FLBooster: A Unified and Efficient Platform for Federated Learning Acceleration

Zhihao Zeng[†], Yuntao Du[†], Ziquan Fang[†], Lu Chen[†], Shiliang Pu[♯], Guodong Chen[♯], Hui Wang[♯], Yunjun Gao[†]
[†]*Zhejiang University, Hangzhou, China*
[♯]*Hikvision, Hangzhou, China*
[†]{*zengzhihao, ytdu, zqfang, luchen, gaoyj*}*@zju.edu.cn* [♯]{*pushiliang, chenguodong, wanghui14*}*@hikvision.com*

***Abstract*—Federated learning (FL) has emerged as a paradigm to train a global machine learning model in a distributed manner while taking privacy concerns and data protection regulations into consideration. Although a variety of FL algorithms have been proposed, the training efficiency of FL remains challenging due to massive mathematical computations and expensive client-server communication costs. However, existing FL-acceleration studies are limited as they can only solve the computation and communication overheads separately, which is suboptimal and constrains their acceleration ability. Moreover, previous studies are typically designed for specific FL scenarios and can support only one or two FL models, thus exhibiting poor generality.**

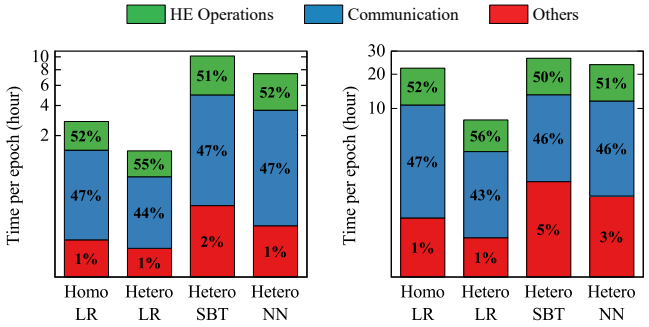
To fill these critical voids, we propose FLBooster, which provides unified and efficient acceleration capacity for a broad range of FL models. This is the first proposal to solve the computation and communication overheads simultaneously. Specifically, we utilize GPUs to boost the computation-intensive homomorphic encryption (HE) operations in a parallel manner, which significantly reduces the computation costs. On the other hand, a simple but efficient compression method is designed to lighten the exchange of data volumes between client and server. Extensive experiments using four standard FL models on three datasets show that FLBooster acquires superior speed-up gains (i.e., $14.3\times$ – $138\times$) over state-of-the-art acceleration systems. Finally, we integrate FLBooster into the open-source FL benchmark FATE and offer user-friendly APIs for development.

***Index Terms*—Federated learning, homomorphic encryption, GPU acceleration, efficient communication**

I. Introduction

In the era of big data, billions of Internet of Thing (IoT) devices and services in a broad range of fields such as internet [24], finance [36], and medicine [12] are generating massive training data. The traditional way of uploading those training instances into the center for centralized model training may encounter privacy leakage and network congestion issues [1]–[3], especially for different participants constrained by the privacy policy. To tackle these issues, federated learning [65] (FL) is proposed, which learns a shared machine learning model in a distributed manner without direct observation of training instances by using a large number of clients with separate data sets. In addition, such a decentralized training paradigm allows FL to address data privacy, data quality, and data oversimplified problems [37], [38].

To deploy FL into industrial applications, there is plenty of works [5], [11], [26], [35], [57], [76] that have been proposed. Among them, the dominant one is FATE [4], which is developed by WeBank [6]. Specifically, FATE enables



(a) RCV1 dataset (b) Avazu dataset

Fig. 1. An illustration of running time per epoch when using FATE to train four standard FL models (Homo LR, Hetero LR, Hetero SBT, Hetero NN) at 1024 key size

distributed model training while providing sufficient privacy protection for a series of machine learning models, which has become the most popular federated learning benchmark in the industry [40]. Despite the privacy protection of existing FL frameworks, training efficiency remains challenging for many applications [32], [38]. Fig. 1 shows the running time of one epoch when using FATE to train four standard FL models, including Homo LR [28], Hetero LR [11], Hetero SBT [17], and Hetero NN [71], from which we yield two observations. First, the training efficiency provided by FATE cannot meet the requirement of industrial applications as they take several hours or even more than a day for one epoch. Second, for any of those FL models, the training bottleneck lies in the massive homomorphic encryption (HE) operations and expensive communication, as detailed below.

- **Computation overhead.** Most existing federated learning frameworks typically utilize homomorphic encryption [54] (HE) for privacy preservation [42], [53]. Taking the most commonly used additive HE Paillier [50] as an example, its implementation relies on trapdoor functions [67] and contains expensive modular exponentiation and modular multiplication operations. However, most existing FL studies rely on Paillier to perform their own HE procedures, resulting in massive computation costs. As shown in Fig. 1, the time spent on HE operations (including encryption, decryption, and homomorphic computation) takes more than 50% of one training epoch. Such computation overhead constrains its deployments in real-world applications.

- **Communication overhead.** To avoid privacy leakage, it is not allowed to exchange raw data between client and server in the federated process. In contrast, the clients usually exchange encrypted data with the server to train a global machine learning model, which involves expensive communication costs. This is because, the key size of homomorphic encryption must be large enough [9] to defeat privacy cracking and it usually generates multiple ciphertexts for each plaintext, meaning that the number of ciphertexts is much more than that of plaintexts. As a result, each client needs to upload redundant ciphertexts during the federated learning, resulting in enormous communication costs. In Fig. 1, the communication takes more than 40% time of one epoch during the training process.

Recent studies have proposed several solutions for federated learning acceleration. Parallel-LR [66] provides a parallel version of logistic regression for federated learning (LR shown in Fig. 1). Similarly, HAFLO [18], a GPU-based implementation of Paillier [50], is developed to boost LR. SecureBoost+ [16] targets reducing the communications for secureboost (SBT shown in Fig. 1) by compressing ciphertexts. As can be noticed, all of the above-mentioned methods are applicable to one or two specific models, which constrains their deployment for generic federated learning. More importantly, none of these methods can solve the computation and communication overheads simultaneously, leaving a big room for efficiency improvements. To effectively deal with the computation and communication overheads of federated learning, as well as provide general acceleration supports for all standard FL models, in this paper, we propose a unified and efficient federated learning acceleration platform, called FLBooster. Specifically, to solve the computation overhead, we present a high-performance parallel implementation of homomorphic encryption via the GPU, where a resource manager is designed to enable fine-grained and balanced GPU resource allocation. To solve the communication overhead, we design a batch compression algorithm to reduce the communications between client and server while ensuring privacy.

- **Unified FL-acceleration.** FLBooster is a unified platform that enables boosting all existing standard federated learning models. To the best of our knowledge, this is the first FL-acceleration proposal that solves both computation and communication overheads simultaneously.
- **GPU-based homomorphic encryption.** Inspired by the independency of homomorphic encryption, we propose a parallel implementation of HE via the GPU to empower parallel computation for FL, where a resource manager is developed for efficient and balanced resource allocation.
- **Privacy quantization and compression.** We design a secure and low-loss quantization scheme to compress encrypted gradients, making full use of the plaintext space. Based on that, we can evidently reduce communication overhead without scarifying accuracy or modifying the communication protocol of federated learning.
- **Extensive experiments.** A practical experimental study using four standard federated learning models on three public datasets validates the efficiency and effectiveness of FLBooster. The results show that FLBooster boosts the performance of federated training by $14.3\times$ – $138\times$.
- **User-friendly APIs.** We have integrated FLBooster into the public FATE benchmark for potential studies. Also, we provide a series of useful APIs for developers to develop efficient models of federated learning as required.

The rest of this paper is organized as follows. Sec. II presents the related work. Sec. III gives the preliminary. In Sec. IV, we detail FLBooster and its key modules. Sec. V presents the pipelined processing of FLBooster. Sec. VI reports the experiments. Sec. VII concludes the paper.

II. Related Work

This section briefly reviews the related work on FL acceleration and GPU-based acceleration studies.

A. Federated Learning Acceleration

To address the training overhead (*i.e.,* computation and communication), several acceleration methods have been proposed, targeting accelerating logistic regression (LR) [66], secureboost (SBT) [16], or neural network (NN) [44], [75], where LR, SBT, and NN are the most commonly used models in federated learning. Basically, these acceleration methods are designed for specific FL models and have different optimization techniques, thus they cannot be deployed effectively to support various FL applications. Another line of studies focuses on the gradient and model compression for reducing communication costs in federated learning, such as Sketched Update [35], STC [55], FTTQ [63] and BatchEncrypt [70]. That is, they quantize and compress the transmitted data without encryption to improve communication speed. Unfortunately, most traditional compression techniques are unsuitable for most applications of federated learning with HE, due to the following two reasons. First, the data transmitted in FL is encrypted, and we cannot directly compress plaintext data to obtain performance gains in communication, such as gradients or model parameters, because the size of the ciphertext is immutable. Second, performing lossy compression on encrypted data will cause incorrect decryption results, and thus affects model effectiveness. It is worth noting that BatchEncrypt uses the packing-based technique to reduce communication overhead, but its' computation of quantization is sophisticated and time-consuming and suffers from the overflow problem in some cases [64]. The other works [59], [64] using this technique are only applicable to specific domains or models. Moreover, several symmetric homomorphic mechanisms have been proposed, including IHC&MRS [14], MORE [62], SFHE [56], ASHE [51], FLASHE [31], many of which have been proved to be insecure and vulnerable to attacks [19], [60]. Consequently, it is still debatable whether the symmetric HE can be applied to industrial applications with high demands for data privacy. *Overall, all of the above approaches either can only deal with the massive computation and expensive*

communication costs separately, making fast FL training remains challenging, or risk privacy leakage. In contrast, we aim to solve these two crucial challenges simultaneously and safely, offering generic acceleration support for a broad range of standard federated learning models.

