	32 bit	IP status, IP run command, reserved for test
Config	5	32 bit	matrix/vector size, compute mode
Address	4	64 bit	base address of KSK/matrix/vector/output

TABLE III
RESOURCE UTILIZATION ON THE XILINX VU9P FPGA.

Module	LUT	FF	BRAM	URAM	DSP
Compute Engine 0	259,318	89,894	640	294	986
Compute Engine 1	259,502	90,043	640	294	986
Platform	234,066	302,670	278	7	14
Total*	63.68%	20.41%	72.13%	61.98%	29.04%

* Measured by percentage of the total FPGA resource.

We set up a prototyping system with Intel Xeon Gold 6130@2.1 GHz and Xilinx VU9P FPGA. The benchmarks listed in Section VII are also evaluated based on this system. Moreover, we implement two SAFE compute engines in the FPGA board to achieve better performance. The SAFE floorplan in Figure 10 shows two computing engines. To balance the utilization rate of BRAM, URAM, and look-up table (LUT), we make a fine-grained tuning to optimize all these percentages below 75%. The final resource utilization is shown in Table III.

VII. PERFORMANCE EVALUATION

SAFE is implemented at $300\,\mathrm{MHz}$. We evaluate the performance with a variety of benchmarks and task settings. The baseline performance on CPU and GPU mentioned in this section are evaluated based on Intel Xeon Gold 6130@2.1 GHz with the SEAL v3.5 [58] and NVIDIA Tesla V100@1.29 GHz with CUDA library[1], respectively.

A. Comparisons on Functional Performances

In Table IV, we present the latency, resource utilization, and efficiency of the NTT and INTT units, which are the most time-consuming components. We explore three implementation strategies for the twiddle-factor ROMs and the NTT local buffer by a flexible allocation of block RAMs (BRAMs) or LUT-based distributed RAMs (dRAMs).

Besides, we compare our implementation to existing work, *i.e.*, HEAX [27], F1 [41] and Medha [48]. The NTT module of HEAX consumes the same clock cycles as this work. Nevertheless, SAFE is more compact due to the use of hardware-friendly moduli and constant-geometry dataflow. In addition, SAFE achieves high throughput because it has a total number of 60 NTT units, which can achieve $2\times$ throughput and better ATP efficiency under the same $N = 2^{12}$ settings than HEAX. F1, a high-performance implemented on ASICs, shows a significant advantage in performance. The NTT module is too large on the FPGA platform, causing more than 100% DSP slices to be consumed when instantiating the modules. Medha [48] is one of the state-of-the-art accelerators that

[1] https://github.com/vernamlab/cuFHE

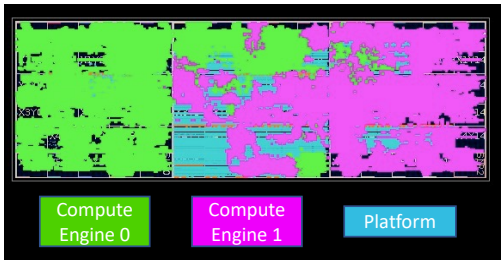


Fig. 10. Floorplan result of SAFE based on Xilinx VU9P FPGA.

support general HE operations over polynomial degrees $N = 2^{14}$ and 2^{15}. On LUT efficiency, SAFE is 2.43× better than Medha. This is because the NTT unit has simplified control logic and data path with dedicated moduli.

TABLE IV
COMPARISON OF A SINGLE NTT MODULE.

Accelerator	Latency (l)	#Mult. (p)	ATP[1] ($l \times p$)	LUT[2] (u)	BRAM	ATP[1] ($l \times u$)
SAFE (BRAM only)	6144	4	1×	3324	14	1×
SAFE (BRAM+dRAM)[3]	6144	4	1×	6508	6	1.96×
SAFE (dRAM only)	6144	4	1×	9248	0	2.78×
HEAX [27]	6144	4	1×	22316	11	6.71×
Medha [48][4]	7200	16	1×	32214	32	2.43×
F1 [41]	202	896	7.36×	-	-	-

[1] Area-time product is normalized.
[2] SAFE deploys on Xilinx FPGAs with 6-input LUTs and 36 kbit BRAMs, while HEAX deploys on Intel FPGAs with 8-input LUTs and 20 kbit BRAMs.
[3] Twiddle-factor ROM uses dRAM, and the local buffer uses BRAM.
[4] Medha uses the parameter N=2^{14}. We have normalized ATP by a computational complexity factor of 4.67 for a fair comparison.

