IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received 16 November 2023, accepted 9 December 2023, date of publication 26 December 2023,
date of current version 3 January 2024.

Digital Object Identifier 10.1109/ACCESS.2023.3347521

RESEARCH ARTICLE

UniFL: Accelerating Federated Learning Using Heterogeneous Hardware Under a Unified Framework

BIYAO CHE[1], ZIXIAO WANG[1], YING CHEN[1], LIANG GUO[2], (Member, IEEE), YUAN LIU[1], YUAN TIAN[1], AND JIZHUANG ZHAO[1]

[1]China Telecom Research Institute, Beijing 102209, China
[2]Institute of Cloud Computing and Big Data of CAICT, Beijing 100191, China

Corresponding author: Zixiao Wang (wangzx15@chinatelecom.cn)

This work was supported in part by the National Key Research and Development Program of China under Grant 2021ZD0113003, and in part by the China Telecom Cloud Computing Key Research Program under Grant T-2023-03.

ABSTRACT Federated learning (FL) is now considered a critical method for breaking down data silos. However, data encryption can significantly increase computing time, limiting its large-scale deployment. While hardware acceleration can be an effective solution, existing research has largely focused on a single hardware type, which hinders the acceleration of FL across the various heterogeneous hardware of the participants. In light of this challenge, this paper proposes a novel FL acceleration framework that supports diverse types of hardware. Firstly, we conduct an analysis of the key elements of FL to clarify our accelerator design goals. Secondly, a unified acceleration framework is proposed, which divides FL into four layers, providing a basis for the compatibility and implementation of heterogeneous hardware acceleration. After that, based on the physical properties of three mainstream acceleration hardware, i.e., GPU, ASIC and FPGA, the architecture design of corresponding heterogeneous accelerators under the framework is detailed. Finally, we validate the effectiveness of the proposed heterogeneous hardware acceleration framework through experiments. For specific algorithms, our implementation achieves a state of the art acceleration effect compared to previous work. For the end-to-end acceleration performance, we gain 12×, 7.7× and 2.2× improvement on GPU, ASIC and FPGA respectively, compared to CPU in large-scale vertical linear regression training tasks.

INDEX TERMS Federated learning, hardware acceleration, homomorphic encryption, privacy preserving.

I. INTRODUCTION

With the widespread application of big data and cloud computing technologies, the demand for machine learning and deep learning data is ever-increasing. As a result, the need for data sharing and fusion is becoming more and more pronounced. However, the emergence and strict enforcement of laws and regulations related to privacy data protection have severely constrained the collection, transmission, retention, and processing of personal privacy data. In view of the demand for data privacy and security protection, as well as the need to break down data silos, federated learning (FL) technology [1], [2] has emerged as an ideal solution. FL which has attracted widespread attention from both academia and industry, is a special distributed learning scheme that enables multiple organizations to jointly model their data while meeting requirements related to user privacy protection, data security, and government regulations.

FL adopts various technologies such as garbled circuits [3], differential privacy [4], and homomorphic encryption [5] to protect data security, with the latter being the most practical as it allows for algebraic operations to be performed on ciphertext without decryption. While fully homomorphic encryption (FHE) is relatively resource-intensive in terms of computation and time consumption, partially homomorphic encryption (PHE) is more widely used in industry.

The associate editor coordinating the review of this manuscript and approving it for publication was Long Wang.

Particularly, the RSA [6] and Paillier [7] algorithms based on large integer modular operations, which provide homomorphic multiplication and homomorphic addition properties respectively, play an important role in algorithm design for FL tasks such as sample alignment and model aggregation [8], [9], [10], [11], [12].

However, the PHE algorithms come with additional computational and I/O overhead, which limits the commercial deployment of FL. Firstly, there are a large number of modular operations involved in the process of encryption, decryption, and homomorphic computation, which are not computationally friendly to CPU and would significantly increase the clock cycles consumed by calculations. Secondly, to ensure security, it is usually required that the key bit width is greater than 1024 or 2048, which causes significant data inflation during encryption and makes data access an additional bottleneck. Taking the 1024-bit Paillier algorithm as an example, the maximum bit width of the ciphertext after encryption is twice that of the key bit width, i.e., 2048. Gradients are usually 32-bit floating-point numbers in plain text, and encrypting them may result in a 64× increase in bit width. These bottlenecks severely slow down the training speed of FL when the PHE algorithm is deployed directly on CPU. Previous work [13] has shown that when performing a two-party vertical FL task based on the FL framework Federated AI Technology Enabler (FATE) [14] on the CPU, encryption, decryption, and ciphertext space computations account for 86.4% of the total time, while key generation and other operations (including some plaintext computations, gradient updates, and local data access time) only account for 13.6%.

With the development of heterogeneous computing hardware, GPU, ASIC, and FPGA have demonstrated significant speed-up performance in the field of deep learning [15], [16], [17], [18], [19], [20]. How to apply them to FL to address the above bottlenecks has become a hot research topic in academia and industry [21], [22], [23], [24], [25], [26]. Among them, the GPU has the best flexibility and scalability, and has always received the most attention [21], [22]. FPGA, with its reconfigurable characteristics and rich logic resources, has also gained considerable research interest in recent years [24], [25], [26]. For large-scale commercial applications, ASIC has gradually received attention due to its advantages of low cost and power consumption [23]. Several vendors have developed multiple commercial ASIC acceleration cards designed for specific cryptographic operations, such as Intel QuickAssist Technology (QAT) [27], Cavium's crypto offload adaptors (Nitrox) [28], and Exar Compression and Security Acceleration Card (DX2040) [29].

Existing research has mostly focused on using a single type of hardware (i.e., one of GPU, ASIC, or FPGA) to accelerate FL tasks. However, FL requires multiple participants to collaborate on the training process, and these participants often come from different industries. In many practical cases of training, different participants may possess heterogeneous computing devices, and the available hardware resources within a data center may also change over time. For instance, when two enterprises collaborate on training a federated model, enterprise A may have available GPU cards, while enterprise B may have FPGA cards. Existing studies have not provided a clear definition on data exchange mechanisms for this scenario, which makes it difficult to accelerate training. Furthermore, if the available hardware resources of a participant change before a training task starts, for instance, if enterprise A's GPU resources are occupied by other workloads but they still have idle ASIC cards, this requires the training task to seamlessly switch to the ASIC acceleration card to make full use of the resources.

Therefore, developing an acceleration framework that supports multiple heterogeneous hardware and effectively utilizes existing hardware resources of different participants is a crucial challenge that needs to be addressed. The main contributions of this paper are as follows:

- We conduct a theoretical analysis of the key elements in FL acceleration, providing a theoretical basis for the proposed unified acceleration framework and laying a foundation for future work in this area.
- We propose a unified acceleration framework that supports multiple types of heterogeneous hardware, allowing different participants to use different types of acceleration cards. The various heterogeneous hardware in this framework supports flexible loading and unloading. Furthermore, based on the characteristics of GPU, ASIC, and FPGA, corresponding accelerator architectures are designed.
- We validate the effectiveness of the proposed hardware acceleration solution through experiments. In the implementation on our experimental platform, which is based on NVIDIA Tesla T4 GPU, Intel QAT 8970 ASIC, and Intel Stratix 10 GX2800 FPGA, we verified their performance on modular multiplication (ModMult), modular exponentiation (ModExp), and the Paillier encryption operations. Comparative experiments showed that our design achieved state-of-the-art acceleration effects.
- We evaluated the actual performance of the designed accelerator on practical FL training tasks. We achieve end-to-end acceleration performance gains of 12×, 7.7×, and 2.2× for GPU, ASIC, and FPGA, respectively, compared to CPU in large-scale vertical linear regression training tasks.

The remainder of the paper is organized as follows. Section II presents related work. Section III briefly reviews common PHE algorithms and analyzes the key elements of accelerator design for FL. In Section IV, we propose a unified FL heterogeneous acceleration framework, and in Section V, we provide detailed descriptions of the architecture designs for various heterogeneous hardware. Section VI conducts experiments and performance comparison analysis. Section VII summarizes the paper and suggests future research directions.

II. RELATED WORK

In recent years, researchers have proposed hardware-accelerated solutions for the high computational overhead of PHE, which can be categorized into three types: GPU, ASIC, and FPGA-based acceleration.

GPUs have high parallelism and large computational resources, which can process large amounts of data and complex mathematical operations in a short time, and have been proven to greatly improve the efficiency and accuracy of machine learning applications. It is a natural idea to accelerate big number modular operations, and consequently RSA and other cryptographic algorithms, based on GPUs. Feasible solutions were proposed more than a decade ago [30], [31], [32], and subsequent researchers attempted to further optimize performance based on newer hardware [21], [33], [34]. Cooperative Groups Big Numbers (CGBN) [35] is an open-source library proposed by NVIDIA Research, which accelerates multiple precision arithmetic (big numbers) based on Compute Unified Device Architecture (CUDA), providing implementations of Montgomery multiplication(MontMult) [36] and big number modular operations, but does not implement commonly used cryptographic algorithms in FL. In recent work, some new FL architectures [37], [38], [39] support GPU acceleration, but these works do not focus on PHE-based FL algorithms. Mainstream AI frameworks such as Pytorch [40] and Tensorflow [41] do not provide support for big number ModMult and ModExp, making it incompatible with FL's PHE algorithms. Currently only relatively few works consider engineering optimizations based on GPUs. In the work of Cheng et al. [22], they attempted to use GPUs to accelerate the federated logistic regression training process, optimizing the storage, computation, and IO processes, and achieved an acceleration effect of 88.4× and 44.9×, respectively, in horizontal and vertical logistic regression training tasks using NVIDIA V100 cards. In the work of Zeng et al. [42], both computational and communication costs were considered, achieving even better acceleration results.

Recently, some studies have focused on ASIC implementation of ModMult, ModExp, and cryptographic operation [43], [44], [45], [46]. Due to the complexity of FL algorithm protocols and high cost of chip fabrication, more enterprises tend to use commercial ASIC cryptographic accelerator cards to improve computing efficiency. Several vendors provide ASIC solutions, such as Intel QAT, Nitrox, and DX2040, for cryptographic algorithm unloading. Previous research has investigated the offloading of the Paillier algorithm in FL using Intel QAT [23], [47]. QAT only provides ModExp interface, not a Paillier algorithm interface. Zhuang [47] considered using QAT for ModExp operations in Paillier algorithm, while other calculations will still be processed by the CPU. This was used to speed up the encryption process of Int data for the encrypted database CryptDB, achieving a 3-4× speedup. Zhou and Hua [23] proposed a PHE offloading framework QHCS (QAT-based Homomorphic Encryption Scheme) based on QAT accelerator, focusing on asynchronous offloading of Paillier algorithm. It is verified that using QAT offloading homomorphic encryption calculation can greatly shorten the computing time and improve the Paillier encryption throughput to 26.87KOPS. Although Zhou et al. have considered applying QAT to FL modeling, they only focused on offloading Paillier encryption calculations, and did not study other computationally intensive calculations in FL.