B. GPU-based Acceleration

The increasing deployment of GPUs has enabled researchers to design various acceleration algorithms used in machine learning [49], [58], databases [22], [43], artificial intelligence [48], [74], etc. In fact, many popular and high-performance AI frameworks are implemented on GPUs, such as Caffe [29], Tensorflow [7], and Pytorch [52]. In terms of federated learning, the community has also proposed several GPU-based acceleration studies for model training. For example, TrustFL [72] transfers computations in the training process to GPUs to accelerate federated learning with high confidence. IBM federated learning [41] utilizes the GPU to accelerate neural network models (NN model in Fig. 1) used in federated learning. HAFLO [18] extracts the performance-critical homomorphic operations in model training and performs them on GPUs to accelerate logistic regression (LR model in Fig. 1). In other to improve the energy efficiency of federated edge learning, C^{2}RM [69] proposes energy-efficient resource management based on CPU-GPU heterogeneous computing. Moreover, to improve the throughput of federated learning, Full-Stack FL [68] designs a resource-sharing mechanism to manage the scheduling and competition of GPU resources. *As can be noticed, all these GPU-based optimizations are only applicable to specific models or scenarios in federated learning. In contrast, we propose FLBooster to offer general acceleration and support for all popular FL models.*

III. Preliminary

We proceed to introduce key concepts related to the studied problem, including stochastic gradient descent (SGD) used in FL, homomorphic encryption, and GPU processing.

A. SGD used in Federated Learning

Similar to centralized model training, SGD is also widely used in federated learning for distributed model optimization. Almost all FL models, *i.e.,* LR, SBT, NN, etc, support SGD. The model's update process using SGD is formulated below.

$$W_{t+1} = W_t - \alpha_t \nabla G_t, \tag{1}$$

where W denotes the model's parameters, α represents the learning rate, ∇G is the derivative gradients via objective function, and t is the iterative round.

For better understanding, a typical SGD-based FL model training is shown in Fig. 2. Specifically, each client (*i.e.,* participant) encrypts their local gradients using the public key and uploads the ciphertext (*i.e.,* encrypted gradients) to the central server, which is denoted by solid black arrows. At the same time, the server aggregates the encrypted gradients and then distributes the aggregated results to each client. After that, the client decrypts the received aggregated gradients by using

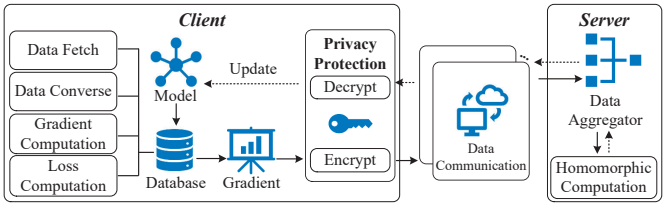


Fig. 2. The training process of federated learning models via SGD

the private key and updates the local model, which is denoted by dotted black arrows. The above process is repeated until the model converges. As clients only upload the encrypted gradients instead of raw values, no personal information is available to the server or external parties during the data transfer and data aggregation. In this paper, FLBooster also utilizes SGD for FL model training.

B. Homomorphic Encryption

To ensure model accuracy, for clients, the training results yielded by encrypted gradients should be the same as the training results yielded by plaintext gradients. To deal with that, homomorphic encryption is popularly used and has become a mainstream method to support privacy protection and model accuracy in federated learning. Homomorphic encryption is generally defined as below.

$$E(m_1) \otimes E(m_2) = E(m_1 \oplus m_2), \tag{2}$$

where $\otimes$ and $\oplus$ denote mathematical operations, m_1 and m_2 are the message, and E is the encryption function. In federated learning, additive homomorphic encryption [50] is effective to perform secure federated averaging, which allows a particular operation on the ciphertext to perform addition on the plaintext. In practice, many additive homomorphic variants have been proposed in the literature [10], [21], [50], [51], among which, Paillier [50] has become the most popular one due to its simple implementation and reliable security.

As our FLBooster also utilizes Paillier, we briefly detail Paillier, which mainly consists of four processes.

- *Key generation.* First, Paillier selects two large primes p and q, and computes $n = p \cdot q$ and $\lambda = lcm(p-1, q-1)$, where lcm is the least common multiple function. Then, Paillier chooses a random integer $g \in Z^*_{n^2}$ satisfying $gcd(n, L(g^\lambda \mod n^2)) = 1$, where gcd is the greatest common divisor function and L function is defined as $L(x) = (x-1)/n$. Based on that, Paillier generates the public key (g, n) and the private key (p, q).
- *Encryption.* For each message m, Paillier selects a random integer $r \in Z^*_{n^2}$, and encrypts m as follows:

$$E(m) = g^m r^n \mod n^2 \tag{3}$$

- *Decryption.* For each ciphertext c, Paillier decrypts it as:

$$D(c) = \frac{L(c^\lambda \mod n^2)}{L(g^\lambda \mod n^2)} \mod n \tag{4}$$

Algorithm 1: The basic Montgomery multiplication

Input: integers A, B, N, and $R = 2^w$ (N<R), where w is the size of a word in the system; $N^{'} = -N \mod R$

Output: Montgomery multiplication $ABR^{-1} \mod N$, where $RR^{-1} \mod N = 1$

1 $T \leftarrow AB \mod R$; $M \leftarrow TN^{'} \mod R$;
2 $U \leftarrow (AB + MN)/R$;
3 **if** $U \geq N$ **then**
4 | **return** $U - N$;
5 **else**
6 | **return** U;

- *Homomorphic property.* Paillier guarantees that the decryption results of multiplying two ciphertexts are equal to the addition of two plaintexts:

$$E(m_1) * E(m_2) = E(m_1 + m_2) \tag{5}$$

The most costable operations in homomorphic encryption are modular operations. To deal with the high complexity of modular multiplication, the Montgomery multiplication [47] is proposed. Its core idea is to replace the complex division and modular operations with more efficient operations. Based on that, it significantly improves the speed of modular multiplication. The details of Montgomery multiplication are shown in Algorithm 1, where modular and division operations are replaced by *AND* (line 1) and *Shift* (line 2) efficient operations since R is the power of 2. Given four integers A, B, N and $R = 2^w$ (w is the word size and $N < R$), Algorithm 1 computes Montgomery multiplication result $ABR^{-1} \mod N$. Note that, N' can be reused for all Montgomery multiplications, and thus, it is pre-computed as input for Algorithm 1. However, the basic Montgomery multiplication does not directly support large integers. In this paper, for more efficient homomorphic encryption, we propose a parallelized Montgomery algorithm via the GPU, to be detailed in Section IV-A3.

C. GPU Processing

GPUs have far more cores than CPUs, which enables parallel computing for high-density, high-intensity, and independent data. In FL, homomorphic encryption involves large-scale gradient encryption, decryption, and homomorphic computation, while the computational power of CPUs is far from adequate for the application. In contrast, GPUs can handle such tasks efficiently by pipelined processing. Specifically, the computation between large integers can be divided into disjointed limbs and then assigned to different threads for GPU acceleration. With a reasonable allocation of GPU resources, multiple training instances can run simultaneously, thus significantly improving the convergence speed of FL models. Overall, we are inspired to conduct computation-intensive HE operations via the GPU to overcome the computation overhead bottleneck. To make full use of GPU computing resources, we propose a powerful resource allocation manager, to be discussed in IV-A2.

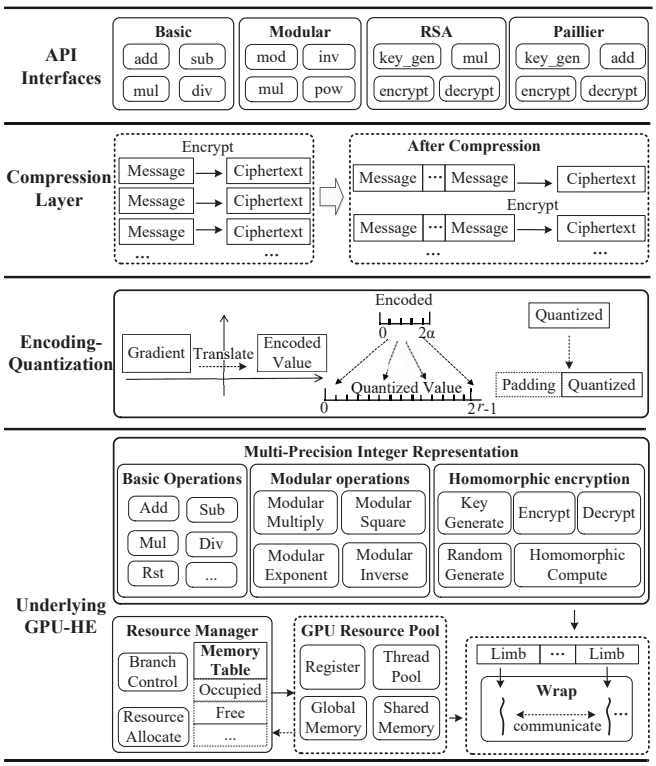


Fig. 3. The architecture of FLBooster

IV. FLBooster

In Fig. 3, FLBooster comprises four layers, including GPU-HE, Encoding-Quantization, Compression, and API Interfaces.

A. GPU-HE (GHE)

GPU-HE is an underlying computation layer of FLBooster, which offers parallel homomorphic encryption ability (including encryption, decryption, and homomorphic computation). To design acceleration algorithms in GPU environments, two challenges exist. In industrial federated learning scenarios, due to the high-security criteria of HE, only HE with enough large key size can be allowed to use, *i.e.,* 1024 bits or even larger. Thus, the first challenge of GPU-HE is how to represent multi-precision integers including the ciphertext, the public key, and the private key, and define the arithmetic operations on multi-precision integers, such as basic operations and modular operations, as shown in Fig. 3. The second challenge is, how to maximize the use of GPU resources, including the number of threads, the number of registers, the size of GPU memory, etc. To address the above two technical challenges, we implement the multi-precision integer representation method and an efficient GPU resource manager. Based on that, we can perform GPU-based homomorphic encryption efficiently.