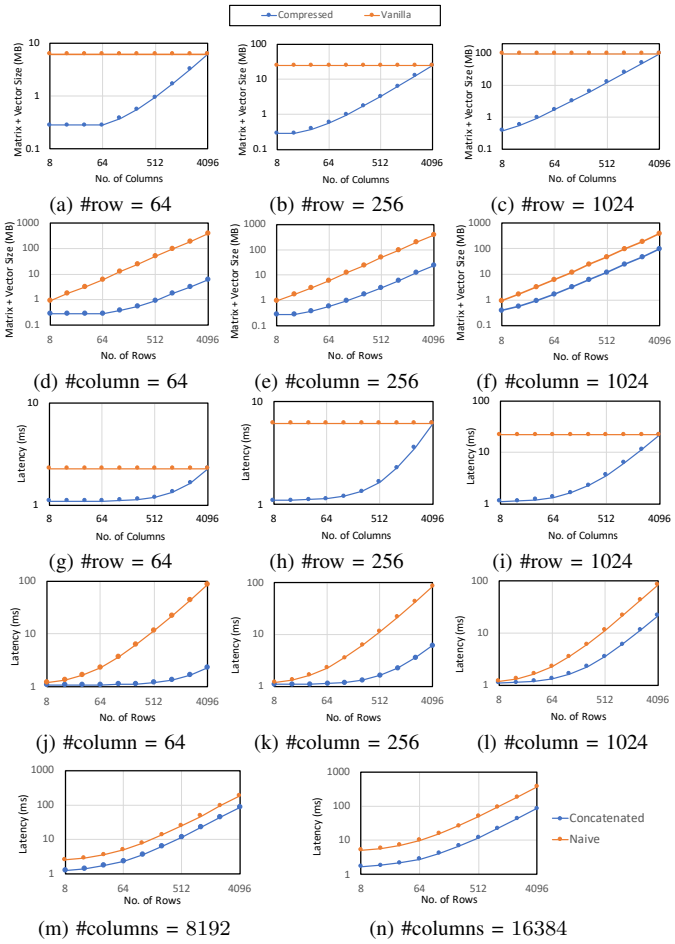


Fig. 11. The performance analysis of SAFE for different HMVP parameter settings.

B. Evaluations on HMVP

We conduct ablation studies on row and column numbers and analyze the effect on data size, computation overhead, and overall task acceleration.

1) On The Data Size: We measure the size of data, *i.e.*, the plaintext matrix and the encrypted vector, transferred between the CPU and the FPGA via PCIe. As discussed in Section III, the scalable HMVP algorithm enables the compression of multiple rows into a single plaintext, thereby effectively reducing the data size.

Fig. 11(a)∼ 11(f) demonstrates the total plaintext and ciphertext size corresponding to various task parameters. We first fix the row numbers as 64, 256, and 1024 in Fig. 11(a)∼ 11(c). The data size of the vanilla coefficient-wise encoded HMVP is kept the same despite the small column number since the padding introduces an extra dummy. The proposed compressed mode shrinks the size by packing multiple rows into one plaintext, reducing the number of encoded plaintexts to a minimum of one. Then we fix the number of columns and find that the data size is linearly related to the row number as Fig. 11(d)∼ 11(f). The compressed mode reduces about 64×, 16×, and 4× for column settings 64, 256, and 1024.

2) On The Computation Latency: As we analyzed in Section III, the computation overhead depends near-linearly on the number of rows m of the matrix. Given the fixed row number in Fig. 11(g)∼ 11(i), the latency keeps constant in the vanilla mode since the plaintext number equals the row number. In contrast, the compressed mode reduces the latency because the plaintext number is reduced to m/k, where k is the compression degree. The computation latency is reduced in the compressed mode for different row numbers, particularly when the row number is large in Fig. 11(j)∼ 11(l). This is due to data preparation and pipeline filling dominance when the row number is very small. When the column is larger than 4096 as Fig. 11(m)∼ 11(n), the concatenated modes reduce the computation overhead by 2× and 4×, respectively.

3) On the HMVP Full Task Acceleration: We evaluated the full HMVP on a CPU-FPGA platform, measuring the latency and throughput as shown in Figure 12. In Figure 12(a)(b), we compared SAFE to the CPU baseline and observed a significant offloading of computation to the FPGA, resulting in a speed-up of over 10×. Furthermore, when compared to the cuFHE implementation on V100 GPU, SAFE exhibited approximately 0.3× to 0.7× latency as shown in Figure 12(a)(b) and 4.5× throughput as Figure 12(c). Notably, the throughput showed a near-linear scaling with the m in the matrix.

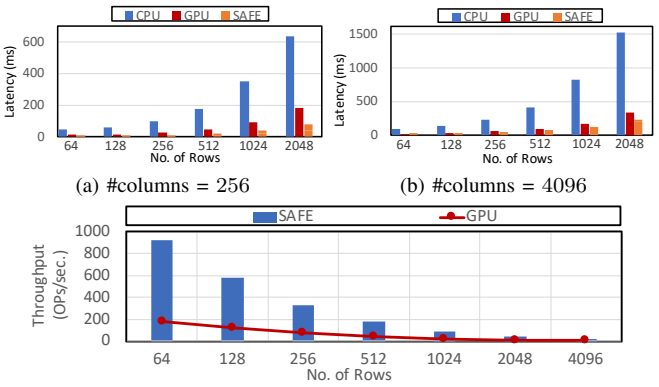


Fig. 12. Performance comparison for HMVP.

