An Efficient Parallel Secure Machine Learning Framework on GPUs

Feng Zhang, Zheng Chen, Chenyang Zhang, Amelie Chi Zhou, Jidong Zhai, and Xiaoyong Du

Abstract—Machine learning is widely used in our daily lives. Large amounts of data have been continuously produced and transmitted to the cloud for model training and data processing, which raises a problem: how to preserve the security of the data. Recently, a secure machine learning system named SecureML has been proposed to solve this issue using two-party computation. However, due to the excessive computation expenses of two-party computation, the secure machine learning is about 2× slower than the original machine learning methods. Previous work on secure machine learning mostly focused on novel protocols or improving accuracy, while the performance metric has been ignored. In this article, we propose a GPU-based framework ParSecureML to improve the performance of secure machine learning algorithms based on two-party computation. The main challenges of developing ParSecureML lie in the complex computation patterns, frequent intra-node data transmission between CPU and GPU, and complicated inter-node data dependence. To handle these challenges, we propose a series of novel solutions, including profiling-guided adaptive GPU utilization, fine-grained double pipeline for intra-node CPU-GPU cooperation, and compressed transmission for inter-node communication. Moreover, we integrate architecture specific optimizations, such as Tensor Cores, into ParSecureML. As far as we know, this is the first GPU-based secure machine learning framework. Compared to the state-of-the-art framework, ParSecureML achieves an average of 33.8× speedup. ParSecureML can also be applied to inferences, which achieves 31.7× speedup on average.

Index Terms—Two-party computation, GPU acceleration, secure training, secure inference, machine learning

1 Introduction

Secure machine learning, or security in machine learning, is increasingly important with the development of information technology, since current large-scale machine learning tasks are usually conducted on HPC servers, which could be provided by a third party. Multiparty computation, as an important means of protecting data security, has been widely used in various application scenarios [1], such as databases [2], [3], auctions [4], [5], and corporate secret protection [6]. Currently, multiparty computation has been widely applied to machine learning [7], [8], [9], [10], [11], [12], [13]. Different from differential privacy [14], which inserts disturbing data in the original data, multiparty computation divides the original dataset into several encrypted copies with partial data, and then distributes these encrypted copies to different servers. By processing encrypted data on multiple servers, even if these servers are untrusted, multiparty computation provides high-level data security protection.

To secure machine learning algorithms, performance is an important metric as it decides the applicability of those algorithms. Two-party computation, as a special case in multiparty computation, has relatively low complexity. However, we find that the low performance problem exists even with two-party computation, which is mainly due to the following reasons. First, two-party computation still greatly increases the amount of computations compared to traditional security-ignorant machine learning algorithms. Second, in two-party computation, two servers need to communicate with each other, which leads to communication overhead. Third, dependencies between different steps in two-party computation still exit, which limits the parallel acceleration.

Many two-party computation based secure machine learning applications have been developed in recent years, such as linear and logistic regressions [7], neural networks [8], [9], [10], and k-nearest neighbor classification [11], [12]. Among these works, SecureML [10], proposed by Mohassel and others, is the state-of-the-art machine learning framework based on two-party computation. SecureML includes a set of novel protocols and models, and it can be used in various machine learning applications. However, SecureML is about 2× slower than the original machine learning methods. Recently, GPUs have been widely used as a powerful accelerator to machine learning algorithms [15], [16]. However, none of existing studies has focused on the acceleration of secure machine learning algorithms using GPUs.

Building a GPU-based secure machine learning framework requires handling three challenges. The first challenge is the complex triplet multiplication based computation

- *Feng Zhang, Zheng Chen, Chenyang Zhang, and Xiaoyong Du are with the Key Laboratory of Data Engineering and Knowledge Engineering (MOE), and the School of Information, Renmin University of China, Beijing 100872, China. E-mail: {fengzhang, chenzheng123, chenyangzhang, duyong}@ruc.edu.cn.*
- *Amelie Chi Zhou is with the Guangdong Province Engineering Center of China-Made High Performance Data Computing System, Shenzhen University, Shenzhen 518061, China. E-mail: chi.zhou@szu.edu.cn.*
- *Jidong Zhai is with the Department of Computer Science and Technology, Tsinghua University, Beijing 100084, China. E-mail: zhaijidong@tsinghua.edu.cn.*