*1) **Multi-precision integer representation**:* We implement a radix-based multi-precision number system named FRNS to represent multi-precision integers on GPUs. Specifically, we split the multi-precision integer into multiple limbs, which are then processed by multiple threads in parallel. Compared with other multi-precision number systems like RNS [25],

FRNS shows much better efficiency when performing complex operations such as comparison, division, and modular operations [23]. When performing the addition or subtraction of two multi-precision integers, we store the overflow result in the thread locally and then propagate the overflow result to other threads for the carry and borrow operations via inter-thread communication. When performing multiplication of two multi-precision integers, we multiply the limbs with the limbs in other threads one by one, aggregate and propagate the result to other threads, and use two multi-precision integers of the same size with others to represent the more significant words and less significant words of the final result, respectively. In addition, we replace complex division and rest operations with multiple subtraction and multiplication operations. The quotient is obtained by dividing two multi-precision integers using more significant words. After that, we subtract the product of the quotient and the denominator from the numerator. If the result of subtraction overflows, then we recover it by addition. This process is repeated until the numerator is small than the denominator. In FLBooster, we use the unsigned number system with base 2^w to represent integers, where w is the size of a word. For example, $w = 32$ is used in 32-bit systems and $w = 64$ is used in 64-bit systems. An integer m is represented by $s = \lceil k/w \rceil$ words, where $k = \lceil \log_2 m \rceil$ is the number of bits of m. For a GPU program with d threads, each thread processes s/d limbs or words.

*2) **Resource manager**:* To fully release the computation power of GPUs, we develop a GPU-resource manager to manage and balance the GPU resources, *i.e.,* the number of threads, the number of registers, and the size of memory. As shown in Fig. 3, the resource manager stores the common block sizes and adjusts the block size by allocating the corresponding thread numbers in stream multiprocessors (SMs) according to the number of tasks, fully using the resources in the thread pool. Also, it marks the allocated GPU memory addresses to reduce memory allocation costs. When a thread calls for memory, it looks for a free address in the memory table to allocate and marks it occupied. Besides, the resource manager allocates an appropriate number of registers and memory size used by each thread based on tasks to make full use of resources on the GPU. Moreover, the resource manager can also manage branches. If threads encounter an unexpected branch issue, the threads in a warp will be split into several parts to execute the program separately, resulting in double or even several times the number of registers. In view of this, the resource manager can improve performance by combining branch issues or executing the branch code as a warp. Based on that, it can significantly reduce the number of registers used.

*3) **Parallel homomorphic encryption**:* Based on multi-precision representation and resource manager in FLBooster, we develop an efficient GPU-based parallel homomorphic encryption. Basically, homomorphic encryption includes two most important modular operations, *i.e.,* modular multiplication and modular exponentiation operations.

To support multi-precision modular multiplication, five

Algorithm 2: The parallel Montgomery multiplication

Input: three x-word integers a, b and n (x is the number of words assigned for each thread); $r = 2^{wx}$ (w is the word size in the system and $n<r$); $n_0^{'} = -n_0[0] \mod 2^w$; the number of threads T

Output: Montgomery multiplication $abr^{-1} \mod n$, where $rr^{-1} \mod n = 1$

```
1  for i ← 0 to T − 1 do
2      for j ← 0 to x − 1 do
3          S ← 0 // S is used to store the sum result;
4          C ← 0 // C is used to store the carry out;
5          for k ← 0 to x − 1 do
6              (C, S) ← t[k] + a[k]b_i[j] + C;
7              t[k] ← S;
8          (C, S) ← t[x] + C;
9          (t[x], t[x + 1]) ← (S, t[x + 1] + C);
10         m ← t_0[0]n_0^{'};
11         for k ← 0 to x − 1 do
12             (C, S) ← t[k] + mn[k] + C;
13             t[k] ← S;
14         (C, S) ← t[x] + C;
15         (t[x], t[x + 1]) ← (S, t[x + 1] + C);
16         for k ← 0 to x do
17             t[k] ← t[k + 1];
18 if t_{T−1} > 0 then
19     C ← 0;
20     for i ← 0 to x − 1 do
21         (C, S) ← t[i] − n[i] + C;
22         t[i] ← S;
23 return t;
```

CPU-based implementations of Montgomery multiplication (SOS, CIOS, FIOS, FIPS, and CIHS) [34] have been proposed, and among them, the CIOS method has the lowest running time and takes the least storage space. In this paper, we implement the CIOS method on the GPU for parallel computation, as detailed in Algorithm 2. For large integers, each of them can be represented as s words as discussed in the multi-precision integer representation IV-A1. Assuming that we have T threads in total, Algorithm 2 runs for each thread, which takes $x = s/T$ words as inputs to represent each large integer. Algorithm 2 takes three x-word integers a, b and n, $r = 2^{wx}$ (w is the word size and $n < r$), $n_0^{'} = -n_0[0] \mod 2^w$, and the number of threads T as inputs. It outputs the Montgomery multiplication result $abr^{-1} \mod n$. Note that, as we have T threads, we use a_i, b_i, and n_i to denote the x-word run on i-th thread, while we use $a[j]$, $b[j]$ and $n[j]$ to denote the j-the word in the thread. Algorithm 2 first computes $t \leftarrow ab \mod r$ on GPUs (lines 3–9). Specifically, the algorithm gets b_i (the $i-th$ thread of b) via inter-thread communication and multiply b_i with each word of a. Next, to implement $m \leftarrow tn^{'} \mod r$, the lowest significant words of the multi-precision integer t is multiplied with $n_0^{'}$ to compute m (line 10). Then, according to the property of modular inverse (*i.e.,* in the 0-th thread, $mn[0] = t_0[0]n_0^{'}n[0]$ equals to $-t_0[0]$), when multiplying m with n and adding it with t, the lowest

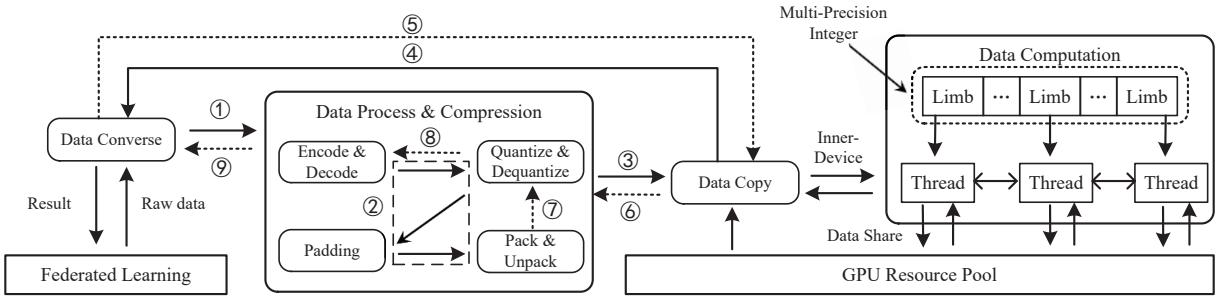


Fig. 4. The pipelined data processing in FLBooster

significant words of t can be processed as 0 (lines 11–15), and a shift operation can be performed (lines 16–17). After that, if there is an overflow (*i.e.,* $t_{T-1} > 0$), a subtraction is performed (lines 19–22). Finally, the result of Montgomery multiplication $abr^{-1} \mod n$ is returned (line 23).

To support modular exponentiation, we integrate our GPU implementation of Montgomery modular multiplication with an extension of the sliding window exponential method [45], successfully reducing the complexity of modular exponentiation from e to $\log_{2^b} e$, where e is the power exponent and b is the number of bits selected for modular multiplication.

In addition, we develop a random number generator for large integers (including Miller-Rabin [46] large prime number generator), assigning a random number generator for each thread in a warp. The Miller-Rabin large prime number generator is used in the key generation phase and the large prime numbers p and q are generated using the Miller-Rabin primality test. To ensure consistency between threads and the data, and simultaneously improve the efficiency of computing multi-precision integers, the lengths of the large prime number p and q are the same as the length of other large integers. After implementing the core arithmetic described above, it is convenient to realize the homomorphic computation, which only requires a modular multiplication or modular addition with the public key for additive or multiplicative homomorphism.

B. Encoding-Quantization

Since the homomorphic encryption used by federated learning only supports unsigned integers, the signed gradient data must be converted to unsigned integers before being encrypted. However, existing federated learning solutions tend to use an insecure manner to quantize the gradient values by encoding and encrypting the significand, leaving the exponent in plaintexts [31], *i.e.,* (*encrypt(significand), exponent*). It leaks the approximate interval of data and the model to adversaries, which also causes data redundancy, increasing communication costs. Thus, it is required to design a new safe quantization method to convert gradient values into unsigned integers. In this paper, we propose an encoding-quantization method.

According to the distribution of gradients, for a real number $m \in [-\alpha, \alpha]$, we first map the gradient to non-negative numbers by linear translation:

$$e = m + \alpha, \tag{6}$$

where α denotes the bound of the gradient, usually smaller than 1. Then we use r bits to amplify the encoded gradient:

$$q = e * (2^r - 1) \tag{7}$$

To avoid overflow in gradients aggregation, we keep the minimum number of overflow bits $b = \lceil \log_2 p \rceil$ for the number of participants p and use the remaining number of bits to quantize the gradient value to integers with less error. Therefore, a total of $b + r$ bits are required:

$$z = \underbrace{[00...]}_{b\text{-}bits} \underbrace{[q]}_{r\text{-}bits} \tag{8}$$

The proposed encoding-quantization method has many benefits. First, it quantizes the gradient in a secure manner without any data leakage compared to existing solutions. Second, it does not need to store the significand and exponent of floating points separately, saving storage occupation and communication costs. Third, the quantization error is small enough to be negligible. This is because, the key of homomorphic encryption reaches at least 1024 bits, and the number of quantization bits chosen is usually no less than 32 bits. Thus, it does not cause a large quantization error. Moreover, since the number of participants is known in advance, a certain number of overflow bits are reserved so that no overflow or inconsistent decryption results occur. Finally, our encoding-quantization technique can serve as a lightweight plug-in that does not need complex arithmetic operations, thus the time spent on encoding and quantization is extremely small.