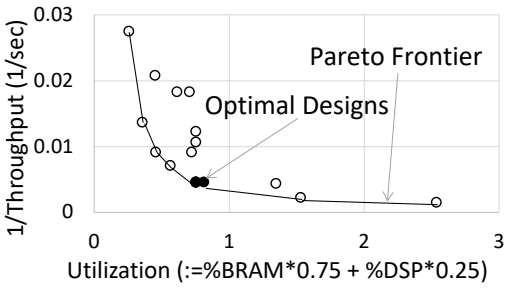


Fig. 14. Various design decisions are explored to select the best-performed designs that fit the FPGA.

C. Performance on End-to-End LR Training

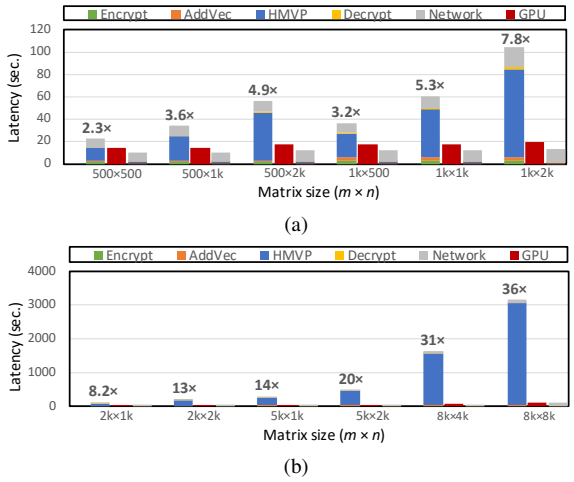


Fig. 13. Performance comparison for LR training under 10Mbps network settings.

We then evaluate the performance for heterogeneous logistic regression (*HeteroLR*) where data is partitioned vertically across parties [4]. The *HeteroLR* carries out a sample alignment process for identifying overlapping samples owned by the two parties before training. The federated model is then built based on those overlapping samples as Section II-D. Both agencies A and B need to compute the HMVP. The framework of Federated AI Technology Enabler (FATE) originally uses *Paillier*, a semi-HE algorithm [15]. In this work, we replaced *Paillier* with BFV for better algorithm and hardware acceleration. Moreover, if combined with mini-batch and matrix tiling techniques, our algorithm can support data of any scale and be deployed in multiple hardware accelerators.

We evaluate the *HeteroLR* performance using datasets of varying sizes under 10Mbps network settings, as shown in Figure 13. SAFE significantly reduces the computation overhead of the major step in *HeteroLR*. As a result, the end-to-end *HeteroLR* computation was accelerated by 2.3× to 36×. It is worth noting that cases involving large matrices (*e.g.*, 8192×4096 and 8192×8192) exhibited a higher speed-up. This is primarily due to the overhead between HMVP computation and communication in the overall process. We recommend using larger datasets to fully leverage the hardware acceleration capabilities.

D. Design Space Exploration

We thoroughly explored the design space for implementing HMVP, considering various factors such as pipeline splitting, the number of compute engine modules, the parallelism of each FU, and buffer size. The results of our exploration are illustrated in Figure 14, where each design choice is positioned based on its performance and resource utilization. From the plot in Figure 14, we identified two optimal design choices: (9-stages, 1×PACKTWOLWES, 6×NTT, 4-PE NTT, 2×compute engines) and (9-stages, 1×PACKTWOLWES, 6×NTT, 8-PE NTT, 1×compute engine). SAFE corresponds to the first design choice, which facilitates parallel compute engines. It is important to note that, to maximize overall performance, all pipeline stages have similar latency and throughput.

VIII. RELATED WORK

A. FHE accelerators

Defense Advanced Research Projects Agency (DARPA) deems FHE to be one of the most promising techniques for the future. In particular, they expect that by the year 2025, computation on ciphertext will be as fast as that on plaintext due to the improvement of the FHE algorithm and the development of highly parallel hardware [59]. CraterLake [43] builds a vector processor in a commercial 14/12nm process. It supports ResNet-20 and simple training tasks using a VLIW instruction architecture. ARK [45] develops a 7nm FHE accelerator that balances the memory overhead by on-the-fly computing and data reuse. Another 7nm accelerator, SHARP [46], balances the accuracy with the overhead by tuning the word size. Some FPGA-based accelerators [18], [60] also present comparable performance on benchmarks with the ASIC accelerators.

Besides the overhead caused by bootstrapping, HE also falls short of supporting different types of functions. In particular, BFV and CKKS cannot support non-linear functions efficiently, *e.g.*, activation function in neural networks. A solution is to approximate these functions using high-order polynomials [61]. However, this solution has the drawback of modifying an already-trained model, resulting in a degradation of model accuracy.