FPGA provides flexibility between GPU and ASIC with its hardware programmability, and has attracted increasing attention from academia in recent years. FPGA supports hardware circuit programming, and can overcome the defects of fixed algorithms of ASIC. With sophisticated circuit design, FPGA can also achieve good acceleration effects and serve as a prototype design for ASIC chip. Like GPU and ASIC accelerators for FL, most FPGA accelerators adopt hardware-optimized Montgomery modular multiplication algorithm. San and At [24] accelerated ModMult and ModExp on Xilinx Zynq 7000, and achieved throughput of 366.3KOPS and 176.1OPS, respectively. In the more recent work of this research team [25], Paillier algorithm was implemented on the same hardware platform, with throughput of 680.3OPS for 1024-bit Paillier. Yang et al. [48] accelerated Paillier algorithm on Xilinx XCVU9P, and reported a Paillier acceleration throughput of about 5200OPS. Previous works mostly focused on designing single circuits to support a specific algorithm, i.e., ModMult, ModExp or Paillier, which greatly impaired their performance in end-to-end FL. In our previous work [13], we designed a flexible FPGA accelerator circuit that can simultaneously accelerate ModMult, ModExp and Paillier algorithms. Therefore, on Intel Stratix 10 GX2800 FPGA with similar on-chip resources as the work of Yang et al. [48], we achieved a performance improvement of about 3× compared with their work in end-to-end FL experiments.

The accelerator designs in all of the above studies are based on one kind of specific hardware. However, FL contains two or more participants, so there exists the problem of device heterogeneity. Although there have been study evaluating the impact of heterogeneity on FL from an algorithmic perspective [49], as our best known, no study has focused on designing a general acceleration framework applicable to heterogeneous hardware in FL. Therefore, in this paper, we propose a unified FL acceleration framework that supports GPU, ASIC, and FPGA hardware simultaneously, which has the following advantages:

1) The framework promotes ease of adoption and scalability of FL on various devices, allowing commonly used models to be trained across different hardware without the need for additional hardware-specific optimizations. This can make full use of the heterogeneous hardware available to the participants, providing a higher training acceleration ratio compared to traditional single hardware acceleration.

2) The unified acceleration framework provides a uniform interface for model developers, which means that they do not need to be concerned about specific hardware implementations, allowing them to focus more on algorithm innovation.

In summary, the proposed unified FL acceleration framework promotes ease of adoption, enhances computing flexibility, and reduces development complexity and costs.

III. ANALYSIS OF KEY ELEMENTS FOR ACCELERATING FEDERATED LEARNING

As mentioned earlier, mature FL frameworks typically utilize a variety of PHE algorithms. To improve the performance of PHE computation, it is essential to understand these algorithms. RSA and Paillier are two classic PHE algorithms, respectively satisfying the homomorphic properties of multiplication and addition. They are widely used in FL frameworks such as Tensorflow Federated (TFF) [50], FATE [14], and PySyft [51]. These two algorithms are also frequently considered as acceleration targets by FL hardware accelerator designers. Before discussing the specific design and implementation of the accelerator, we briefly review the principles of these two algorithms and analyze the acceleration potential of FL schemes based on them, which leads us to our design goals and architecture.

A. REVIEW AND ANALYSIS OF PHE ALGORITHMS IN FEDERATED LEARNING

1) REVIEW OF THE RSA ALGORITHM

RSA algorithm [6] was proposed in 1977, and then widely used in the field of encryption and digital signatures. Given RSA public key (n, e) and private key (n, d), the encryption process for plaintext $m(m < n)$ is as follows:

$$c = [\![m]\!] = m^e \bmod n \tag{1}$$

The notation $[\![\cdot]\!]$ is used in this paper to represent encrypted data. The decryption calculation is:

$$m = c^d \bmod n \tag{2}$$

The RSA algorithm satisfies the homomorphic multiplication property. Given two ciphertexts $[\![m_1]\!]$ and $[\![m_2]\!]$ encrypted with RSA, then

$$[\![m_1]\!] \otimes [\![m_2]\!] = \left([\![m_1]\!] \cdot [\![m_2]\!]\right) \bmod n = [\![m_1 \cdot m_2]\!] \tag{3}$$

where "$\otimes$" represents the ciphertext multiplication. The equation (3) shows that each RSA ciphertext multiplication can be implemented with one ModMult operation.

2) REVIEW OF THE PAILLIER ALGORITHM

The Paillier algorithm [7] is a public-key encryption scheme introduced by Pascal Paillier in 1999. It is based on the Composite Residuosity Class Problem and is characterized by additively homomorphism.

Given the public key (n, g) and private key (λ, μ) for Paillier encryption, where the plaintext to be encrypted is denoted as m, and $m < n$. r is a random number selected from $\mathbb{Z}^*_{n^2}$ (multiplicative group of integers modulo n^2), and the encryption operation is as follows:

$$c = [\![m]\!] = g^m r^n \bmod n^2 \tag{4}$$

To reduce computational complexity, the privacy computing platforms such as FATE [14] usually use $g = n + 1$, then the encryption operation is simplified to:

$$c = [\![m]\!] = (n+1)^m r^n \bmod n^2 = (m \cdot n + 1) \cdot r^n \bmod n^2 \tag{5}$$

Decryption can be expressed as follows:

$$m = \mu \cdot L(c^\lambda \bmod n^2) \bmod n \tag{6}$$

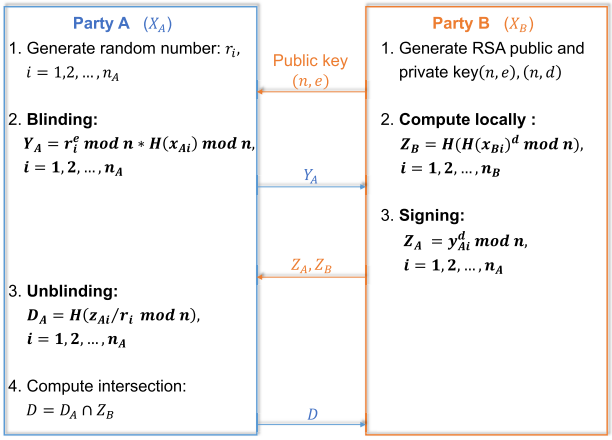


FIGURE 1. Flow diagram of PSI_RSA algorithm.

where $L(x) = (x-1)/n$. The homomorphic addition of two Paillier ciphertexts satisfies:

$$[\![m_1]\!] \oplus [\![m_2]\!] = \left([\![m_1]\!] \cdot [\![m_2]\!]\right) \bmod n^2 = [\![m_1 + m_2]\!] \tag{7}$$

where "$\oplus$" represents the ciphertext addition, and it can be implemented using ModMult. Scalar multiplication can be decomposed into multiple addition operations, therefore, the Paillier algorithm satisfies the homomorphic scalar multiplication, which can be achieved using ModExp.

3) APPLICATIONS OF PHE ALGORITHM IN FEDERATED LEARNING

FL includes horizontal federated learning (HFL) and vertical federated learning (VFL). In HFL, participants possess different samples with overlapping features. Collaborative modeling allows for an expanded training sample set. In VFL, participants have overlapping samples, but the sample features are different. Collaborative training enablesthe model to consider more feature information. In this section, we take the example of a two-party vertical federated logistic regression (VFL LR) training task to introduce the applications of these two PHE algorithms, RSA and Paillier, in FL.

The Private Set Intersection algorithm based on RSA (PSI_RSA) [8] plays a role in sample alignment. In a vertical

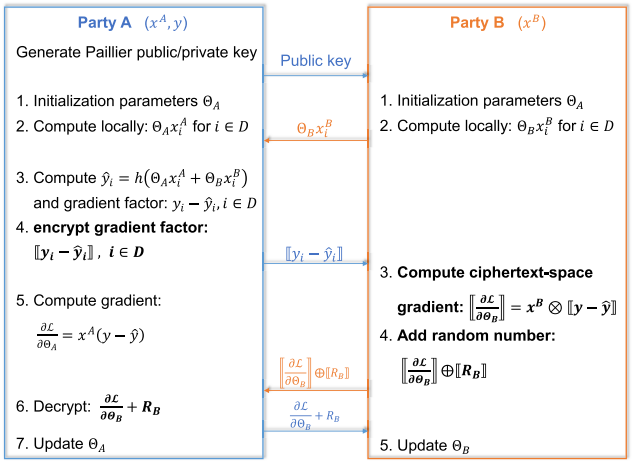


FIGURE 2. Flow diagram of VFL LR training with Paillier alogrithm.

FL scenario, assuming that participant A has data and labels ($\tilde{x}^A$, $\tilde{y}$), and participant B has data ($\tilde{x}^B$). There is partial overlap in the data sample IDs, but the features are different. Therefore, sample alignment is required before training, i.e., to identify the common samples owned by the participants based on their IDs. We denote X_A, X_B as the ID sets of A and B's data, and n_A, n_B as the data sizes of A and B. Fig. 1 shows the process of using PSI_RSA to find the intersection D of X_A and X_B without revealing their all data IDs, where $H(\cdot)$ is the Hash function.

The Paillier algorithm is used to protect intermediate results for backpropagation and parameter update without revealing raw data in FL. After PSI_RSA, the data sets of both parties are denoted as (x^A, y), (x^B), and the training process for VFL LR [9] is shown in Fig. 2. Here, $h(\cdot)$ represents the Sigmoid function.

As can be seen from the algorithm flow in Fig. 1 and Fig. 2, RSA encryption and decryption are used in PSI_RSA to achieve blinding and signing, while Paillier is used in VFL LR to encrypt gradient factors to prevent leaking raw data. The gradient is calculated through matrix multiplication in the Paillier encryption space (multiplication of plaintext matrix with encrypted gradient factor vector), and random perturbations are added to the gradient via Paillier homomorphic addition. Similarly, vertical linear regression and vertical XGBoost training tasks can also be implemented based on RSA and Paillier. In HFL algorithms, data protection is also achieved through PHE, such as encrypting the gradients calculated locally by each participant using Paillier, summing the encrypted gradients by the agency and sending them back to each participant to update their respective parameters.

B. KEY ELEMENTS OF FEDERATED LEARNING ACCELERATOR DESIGN

By reviewing the cryptographic algorithms involved in FL in the previous subsection, we find the following characteristics:

1) Different cryptographic algorithms will be used in different stages of FL tasks. Specifically, RSA will be used for data processing, while Paillier algorithm will be used during training. In addition, computations in the encryption space and computations between plaintext and encrypted spaces will be involved at various stages of the process.
2) Various cryptographic algorithms take ModMult or ModExp as their fundamental operations, and create different algorithms through different permutations and combinations. Previous researchers [52] have categorized cryptographic operations in FL into nine types, and through a time slice analysis, their conclusion suggests that 95% of computation time spent on cryptographic operations involved in FL is consumed by ModMult or ModExp calculations.
3) FL requires at least two participants, and existing FL frameworks only define the relevant algorithms, without defining the lower-level hardware implementation. As a result, participants may have different types of heterogeneous computing hardware, which poses a challenge for designing a unified accelerator architecture.