C. Batch Compression (BC)

To address the communication overhead resulting from homomorphic encryption with a large key size, we develop a batch compression module in FLBooster.

Based on the encoding-quantization method, we develop batch compression to pack multiple plaintexts into a multi-precision integer and encrypt it at one time. It can not only reduce the number of ciphertexts in communication, but also reduce the complex encryption, decryption, and homomorphic computation. After the encoding and quantization, the gradient will be converted into unsigned integers with a fixed number of bits. Given key size k and the number of participants p,

TABLE I
THE PROVIDED APIS OF FLBOOSTER

FLBooster APIs	Description
add/sub(values1, values2)→ res	Computes the *addition/subtraction* result of values1 and values2
mul/div(values1, values2)→ res	Computes the *multiplication/division* result of values1 and values2
mod(x, n)→ res	Computes the *remainder* result of x % n
mod_inv(x, n)→ res	Computes the *modular inverse* result of x and n
mod_mul(values1, values2, n)→ res	Computes the *modular multiplication* result of values1 * values2 % n
mod_pow(x, p, n)→ res	Computes the *modular exponentation* result of $x^p\%n$
RSA::key_gen(size)→ (pri_key, pub_key)	Generates the *keypair* (public key, private key) of RSA with size
RSA::encrypt(pub_key, plaintext)→ c	*Encrypts* the plaintext with the public key
RSA::decrypt(pri_key, c)→ plaintext	*Decrypts* the ciphertext with the private key
RSA::mul(pub_key, ciphertext1, ciphertext2)→ c	Computes the *homomorphic multiplication* result of ciphertext1 and ciphertext2 with the public key
Paillier::key_gen(size) → (pri_key, pub_key)	Generates the *keypair* (public key, private key) of Paillier with size
Paillier::encrypt(pub_key, plaintext)→ c	*Encrypts* the plaintext with the public key
Paillier::decrypt(pri_key, c)→ plaintext	*Decrypts* the ciphertext with the private key
Paillier::add(pub_key, ciphertext1, ciphertext2)→ c	Computes the *homomorphic addition* result of ciphertext1 and ciphertext2 with the public key

$n = \lfloor \frac{k}{r+\lceil \log_2 p \rceil} \rfloor$ plaintexts will be packed into a large integer, and then encrypted for transmission:

$$Z = \underbrace{[00...]}_{b\text{-}bits}\ \underbrace{[q_0]}_{r\text{-}bits}[00...][q_1]...[00...][q_i]...[00...][q_{n-1}] \quad (9)$$

Batch compression has several benefits. First, it can not only reduce the number of ciphertexts in communication but also reduce the times of homomorphic encryption operations. Second, it effectively reduces the number of ciphertexts and increases the compression ratio. If we use $r + b = 32$ bits, for homomorphic encryption with key size $k = 1024$, we can pack 32 plaintexts into a single one and theoretically achieves compression rate of 32×, 64× at 2048 key size, and 128× at 4096 key size. Overall, it fully utilizes the plaintext space and compresses data to almost twice the unencrypted message size. Furthermore, batch compression does not compress the ciphertext directly and thus no erroneous decryption results occur. Combined with the GPU-HE module in Section IV-A, the BC module used in our FLBooster acceleration system can significantly improve the speedup by two or three orders of magnitude with theoretical guarantees.

D. API Interfaces

We wrap commonly used arithmetic operations of FLBooster into user-friendly APIs, including fundamental operations of arithmetic, modular operations, and homomorphic encryption operations (key generation, encryption, decryption, and homomorphic computation) for developers to accelerate federated learning applications. The provided APIs are summarized as follows, also shown in Table I.

- **add/sub/mul/div** are the most fundamental APIs, which can efficiently compute the addition, subtraction, multiplication, and division of two multi-precision integer arrays.
- **mod** computes the remainder of two multi-precision integer arrays and is the basic method of other modular operations.
- **mod_inv** computes the modular inverse of two multi-precision integer arrays, used to generate key pair (*i.e.,* public key and private key) of RSA and Paillier.

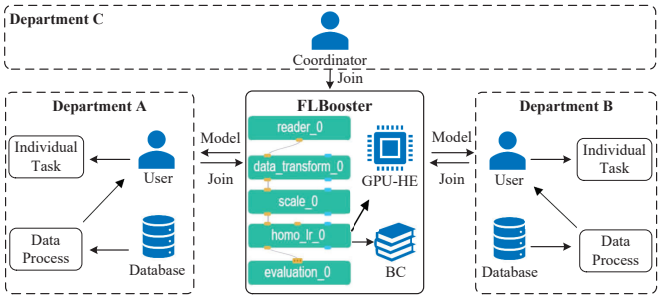


Fig. 5. The deployment of FLBooster in HIK

- **mod_mul** is the modular multiplication function implemented on the GPU by Montgomery multiplication and plays a significant role in modular exponentiation and HE.
- **mod_pow** is one of the most essential functions for the implementation of homomorphic encryption on the GPU.
- **Paillier::key_gen/RSA::key_gen** uses the random multi-precision integer generator to generate the Paillier/RSA key pair of specified key size.
- **Paillier::encrypt/RSA::encrypt** consists of modular operations defined above to encrypt the plaintext.
- **Paillier::decrypt/RSA::decrypt** consists of modular operations defined above to decrypt the ciphertext.
- **Paillier::add/RSA::mul** computes the modular multiplication of ciphertexts to calculate Paillier/RSA homomorphic addition/multiplication of plaintexts.

As shown in Fig. 5, we show the deployment of FLBooster in HIKVISION. Specifically, users of each department load their raw data from their own database and transfer that data into FLBooster for distributed federated learning, and such joint model training of department-to-FLBooster and FLBooster-to-department is accelerated by GPU-HE and batch compression without privacy leakage.

V. PIPELINED PROCESSING AND THEORETICAL ANALYSIS

In this section, we first present the pipelined processing of FLBooster and then provide the theoretical analysis.

A. Pipelined Processing

We proceed to show the data processing pipeline of FLBooster, as depicted in Fig. 4, including key generation, encryption, decryption, and homomorphic computation.

(i) Encryption. In the encryption phase (denoted by ①-④ in Fig. 4), FLBooster loads the gradient data from clients and performs data conversion. Then the data is encoded, quantified, padded, and packed in the data processing and data compression process. FLBooster copies the data to the GPU for data computation and then copies the result back from the GPU. During data computation, the resource manager makes an allocation of resources and manages the branch according to task parameters. Finally, we return the data result to federated learning after the data conversion.

(ii) Decryption. In the decryption phase (denoted by ⑤-⑨ in Fig. 4), FLBooster loads the ciphertext from clients and performs data conversion. After that, the data is copied to the GPU in data computation and then the plaintext result is copied back from the GPU. Note that, the plaintext is unpacked, unquantified and decoded during the data processing and data compression. Finally, the data is returned to federated learning after the data conversion.

(iii) Homomorphic computation. In the homomorphic computation phase (denoted by ④ and ⑤ in Fig. 4), FLBooster loads the key parameters from the server and performs data conversion. Since the raw data and the result are both ciphertexts, FLBooster copies the data to the GPU and the result from the GPU in data computation without data processing and compression. Finally, we return the data to federated learning after the data conversion.

B. Theoretical Analysis

In this subsection, we give a theoretical analysis of FLBooster. In terms of the GPU-HE module, we divide the GPU processing into three stages: (i) The CPU copies the data into GPU memory; (ii) The GPU performs the parallel computation of HE operations with the data; (iii) The GPU copies the result of the computation into the CPU memory. We assume that β_{cpu} is the time for the CPU to process a HE operation, $\beta_{transfer}$ is the time per byte of data copied between the CPU and the GPU, and β_{gpu} is the time for the GPU to process a HE operation. In a 32-bit system, we use the CPU total time t_{cpu} and the GPU total time t_{gpu} to compute the acceleration ratio of the GPU-HE module AC_{ghe}.

$$AC_{ghe} = \frac{t_{cpu}}{t_{gpu}} = \frac{n\beta_{cpu}}{(\frac{L_{before}}{8} + \frac{L_{after}}{8}) * \beta_{transfer} + \frac{32T_{max}}{L_{after}} * \beta_{gpu}}, \tag{10}$$

where n is the cardinality of data, L_{before} is the size of data before being processed, L_{after} is the size of data after being processed, and T_{max} is the maximum number of threads running simultaneously in the GPU. For encryption, the input data is plaintext and L_{before} equals 32 bits. The output data is the ciphertext and L_{after} equals the length of the ciphertext.

TABLE II
STATISTICS OF THE EVALUATION DATASETS.

Dataset	Instances	Features	Size	Domain
RCV1	677,399	47,236	1.1 GB	NLP
Avazu	1,719,304	1,000,000	321 MB	Advertising
Synthetic	100,000	10,000	18 GB	-

For decryption, the input data is the ciphertext and L_{before} equals to the length of the ciphertext. The output data is the plaintext and L_{after} equals 32 bits. For homomorphic computation, both input and output data are ciphertext data. If we want to increase the acceleration ratio of the GPU-HE module, we have to increase the GPU resource utilization and SM utilization, which determines the number of threads running at a time.