B. Trusted Execution Environment (TEE)

TEEs construct secure enclaves within untrusted resources such as cloud services. The enclaves can provide attestation on verifying the required security configuration and ensure the isolation of code and data from untrusted external resources during task execution. The isolation is implemented using standard cryptographic techniques between processes or virtual machines. Ohrimenko *et al.* designed data-oblivious models to prevent exploitation of side channels induced by data-dependent access patterns on Intel SGX environments [62]. Flatee is a framework for privacy-preserving FL based on TEEs. It allows efficient training of distributed models while also providing strict privacy guarantees against data-poisoning and model-poisoning attacks [63].

Although TEEs can generally meet performance requirements, they rely on trusted hardware manufacturers and are hindered by the risk of side-channel leakage. Therefore, utilizing TEEs is not feasible under certain security assumptions.

IX. Conclusion

In this work, we design a scalable HE accelerator, named SAFE, for high-performance vertical federated learning. SAFE employs an algorithm-hardware co-design approach. A low-complexity coefficient-wise encoded HMVP algorithm is adopted. Despite the vanilla mode, we further explore the compressed concatenated modes, which can encode the plaintext matrix more compactly. The proposed hardware architecture, customized for HMVP data flow, supports both spatial and temporal parallelization of function units. As part of this design, we incorporate a low-area constant-geometry NTT unit to accelerate the computationally expensive polynomial functions. SAFE is deployed as a part of the heterogeneous acceleration solution on the CPU-FPGA system. The experimental results demonstrate up to $36\times$ speed-up for LR training.

References

[1] F. Kerschbaum, *Privacy-Preserving Computation*. Springer Berlin Heidelberg, 2014.

[2] J. Cabrero-Holgueras and S. Pastrana, "SoK: Privacy-preserving computation techniques for deep learning," *Proc. Priv. Enhancing Technol.*, vol. 2021, no. 4, pp. 139–162, 2021.

[3] N. Agrawal, R. Binns, M. V. Kleek, K. Laine, and N. Shadbolt, "Exploring design and governance challenges in the development of privacy-preserving computation," in *Proc. CHI '21*. ACM, 2021, pp. 68:1–68:13.

[4] S. Hardy, W. Henecka, H. Ivey-Law, R. Nock, G. Patrini, G. Smith, and B. Thorne, "Private federated learning on vertically partitioned data via entity resolution and additively homomorphic encryption," *CoRR*, vol. abs/1711.10677, 2017.

[5] C. Chen, J. Zhou, L. Wang, X. Wu, W. Fang, J. Tan, L. Wang, A. X. Liu, H. Wang, and C. Hong, "When homomorphic encryption marries secret sharing: Secure large-scale sparse logistic regression and applications in risk control," in *Proc. KDD '21*. ACM, 2021, pp. 2652–2662.

[6] Y. Liu, T. Fan, T. Chen, Q. Xu, and Q. Yang, "FATE: An Industrial Grade Platform for Collaborative Learning with Data Protection," *Journal of Machine Learning Research*, vol. 22, pp. 1–6, 2021.

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

[8] C. Gentry, "A fully homomorphic encryption scheme," Ph.D. dissertation, Stanford University, USA, 2009. [Online]. Available: https://searchworks.stanford.edu/view/8493082

[9] Z. Brakerski, "Fully homomorphic encryption without modulus switching from classical GapSVP," in *Proc. CRYPTO '12*, vol. 7417. Springer, 2012, pp. 868–886.

[10] J. Fan and F. Vercauteren, "Somewhat practical fully homomorphic encryption," Cryptology ePrint Archive, Paper 2012/144, 2012, https://eprint.iacr.org/2012/144.

[11] Z. Brakerski, C. Gentry, and V. Vaikuntanathan, "(Leveled) fully homomorphic encryption without bootstrapping," *ACM Trans. Comput. Theory*, vol. 6, no. 3, pp. 13:1–13:36, 2014.

[12] J. H. Cheon, A. Kim, M. Kim, and Y. S. Song, "Homomorphic encryption for arithmetic of approximate numbers," in *Proc. ASIACRYPT '2017*, vol. 10624. Springer, 2017, pp. 409–437.

[13] J. H. Cheon, K. Han, A. Kim, M. Kim, and Y. Song, "A full RNS variant of approximate homomorphic encryption," in *Proc. SAC '18*, vol. 11349. Springer, 2018, pp. 347–368.

[14] ——, "Bootstrapping for approximate homomorphic encryption," in *Proc. EUROCRYPT '18*, vol. 10820. Springer, 2018, pp. 360–384.

[15] P. Paillier, "Public-key cryptosystems based on composite degree residuosity classes," in *Proc. EUROCRYPT '99*, vol. 1592. Springer, 1999, pp. 223–238.