Manuscript received 31 Aug. 2020; revised 23 Jan. 2021; accepted 5 Feb. 2021. Date of publication 12 Feb. 2021; date of current version 25 Mar. 2021.
(Corresponding authors: Jidong Zhai and Xiaoyong Du)
Recommended for acceptance by R. Ge.
Digital Object Identifier no. 10.1109/TPDS.2021.3059108

patterns, which needs to identify the part of computations that are suitable for GPU accelerations from the entire algorithms. The second challenge is how to handle the PCIe transmission overhead caused by frequent intra-node data transmission between CPU and GPU. The third challenge is the complicated inter-node data dependence. The architecture of GPUs determines that they are good at accelerating parallel computations, not for accelerating communications.

We propose *ParSecureML*, an efficient **Par**allel **Secure** **M**achine **L**earning framework on GPUs for processing machine learning tasks with security. ParSecureML includes three novel technologies. First, we develop a profiling-guided adaptive GPU engine. With profiling the two-party computation process, we identify the most compute-intensive part, and accelerate this part by applying GPUs adaptively (Section 4.2). Second, we propose a double pipeline design for intra-node CPU-GPU fine-grained cooperation, which can overlap not only the GPU computation and PCIe data transmission, but also potential steps among different neural network layers (Section 4.3). Third, to accelerate the data communication between two servers, we develop a novel compression-based transmission method (Section 4.4). Our preliminary work has been presented in [17], which only includes a simple design without deep optimizations and analysis. Compared to [17], we provide new observations, implementation, and insights through extra experiments. Moreover, we also add new benchmarks and datasets.

We provide a series of deep optimizations on both CPUs and GPUs in this work (Section 5). First, ParSecureML involves a large amount of random number generation on CPUs; to parallel such processes, we utilize a thread-safe random number generation design. Second, we port the compute-intensive parts of the programs on GPUs, but leave partial matrix addition and subtraction operations on CPUs; we parallel these operations with cache optimizations. Third, we involve architecture specific optimizations, such as Tensor Cores [18], into GPUs. Finally, we integrate these modules as a framework that can be applied to different scenarios.

We evaluate ParSecureML with six typical machine learning algorithms, including convolutional neural network (CNN) [19], multilayer perceptron (MLP) [20], linear regression [21], logistic regression [22], recurrent neural network (RNN) [23], and Support Vector Machine (SVM) [24], and five datasets, including MNIST [25], VGGFace2 [26], NIST [27], CIFAR-10 [28], and a synthetic dataset. Compared to the state-of-the-art secure machine learning framework SecureML, ParSecureML achieves an average of 33.8× speedup. SecureML involves online phase for servers and offline phase for clients, and the online phase occupies the majority of the time; for the online phase, ParSecureML achieves an average of 64.5× speedup. ParSecureML can also be applied to inferences, which achieves 31.7× speedup on average.

As far as we know, this is the first GPU acceleration work for two-party computation. We summarize our contributions of this work as follows.

- We exhibit our observations and insights in the two-party computation, and point out that the computation and communication overhead can be reduced by using GPUs and efficient data structures.
- We develop ParSecureML, the first parallel secure machine learning framework on GPUs, which involves profiling-guided adaptive GPU acceleration, double pipeline optimization, and compression-based communication.
- We involve CPU parallelism and GPU architecture specific optimization of Tensor Cores into ParSecureML, which further improves the efficiency of our system.
- We evaluate ParSecureML with six typical machine learning tasks and five datasets, and demonstrate its benefits over the state-of-the-art secure machine learning framework.

The remainder of the paper is organized as follows. Section 2 introduces the background of secure machine learning, operations in two-party computation, and GPU acceleration. Section 3 presents the motivation of this work, followed by the design of ParSecureML in Section 4. We present the optimization in Section 5 and implementation in Section 6. We show our evaluation in Section 7 and the related work in Section 8. Finally, we show the conclusion in Section 9.