In terms of the compression module, we first analyze its compression ratio and plaintext space utilization (PSU):

$$Compression\ Ratio = \frac{n}{\lceil \frac{n}{\lfloor \frac{k}{r+\lceil \log_2 p \rceil} \rfloor} \rceil} \leq \frac{k}{r + \lceil \log_2 p \rceil}, \tag{11}$$

$$PSU = \frac{n * (r + \lceil \log_2 p \rceil)}{k * \lceil \frac{n}{\lfloor \frac{k}{r+\lceil \log_2 p \rceil} \rfloor} \rceil} \leq 1, \tag{12}$$

where n denotes the cardinality of data, k is the key size, p is the number of participants, and r is the number of quantization bits. The acceleration ratio of the BC module in HE operations AC_{bc} equals the above compression ratio, as the BC module not only reduces the amount of communication but also brings a reduction in the number of HE operations.

$$AC_{bc} = Compression\ Ratio = \frac{n}{\lceil \frac{n}{\lfloor \frac{k}{r+\lceil \log_2 p \rceil} \rfloor} \rceil} \tag{13}$$

According to Eqs. 11–13, if we want to improve the compression ratio, PSU, and AC_{bc}, while ensuring the model accuracy, we have to choose an appropriate r according to other parameters. We give an optional solution that for most cases in federated learning, the model accuracy, compression rate, and plaintext space utilization are satisfied when $r + \lceil \log_2 p \rceil$ is chosen as a multiple of 32. In the HE operation, the total acceleration ratio of FLBooster is the multiplication of the acceleration ratio of GHE module by that of BC module:

$$AC = AC_{ghe} * AC_{bc} \tag{14}$$

VI. EXPERIMENT

In this section, we provide empirical results to demonstrate the effectiveness and efficiency of proposed FLBooster. The extensive experiments are designed to answer the following research questions (RQs):

- **RQ1:** How does FLBooster perform compared to state-of-the-art methods?
- **RQ2:** To what extent does FLBooster improve the throughput and hardware utilization compared to existing methods?
- **RQ3:** How does each module affect FLBooster?
- **RQ4:** How does each component of FLBooster cost?
- **RQ5:** How does FLBooster affect the convergence of FL models?

TABLE III
THE AVERAGE RUNNING TIME IN SECONDS

Dataset		RCV1			Avazu			Synthetic		
Model	**Key Size**	FATE	HAFLO	**FLBooster**	FATE	HAFLO	**FLBooster**	FATE	HAFLO	**FLBooster**
Homo LR	1024	10009.9	5472.6	**57.8**	79457.9	44747.4	**370.5**	1327.2	691.6	**9.2**
	2048	35857.9	8695.6	**73**	284661.6	70177.4	**541.1**	4747.2	1300.1	**11.1**
	4096	175666.1	17234.4	**163.4**	1394577.8	138542	**1134.7**	23245.9	2581.8	**24.2**
Hetero LR	1024	4760	2139.6	**33.6**	25109.8	12151.5	**172.5**	706.6	319.8	**8.4**
	2048	17596.3	3879.9	**42.4**	89817.7	21987.4	**236.3**	2737.6	712.8	**9.7**
	4096	87857.2	7765.7	**92.9**	439813.5	43744.6	**465.8**	13233.9	1410.5	**17.1**
Hetero SBT	1024	36489.2	17931.3	**775.8**	92526.3	45424.6	**1882.3**	5462.3	2722.7	**190.1**
	2048	129159	32017.5	**838.1**	327731.2	81176.8	**2040.5**	19142.5	4802.1	**199.3**
	4096	630396.6	63176.5	**1195.8**	1599920.8	160261.4	**2948.3**	93137	9402	**252.1**
Hetero NN	1024	26696.7	14366.7	**222.7**	83324.7	47036.6	**648.4**	3974.2	2195.5	**66**
	2048	95391.9	23196.7	**291.1**	297855.2	73622.4	**867.4**	14115.3	3894.1	**71.8**
	4096	466954.4	45666.3	**543.6**	1458219.2	145094	**1625.9**	68966.6	7694.3	**110.6**

A. Experimental Settings

Datasets. To support various federated learning scenarios, we use three public datasets to evaluate FLBooster in the following experiments, namely RCV1, Avazu, and Synthetic.

- **RCV1**[1] is a benchmark dataset on text categorization. It contains manually labeled newswire documents which are categorized with respect to three controlled vocabularies: industries, topics, and regions, widely used in natural language processing and federated learning [20], [73].
- **Avazu**[2] is a public online advertising dataset, which is used for click-through rate (CTR) prediction. Specifically, the task of the Avazu dataset is to predict how likely a user would click an item given his/her features. It is widely used in machine learning and federated learning [15], [27].
- **Synthetic**[3] is one of the benchmark datasets for federated learning, which is proposed in [39]. It is used for a classification problem that considers resource allocation and multiple centers in federated learning [8], [13], [30].

These datasets vary in terms of domain, size, sparsity, and features. RCV1 and Avazu are sparse datasets and smaller in size than the Synthetic dense dataset. Table II summarizes the statistics of three datasets.

Benchmark FL Models. To illustrate the generality of FLBooster, we choose four representative federated learning models: homogeneous logistic regression (Homo LR), heterogeneous logistic regression (Hetero LR), heterogeneous secureboost (Hetero SBT), and heterogeneous neural network (Hetero NN). For the homogeneous model, we horizontally divide three datasets into subsets of the same number of data instances where each participant shares the same feature space but is different in samples. For heterogeneous models, we vertically divide three datasets into subsets of the same number of features, where each participant shares the same sample ID space but differs in feature space.

Competitors. We compare FLBooster with two representative federated learning frameworks:

- **FATE** [4] is the world's first industrial-grade federated learning framework. It implements secure computation protocols based on homomorphic encryption and multi-party computation, supporting many federated learning scenarios.
- **HAFLO** [18] is the state-of-the-art FL acceleration system. It summarizes the performance-critical homomorphic encryption operations in the model training and uses the GPU to accelerate these operations.

B. Evaluation Metrics and Implementation

Evaluation Metrics. We evaluate FLBooster from both computation and communication aspects.

- **Computation evaluation**. We utilize the average running time per epoch to show the training efficiency and report the time usage and proportions of each critical component during federated learning.
- **Communication evaluation**. We adopt the compression ratio and time usage to analyze the effectiveness of batch compression. We also introduce the convergence bias to evaluate the model error due to the quantization:

$$Convergence\ \ Bias = \frac{|\mathcal{L} - \mathcal{L}_{FLBooster}|}{\mathcal{L}}, \qquad (15)$$

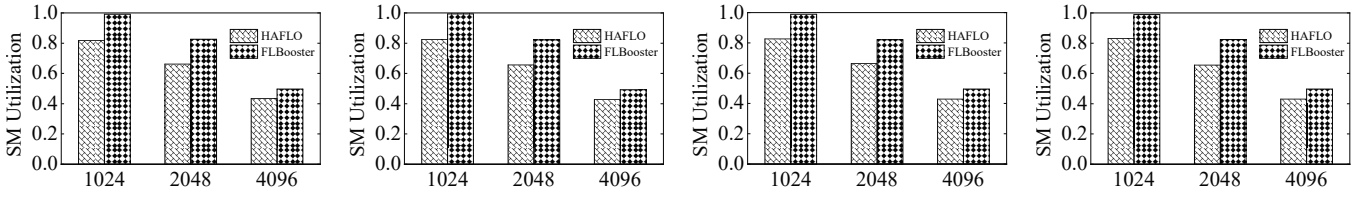
where $\mathcal{L}$ is the loss of the FL model trained without compression techniques, $\mathcal{L}_{FLBooster}$ is the loss of the model optimized by our method. Low convergence bias means that the trained model is less influenced by FLBooster.

Implementation Details. FLBooster is implemented on FATE [4], an open-source and benchmark framework of industrial federated learning. FLBooster uses the CUDA toolkit to write all codes in C++ and wraps the crucial operation with simple Python APIs as plugin acceleration components. All experiments are conducted on four computing servers in the federated learning environment of the company, each of which connects in Gigabit Ethernet and is equipped with two Intel(R) Xeon(R) CPU E5-2650 v4 @ 2.20GHz processors with 12 cores, 128GB RAM, and an NVIDIA GeForce RTX 3090. Although we implement it with FATE, it can be easily adapted to other federated learning frameworks like TensorFlow Federated [5], PySyft [77], FederatedScope [61], etc.

[1]https://trec.nist.gov/data/reuters/reuters.html
[2]https://www.kaggle.com/c/avazu-ctr-prediction/data
[3]https://leaf.cmu.edu/

TABLE IV
THE THROUGHPUT IN HE OPERATIONS (INSTANCES PER SECOND)

Dataset		RCV1			Avazu			Synthetic		
Model	**Key Size**	FATE	HAFLO	**FLBooster**	FATE	HAFLO	**FLBooster**	FATE	HAFLO	**FLBooster**
Homo LR	1024	363	58823	**398309**	392	61652	**515576**	357	54290	**331925**
	2048	69	9783	**64782**	65	10634	**93488**	67	9224	**52870**
	4096	12	1709	**11316**	12	1793	**15303**	12	1614	**9098**
Hetero LR	1024	364	58717	**512111**	364	62841	**536499**	341	61369	**455210**
	2048	68	9969	**83291**	70	10710	**83255**	66	10975	**74037**
	4096	12	1873	**14548**	12	1902	**15465**	12	1816	**12932**
Hetero SBT	1024	389	59572	**516886**	354	62754	**525673**	391	62936	**459454**
	2048	73	10809	**77234**	72	10330	**91798**	72	9883	**68652**
	4096	11	1870	**14369**	12	1921	**16838**	12	1821	**12772**
Hetero NN	1024	359	61145	**498254**	369	61442	**515283**	377	65338	**442893**
	2048	67	10446	**79788**	65	10932	**90516**	74	11272	**70922**
	4096	12	1779	**14806**	12	1648	**16303**	12	1786	**13161**



(a) Homo LR (b) Hetero LR (c) Hetero SBT (d) Hetero NN

Fig. 6. The GPU utilization in HE operations

Parameter Settings. We set all model and optimizer parameters to their default values: the penalty method is set to $L2$ normalization with a coefficient equal to 0.01 for all models; the batch size is set as 1024, and Adam [33] optimizer is used to train the models. For batch compression, 32 bits are used to quantize 32-bit float gradients, where the last two bits are used for computational overflow. The tolerance of convergence is set to 10^{-6}, which means that if the loss difference between two successive epochs is less than 10^{-6}, the model reaches convergence. We divide each dataset into 64 partitions and upload them to each server for federated learning.