[16] T. E. Gamal, "A public key cryptosystem and a signature scheme based on discrete logarithms," *IEEE Trans. Inf. Theory*, vol. 31, no. 4, pp. 469–472, 1985.

[17] M. Albrecht, M. Chase, H. Chen, J. Ding, S. Goldwasser, S. Gorbunov, S. Halevi, J. Hoffstein, K. Laine, K. Lauter *et al.*, "Homomorphic Encryption Standard," in *Protecting Privacy through Homomorphic Encryption*. Springer, 2021, pp. 31–62.

[18] Y. Yang, H. Zhang, S. Fan, H. Lu, M. Zhang, and X. Li, "Poseidon: Practical homomorphic encryption accelerator," in *Proc. HPCA '23*. IEEE, 2023, pp. 870–881.

[19] S. Halevi and V. Shoup, "Design and implementation of helib: a homomorphic encryption library," Cryptology ePrint Archive, Paper 2020/1481, 2020, https://eprint.iacr.org/2020/1481.

[20] A. A. Badawi, Y. Polyakov, K. M. M. Aung, B. Veeravalli, and K. Rohloff, "Implementation and performance evaluation of RNS variants of the BFV homomorphic encryption scheme," *IEEE Trans. Emerg. Top. Comput.*, vol. 9, no. 2, pp. 941–956, 2021.

[21] A. Brutzkus, R. Gilad-Bachrach, and O. Elisha, "Low latency privacy preserving inference," in *Proc. ICML '19*, vol. 97. PMLR, 2019, pp. 812–821.

[22] D. Kim, J. Park, J. Kim, S. Kim, and J. H. Ahn, "Hyphen: A hybrid packing method and its optimizations for homomorphic encryption-based neural networks," *IEEE Access*, vol. 12, pp. 3024–3038, 2024.

[23] E. Lee, J. Lee, J. Lee, Y. Kim, Y. Kim, J. No, and W. Choi, "Low-complexity deep convolutional neural networks on fully homomorphic encryption using multiplexed parallel convolutions," in *Proc. ICML '22*, vol. 162. PMLR, 2022, pp. 12 403–12 422.

[24] D. Soni, M. Nabeel, H. Gamil, O. Mazonka, B. Reagen, R. Karri, and M. Maniatakos, "Design space exploration of modular multipliers for ASIC FHE accelerators," in *Proc. ISQED '23*. IEEE, 2023, pp. 1–8.

[25] K. Han and D. Ki, "Better bootstrapping for approximate homomorphic encryption," in *Proc. CT-RSA '20*, vol. 12006. Springer, 2020, pp. 364–390.

[26] J. W. Lee, H. Kang, Y. Lee, W. Choi, J. Eom, M. Deryabin, E. Lee, J. Lee, D. Yoo, Y. S. Kim, and J. S. No, "Privacy-preserving machine learning with fully homomorphic encryption for deep neural network," *IEEE Access*, vol. 10, no. 1, pp. 30 039–30 054, 2022.

[27] M. S. Riazi, K. Laine, B. Pelton, and W. Dai, "HEAX: an architecture for computing on encrypted data," in *Proc. ASPLOS '20*. ACM, 2020, pp. 1295–1309.

[28] S. Kim, K. Lee, W. Cho, Y. Nam, J. H. Cheon, and R. A. Rutenbar, "Hardware architecture of a number theoretic transform for a bootstrappable RNS-based homomorphic encryption scheme," in *Proc. FCCM '20*. IEEE, 2020, pp. 56–64.

[29] S. Sinha Roy, F. Turan, K. Jarvinen, F. Vercauteren, and I. Verbauwhede, "FPGA-based high-performance parallel architecture for homomorphic computing on encrypted data," in *Proc. HPCA '19*, 2019, pp. 387–398.

[30] S. Kim, K. Lee, W. Cho, J. H. Cheon, and R. A. Rutenbar, "FPGA-based Accelerators of Fully Pipelined Modular Multipliers for Homomorphic Encryption," in *Proc. ReConFig '19*, 2019.

[31] S. S. Roy, K. Jarvinen, J. Vliegen, F. Vercauteren, and I. Verbauwhede, "HEPCloud: An FPGA-Based Multicore Processor for FV Somewhat Homomorphic Function Evaluation," *IEEE Trans. on Comp.*, vol. 67, no. 11, pp. 1637–1650, 2018.