In light of above findings, we propose three objectives for this paper:

1) The designed heterogeneous accelerator must have good flexibility and can simultaneously support a variety of cryptographic algorithms in FL.
2) As ModMult and ModExp calculation are the main acceleration targets, the accelerator needs to provide a considerable acceleration ratio for these two basic operators.
3) A unified framework needs to be defined for accelerators to achieve compatibility with a variety of heterogeneous computing hardware.

Based on the analysis, the computational cost of FL is mainly concentrated on encryption and homomorphic computation, with a large number of underlying calls to ModMult and ModExp operators. Therefore, we consider designing a software and hardware collaborative acceleration scheme around these two operators. GPU, ASIC and FPGA are all commonly used computing devices. We briefly analyze the advantages and disadvantages of these three hardware in the context of FL computational acceleration as follows:

- GPU has the highest performance and energy consumption, with powerful parallel computing capabilities and high flexibility, enabling the implementation of various cryptography algorithms. However, the underlying hardware of GPU is not programmable, so it cannot directly perform large-width integer calculations and requires custom data structures and corresponding algorithms.
- ASIC has the highest energy efficiency for specific algorithms with the lowest flexibility. Custom operators need to be finely designed for algorithms not supported by the ASIC chip.
- FPGA has moderate performance and energy consumption. Because of its programmable capability, it can

support custom data structures well, but the hardware resources on the chip are limited. Overly complex circuit designs can lead to a sharp decrease in work frequency (Fmax), thereby compromising performance.

In the next section, we propose a unified FL heterogeneous acceleration framework considering the above hardware features.

IV. THE UNIFIED HETEROGENEOUS ACCELERATION FRAMEWORK FOR FEDERATED LEARNING

A. HARDWARE-FRIENDLY ALGORITHMS

Encryption, decryption, and ciphertext computation all depend on the big number ModMult and ModExp. Directly executing the raw operators on hardware would result in significant overhead, so we consider optimizing with hardware-friendly Binary Exponentiation algorithm [53] and Montgomery algorithm [36].

The ModExp is composed of repeated ModMult. Binary Exponentiation algorithm can reduce the number of ModMult in $x^y \bmod n$ from $\mathcal{O}(y)$ to $\mathcal{O}(\log_2 y)$. However, in traditional ModMult, division operations are usually time-consuming on heterogeneous hardware. MontMult is a classical algorithm that accelerates ModMult by converting division into shifts, multiplications, and additions/subtractions that computers excel at. Algorithm 1 shows the MontMult, which we denote as $mont\_mult_b(\cdot,\cdot,\cdot)$ in this paper, i.e., $mont\_mult_b(x,y,n) = x\cdot y\cdot b^{-L} \bmod n$, where b is the radix for MontMult satisfying coprime to n ($gcd(n,b) = 1$), and L is the width of the modulus n in base b.

Algorithm 1 Montgomery Multiplication

Input: $x = (x_{L-1}\dots x_1x_0)_b$, $y = (y_{L-1}\dots y_1y_0)_b$, $n = (n_{L-1}\dots n_1n_0)_b$, with $0 \le x,y \le n$, $R = b^L$ with $gcd(n,b) = 1$
▷L is the width of n in base b
Output: $xyR^{-1} \bmod n$
1: $A \leftarrow 0$ ▷ $A = (a_{L-1}\dots a_1a_0)_b$
2: $n' \leftarrow -n^{-1} \bmod b$
3: **for** $i = 0$ to $L-1$ **do**
4: $u_i \leftarrow (a_0 + x_iy_0)n' \bmod b$
5: $A \leftarrow (A + x_iy + u_in)/b$
6: **end for**
7: **if** $A \ge n$ **then**
8: $A \leftarrow A - n$
9: **end if**
10: return(A)

Due to the high frequency of modulus reuse in FL, to avoid redundant calculations of n', in our design, we directly compute it on the CPU and pass it as a parameter to the hardware operator.

A big number x can be converted into Montgomery Domain by computing MontMult $mont\_mult_b(x, b^{2L}, n)$. Then ModMult can be realized using four MontMult as shown in Algorithm 2. ModExp is composed of multiple ModMult, and performing consecutive ModMult on intermediate results eliminates the need to exit and re-enter the domain. Algorithm 3 shows the Binary Exponentiation algorithm based on Montgomery Multiplication.

Algorithm 2 Modular Multiplication Based on Montgomery Multiplication

Input: $x = (x_{L-1}\dots x_1x_0)_b$, $y = (y_{L-1}\dots y_1y_0)_b$, $n = (n_{L-1}\dots n_1n_0)_b$, with $0 \le x,y \le n$
▷L is the width of n in base b
Output: $x \cdot y \bmod n$
1: $x \leftarrow mont\_mult_b(x, b^{2L}, n)$
2: $y \leftarrow mont\_mult_b(y, b^{2L}, n)$
▷ translate x, y to Montgomery Domain
3: $A \leftarrow mont\_mult_b(x, y, n)$
4: $A \leftarrow mont\_mult_b(A, 1, n)$
▷ retrieve A from Montgomery Domain
5: return(A)

Algorithm 3 Binary Exponentiation Based on Montgomery Multiplication

Input: $x = (x_{L-1}\dots x_1x_0)_b$, $y = (y_{l-1}\dots y_1y_0)_2$, $n = (n_{L-1}\dots n_1n_0)_b$, with $0 \le x,y \le n$
▷L is the width of n in base b
▷l is the width of y in base 2
Output: $x^y \bmod n$
1: $A \leftarrow mont\_mult_b(1, b^{2L}, n)$
▷ translate 1 to Montgomery Domain
2: $x \leftarrow mont\_mult_b(x, b^{2L}, n)$
▷ translate x to Montgomery Domain
3: **for** $i = 0$ to $l-1$ **do**
4: **if** $y_i == 1$ **then**
5: $A \leftarrow mont\_mult_b(A, x, n)$
6: **end if**
7: $x \leftarrow mont\_mult_b(x, x, n)$
8: **end for**
9: $A \leftarrow mont\_mult_b(A, 1, n)$
▷ retrieve A from Montgomery Domain
10: return(A)

B. OVERALL ARCHITECTURE OF THE PROPOSED ACCELERATION FRAMEWORK

As mentioned earlier, FL algorithms use PHE algorithms such as RSA and Paillier to ensure data security during the interaction process. The costs of RSA and Paillier encryption/decryption and homomorphic computation concentrate on ModMult and ModExp, and their hardware implementation relies on MontMult. Therefore, we propose a four-layer architecture of an acceleration framework, as shown in Fig. 3.

- FL Application Layer (FL APP): The FL application layer includes the implementation of various algorithms such as horizontal federated logistic

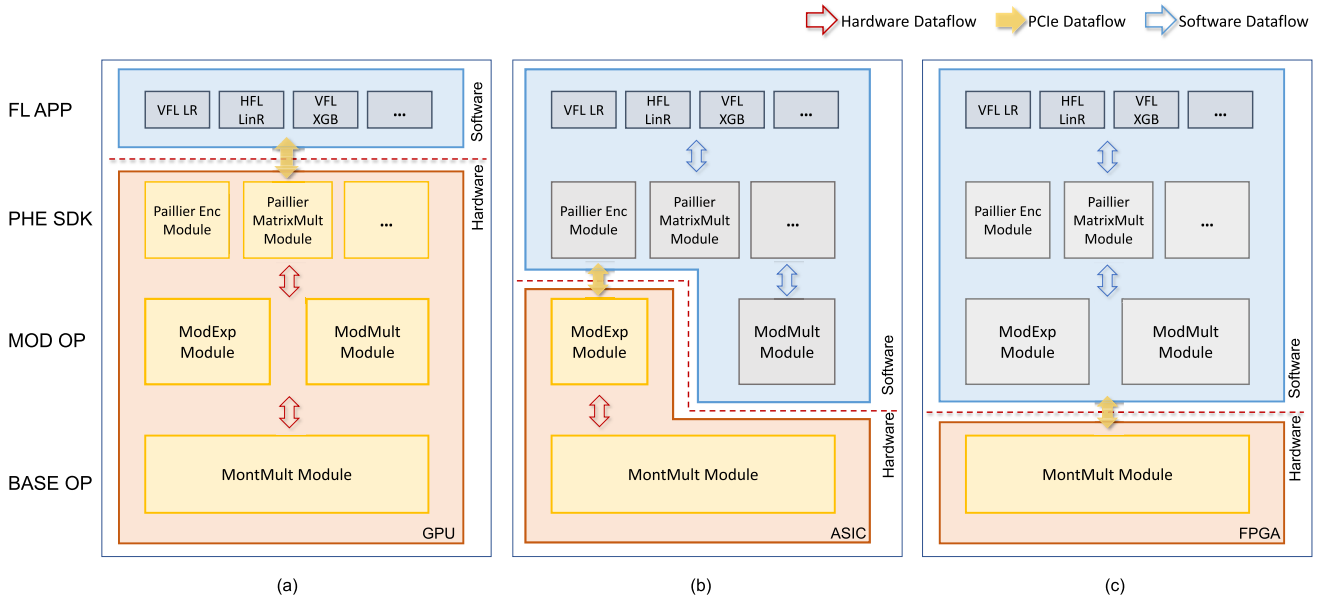


FIGURE 3. Top-level block diagram of the proposed acceleration framework.

regression (HFL LR), horizontal federated linear regression (HFL LinR), VFL LR, vertical federated linear regression (VFL LinR), and vertical federated XGBoost (VFL XGB). Many open-source FL frameworks have already implemented classic FL algorithms, making it possible to implement these algorithms based on existing engineering work.

- PHE Software Development Kit Layer (PHE SDK): This layer mainly implements PHE-related operators used in FL algorithms, such as Paillier encryption (Paillier Enc), point-wise addition of Paillier ciphertext vectors and multiplication of Paillier ciphertext matrices (Paillier MatrixMult) in HLR/VLR. A unified SDK is designed for heterogeneous hardware development, which can be called by FL APP. The selection of heterogeneous hardware for acceleration can be achieved through parameter configuration in the FL APP.
- Mod Operator Layer (MOD OP): This layer includes ModMult and ModExp operators.
- Base Operator Layer (BASE OP): The bottom layer is a hardware-friendly basic operator layer that implements MontMult operators.

As shown in Fig. 3, based on the analysis of different heterogeneous hardware's characteristics in Section III-B, we adopt different software/hardware partitioning strategies for GPU, ASIC, and FPGA at the PHE SDK layer and the MOD OP layer. MontMult Module is designed for GPU and FPGA respectively, while ASIC has already solidified this module in the chip.

For GPU, as shown in Fig. 3.(a), due to the flexibility brought by the mature programming framework, the lower three layers of the proposed architecture are realized using CUDA kernels, thus, the data interaction between these three layers can be achieved through hardware data flow to minimize access latency.