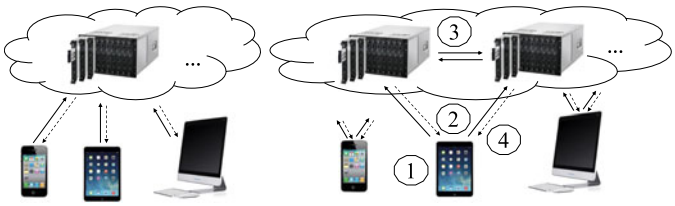
2 Background

2.1 Secure Machine Learning

Many machine learning applications, such as the recently popular deep learning algorithms, require powerful computers to quickly train and update machine learning models [15], [16], [29]. Public clouds provide a good number of powerful servers with different offerings for users to rent [30]. Thus, many machine learning applications have been deployed onto the clouds. As clouds are usually shared among different users, security is an important concern. In this paper, secure machine learning refers to preventing user information from being leaked by protecting the process of machine learning. We mainly focus on using the two-party computation technology [4], [10] to achieve the purpose of secure machine learning. In two-party computation, users partition the input data into two encrypted parts and upload these encrypted parts to separate servers in the cloud. After the servers finish the computation, they transmit their intermediate results back to the client, and the client recovers the final result via two-party computation. Note that during this procedure, the servers need to communicate with each other, which largely increases data processing overhead [31].

The illustration of typical and secure machine learning process is shown in Fig. 1. Fig. 1a shows the typical machine learning process without security protection. The client devices send data to a server, and the server performs computation after receiving the data. Then, when the computation is finished, the server transmits the processed data back to the client. In this process, when the server is insecure, the uploaded data can be leaked and tampered, which is insecure.

A secure machine learning process with two-party computation is shown in Fig. 1b. First, the client device partitions the input data into two encrypted parts: these two parts have the same data size, but with different content. Because this process is conducted on users' devices, we assume it is secure. Second, the client device uploads these



(a) Typical machine learning process.

(b) Machine learning process with two-party computation.

Fig. 1. Secure machine learning.

two parts to two different servers. This process is also secure even if only part of the data is intercepted. Third, after receiving the data, the two servers perform computation separately with limited communication, which is more secure than the typical process in Fig. 1a. Fourth, after the computation is finished, the servers transmit the processed results back to the client device. Note that in this process, the data are not leaked when one server is unsafe, and the data tampering can be detected if happened. The state-of-the-art secure machine learning framework, SecureML [10], also adopts this strategy.

2.2 Operations in Two-Party Computation

We introduce the operations in two-party computation in this part. Typical machine learning models can be protected via protecting *triplet multiplication*, which refers to the multiplication and addition of two matrices to obtain another matrix. This operation is involved in almost all machine learning methods, such as convolutional neural network (CNN), recurrent neural network (RNN), and multilayer perceptron (MLP). Specifically, to protect the triplet multiplication in Equation (1), the client first divides the triplet multiplication into two processes. We randomly divide matrix A into matrices A_0 and A_1, and B into B_0 and B_1. Besides, we randomly generate two matrices, U that has the same dimension with A, and V that has the same dimension with B. The matrix Z is the multiplication of U and V, as shown in Equation (2). U, V, and Z are also be divided into matrices of U_0 and U_1, V_0 and V_1, and Z_0 and Z_1 as shown in Equation (3). The submatrices of A_i, B_i, U_i, V_i, Z_i are uploaded to the related server i, where i belongs to {0,1}. Second, server i computes E_i and F_i, as shown in Equation (4), and both servers compute E and F via communication, as shown in Equation (5). Next, both servers compute their related C_i, as shown in Equation (6), and transmit C_i back to the client. Third, the client obtains C by adding C_0 and C_1