[32] V. Migliore, M. M. Real, V. Lapotre, A. Tisserand, C. Fontaine, and G. Gogniat, "Hardware/Software Co-Design of an Accelerator for FV

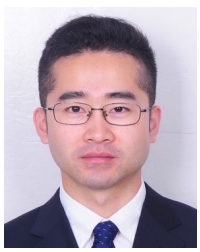
Homomorphic Encryption Scheme Using Karatsuba Algorithm," *IEEE Trans. on Comp.*, vol. 67, no. 3, pp. 335–347, 2018.
[33] E. Ozturk, Y. Doroz, E. Savas, and B. Sunar, "A Custom Accelerator for Homomorphic Encryption Applications," *IEEE Trans. on Comp.*, vol. 66, no. 1, pp. 3–16, 2017.
[34] D. B. Cousins, K. Rohloff, and D. Sumorok, "Designing an FPGA-accelerated Homomorphic Encryption Co-processor," *IEEE Trans. on Emerging Topics in Comp.*, vol. 5, no. 2, pp. 193–206, 2017.
[35] W. Jung, S. Kim, J. H. Ahn, J. H. Cheon, and Y. Lee, "Over 100x Faster Bootstrapping in Fully Homomorphic Encryption through Memory-centric Optimization with GPUs," *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, vol. 2021, no. 4, pp. 114–148, 2021.
[36] W. Jung, E. Lee, S. Kim, J. Kim, N. Kim, K. Lee, C. Min, J. H. Cheon, and J. H. Ahn, "Accelerating Fully Homomorphic Encryption through Architecture-centric Analysis and Optimization," *IEEE Access*, vol. 9, pp. 98 772–98 789, 2021.
[37] P. G. M. R. Alves, J. N. Ortiz, and D. F. Aranha, "Faster homomorphic encryption over gpgpus via hierarchical DGT," in *Proc. FC '21*, vol. 12675. Springer, 2021, pp. 520–540.
[38] A. A. Badawi, C. Jin, J. Lin, C. F. Mun, S. J. Jie, B. H. M. Tan, X. Nan, K. M. M. Aung, and V. R. Chandrasekhar, "Towards the alexnet moment for homomorphic encryption: HCNN, the first homomorphic CNN on encrypted data with gpus," *IEEE Trans. Emerg. Top. Comput.*, vol. 9, no. 3, pp. 1330–1343, 2021.
[39] W. Dai and B. Sunar, "cuHE: A homomorphic encryption accelerator library," in *Proc. BalkanCryptSec 2015*, E. Pasalic and L. R. Knudsen, Eds., vol. 9540. Springer, 2015, pp. 169–186.
[40] Z. Wang, P. Li, R. Hou, Z. Li, J. Cao, X. Wang, and D. Meng, "He-booster: An efficient polynomial arithmetic acceleration on gpus for fully homomorphic encryption," *IEEE Trans. Parallel Distributed Syst.*, vol. 34, no. 4, pp. 1067–1081, 2023.
[41] A. Feldmann, N. Samardzic, A. Krastev, S. Devadas, R. Dreslinski, C. Peikert, and D. Sanchez, "F1: A Fast and Programmable Accelerator for Fully Homomorphic Encryption," in *Proc. MICRO '21'*, 2021, pp. 238–252.
[42] S. Kim, J. Kim, M. J. Kim, W. Jung, M. Rhu, J. Kim, and J. H. Ahn, "BTS: An Accelerator for Bootstrappable Fully Homomorphic Encryption," in *Proc. ISCA '22*, 2022, pp. 711–725.
[43] N. Samardzic, A. Feldmann, N. Manohar, N. Genise, K. Eldefrawy, C. Peikert, A. Arbor, and D. Sanchez, "CraterLake : A Hardware Accelerator for Efficient Unbounded Computation on Encrypted Data," in *Proc. ISCA '22*, 2022.
[44] R. Geelen, M. V. Beirendonck, H. V. L. Pereira, B. Huffman, T. McAuley, B. Selfridge, D. Wagner, G. D. Dimou, I. Verbauwhede, F. Vercauteren, and D. W. Archer, "BASALISC: programmable hardware accelerator for BGV fully homomorphic encryption," *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, vol. 2023, no. 4, pp. 32–57, 2023.
[45] J. Kim, G. Lee, S. Kim, G. Sohn, M. Rhu, J. Kim, and J. H. Ahn, "ARK: fully homomorphic encryption accelerator with runtime data generation and inter-operation key reuse," in *Proc. MICRO '22*. IEEE, 2022, pp. 1237–1254.
[46] J. Kim, S. Kim, J. Choi, J. Park, D. Kim, and J. H. Ahn, "SHARP: A short-word hierarchical accelerator for robust and practical fully homomorphic encryption," in *Proc. ISCA '23*. ACM, 2023, pp. 18:1–18:15.