For FPGA, shown in Fig. 3.(c), as the hardware circuits on it cannot be flexibly switched without re-burning, we only implement the most basic MontMult Module (BASE OP) on hardware and then extend it into MOD OP and higher-level PHE SDK through software encapsulation to provide acceleration services for FL APP. We use OpenCL to implement FPGA architecture to achieve the purpose of rapid deployment.

For ASIC, shown in Fig. 3.(b), the software and hardware partitioning of the ASIC accelerator falls between GPU and FPGA designs. Our currently used Intel QAT 8970 has built-in Montgomery operation and provides the API of ModExp Module. Therefore, we implement the ModMult module directly in software using C language and let it work together with the ModExp Module on hardware to form the MOD OP layer of the ASIC accelerator. It should be noted that when using other models of ASICs, the software and hardware partitioning of each module may differ based on the APIs it can provide, but it can still be unified under the current 4-layer framework.

C. TIMELINE OF SOFTWARE AND HARDWARE COLLABORATION

Following the framework described in the previous section, we have implemented acceleration operators on the three types of hardware to offload computation intensive cryptography operations in FL tasks. Fig. 4 shows the software/hardware collaborative timeline of the proposed framework.

We developed GPU and FPGA accelerators using CUDA and OpenCL, and for ASIC accelerator, we invoked the low-level big number ModExp operator through the API

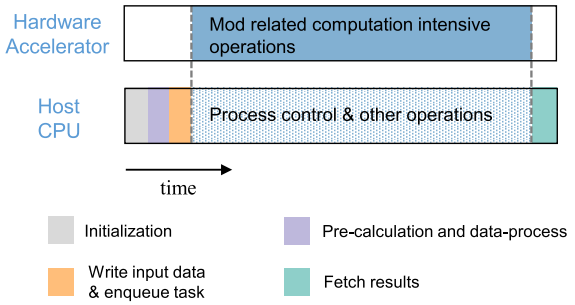


FIGURE 4. Software/hardware collaborative timeline of the proposed framework.

provided by Intel. We take ModExp as an example to illustrate the timeline.

As shown in Fig. 4, on the host CPU side, we perform initialization (e.g., get hardware device instances, create corresponding context and handlers, allocate buffers, etc.), and then carry out precomputation and data processing (e.g., generate keys, prepare hardware input). Subsequently, the CPU software writes the prepared data to the specified memory area that can interact with acceleration devices. For GPU and FPGA, CPU copies the data to the DDR memory on the device, while QAT allocates a segment in the host memory as global memory, which can be read or written by both QAT and host CPU. Finally, the CPU enqueues computational tasks to allocate to the corresponding hardware for execution.

On the hardware side, the corresponding device retrieves the data required for the calculation from the specified memory area, and then performs calculations based on queued tasks.

It is worth noting that when the HW accelerator performs modular-related calculations, the CPU software not only schedules the operator, but also carries out some other operations, such as plaintext operations and loss function calculations for all three kinds of devices, and ModMult operations for ASIC accelerator. Since these operations require very few clock cycles, their execution time can be covered by the calculation time of the hardware accelerator without causing additional delay.

V. ARCHITECTURAL DESIGN OF HETEROGENEOUS ACCELERATORS

Based on the proposed unified heterogeneous acceleration framework, in this section, we introduce the accelerator design of three heterogeneous hardware in detail.

A. DESIGN AND IMPLEMENTATION OF GPU ACCELERATOR

We adopt the MontMult algorithm as BASE OP layer and MOD OP layer is formed through multiple calls to the MontMult Module. To maximize performance, we carefully designed the Paillier Enc and MatrixMult modules inside the PHE SDK layer based on the analysis of the PHE algorithm in Section III.

For example, Paillier encryption calls the ModExp operator to calculate $s = r^n \mod n^2$, and then use the ModMult

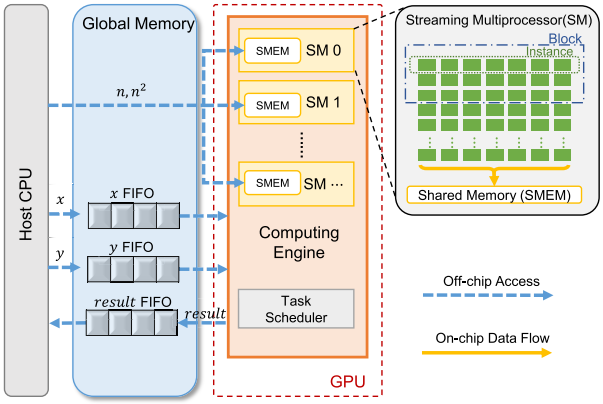


FIGURE 5. Hardware architecture design diagram of GPU accelerator.

operator to calculate $c = (m \cdot n + 1) \cdot s \mod n^2$. Directly calling the complete ModMult and ModExp modules will result in duplicate Montgomery domain entry/exit operations, which lead to more computational complexity and data movement. Therefore, we merge several calls of ModMult and ModExp modules into Paillier Enc Module, making the entire Paillier encryption calculation only require to enter and exit the Montgomery domain once each.

Fig. 5 shows the GPU accelerator architecture proposed in this study. We have designed first-in-first-out (FIFO) queues in the global memory of the GPU for input and output, which helps to buffer data streams. The public key-related parameters that need to be frequently accessed are stored in shared memory (SMEM) on the GPU chip to minimize memory access latency. Streaming Multiprocessors (SMs) are arrays of CUDA cores in the GPU chip used for parallel computation of the BASE OP and MOD OP layer. The total available number of SMs varies depending on the GPU hardware model used. The green square in Fig. 5 represents a thread used for parallel computation. We define a group of threads capable of handling big number calculations as an instance, and each block contains multiple instances. The task scheduler is used for scheduling among multiple SMs to control the operation of each operator.

Algorithm 4 shows the GPU implementation of the Paillier encryption operator based on CUDA, referred to as GPUPE. Due to the modulus width of over 1024 in the encryption operation, it needs to be split into multiple threads for computation. The value of 'threadPerInstance' is related to the width of the big number. For example, in the case of a Paillier key width of 1024, the highest modulus width in the calculation is 2048. When calculated with FP32 precision, 'threadPerInstance' needs to be set to 64. We provide a unified description of the notations used in the subsequent algorithms in Table 1 for ease of reading.

As the public key n remains constant in the Paillier encryption process and to avoid repeat calculations of n^2 in multiple threads, we directly calculate this on the CPU and pass it as an input parameter to the GPU, which is written to shared memory for faster memory access. Additionally, for optimized ModMult and ModExp using the MontMult on

TABLE 1. Notations used in Algorithms.

Notation	Description
X, Y	The plaintext/ciphertext integer matrix or vector, which serves as the input for hardware operator parallel computation.
R	A random integer vector used for Paillier encryption or obfuscation.
n	Modulus or Paillier encryption public key.
shared_*	The data copied to the shared memory of the GPU.
threadPerInstance	In order to achieve large-width integer computation, the GPU splits the data into 'threadPerInstance' threads for parallel processing.

the GPU, the module inverse of n^2 needs to be passed to the GPU. We don't emphasize this in algorithms considering the readability.

Algorithm 4 demonstrates the execution flow for a single thread in GPU. Step 1-9 read input data: Step 1-2 calculates the instance index 'i' for each thread. Step 3-4 reads the input data corresponding to instance. Step 5-8 shows that for each block, only one instance needs to read n and n^2 from global memory to shared memory. Wait for all data to be read before performing subsequent calculation operations. Step 10-13 perform the encryption, As is illustrated at the beginning of this subsection, step 12 should be executed directly in the Montgomery domain after the ModExp in step 11, to reduce the frequency of data transfers between the Montgomery domain, ultimately shortening computation time.

Algorithm 4 Paillier Enc Module Implemented on GPU Accelerator

Input: $X = [x_0, x_1, \cdots, x_{N-1}]$, public key n, $R = [r_0, r_1, \cdots, r_{N-1}]$, $n_{square} = n^2$

Output: $[(x_i n + 1) r_i^n \bmod n^2, i = 1, 2, \cdots, N-1]$

1: $i \leftarrow$ (blockIdx.x · blockDim.x + threadIdx.x)
2: $i \leftarrow i$/threadPerInstance
▷ instance id corresponding to the current thread
3: $x \leftarrow$ asynchronously load x_i
4: $r \leftarrow$ asynchronously load r_i
5: **if** threadIdx.x/threadPerInstance==0 **then**
6: $shared\_n \leftarrow$ asynchronously load n
7: $shared\_n_{square} \leftarrow$ asynchronously load n_{square}
8: **end if**
9: Wait for all threads to finish loading data
10: $x \leftarrow s \cdot shared\_n + 1$
11: $t \leftarrow$ModExp(r, $shared\_n$, $shared\_n_{square}$)
12: $result \leftarrow$ModMult(t, x, $shared\_n_{square}$)
13: asynchronously store $result$ as Out_i

For the commonly used vertical federated linear regression and logistic regression algorithms, we analyzed their algorithm flow and found that dense computation consists of two steps. The first step is the matrix multiplication between the plaintext feature matrix and the ciphertext gradient factors, as shown in the following equation:

$$\begin{bmatrix} x_{11} & x_{21} & \cdots & x_{N1} \\ x_{12} & x_{22} & \cdots & x_{N2} \\ \vdots & \vdots & \cdots & \vdots \\ x_{1k} & x_{2k} & \cdots & x_{Nk} \end{bmatrix} \begin{bmatrix} [\![y_1]\!] \\ [\![y_2]\!] \\ \vdots \\ [\![y_N]\!] \end{bmatrix} = \left[\sum_{i=1}^{N} x_{i1} \otimes [\![y_i]\!], \sum_{i=1}^{N} x_{i2} \otimes [\![y_i]\!], \cdots, \sum_{i=1}^{N} x_{ik} \otimes [\![y_i]\!] \right]^T \tag{8}$$

where N is the batch size and k is the number of features. Here, Σ represents the sum of the ciphertexts. The second step is to select k random numbers $\{r_1, r_2, \cdots, r_k\}$, add them to the ciphertext gradient to ensure data privacy:

$$enc\_grident = \left[\left(\sum_{i=1}^{N} x_{i1} \otimes [\![y_i]\!] \right) \oplus r_1, \cdots, \left(\sum_{i=1}^{N} x_{ik} \otimes [\![y_i]\!] \right) \oplus r_k \right]^T \tag{9}$$

The elements on the right side of the above equation each contain N homomorphic scalar multiplications and N homomorphic additions.

The entire operator is implemented on the GPU, converting all input data to the Montgomery domain and returning the calculation results after completing ModMult and ModExp.