C. Overall Performance Comparison (RQ1)

We begin with the performance comparison *w.r.t.* the average running time per epoch, where we change the key size from 1024 to 4096. The results are reported in Table III, from which we have the following observations.

First, FLBooster can boost federated learning by a large margin and outperform all the baselines on three datasets and four benchmark models. Specifically, FLBooster acquires $14.3\times \sim 138\times$ acceleration ratio, compared with the state-of-the-art HAFLO. We attribute this improvement to the parallel design and batch compression in FLBooster: (i) by devising the parallel mechanism of homomorphic encryption on the GPU, the computation overhead of HE operations can be largely reduced; (ii) by our privacy quantization and compression, the redundant communication can be significantly reduced. In contrast, other baselines are insufficient since they cannot address the inefficiency issues considering both computation and communication aspects.

Second, jointly analyzing the results of FLBooster across different models, we find its acceleration ratios vary. Specifically, the acceleration ratio of Hetero SBT is less than that of Homo LR and Hetero LR. This is because LR is the linear model with much lower complexity (*i.e.,* gradient computation and model update) than Hetero SBT, meaning the majority of running time is spent on HE operations and communication. Thus, FLBooster enables improving the training efficiency significantly by parallel HE and batch compression.

Third, by jointly analyzing the result of FLBooster across different datasets, we can observe that FLBooster achieves the highest acceleration ratios in Avazu. This is because, the dimension of features in Avazu is higher than that of the other two datasets, and high-dimensional features can make full use of the parallelism of GPUs. Moreover, the acceleration ratio of FLBooster increases with the key size. The reasons are two-fold: (i) As the key size increases, the complexity of HE also increases. However, thanks to the parallel design of FLBooster, it can have a higher utilization on GPUs. (ii) The size of ciphertexts is also expanded as the key size increases, which indicates that more data need to be transmitted. And our method can achieve a higher compression ratio and thus reduce data volumes in communication.

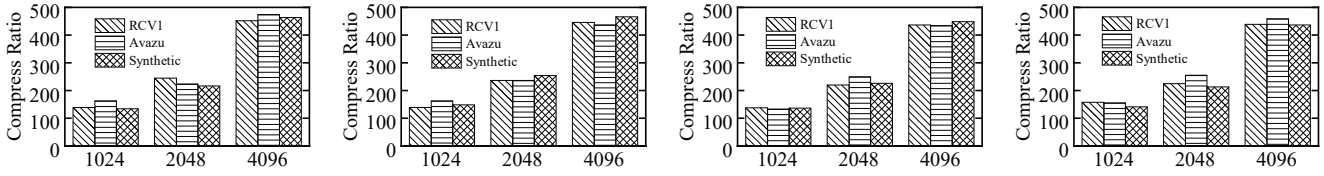
D. Throughput and Hardware Utilization (RQ2)

We proceed to compare FATE, HAFLO, and FLBooster in terms of throughput and hardware utilization in HE operations.

*1) **Throughput**:* As shown in Table IV, the throughput of the CPU in HE operations is quite smaller compared with GPU-based methods, and FLBooster maintains the highest throughput in all models and datasets. Besides, the results of FLBooster on different datasets show different throughput, which contributes to the different dimensions of data features. The larger feature dimension achieves higher throughput due

TABLE V
THE AVERAGE RUNNING TIME OF MODULES IN FLBOOSTER PER EPOCH IN SECONDS

Dataset		RCV1			Avazu			Synthetic		
Model	**Key Size**	FLBooster	w/o GHE	w/o BC	FLBooster	w/o GHE	w/o BC	FLBooster	w/o GHE	w/o BC
Homo LR	1024	**57.8**	217.1	4826.4	**370.5**	1635.3	38306.9	**9.2**	30.2	641.3
	2048	**73**	478.3	7492.5	**541.1**	3759.2	64424.8	**11.1**	64.7	1046.9
	4096	**163.4**	1287.2	17985.4	**1134.7**	10217.7	143821.3	**24.2**	172.9	2571.2
Hetero LR	1024	**33.6**	114.3	2136	**172.5**	571.3	12133.5	**8.4**	20.3	319.3
	2048	**42.4**	247.6	3288.7	**236.3**	1248.1	20451.8	**9.7**	39.9	710.5
	4096	**92.9**	661.8	7725.2	**465.8**	3278.9	43544.2	**17.1**	101.1	1404.6
Hetero SBT	1024	**775.8**	1347	17905.4	**1882.3**	3332	45358.9	**190.1**	274.4	2718.9
	2048	**838.1**	2291.4	31908.9	**2040.5**	9606.8	80901.1	**199.3**	413.8	4786.1
	4096	**1195.8**	5224.5	62889.4	**2948.3**	13173.6	159532.8	**252.1**	846.8	9359.6
Hetero NN	1024	**222.7**	646.1	12920.8	**648.4**	1970.7	40303.4	**66**	128.5	1940.6
	2048	**291.1**	1368.5	23301.4	**867.4**	7748.6	71692.4	**71.8**	230.8	3473
	4096	**543.6**	3530	46267	**1625.9**	10952.9	144441.3	**110.6**	551.5	6863.3



(a) Homo LR (b) Hetero LR (c) Hetero SBT (d) Hetero NN

Fig. 7. The compression ratio of FLBooster

to the parallel design on the GPU. In addition, the results of Homo LR show relatively smaller throughput improvement compared to other models, since the GPU utilization is nearly optimal, as detailed in the following experiments.

*2) **Hardware Utilization**:* We also leverage the SM utilization on the GPU to investigate the hardware utilization for HAFLO and FLBooster, as shown in Fig 6. Although HAFLO also utilizes the GPU for HE operations, our method is able to better leverage the resource and achieve higher utilization. Besides, the demands for registers, memory, and other resources increase with the key size, and thus, the SM performance degrades due to the lack of enough resources.

E. Ablation Study (RQ3)

As the GPU-HE and batch compression modules are two cores of FLBooster, we conduct a comprehensive ablation study to investigate their effectiveness. Specifically, we use the absence of GHE and batch compression modules to illustrate the effects on FLBooster. The results are reported in Table V.

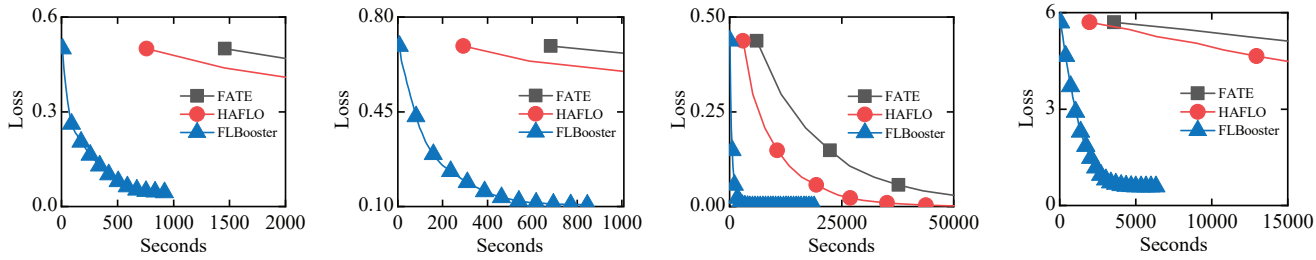
*1) **Impact of GPU-HE**:* We first evaluate the effects of GPU-HE, by discarding the parallel computation for HE operations of FLBooster, called w/o GHE. Compared with the complete method FLBooster, the absence of the parallel HE mechanism dramatically degrades the performance, indicating the necessity of accelerating HE operations with the GPU. Specifically, w/o GHE directly performs the complex HE operations on the CPU, which results in huge computation overhead for FL training. For instance, the average running time increases 4 times on the RCV1 dataset for Homo LR, which is unacceptable in real-world applications.

*2) **Impact of Compression**:* We also investigate the utility of batch compression by removing the BC module during communication, which is denoted as w/o BC in Table V. It is obvious that the training time reduces drastically with the help of batch compression, and FLBooster achieves $14.3\times \sim 126.7\times$ acceleration ratios compared to w/o BC. The second observation is that the acceleration ratio of FLBooster to w/o BC is also different across all datasets. This is because the BC module not only reduces the data size in communication but also reduces the number of HE operations since the amount of data needed to be encrypted also decreases.

TABLE VI
COMPONENT RUNNING TIME IN SECONDS AT 1024 KEY SIZE

Dataset	Method	Others	HE operations	Communication
RCV1	FATE	0.1%	52.0%	47.9%
	HAFLO	0.2%	0.6%	99.2%
	FLBooster	**22.1%**	**5.9%**	**72.0%**
Avazu	FATE	0.1%	52.0%	47.9%
	HAFLO	0.2%	0.5%	99.3%
	FLBooster	**26.8%**	**7.3%**	**67.9%**
Synthetic	FATE	0.3%	51.9%	47.8%
	HAFLO	0.6%	0.6%	98.8%
	FLBooster	**47.9%**	**5.4%**	**46.7%**

Besides, we also analyze the compression ratio of FLBooster *w.r.t.* different key sizes, as shown in Fig. 7. By packing multiple plaintexts into a single one, FLBooster is able to greatly reduce the number of ciphertexts and achieves 2 orders of magnitude of compression ratios. Besides, the compression ratio of FLBooster to FLBooster w/o BC also increases with the key size. This is because it is possible to pack more plaintexts together with a larger key size, thus leading to a high compression ratio. Moreover, the compression ratio is nearly the same across different datasets and models, since it is an independent component in the encoding-quantization process which only relies on the size of the key.