[47] X. Ren, Z. Chen, Z. Gu, Y. Lu, R. Zhong, W. Lu, J. Zhang, Y. Zhang, H. Wu, X. Zheng, H. Liu, T. Chu, C. Hong, C. Wei, D. Niu, and Y. Xie, "CHAM: A customized homomorphic encryption accelerator for fast matrix-vector product," in *Proc. DAC '23*. IEEE, 2023, pp. 1–6.
[48] A. C. Mert, Aikata, S. Kwon, Y. Shin, D. Yoo, Y. Lee, and S. S. Roy, "Medha: Microcoded hardware accelerator for computing on encrypted data," *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, vol. 2023, no. 1, pp. 463–500, 2023.
[49] H. Chen, W. Dai, M. Kim, and Y. Song, "Efficient homomorphic conversion between (ring) LWE ciphertexts," in *Proc. ACNS '21*, vol. 12726. Springer, 2021, pp. 460–479.
[50] C. Juvekar, V. Vaikuntanathan, and A. Chandrakasan, "GAZELLE: A Low Latency Framework for Secure Neural Network Inference," in *Proc. USENIX Security '18*, 2018, pp. 1651–1668.
[51] Z. Huang, W. Lu, C. Hong, and J. Ding, "Cheetah: Lean and fast secure two-party deep neural network inference," in *Proc. USENIX Security '22*. USENIX Association, 2022, pp. 809–826.
[52] P. Longa and M. Naehrig, "Speeding up the number theoretic transform for faster ideal lattice-based cryptography," in *Proc. CANS '16*, S. Foresti and G. Persiano, Eds., vol. 10052, 2016, pp. 124–139.
[53] M. C. Pease, "An Adaptation of the Fast Fourier Transform for Parallel Processing," *Journal of ACM*, vol. 15, no. 2, pp. 252–264, 1968.
[54] D. D. Chen, N. Mentens, F. Vercauteren, S. S. Roy, R. C. C. Cheung, D. C. Pao, and I. Verbauwhede, "High-Speed Polynomial Multiplication Architecture for Ring-LWE and SHE Cryptosystems," *IEEE Trans. Circuits Syst. I Regul. Pap.*, vol. 62-I, no. 1, pp. 157–166, 2015.
[55] J. Cooley and J. Tukey, "An Algorithm for the Machine Calculation of Complex Fourier Series," *Mathematics of Computation*, vol. 19, no. 90, pp. 297–301, 1965.
[56] S. Liu, C. Kuo, Y. Mo, and T. Su, "An area-efficient, conflict-free, and configurable architecture for accelerating NTT/INTT," *IEEE Trans. Very Large Scale Integr. Syst.*, vol. 32, no. 3, pp. 519–529, 2024.
[57] N. Zhang, B. Yang, C. Chen, S. Yin, S. Wei, and L. Liu, "Highly Efficient Architecture of NewHope-NIST on FPGA using Low-Complexity NTT/INTT," *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, vol. 2020, no. 2, pp. 49–72, 2020.
[58] "Microsoft SEAL (release 3.5)," https://github.com/Microsoft/SEAL, Apr. 2020, microsoft Research, Redmond, WA.
[59] DARPA, "Data Protection in Virtual Environments," 2021. [Online]. Available: https://www.darpa.mil/news-events/2021-03-08
[60] R. Agrawal, L. de Castro, G. Yang, C. Juvekar, R. T. Yazicigil, A. P. Chandrakasan, V. Vaikuntanathan, and A. Joshi, "FAB: an fpga-based accelerator for bootstrappable fully homomorphic encryption," in *Proc. HPCA '23*. IEEE, 2023, pp. 882–895.
[61] E. Hesamifard, H. Takabi, and M. Ghasemi, "CryptoDL: Deep neural networks over encrypted data," *CoRR*, vol. abs/1711.05189, 2017.
[62] O. Ohrimenko, F. Schuster, C. Fournet, A. Mehta, S. Nowozin, K. Vaswani, and M. Costa, "Oblivious multi-party machine learning on trusted processors," in *Proc. USENIX Security '16*. USENIX Association, 2016, pp. 619–636.
[63] A. Mondal, Y. More, R. H. Rooparaghunath, and D. Gupta, "Poster: FLATEE: federated learning across trusted execution environments," in *Proc. EuroS&P '21*. IEEE, 2021, pp. 707–709.