Algorithm 5 Paillier MatrixMult Module Implemented on GPU Accelerator

Input: plaintext integers matrix $X_{k \times N}$, encryptext vector$Y = [y_0, y_1, \cdots, y_{N-1}]$, random vector$R = [r_0, r_1, \cdots, r_{N-1}]$, public key n, $n_{square} = n^2$

Output: $X_{k \times N} \otimes Y \oplus R$

1: $i \leftarrow$ (blockIdx.x · blockDim.x + threadIdx.x)
2: $i \leftarrow i$/threadPerInstance
▷ instance id corresponding to the current thread
3: $x \leftarrow$ asynchronously load $x_{i\%k\ i/k}$
4: $y \leftarrow$ asynchronously load $y_{i/k}$
5: **if** threadIdx.x/threadPerInstance==0 **then**
6: $shared\_n \leftarrow$ asynchronously load n
7: $shared\_n_{square} \leftarrow$ asynchronously load n_{square}
8: **end if**
9: Wait for all threads to finish loading data
10: $x \leftarrow$ModExp(y, x, $shared\_n_{square}$)
11: asynchronously store x as $x_{i\%k\ i/k}$
12: **if** i<k **then**
13: $r \leftarrow$ asynchronously load r_i
14: $result \leftarrow r \cdot shared\_n + 1$
▷ encrypt random number r
15: **for** $j = 0$ to $N-1$ **do**
16: $result \leftarrow$ModMult($result$, x_{ij}, $shared\_n_{square}$)
17: **end for**
18: asynchronously store $result$ as Out_i
19: **end if**

Algorithm 5 is the Paillier MatrixMult Module based on CUDA implementation, referred to as GPUPMM. Steps 1-11 use $k \times N$ instances to parallel compute $x_{ij} \otimes y_i (i = 1, 2, \cdots, N,\ j = 1, 2, \cdots, k)$. Step 12-19 sum the results and add the random number. During the calculation process, only the final result is converted out of the Montgomery domain.

It should be known that our GPU architecture design enables high scalability across different GPUs. I.e., our design assigns 8 threads to each 1024bit data, and 16 threads to each 2048bit data. Therefore, in the Nvidia Tesla T4 GPU used in the experiment, all 2560 cores are used, so that we can support 320 1024bit, or 160 2048bit data to compute in parallel.

B. DESIGN AND IMPLEMENTATION OF ASIC ACCELERATOR

Considering the practical situation of the participants in FL, it is more common for them to have purchased ASIC accelerators, rather than custom ASICs specifically for FL tasks. As the Intel QuickAssist Technology (QAT) adapter provides an extendable way to accelerate cryptography and compression capabilities, this accelerator has been widely used in enterprises that require large data storage and machine learning applications, which are also the most likely participants in FL. Therefore, we choose Intel QAT 8970 to develop the ASIC-based accelerator.

The ASIC provides a ModExp calculation interface, but does not support ModMult. Therefore, ModMult Module is designed in CPU. We implemented this module by simply implementing Algorithm 2 in C language.

At this point, the main challenge faced by this accelerator is how to efficiently schedule operators located on ASIC chip hardware and CPU software, respectively. Thus, we designed a task scheduler that can asynchronously call multiple threads, enabling efficient collaboration between the ASIC's ModExp operator and the CPU's ModMult operator. The way the task scheduler works is shown in Fig. 6.

Since QAT does not support parallel calculation of multiple data in a single instruction and multiple data (SIMD) manner, we designed a task scheduler to create two task queues for the ModExp calculation provided by QAT and the ModMult calculation implemented on the CPU respectively. In this way, by splitting multiple data in each batch into independent calculation tasks and adding them to their respective queues through the task scheduler, the ModExp and ModMult calculation stages required by multiple data can form a two-stage pipeline. That is, while the ModMult Module on the CPU gets the result of the QAT ModExp Module and performs ModMult calculations, QAT can perform the ModExp calculation of the next data. For each task, we set a FLAG to indicate the completion of the ModExp calculation to ensure that the ModMult Module in the CPU can correctly obtain the required input data.

Below, we provide a detailed introduction to several PHE algorithms designed specifically for ASIC accelerators.

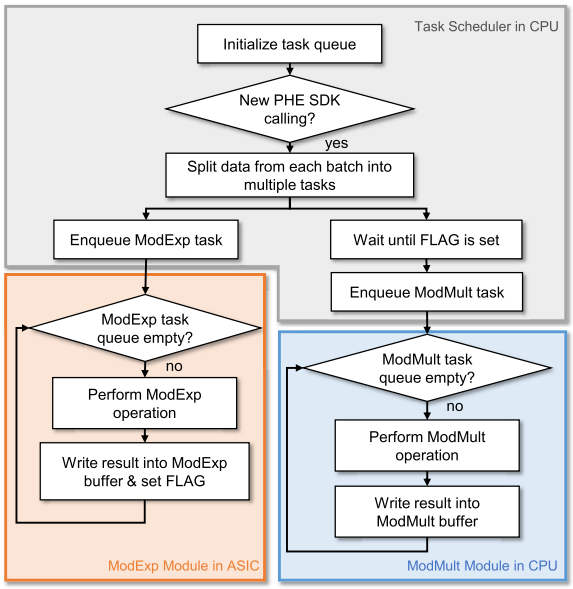


FIGURE 6. Work flow of the proposed ASIC task scheduler.

Algorithm 6 is our implementation of the Paillier encryption algorithm based on the QAT ModExp API. In the algorithm, cpaCyLnModExp is the ModExp Operator API provided by QAT, and ModMult is the ModMult Module API designed to work on CPU.

Algorithm 6 Paillier Enc Module Implemented on ASIC Accelerator

Input: $X = [x_0, x_1, \cdots, x_{N-1}]$, public key n, $R = [r_0, r_1, \cdots, r_{N-1}]$, $n_{square} = n^2$
Output: $[(x_i n + 1) r_i^n \bmod n^2, i = 1, 2, \cdots, N-1]$

1: **for** $i = 0$ to $N-1$ **do**
2: cpaCyLnModExp($r_i, n, n_{square}, tmp_i$)
▷ submit computation tasks asynchronously
3: **end for**
4: **for** $i = 0$ to $N-1$ **do**
5: synchronize the ModExp result tmp_i
6: $x_i \leftarrow x_i \cdot n + 1$
7: $Out_i \leftarrow$ ModMult(tmp_i, x_i, n_{square})
8: **end for**

For ciphertext computations, we use the QAT ModExp API to perform $k \times N$ homomorphic scalar multiplications and combine them with CPU-based ModMult for homomorphic addition. The algorithm implementation process is similar to Algorithm 6, and this implementation also uses the asynchronous calling feature of the task scheduler as shown in Algorithm 7.

In practical work, we found that the time it takes for ASIC to complete one ModExp calculation is longer than the time for CPU to complete one ModMult calculation. Therefore, the time for CPU to perform ModMult can be completely covered by the time it takes for ASIC to perform the next data ModExp calculation. This enables the accelerator to

Algorithm 7 Paillier MatrixMult Module Implemented on ASIC Accelerator

Input: plaintext integers matrix $X_{k\times N}$, encryptext vector$Y = [y_0, y_1, \cdots, y_{N-1}]$, random vector$R = [r_0, r_1, \cdots, r_{N-1}]$, public key n, $n_{square} = n^2$

Output: $X_{k\times N} \otimes Y \oplus R$

1: **for** $i = 0$ to $k-1$ **do**
2: **for** $j = 0$ to $N-1$ **do**
3: cpaCyLnModExp($x_{ij}, y_j, n_{square}, tmp_{ij}$)
 ▷ submit computation tasks asynchronously
4: **end for**
5: **end for**
6: **for** $i = 0$ to $k-1$ **do**
7: $Out_i \leftarrow r_i \cdot n + 1$
8: **for** $j = 0$ to $N-1$ **do**
9: synchronize the ModExp result tmp_i
10: $Out_i \leftarrow$ModMult($tmp_{ij}, result_i, n_{square}$)
11: **end for**
12: **end for**

complete one PHE SDK call in the time for ASIC to complete all ModExp calculations. In other words, the task scheduler's asynchronous calling mode we designed can maximize the utilization of ASIC hardware and achieve the optimal acceleration effect.

C. DESIGN AND IMPLEMENTATION OF FPGA ACCELERATOR

The FPGA accelerator implementation is based on our prior work [13]. As analyzed in Section III-B, since the circuit on the FPGA is relatively fixed during runtime, directly implementing high-level operators (i.e., PHE SDK layer operators) on the FPGA will not be conducive to reusing on-chip computing resources, thus compromising maximum performance.

Therefore, as shown in Fig. 3(c), the architecture proposed in this paper implements the bottom layer, i.e., BASIC OP layer, in which is a MontMult Module on the FPGA. Support for various PHE algorithms in FL is achieved by making multiple calls to the FPGA MontMult Module.

Fig. 7 shows the FPGA accelerator architecture proposed in this paper. The Data Load Unit (DLU) and Data Store Unit (DSU) are implemented as FPGA logic circuits, responsible for reading input data for Montgomery calculation from global memory and writing the FPGA-calculated output result back. Since the FPGA provides programmable RAM resources on-chip, the latency of on-chip RAM access is much lower than that of off-chip global memory access. Therefore, we use on-chip RAM to implement input and output FIFOs to reduce the number of read/write interactions between the FPGA and global memory, thereby maximizing memory read/write efficiency. Relevant control signals and key parameters are passed on-chip through OpenCL channels. Similar to the definition of instances in GPU implementation, we design multiple Processing Elements

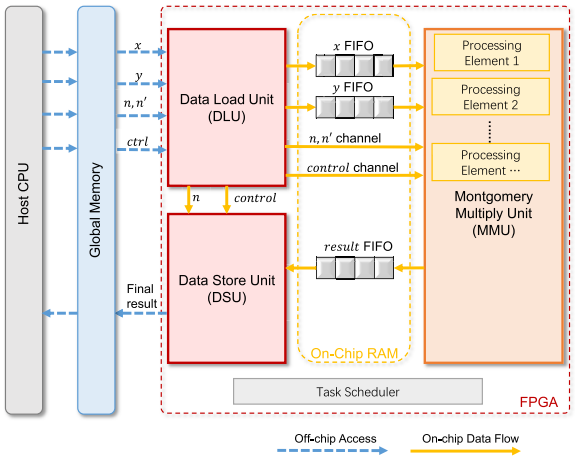


FIGURE 7. Hardware architecture design diagram of FPGA accelerator. The red dashed box shows the designed hardware circuit, where each unit in the circuit together forms the MontMult Module and is called by higher layers as the BASIC OP layer.

(PEs) in FPGA to achieve SIMD parallel calculation of a group of big numbers. At the same time, each unit in FPGA has its own task scheduler to control the calculation process.

In the Montgomery Multiply Unit (MMU), we implemented a hardware-aware Montgomery algorithm that we designed based on the Montgomery algorithm mentioned in Algorithm 1 in Section IV-A, adapted to the hardware characteristics of the FPGA.

The specific optimizations we made are as follows:

- Advance the timing of calculating the required u_{i+1} (step 4) for the next loop to coincide with step 5 of the current loop. In this way, the calculation time of step 4 and step 5 in the original Algorithm 1 can overlap in multiple loops.
- Vectorizing the input/output in Algorithm 1 to support SIMD parallelism, each big number in a SIMD vector is calculated on the corresponding PE.
- The loop in Algorithm 1 is implemented using a pipeline approach.