(a) Homo LR (b) Hetero LR (c) Hetero SBT (d) Hetero NN

Fig. 8. Convergence performance on Synthetic dataset at 1024 key size

F. Component Running Time (RQ4)

Next, we analyze the running time of FLBooster *w.r.t.* different components, including HE operations, communication, and the others (*e.g.,* model computing). The results of Homo LR with 1024 key size are demonstrated in Table VI, while the other models are omitted due to similar observations and limited space. First, HE operations and communication components take up the majority of the training time, the sum of which accounts for more than 99.5% in FATE. It suggests that these two components are the main bottleneck in the training process of federated learning. When HE operations are accelerated using the GPU, there is a significant decrease in the time of HE operations. However, the overall acceleration of HAFLO is inefficient. Because a huge amount of communication cost still exists as the time percentage of communication cost is more than 98%. Therefore, optimization on only one of the HE operations or communication could not achieve a high acceleration ratio in the training process. In FLBooster, we not only accelerate HE operations with the GPU but also compress the huge communication during the training process using the BC module, thus bringing in significant improvements for both HE operations and communication, i.e., the training efficiency improves from $144.3\times \sim 1229\times$ compared to FATE.

G. Model Convergence Analysis (RQ5)

Finally, we compare FATE, HAFLO, and FLBooster for the convergence performance. Note that we only show the convergence results of all models at 1024 key size using Synthetic dataset, since the results of other datasets are similar. As shown in Fig. 8, FLBooster exhibits the fastest convergence, followed by HAFLO, while FATE without any optimizations converges slowly. Specifically, FLBooster achieves an acceleration ratio of $28.7\times$ to $144.3\times$ compared to FATE and has an acceleration ratio of 14.3 to 75.2 times compared to HAFLO. In the Homo LR model, FATE has not yet completed the first epoch of training while FLBooster has already finished its training. The high efficiency of FLBooster is achieved by reducing the computation and communication overhead simultaneously.

In addition, we perform comparisons of convergence bias, which is important for model convergence. We compare the convergence value of FLBooster and FATE to show the impact of the encoding-quantization method in FLBooster at 1024 key size. The metric is the convergence bias, and the formula has been defined in Eq. 15. As shown in Table VII, FLBooster achieves almost the same model quality as FATE and the convergence bias caused by our acceleration system FLBooster is small enough (i.e., much less than 5%) and thus can be ignored across all models and datasets. This is because compared to the large key size, the number of bits we use to quantize the float-point data in the encoding-quantization is enough to store most gradients with less loss. The experiments also indicate the effectiveness of FLBooster. It is also can be observed that the convergence bias of FLBooster is small in Homo and Hetero LR models, while there is a relatively large convergence bias in Hetero SBT and NN models. In this sense, we conjecture that SBT and NN models are more sensitive to model accuracy than the simpler LR linear model.

TABLE VII
THE CONVERGENCE BIAS AT 1024 KEY SIZE

Model	RCV1	Avazu	Synthetic
Homo LR	0.3%	0.5%	0.3%
Hetero LR	0.2%	0.3%	0.2%
Hetero SBT	2.1%	3.3%	1.7%
Hetero NN	1.3%	0.8%	0.8%

VII. CONCLUSION

In this paper, we propose FLBooster, a framework and GPU-accelerated system for federated learning. To maximize the computational resources of the GPU, we parallelize encryption, decryption, and homomorphic computation of homomorphic encryption, flattening the computation overhead. Also, to reduce the communication overhead caused by ciphertext expansion, combined with low-less quantization, batch compression is used to pack multiple plaintexts into a single one. It maximizes the redundant space of the plaintext and reduces not only the amount of data communication but also homomorphic operations. The experiment shows that our system achieves significant performance improvements in federated learning.

VIII. ACKNOWLEDGEMENT

This work was supported in part by the NSFC under Grants No. (62025206, 61972338, and 62102351), the Ningbo Science and Technology Special Innovation Projects with Grant Nos. 2022Z095 and 2021Z019, and the Hangzhou Technology Bureau AI Research Project under Grants No. 2022AIZD0116. Lu Chen is the corresponding author of the work.

REFERENCES

[1] Regulation (eu) 2016/679 of the european parliament and of the council of 27 april 2016 on the protection of natural persons with regard to the processing of personal data and on the free movement of such data, and repealing directive 95/46/ec (general data protection regulation). https://eur-lex.europa.eu/eli/reg/2016/679/oj, 2016.

[2] Cybersecurity law of the people's republic of china. http://www.lawinfochina.com/display.aspx?id=22826&lib=law, 2017.

[3] California consumer privacy act (ccpa). https://oag.ca.gov/privacy/ccpa, 2018.

[4] Fate (federated ai technology enabler). https://github.com/FederatedAI/FATE, 2019.

[5] Tensorflow federated. https://www.tensorflow.org/federated, 2019.

[6] Webank. https://www.webank.com/, 2019.

[7] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, et al. {TensorFlow}: a system for {Large-Scale} machine learning. In *12th USENIX symposium on operating systems design and implementation (OSDI 16)*, pages 265–283, 2016.

[8] A. M. Abdelmoniem and M. Canini. Towards mitigating device heterogeneity in federated learning via adaptive model quantization. In *Proceedings of the 1st Workshop on Machine Learning and Systems*, pages 96–103, 2021.

[9] E. Barker, E. Barker, W. Burr, W. Polk, M. Smid, et al. *Recommendation for key management: Part 1: General*. 2006.

[10] J. Benaloh. Dense probabilistic encryption. In *Proceedings of the workshop on selected areas of cryptography*, pages 120–128, 1994.

[11] K. Bonawitz, V. Ivanov, B. Kreuter, A. Marcedone, H. B. McMahan, S. Patel, D. Ramage, A. Segal, and K. Seth. Practical secure aggregation for privacy-preserving machine learning. In *proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 1175–1191, 2017.

[12] Q. Cai, K. Zheng, B. C. Ooi, W. Wang, and C. Yao. Elda: Learning explicit dual-interactions for healthcare analytics. In *2022 IEEE 38th International Conference on Data Engineering (ICDE)*, pages 393–406. IEEE, 2022.

[13] S. Caldas, S. M. K. Duddu, P. Wu, T. Li, J. Konečnỳ, H. B. McMahan, V. Smith, and A. Talwalkar. Leaf: A benchmark for federated settings. *arXiv preprint arXiv:1812.01097*, 2018.

[14] A.-F. Chan. Symmetric-key homomorphic encryption for encrypted data processing. In *2009 IEEE International conference on communications*, pages 1–5, 2009.

[15] P. P. Chan, X. Hu, L. Zhao, D. S. Yeung, D. Liu, and L. Xiao. Convolutional neural networks based click-through rate prediction with multiple feature sequences. In *IJCAI*, pages 2007–2013, 2018.

[16] W. Chen, G. Ma, T. Fan, Y. Kang, Q. Xu, and Q. Yang. Secureboost+: A high performance gradient boosting tree framework for large scale vertical federated learning. *arXiv preprint arXiv:2110.10927*, 2021.

[17] K. Cheng, T. Fan, Y. Jin, Y. Liu, T. Chen, D. Papadopoulos, and Q. Yang. Secureboost: A lossless federated learning framework. *IEEE Intelligent Systems*, 36(6):87–98, 2021.

[18] X. Cheng, W. Lu, X. Huang, S. Hu, and K. Chen. Haflo: Gpu-based acceleration for federated logistic regression. *arXiv preprint arXiv:2107.13797*, 2021.

[19] J. H. Cheon, W.-H. Kim, and H. S. Nam. Known-plaintext cryptanalysis of the domingo-ferrer algebraic privacy homomorphism scheme. *Information Processing Letters*, 97(3):118–123, 2006.

[20] R. Collobert, J. Weston, L. Bottou, M. Karlen, K. Kavukcuoglu, and P. Kuksa. Natural language processing (almost) from scratch. *Journal of machine learning research*, 12(ARTICLE):2493–2537, 2011.

[21] I. Damgård, M. Jurik, and J. B. Nielsen. A generalization of paillier's public-key system with applications to electronic voting. *International Journal of Information Security*, 9(6):371–385, 2010.

[22] H. Doraiswamy and J. Freire. Spade: Gpu-powered spatial database engine for commodity hardware. In *2022 IEEE 38th International Conference on Data Engineering (ICDE)*, pages 2669–2681. IEEE, 2022.

[23] N. Emmart. A study of high performance multiple precision arithmetic on graphics processing units. 2018.

[24] Y. Gao, M. Kim, C. Thapa, A. Abuadbba, Z. Zhang, S. Camtepe, H. Kim, and S. Nepal. Evaluation and optimization of distributed machine learning techniques for internet of things. *IEEE Transactions on Computers*, 71(10):2538–2552, 2021.

[25] H. L. Garner. The residue number system. In *Papers presented at the the March 3-5, 1959, western joint computer conference*, pages 146–153, 1959.

[26] R. C. Geyer, T. Klein, and M. Nabi. Differentially private federated learning: A client level perspective. *arXiv preprint arXiv:1712.07557*, 2017.

[27] A. Hard, K. Rao, R. Mathews, S. Ramaswamy, F. Beaufays, S. Augenstein, H. Eichner, C. Kiddon, and D. Ramage. Federated learning for mobile keyboard prediction. *arXiv preprint arXiv:1811.03604*, 2018.

[28] 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. *arXiv preprint arXiv:1711.10677*, 2017.

[29] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell. Caffe: Convolutional architecture for fast feature embedding. In *Proceedings of the 22nd ACM international conference on Multimedia*, pages 675–678, 2014.

[30] J. Jiang and L. Hu. Decentralised federated learning with adaptive partial gradient aggregation. *CAAI Transactions on Intelligence Technology*, 5(3):230–236, 2020.

[31] Z. Jiang, W. Wang, and Y. Liu. Flashe: Additively symmetric homomorphic encryption for cross-silo federated learning. *arXiv preprint arXiv:2109.00675*, 2021.