Zhaohui Chen received a Ph.D. degree in computer science and technology from the University of Chinese Academy of Sciences, China.

He is currently working with DAMO Academy, Alibaba Group. His research interests include applied cryptography, hardware security, and homomorphic encryption.

Zhen Gu received the bachelor's degree and the Ph.D. degree in microelectronics from School of Integrated Circuits, Tsinghua University, Beijing, China. After graduation, he was employed as Research Scientist in DAMO Academy of Alibaba Group, China. His research interests include cryptographic and privacy enhancing algorithms and VLSI designs.

Yanheng Lu received the B.S. and M.S. degrees in microelectronics from Fudan University, Shanghai, China, in 2013 and 2016, respectively. He is currently a Research Scientist in Computing Technology Lab at Alibaba DAMO Academy. Before joining Alibaba, he also worked with the China System Lab, IBM. His current research interests include computer architecture, storage systems, and domain-specific architectures.



Xuanle Ren received the B.S. degree in microelectronics from Peking University, Beijing, China, in 2012, and the Ph.D. degree in electrical and computer engineering from Carnegie Mellon University, Pittsburgh, PA, USA, in 2018.

He has been working as a Research Scientist with Alibaba Group, Hangzhou, China, and Bitmain, Beijing, China. His research is focused on privacy-preserving computing, design of secure computer architecture and hardware security.

Ruiguang Zhong received the B.S and M.S. degrees from the Beijing University of Posts and Telecommunications in 2016 and 2019. He is now a researcher in the technology planning department of NIO, where his main research interests encompass heterogeneous computing, large language model compression, and privacy computing.

Wen-jie Lu received his Doctoral Degree from Tsukuba University, Japan. His research interests include privacy-preserving machine learning and applied homomorphic encryption.

Jiansong Zhang received his bachelor's and master's degrees from Tsinghua University, Beijing, China, in 2004 and 2006, respectively, and the Ph.D. degree from the Hong Kong University of Science and Technology in 2014.

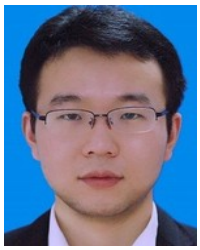
He is a Research Scientist in China Telecom and working on networking and cloud computing. Before that, he was research scientist in NIO, Alibaba Damo Academy and Microsoft Research Asia. His research interests include heterogeneous computing for Cloud and AIoT systems, networking systems, software-hardware co-design and vehicle computing. He has published dozens of papers in top-tier conferences like Sigcomm, NSDI, Mobicom, Ubicomp, Mobisys, HotNets, HotChips, FCCM, Infocom, and MLSys.

Yichi Zhang received the BS degree in microelectronics from Tsinghua University, Beijing, China, in 2022. He is currently working toward the PhD degree with the School of Integrated Circuits, Tsinghua University. His research interests include reconfigurable computer architecture, domain-specific accelerators, and computational genomics.

Hanghang Wu received his bachelor's degree in computer science and technology from Tsinghua University, Beijing, China. WU is a staff engineer with Ant Group, Beijing, 100020, China. His research interests include privacy-preserving computation and verifiable computation. Contact him at hanghang.whh@antgroup.com.

Xiaofu Zheng received a master's degree in Communication Engineering from the Xidian University, Xi'an, China. He works with Ant Group, Beijing, China. His research interests include privacy-preserving computation and verifiable computation.

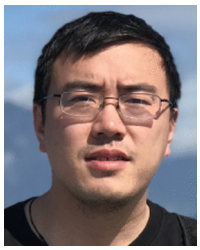
Heng Liu received a master's degree in circuits and systems from Zhejiang University, China. She currently works with SiOrigin Intelligent Technology Co., Ltd. Her research interests include computer architecture, power management, and SoC integration.



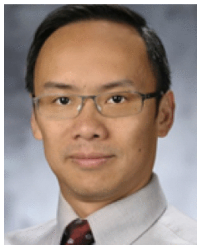
Tingqiang Chu received his M.S. degree in electronics & communication engineering from Nanjing University of Posts and Telecommunications, Nanjing, China. CHU is a staff engineer with Ant Group, Shanghai, 200010, China. His research interests include blockchain, privacy-preserving computation and verifiable computation. Contact him at chutingqiang.ctq@antgroup.com.

Cheng Hong received his PhD degree from the University of Chinese Academy of Sciences, China in 2012. He is currently the director of Cryptography and Privacy Research of Ant Group, China. His research interests include information security and applied cryptography.

Changzheng Wei received a master's degree in microelectronics from the Chinese Academy of Sciences, Shanghai, China. WEI is a senior staff engineer with Ant Group, Shanghai, 200010, China. His research interests include blockchain, privacy-preserving computation and verifiable computation. Contact him at changzheng.wcz@antgroup.com.



Dimin Niu Dimin Niu received the PhD degree in computer science from the Pennsylvania State University, University Park, in 2012. He is a research scientist with Computing Technology Lab, Alibaba DAMO Academy. Prior to joining Alibaba, he was a staff memory architecture with Memory Solutions Laboratory, Samsung Semiconductor Inc. His current research interests include computer architecture, memory architecture, storage system, process-in-memory, and domain specific architectures.



Yuan Xie (Fellow, IEEE) received the B.S. degree from Tsinghua University, Beijing, China, and the M.S. and Ph.D. degrees from the Department of Electrical Engineering, Princeton University, Princeton, NJ, USA, in 2002.

He was with IBM Microelectronics Division's Worldwide Design Center, Burlington, NJ, USA; Pennsylvania State University, State College, PA, USA; AMD Research, Beijing; and University of California at Santa Barbara, Santa Barbara, CA, USA.

Dr. Xie is a recipient of the NSF CAREER Award and the IEEE Computer Society Edward J. McCluskey Technical Achievement Award in 2020. He is a Fellow of ACM and AAAS.