After implementing the MontMult module on the FPGA hardware, it is integrated with the unified FL heterogeneous acceleration framework as the BASIC OP layer, and encapsulated as a high-level operator in the PHE SDK layer for interacting with the FL APP layer.

VI. EXPERIMENTAL RESULTS

A. EXPERIMENTAL METHODOLOGY

The GPU platform we used is the 12nm NVIDIA Tesla T4 GPU. The FPGA platform is the Intel Programmable Acceleration Card (PAC) D5005, which features the Intel Stratix 10 GX2800 FPGA. The ASIC platform is the Intel QuickAssist Technology (QAT) adapter 8970, which provides an interface for accelerating ModExp operations.

We have verified the performance of cryptographic operations in our development environment, in which the host server equips an Intel Xeon E7-4830 V3 processor

with 128GB memory. We connected three types of acceleration devices to the server through the PCIe interface. The comparison of the three accelerators, as well as the cross-sectional comparison of the existing studies, are based on this experimental environment. The results are presented in detail in the next Section VI-B.

Furthermore, in order to validate the effectiveness of our accelerator in actual FL applications compared to powerful CPUs, we test the end-to-end training performance in our production environment. Each host server in the production environment is equipped with an AMD EPYC 7A23 48-Core Processor CPU with 128GB memory, and can provide several times the computing power of the development environment. We integrated our accelerator into the existing open-source FL training framework FATE [14], and the results are shown in Section VI-C.

The software environment used in the experiment is as follows. Our environment uses the Linux operating system with kernel version 3.10 and GCC version 9.1.0. For the GPU accelerator, we use the CUDA 10.0 to design CUDA kernels and provide runtime environment. For the FPGA accelerator, we use Intel OpenCL SDK v20.3 to design OpenCL kernels for the BASE OP Layer, and synthesis them into FPGA bitstream file which can be loaded into FPGA fabric. PyOpenCL is used to implement the call interface between the host CPU and the FPGA hardware. For QAT, in the MOD OP layer, we use the API provided by the manufacturer to call the ModExp operator, and use the implementation of the ModMult operator in the python PHE library [54]. PHE SDK layer operators are encapsulated by MOD OP layer operators through python. Our baseline CPU implementation is based on the open-source CPU libraries gmpy2 [55] and PHE.

As for compiler optimizations, for the code runs on CPU, we only use the -O1 optimization. For the code runs on heterogenous hardware, we just use vendors' default compilation optimizations for SDKs, e.g., NVCC default optimizations for GPU, and OpenCL default optimizations for FPGA.

B. PERFORMANCE OF CRYPTOGRAPHIC OPERATIONS

To compare the performance of our operators with other works, we chose ModMult, ModExp (RSA enc/dec), and the Paillier encryption algorithm as our benchmark targets.

1) THE PERFORMANCE OF HETEROGENEOUS ACCELERATORS UNDER A UNIFIED FRAMEWORK

In the 2048-bit ModMult and ModExp experiments, we randomly selected a 2048-bit big integer as the modulus n and chose x and y slightly smaller than the modulus (2047-bit number) as the operator input to simulate the most complex scenario in actual operations. We tested the CPU ModMult and ModExp operators performance using the gmpy2 library.

For the Paillier Enc test, the modulus is the square of the public key n, so we generated a 1024-bit key n, randomly selected float32 data as plaintext. For CPU, we conducted tests using the PHE library. And we referred to the encoding method in the PHE library to convert the plaintext into an integer as input to the hardware Paillier Enc operator.

For all tests, we processed 30,000 data at a time, started to statistic performance after preprocessing 10,000 data, and repeated each test 5 times to obtain the average result. Table 2 shows the performance of proposed accelerators in the three benchmark tests with 2048 modular bit width.

As shown in Table 2, the GPU, ASIC, and FPGA accelerators we designed can achieve excellent acceleration

TABLE 2. Throughput of cryptographic operations with 2048 modular bit width of the three proposed accelerators.

Throughput	ModMult (KOPS)	ModExp (OPS)	Paillier Enc. (OPS)
CPU(Intel Xeon CPU E7-4830 v3)	209.7	246.6	467.2
GPU (Nvidia Tesla T4)	3294.0	32398.4	65091.1
ASIC(Intel QAT 8970)	209.7*	32930.8	39215.7
FPGA (Intel PAC D5005)	1121.2	743.1	1488.1

* As QAT platform does not natively support ModMult calculation, we used CPU to implement the ModMult operator for the ASIC accelerator, and the performance was tested based on Intel Xeon CPU E7-4830 v3.

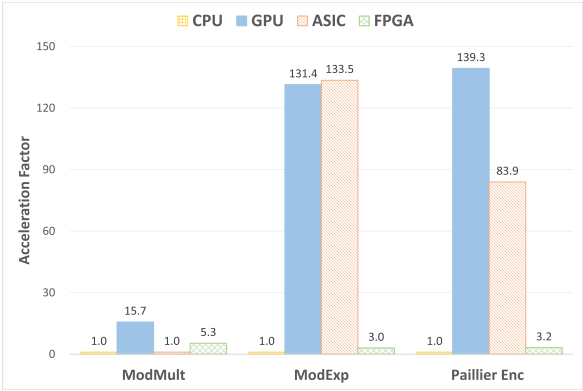


FIGURE 8. Acceleration effect of three proposed accelerators for cryptographic operations with 2048 modular bit width.

TABLE 3. Energy efficiency (Throughput/Watt) of cryptographic operations with 2048 modular bit width of the three proposed accelerators.

Energy Efficiency	ModMult (KOPS/W)	ModExp (OPS/W)	Paillier Enc. (OPS/W)
CPU(Intel Xeon CPU E7-4830 v3)	3.8	4.4	8.4
GPU (Nvidia Tesla T4)	99.8	426.8	929.8
ASIC(Intel QAT 8970)	-	1431.7	1705.0
FPGA (Intel PAC D5005)	44.8	29.7	59.5

TABLE 4. The GPU accelerators performance of cryptographic operations with 2048 modular bit width.

Implementation		Emmart et al. [21]	Cheng et al. [22]	Ours
Hardware Info	Target GPU	GTX Titan Black	Tesla V100	Tesla T4
	computing power(TFLOPS@FP32)	5.6	14.1	8.1
	TDP(W)	250	250	70
Throughput (OPS)	ModExp	17.5	-	**32.4**
	Paillier Enc	-	**83.9**	65.1
Scaled Throughput (OPS)	ModExp	25.3	-	**32.4**
	Paillier Enc	-	48.2	**65.1**
Scaled Energy Efficiency (OPS/W)	ModExp	101.2	-	**462.8**
	Paillier Enc	-	192.8	**930.9**

TABLE 5. The ASIC accelerators performance of cryptographic operations with 2048 modular bit width.

Implementation	Zhou et al. [23]	Ours
Target ASIC	Intel QAT 8970	Intel QAT 8970
Paillier Enc Throughput (OPS)	26.87	39.22

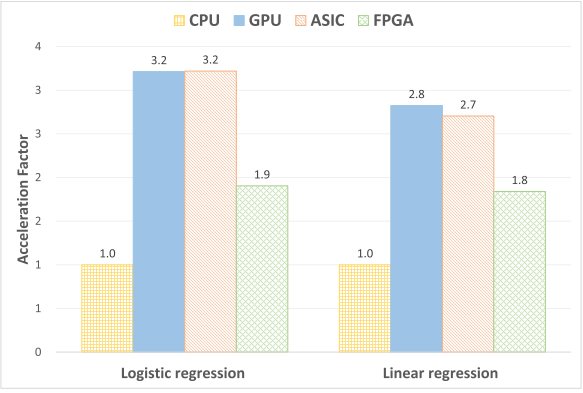


FIGURE 9. Accelerated effect of integrating the accelerator into the FATE framework for small-scale end-to-end federated learning.

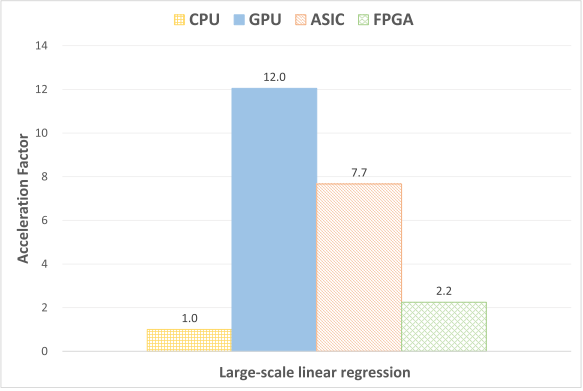


FIGURE 10. Accelerated effect of integrating the accelerator into the FATE framework for large-scale end-to-end federated learning.

ratios under the proposed unified acceleration framework. The acceleration effect is visually demonstrated in Fig. 8. For the sake of demonstration, we standardize the CPU compute time and report the acceleration times of various heterogeneous hardware compared to the CPU. Due to its greater computing resources and parallelism, the GPU can provide acceleration ratios of 15.7×, 131.4×, and 139.3× for ModMult, ModExp, and Paillier encryption operators respectively compared to the CPU. The ASIC can achieve similar performance to the GPU in ModExp operations due to its advantages in solidified dedicated circuit, and can bring a performance improvement of 83.9× compared to the CPU in Paillier encryption operations. In contrast, the acceleration effect of the FPGA accelerator is not as significant as the other two accelerators, but it can bring acceleration ratios of 5.3×, 3.0×, and 3.2× for ModMult, ModExp, and Paillier encryptions respectively compared to the CPU. Considering that in general, the performance improvement of about 30× can be achieved by converting FPGA design prototypes into ASICs, and the expected performance of our design after tape-out can be comparable to the current ASIC accelerator.

The energy efficiency are presented in Table 3. We conduct experiments on GPU, ASIC and FPGA and monitor the hardware power consumption during the invocation of operators, taking the average value of the measurements. As a result, the power consumption of the ModMult operator is 33W on GPU and 25W on FPGA. The ModExp and Paillier Enc operators lead the GPU power to 70W during run time while FPGA's power remains stable. Since it's hard to measure the real-time power of QAT, we conservatively calculate the energy efficiency based on its TDP of 23W.

As for the experimental results in Table 3, it is expected that ASIC has the best energy efficiency, and the pure hardware implementation of ModExp operator energy efficiency is about 325× of CPU, 3.4× of GPU. It should be noted that our FPGA uses a 14nm process, which is more backward than the 12nm process of the GPU, so the ModMult energy efficiency is about half that of the GPU. For ModExp and Paillier operators, due to the need for more CPU participation in our FPGA implementation, frequent data moving may causes the loss of energy efficiency.

2) PERFORMANCE COMPARISON WITH THE EXISTING WORKS

Table 4 presents the performance comparison of the GPU accelerators. Our designed GPU accelerator can easily scale according to the GPU computing power. Therefore, considering the hardware differences used in different works, we report the scaled accelerator performance. The basis for

TABLE 6. The FPGA accelerators performance of cryptographic operations.