[32] P. Kairouz, H. B. McMahan, B. Avent, A. Bellet, M. Bennis, A. N. Bhagoji, K. Bonawitz, Z. Charles, G. Cormode, R. Cummings, et al. Advances and open problems in federated learning. *Foundations and Trends® in Machine Learning*, 14(1–2):1–210, 2021.

[33] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2015.

[34] C. K. Koc, T. Acar, and B. S. Kaliski. Analyzing and comparing montgomery multiplication algorithms. *IEEE micro*, 16(3):26–33, 1996.

[35] J. Konečnỳ, H. B. McMahan, F. X. Yu, P. Richtárik, A. T. Suresh, and D. Bacon. Federated learning: Strategies for improving communication efficiency. *arXiv preprint arXiv:1610.05492*, 2016.

[36] S. Kumar, L. Akoglu, N. Chawla, J. A. Rodriguez-Serrano, T. Faruquie, and S. Nagrecha. Machine learning in finance. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, pages 4139–4140, 2021.

[37] Q. Li, Z. Wen, Z. Wu, S. Hu, N. Wang, Y. Li, X. Liu, and B. He. A survey on federated learning systems: vision, hype and reality for data privacy and protection. *IEEE Transactions on Knowledge and Data Engineering*, 2021.

[38] T. Li, A. K. Sahu, A. Talwalkar, and V. Smith. Federated learning: Challenges, methods, and future directions. *IEEE Signal Processing Magazine*, 37(3):50–60, 2020.

[39] T. Li, M. Sanjabi, A. Beirami, and V. Smith. Fair resource allocation in federated learning. In *ICLR*, 2019.

[40] Y. Liu, T. Fan, T. Chen, Q. Xu, and Q. Yang. Fate: An industrial grade platform for collaborative learning with data protection. *J. Mach. Learn. Res.*, 22:226:1–226:6, 2021.

[41] H. Ludwig, N. Baracaldo, G. Thomas, Y. Zhou, A. Anwar, S. Rajamoni, Y. Ong, J. Radhakrishnan, A. Verma, M. Sinn, et al. Ibm federated learning: an enterprise framework white paper v0. 1. *arXiv preprint arXiv:2007.10987*, 2020.

[42] Z. Ma, J. Ma, Y. Miao, Y. Li, and R. H. Deng. Shieldfl: Mitigating model poisoning attacks in privacy-preserving federated learning. *IEEE Transactions on Information Forensics and Security*, 17:1639–1654, 2022.

[43] V. Mageirakos, R. Mancini, S. Karthik, B. Chandra, and A. Ailamaki. Efficient gpu-accelerated join optimization for complex queries. In *2022 IEEE 38th International Conference on Data Engineering (ICDE)*, pages 3190–3193. IEEE, 2022.

[44] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas. Communication-efficient learning of deep networks from decentralized data. In *Artificial intelligence and statistics*, pages 1273–1282, 2017.

[45] A. J. Menezes, P. C. Van Oorschot, and S. A. Vanstone. *Handbook of applied cryptography*. CRC press, 2018.

[46] G. L. Miller. Riemann's hypothesis and tests for primality. *Journal of computer and system sciences*, 13(3):300–317, 1976.

[47] P. L. Montgomery. Modular multiplication without trial division. *Mathematics of computation*, 44(170):519–521, 1985.

[48] H. Mustafa, C. Barrus, E. Leal, and L. Gruenwald. Gtraclus: A local trajectory clustering algorithm for gpus. In *2021 IEEE 37th International Conference on Data Engineering Workshops (ICDEW)*, pages 30–35. IEEE, 2021.

[49] X. Nie, X. Miao, Z. Yang, and B. Cui. Tsplit: Fine-grained gpu memory management for efficient dnn training via tensor splitting. In *2022 IEEE 38th International Conference on Data Engineering (ICDE)*, pages 2615–2628. IEEE, 2022.

[50] P. Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *International conference on the theory and applications of cryptographic techniques*, pages 223–238, 1999.

[51] A. Papadimitriou, R. Bhagwan, N. Chandran, R. Ramjee, A. Haeberlen, H. Singh, A. Modi, and S. Badrinarayanan. Big data analytics over encrypted datasets with seabed. In *12th USENIX symposium on operating systems design and implementation (OSDI 16)*, pages 587–602, 2016.

[52] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Köpf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala. Pytorch: An imperative style, high-performance deep learning library. In *NeurIPS*, 2019.

[53] L. T. Phong, Y. Aono, T. Hayashi, L. Wang, and S. Moriai. Privacy-preserving deep learning via additively homomorphic encryption. *IEEE Transactions on Information Forensics and Security*, 13:1333–1345, 2018.

[54] R. L. Rivest, L. Adleman, M. L. Dertouzos, et al. On data banks and privacy homomorphisms. *Foundations of secure computation*, 4(11):169–180, 1978.

[55] F. Sattler, S. Wiedemann, K.-R. Müller, and W. Samek. Robust and communication-efficient federated learning from non-iid data. *IEEE transactions on neural networks and learning systems*, 31(9):3400–3413, 2019.

[56] I. Sharma. Fully homomorphic encryption scheme with symmetric keys. *arXiv preprint arXiv:1310.2452*, 2013.

[57] V. Smith, C.-K. Chiang, M. Sanjabi, and A. S. Talwalkar. Federated multi-task learning. *Advances in neural information processing systems*, 30, 2017.

[58] C. Taylor and M. Gowanlock. Accelerating the yinyang k-means algorithm using the gpu. In *2021 IEEE 37th International Conference on Data Engineering (ICDE)*, pages 1835–1840. IEEE, 2021.

[59] S. L. Tu, M. F. Kaashoek, S. R. Madden, and N. Zeldovich. Processing analytical queries over encrypted data. 2013.

[60] D. Vizár and S. Vaudenay. Cryptanalysis of chosen symmetric homomorphic schemes. *Studia Scientiarum Mathematicarum Hungarica*, 52(2):288–306, 2015.

[61] Z. Wang, W. Kuang, Y. Xie, L. Yao, Y. Li, B. Ding, and J. Zhou. Federatedscope-gnn: Towards a unified, comprehensive and efficient package for federated graph learning. *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, 2022.

[62] L. Xiao, O. Bastani, and I.-L. Yen. An efficient homomorphic encryption protocol for multi-user systems. *IACR Cryptol. ePrint Arch.*, 2012:193, 2012.

[63] J. Xu, W. Du, Y. Jin, W. He, and R. Cheng. Ternary compression for communication-efficient federated learning. *IEEE Transactions on Neural Networks and Learning Systems*, 33:1162–1176, 2022.

[64] W. Xu, H. Fan, K. Li, and K. Yang. Efficient batch homomorphic encryption for vertically federated xgboost. *arXiv preprint arXiv:2112.04261*, 2021.

[65] Q. Yang, Y. Liu, T. Chen, and Y. Tong. Federated machine learning: Concept and applications. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 10(2):1–19, 2019.

[66] S. Yang, B. Ren, X. Zhou, and L. Liu. Parallel distributed logistic regression for vertical federated learning without third-party coordinator. *arXiv preprint arXiv:1911.09824*, 2019.

[67] A. C. Yao. Theory and application of trapdoor functions. In *23rd Annual Symposium on Foundations of Computer Science (SFCS 1982)*, pages 80–91, 1982.

[68] Y. Yu, F. Yu, Z. Xu, D. Wang, M. Zhang, A. Li, S. Bray, C. Liu, and X. Chen. Powering multi-task federated learning with competitive gpu resource sharing. In *Companion Proceedings of the Web Conference 2022*, pages 567–571, 2022.

[69] Q. Zeng, Y. Du, K. Huang, and K. K. Leung. Energy-efficient resource management for federated edge learning with cpu-gpu heterogeneous computing. *IEEE Transactions on Wireless Communications*, 20(12):7947–7962, 2021.

[70] C. Zhang, S. Li, J. Xia, W. Wang, F. Yan, and Y. Liu. {BatchCrypt}: Efficient homomorphic encryption for {Cross-Silo} federated learning. In *2020 USENIX annual technical conference (USENIX ATC 20)*, pages 493–506, 2020.

[71] Q. Zhang, C. Wang, H. Wu, C. Xin, and T. V. Phuong. Gelu-net: A globally encrypted, locally unencrypted deep neural network for privacy-preserved learning. In *IJCAI*, pages 3933–3939, 2018.

[72] X. Zhang, F. Li, Z. Zhang, Q. Li, C. Wang, and J. Wu. Enabling execution assurance of federated learning at untrusted participants. In *IEEE INFOCOM 2020-IEEE Conference on Computer Communications*, pages 1877–1886, 2020.

[73] X. Zhang, J. Zhao, and Y. LeCun. Character-level convolutional networks for text classification. *Advances in neural information processing systems*, 28, 2015.

[74] Y. Zhang, L. Chen, S. Yang, M. Yuan, H. juan Yi, J. Zhang, J. Wang, J. Dong, Y. Xu, Y. Song, Y. Li, D. Zhang, W. Lin, L. Qu, and B. Zheng. Picasso: Unleashing the potential of gpu-centric training for wide-and-deep recommender systems. *2022 IEEE 38th International Conference on Data Engineering (ICDE)*, pages 3453–3466, 2022.

[75] Y. Zhang and H. Zhu. Additively homomorphical encryption based deep neural network for asymmetrically collaborative machine learning. *arXiv preprint arXiv:2007.06849*, 2020.

[76] Y. Zhao, M. Li, L. Lai, N. Suda, D. Civin, and V. Chandra. Federated learning with non-iid data. *arXiv preprint arXiv:1806.00582*, 2018.

[77] A. Ziller, A. Trask, A. Lopardo, B. Szymkow, B. Wagner, E. Bluemke, J.-M. Nounahon, J. Passerat-Palmbach, K. Prakash, N. Rose, et al. Pysyft: A library for easy federated learning. In *Federated Learning Systems*, pages 111–139. Springer, 2021.