Implementation		San et al. [24]				San et al. [25]		Milad et al. [26]	Ours	
Hardware Info	Target FPGA	Xilinx XC7VX330T-3				Xilinx XC7VX330T-3		Xilinx Zynq 7020	Intel Stratix 10 GX2800	
FPGA Resource ultilization	Fmax(MHz)	211	201	403	399	386	386	122	257	187
	Logic	2%	6%	4%	7%	6%	7%	98%	38%	65%
	DSP	0%	5%	2%	5%	4%	4%	87%	8%	16%
	RAM	0%	0%	0%	0%	-	-	59%	24%	50%
Throughput Performance(OPS)	1024bit ModMult	719.4	-	-	-	-	-	-	1863.3	-
	1024bit ModExp	-	657.9	-	-	-	-	-	3593.7	-
	1024bit Paillier	-	-	-	-	680.3	-	-	7029.9	-
	2048bit ModMult	-	-	366.3	-	-	-	730.9	-	1121.1
	2048bit ModExp	-	-	-	176.1	-	-	237.0	-	743.1
	2048bit Paillier	-	-	-	-	-	88.7	-	-	1488.1

standardization is the computing power of the Tesla T4 provided by the vendor. standardized results in the table are denoted as 'Scaled'. Based on this, due to the fact that the compared works all utilized PCIe interface GPUs deployable in data center servers, we also considered the power consumption factors of different boards, and the comparison results are presented in the last two rows of the table. In all tables, '-' marked cells indicate that the current design does not support the algorithm or that relevant information has not been disclosed.

From the comparison, it can be seen that for ModExp with a modulus bit width of 2048, our designed GPU accelerator's performance is increased by about 1.9× compared to previous work [21], and after standardization based on GPU computing power and power consumption, our relative performance can reach 4.6× of theirs. For Paillier encryption with a key bit width of 1024 (a modular bit width of 2048), the scaled energy efficiency of our designed accelerator can reach 4.8× compared to the implementation by Cheng et al. [22] on the V100 GPU.

In the work by Zhou et al. [23], they considered using QAT to offload ModExp in Paillier encryption. As shown in Table 5, with the help of a sophisticated asynchronous task scheduler design, as described in Section V-B, we achieved about a 46% improvement in the performance of paillier encryption implemented on the same model ASIC compared to previous work.

Table 6 shows the comparison of FPGA accelerators. It can be seen that the previous works [24], [25] designed relatively fixed circuits, which means that their FPGA bitstreams can only support one specific algorithm at a time, thus unable to accelerate all cryptographic performance bottlenecks related to FL. In contrast, our design can flexibly support multiple FL algorithms, thereby meeting end-to-end FL acceleration needs.

In terms of performance, existing works [24], [25], [26] are all based on embedded FPGAs, and [24], [25] did not make full use of FPGA hardware resources. However, in the actual FL deployment environment, FPGAs are usually chosen for more powerful devices deployed on servers. This makes their research and actual application scenarios quite different, leading to much less acceleration effect than our accelerator.

C. END-TO-END PERFORMANCE OF FEDERATED LEARNING TASK

Two commonly used public datasets, Kaggle datasets on breast cancer [56] and motor temperature [57] are used, and we selected two of the most common FL training tasks to evaluate the end-to-end performance of accelerators: logistic regression and linear regression. The breast cancer dataset has a size of 569, and 800 data samples were randomly selected from the motor temperature dataset for testing, with the training set and test set divided in 7:3. The test results are shown in Fig. 9.

From the comparison results in Fig. 9, our accelerator exhibits considerable acceleration performance in end-to-end linear regression and logistic regression tasks. Compared with the software (SW) implementation, the GPU and ASIC (QAT) implementations achieve around 3× acceleration, and the FPGA is close to 2× acceleration.

To compare the hardware acceleration performance in large-scale training, we also trained a linear regression task using the entire motor temperature dataset, which consists of 133016 data samples. The test results are shown in Fig. 10.

With an increasing amount of computation, the hardware operator acceleration effect becomes more pronounced. Compared with the SW's single iteration time, the time overheads for using GPU, ASIC (QAT), and FPGA operators are reduced by 12.0×, 7.7×, and 2.2×, respectively. In summary, through these experiments, we verified the performance improvement effect of our proposed unified acceleration architecture and three hardware operators on end-to-end training tasks in FL.

In Section VI-B, we reported that our proposed cryptographic acceleration operators' performance outperforms existing work. And the computational overhead of cryptographic operations during FL tasks accounts for over 85% of the total time. SectionVI-C demonstrates how our consistency framework effectively improves the running speed of federated learning tasks. It should be noted that our focus is to provide a platform that supports various heterogeneous hardware, rather than aiming for maximum acceleration for each hardware design. While ensuring generality may result in a slight performance loss, we believe that supporting heterogeneity in practical business

environments and simplifying deployment difficulties is highly
valuable.

VII. CONCLUSION AND FUTURE WORK

In this work, we proposed a unified FL acceleration framework that can handle the hardware heterogeneity challenge of multiple participants, by supporting various hardware devices simultaneously. The proposed framework defines FL as four layers, namely BASIC OP layer, MOD OP layer, PHE SDK layer and FL APP layer. With this definition, the FL application can be successfully decoupled from the underlying hardware operators, so that the framework can support multiple heterogeneous computing devices at the same time.

Furthermore, based on our proposed unified framework, we designed acceleration solutions aiming at three types of hardware, namely GPU, ASIC, and FPGA. We verified the effectiveness of these solutions in improving the performance of specific PHE algorithms and end-to-end FL tasks.

This research, however, is subject to several limitations. In our experimental and production environments, each compute node involved in FL is located in the same data center, which makes the communication delay of data in the network very low, and the computing power becomes the main bottleneck limiting the training performance. However, in actual FL tasks, long-distance network transmission may introduce additional latency, creating new bottlenecks. Due to the limited experimental environment, although we demonstrate that the proposed architecture is effective in improving computational performance, the impact of network latency has not been fully evaluated in this research.

As for future work, on the one hand, we will continue to work on improving the effectiveness of compute acceleration, for example, considering the use of specific modules provided in advanced heterogeneous hardware, such as Tenser Core and High Bandwidth Memory. On the other hand, it can be combined with the relevant research results in the field of high-performance networks to study the specific needs of the network in FL, and perform cooperative optimization for both end-point side and network side, which will be conducive to further improving the end-to-end performance in practical applications.

ACKNOWLEDGMENT

The authors would like to thank the anonymous reviewers for their valuable comments and constructive suggestions, which helped in improving the quality of the article.

REFERENCES

[1] H. B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. Y. Arcas, "Communication-efficient learning of deep networks from decentralized data," 2023, *arXiv:1602.05629*.
[2] Q. Yang, Y. Liu, T. Chen, and Y. Tong, "Federated machine learning: Concept and applications," *ACM Trans. Intell. Syst. Technol. (TIST)*, vol. 10, no. 2, pp. 1–19, 2019.
[3] A. C.-C. Yao, "How to generate and exchange secrets," in *Proc. 27th Annu. Symp. Found. Comput. Sci. (SFCS)*, Oct. 1986, pp. 162–167.
[4] C. Dwork, F. McSherry, K. Nissim, and A. Smith, "Calibrating noise to sensitivity in private data analysis," in *Theory Cryptography*. Berlin, Germany: Springer, 2006, pp. 265–284.
[5] R. L. Rivest and M. Dertouzos, "On data banks and privacy homomorphisms," *Found. Secure Comput.*, vol. 4, pp. 160–178, Jan. 1978.
[6] R. L. Rivest, A. Shamir, and L. Adleman, "A method for obtaining digital signatures and public-key cryptosystems," *Commun. ACM*, vol. 26, no. 1, pp. 96–99, Jan. 1983.
[7] P. Paillier, "Public-key cryptosystems based on composite degree residuosity classes," in *Advances in Cryptology—EUROCRYPT*. Berlin, Germany: Springer, 1999.
[8] E. De Cristofaro and G. Tsudik, "Practical private set intersection protocols with linear complexity," in *Financial Cryptogr. Data Secur.*, R. Sion, Ed. Berlin, Germany: Springer, 2010, pp. 143–159.
[9] S. Yang, B. Ren, X. Zhou, and L. Liu, "Parallel distributed logistic regression for vertical federated learning without third-party coordinator," 2019, *arXiv:1911.09824*.
[10] L. T. Phong, Y. Aono, T. Hayashi, L. Wang, and S. Moriai, "Privacy-preserving deep learning via additively homomorphic encryption," *IEEE Trans. Inf. Forensics Security*, vol. 13, no. 5, pp. 1333–1345, May 2018.
[11] C. Liu, S. Chakraborty, and D. Verma, "Secure model fusion for distributed learning using partial homomorphic encryption," in *Policy-Based Autonomic Data Governance*, vol. 11550. Cham, Switzerland: Springer, 2019, pp. 154–179.
[12] H. Fang and Q. Qian, "Privacy preserving machine learning with homomorphic encryption and federated learning," *Future Internet*, vol. 13, no. 4, p. 94, Mar. 2021. [Online]. Available: https://www.mdpi.com/1999-5903/13/4/94
[13] Z. Wang, B. Che, L. Guo, Y. Du, Y. Chen, J. Zhao, and W. He, "PipeFL: Hardware/software co-design of an FPGA accelerator for federated learning," *IEEE Access*, vol. 10, pp. 98649–98661, 2022.
[14] WeBank Corp. (2023). *An Industrial Grade Federated Learning Framework*. [Online]. Available: https://fate.readthedocs.io/en/latest
[15] W. Jung, T. T. Dao, and J. Lee, "Deepcuts: A deep learning optimization framework for versatile GPU workloads," in *Proc. 42nd ACM SIGPLAN Int. Conf. Program. Lang. Design Implement. (PLDI)*. New York, NY, USA: Association for Computing Machinery, 2021, pp. 190–205, doi: 10.1145/3453483.3454038.
[16] S. Tan, B. Knott, Y. Tian, and D. J. Wu, "CryptGPU: Fast privacy-preserving machine learning on the GPU," 2021, *arXiv:2104.10949*.
[17] Z. Wang, K. Xu, S. Wu, L. Liu, L. Liu, and D. Wang, "Sparse-YOLO: Hardware/software co-design of an FPGA accelerator for YOLOv2," *IEEE Access*, vol. 8, pp. 116569–116585, 2020.
[18] D. Wang, J. An, and K. Xu, "PipeCNN: An OpenCL-based FPGA accelerator for large-scale convolution neuron networks," 2016, *arXiv:1611.02450*.
[19] D. Wang, K. Xu, Q. Jia, and S. Ghiasi, "ABM-SpConv: A novel approach to FPGA-based acceleration of convolutional neural network inference," in *Proc. 56th ACM/IEEE Design Autom. Conf. (DAC)*, 2019, pp. 1–6.
[20] S. D. Manasi and S. S. Sapatnekar, "DeepOpt: Optimized scheduling of CNN workloads for ASIC-based systolic deep learning accelerators," in *Proc. 26th Asia South Pacific Design Autom. Conf. (ASP-DAC)*, Jan. 2021, pp. 235–241.
[21] N. Emmart, F. Zheng, and C. Weems, "Faster modular exponentiation using double precision floating point arithmetic on the GPU," in *Proc. IEEE 25th Symp. Comput. Arithmetic (ARITH)*, Jun. 2018, pp. 130–137.
[22] X. Cheng, W. Lu, X. Huang, S. Hu, and K. Chen, "HAFLO: GPU-based acceleration for federated logistic regression," 2021, *arXiv:2107.13797*.
[23] H. Zhou and B. Hua, "Homomorphic encryption offloading and its application in privacy-preserving computing," *J. Chin. Comput. Syst.*, vol. 42, no. 3, pp. 595–600, 2021. [Online]. Available: http://xwxt.sict.ac.cn/EN/Y2021/V42/I3/595
[24] I. San and N. At, "Improving the computational efficiency of modular operations for embedded systems," *J. Syst. Archit.*, vol. 60, no. 5, pp. 440–451, May 2014.
[25] I. San, N. At, I. Yakut, and H. Polat, "Efficient Paillier cryptoprocessor for privacy-preserving data mining," *Secur. Commun. Netw.*, vol. 9, no. 11, pp. 1535–1546, Jul. 2016.
[26] M. Bahadori and K. Järvinen, "A programmable SoC-based accelerator for privacy-enhancing technologies and functional encryption," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 28, no. 10, pp. 2182–2195, Oct. 2020.

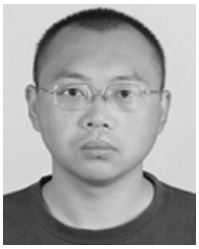
[27] Intel Corp. (2023). *Intel Quickassist Technology (Intel QAT)*. [Online]. Available: https://www.intel.com/
[28] Cavium Inc. (2020). *NITROX XL CNN35XX Security Adapter Family*. Technical Report. [Online]. Available: http://caxapa.ru/thumbs/311005/Nitrox-XL_CNN35XX_Rev1.0.pdf
[29] (2014). *DX2040—High Performance Scalable Solutions for Data Analytics, Storage, and Networking, Technical Report*. [Online]. Available: https://assets.maxlinear.com/
[30] S. Fleissner, "GPU-accelerated Montgomery exponentiation," in *Proc. 7th Int. Conf. Comput. Sci.* Berlin, Germany: Springer-Verlag, 2007, pp. 213–220.
[31] O. Harrison and J. Waldron, "Efficient acceleration of asymmetric cryptography on graphics hardware," in *Proc. Prog. Cryptol. (AFRICACRYPT)*. Berlin, Germany: Springer, 2009, pp. 350–367.
[32] S. Neves and F. Araujo, "On the performance of GPU public-key cryptography," in *Proc. 22nd IEEE Int. Conf. Appl.-Specific Syst., Archit. Processors (ASAP)*, Sep. 2011, pp. 133–140.
[33] J. Dong, G. Fan, F. Zheng, T. Mao, F. Xiao, and J. Lin, "TEGRAS: An efficient Tegra embedded GPU-based RSA acceleration server," *IEEE Internet Things J.*, vol. 9, no. 18, pp. 16850–16861, Sep. 2022.
[34] G. Yudheksha, P. Kumar, and S. Keerthana, "A study of AES and RSA algorithms based on GPUs," in *Proc. Int. Conf. Electron. Renew. Syst. (ICEARS)*, Mar. 2022, pp. 879–885.
[35] NVIDIA Corp. (2023). *CGBN: CUDA Accelerated Multiple Precision Arithmetic (Big Num) Using Cooperative Groups*. [Online]. Available: https://github.com/NVlabs/CGBN
[36] P. L. Montgomery, "Modular multiplication without trial division," *Math. Comput.*, vol. 44, no. 170, pp. 519–521, Apr. 1985.
[37] K. Burlachenko, S. Horváth, and P. Richtárik, "FL_PyTorch: Optimization research simulator for federated learning," 2022, *arXiv:2202.03099*.
[38] J. H. Ro, A. T. Suresh, and K. Wu, "FedJAX: Federated learning simulation with JAX," 2021, *arXiv:2108.02117*.
[39] Z. Tang, X. Chu, R. Y. Ran, S. Lee, S. Shi, Y. Zhang, Y. Wang, A. Q. Liang, S. Avestimehr, and C. He, "FedML Parrot: A scalable federated learning system via heterogeneity-aware scheduling on sequential and hierarchical training," 2023, *arXiv:2303.01778*.
[40] A. Paszke et al., "PyTorch: An imperative style, high-performance deep learning library," in *Advances in Neural Information Processing Systems*. Red Hook, NY, USA: Curran Associates, 2019.
[41] M. Abadi et al. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. [Online]. Available: https://tensorflow.org
[42] Z. Zeng, Y. Du, Z. Fang, L. Chen, S. Pu, G. Chen, H. Wang, and Y. Gao, "FLBooster: A unified and efficient platform for federated learning acceleration," in *Proc. IEEE 39th Int. Conf. Data Eng. (ICDE)*, Apr. 2023, pp. 3140–3153.
[43] R. B. Nti and K. Ryoo, "ASIC design of low area RSA crypto-core based on Montgomery multiplier," *Int. J. Eng. Technol.*, vol. 7, no. 3, pp. 278–283, 2018.
[44] C. Cai, H. Awano, and M. Ikeda, "High-speed ASIC implementation of Paillier cryptosystem with homomorphism," in *Proc. IEEE 13th Int. Conf. ASIC (ASICON)*, Oct. 2019, pp. 1–4.
[45] A. C. Mert, E. Öztürk, and E. Savas, "Low-latency ASIC algorithms of modular squaring of large integers for VDF evaluation," *IEEE Trans. Comput.*, vol. 71, no. 1, pp. 107–120, Jan. 2022.
[46] D.-T. Nguyen-Hoang, K.-M. Ma, D.-L. Le, H.-H. Thai, T.-B.-T. Cao, and D.-H. Le, "Implementation of a 32-bit RISC-V processor with cryptography accelerators on FPGA and ASIC," in *Proc. IEEE 9th Int. Conf. Commun. Electron. (ICCE)*, Jul. 2022, pp. 219–224.
[47] Y. Zhuang, "Performance enhanced for encrypted database CryptDB," Ph.D. dissertation, School Softw., Shanghai Jiao Tong Univ., Shanghai, China, 2017.
[48] Z. Yang, S. Hu, and K. Chen, "FPGA-based hardware accelerator of homomorphic encryption for efficient federated learning," 2020, *arXiv:2007.10560*.
[49] C. Yang et al., "FLASH: Heterogeneity-aware federated learning at scale," *IEEE Trans. Mobile Comput.*, vol. 23, no. 1, pp. 483–500, Jan. 2024, doi: 10.1109/TMC.2022.3214234.
[50] G. Corp. (2023). *Tensorflow Federated: Machine Learning on Decentralized Data*. [Online]. Available: https://www.tensorflow.org/federated
[51] T. Ryffel, A. Trask, M. Dahl, B. Wagner, J. Mancuso, D. Rueckert, and J. Passerat-Palmbach, "A generic framework for privacy preserving deep learning," 2018, *arXiv:1811.04017*.
[52] J. Zhang, X. Cheng, W. Wang, L. Yang, J. Hu, and K. Chen, "FLASH: Towards a high-performance hardware acceleration architecture for cross-silo federated learning," in *Proc. 20th USENIX Symp. Networked Syst. Design Implement. (NSDI)*. Boston, MA, USA: USENIX Association, Apr. 2023, pp. 1057–1079. [Online]. Available: https://www.usenix.org/conference/nsdi23/presentation/zhang-junxue
[53] A. Daly and W. Marnane, "Efficient architectures for implementing Montgomery modular multiplication and RSA modular exponentiation on reconfigurable logic," in *Proc. ACM/SIGDA Int. Symp. Field-Program. Gate Arrays (FPGA)*, Feb. 2002, pp. 40–49.
[54] Python Paillier Library. (2022). *CSIRO Data61 Engineering & Design*. [Online]. Available: https://github.com/data61/python-paillier
[55] A. Martelli. (2022). *General Multi-Precision Arithmetic for Python*. [Online]. Available: https://github.com/aleaxit/gmpy
[56] KIRGSN. (2023). *Breast Cancer Wisconsin (Diagnostic) Data Set*. [Online]. Available: https://www.kaggle.com/uciml/breast-cancer-wisconsin-data
[57] UCI Machine Learning. (2023). *Electric Motor Temperature*. [Online]. Available: https://www.kaggle.com/wkirgsn/electric-motor-temperature

BIYAO CHE received the B.S. degree in financial mathematics from the Beijing University of Chemical Technology, Beijing, China, in 2018, and the M.S. degree in statistics from the Beijing Jiaotong University, Beijing, in 2021. She is currently with the China Telecom Research Institute, Beijing, as an AI Algorithm Engineer. Her research interests include mathematics, statistics, deep learning, and the interpretability of neural networks.

ZIXIAO WANG received the M.S. degree in signal and information processing from the Beijing Key Laboratory of Advanced Information Science and Network Technology, Beijing Jiaotong University, Beijing, China, in 2021. He is currently with the China Telecom Research Institute, Beijing, as an AI Algorithm Engineer. His research interests include heterogeneous computing systems, neural network compression, and high-performance computing architectures for deep learning applications.

YING CHEN received the B.S. and M.S. degrees in control theory from the Institute of Robotics and Intelligent Equipment, Beijing University of Technology, Beijing, China, in 2012 and 2019, respectively. From 2019 to 2021, he was a Computer Vision Algorithm Engineer with Beijing Vion Technology Inc., Beijing. He is currently with the China Telecom Research Institute, Beijing, as an AI algorithm Engineer. His research interests include neural network algorithms and architecture, computer vision, high-performance computing, and optimization theory.

LIANG GUO (Member, IEEE) is currently a professor-level Senior Engineer and the Deputy Chief Engineer of the Institute of Cloud Computing and Big Data of CAICT, Beijing, China. He is mainly engaged in policy support, technical research, and standard-setting related to computational infrastructure. He wrote four monographs and published more than 20 journal articles.



YUAN LIU received the Ph.D. degree in computer science from the University of Science and Technology of China. Since 2011, he has been a Staff Engineer with the China Telecom Research Institute. His research interests include cloud computing and AI.

YUAN TIAN received the M.S. degree in computer science from The University of Waikato, Hamilton, New Zealand, in 2016. He is currently with the China Telecom Research Institute, Beijing, China, as a Data Engineer and a Network Algorithm Engineer. His research interests include computer systems, parallel computing, distributed database systems, and high-performance computing architectures.

JIZHUANG ZHAO received the M.S. degree from the Beijing Institute of Technology, Beijing, China, in 2006. He is currently the Deputy Leader of the Advance Technology Testing Group, Open Data Center Committee (ODCC), and the Chief of the Requirements and Architecture Group, Diversified Computing Industry Alliance (DICA). He is also with the China Telecom Research Institute, Beijing, as a Senior Engineer. His research interests include cloud computing and software/hardware optimization of high-performance computing.