$$C = A \times B \tag{1}$$

$$Z = U \times V \tag{2}$$

$$U = U_0 + U_1, V = V_0 + V_1, Z = Z_0 + Z_1 \tag{3}$$

$$E_i = A_i - U_i, F_i = B_i - V_i \tag{4}$$

$$E = E_0 + E_1, F = F_0 + F_1 \tag{5}$$

$$C_i = (-i) \times E \times F + A_i \times F + E \times B_i + Z_i. \tag{6}$$

2.3 GPU Acceleration

GPUs have been considered as one of the important foundations for the resurgence of machine learning, because the parallel architecture of GPUs is very suitable for dense matrix operations in machine learning tasks. With the increase of the scale of deep neural networks, the network generation and training processes often require tens of thousands of GPU computing hours [32]. The HPC servers equipped with GPUs should provide support for machine learning tasks, and expand the machine learning application boundaries, including secure machine learning technology of two-party computation. Currently, although new memory technologies are constantly being proposed [33], [34], typical GPUs still have their independent memories; hence, machine learning workloads need to be uploaded to GPUs for computation. Many machine learning frameworks, such as TensorFlow [35], PyTorch [36], and Caffe [37], support GPU acceleration. However, there is no GPU-based research of two-party computation, and our work can fill this gap.

GPUs are suitable for accelerating two-party computation. First, GPUs contain a large number of lightweight cores, which is suitable for matrix operations. Second, from Section 3.1, we can see that the most time-consuming operations are the matrix-related computations. These operations usually contain a large amount of regular dense computation, which shows the huge performance potential of applying GPUs. Third, the GPU memory usually provides much higher bandwidth.

3 Motivation

In this section, we analyze the performance degradation caused by security preserving technologies, which is also the motivation of using GPUs for acceleration. Then, we analyze the challenges of using GPUs to accelerate two-party computation.

3.1 Performance Degradation

> *Observation:* Compared to the original machine learning without security, current two-party computation seriously drags down the overall performance, which exhibits great potentials and necessities for GPU acceleration.

In this part, we analyze the performance degradation caused by security-related operations. We show the performance of the original version and the SecureML version of convolution neural network (CNN), multilayer perceptron (MLP), linear regression, and logistic regression in Table 1. We use the MNIST [25] dataset for validation (detailed in Section 7.1). On average, SecureML generates a 2× performance degradation compared to the original implementation. The reason is that as discussed in Section 2.1, the two-party computation involves extra computation and communication overheads: 1) The client needs to partition the input matrix into two parts, which decreases 4.4 percent performance. 2) The servers need extra computation for E_i, F_i, E, F, and C_i, which decreases the performance by a factor of 1.5. 3) During computation, the two servers need

TABLE 1
Performance Comparison Between the Original Implementation and the SecureML Implementation

Method	Original (s)	SecureML (s)	Slowdown (x)
CNN	30.02	74.77	2.49
MLP	63.74	114.85	1.80
Linear regression	32.66	62.94	1.93
Logistic regression	32.1	63.29	1.97

communication to calculate E and F, which decreases 1.43 percent performance.

We use Fig. 2 for detailed illustration. Fig. 2 shows the detailed process of MLP with the MNIST dataset. We divide the process into offline and online according to [10]. In the offline phase, we put the MNIST dataset, which has 60,000 samples in 28×28 size, into one batch. The client generates two encrypted data, which takes 62.68s, and then, transmits them to two different servers, which takes 0.21s. In the online phase, after the server receives the input data, it first computes partial results ("compute1"), such as the E_i and F_i as discussed in Section 2.1, which takes about 0.19s. Second, *server1* and *server2* communicate E_i and F_i with each other ("communicate"), which takes about 0.24s. Third, the server computes the C_i ("compute2"), which takes about 95.52s. Then, C_i shall be transmitted back to the client to merge to obtain the final result.

3.2 Solution to Performance Degradation From GPU Perspective

Insight. A GPU-based two-party computation that considers both the GPU characteristics and features of two-party computation shall have better performance acceleration effects.

This work introduces a GPU-based two-party computation framework for secure machine learning. GPUs are suitable for accelerating two-party computation. First, GPUs contain a large number of lightweight cores, which is suitable for matrix operations. Second, from Section 3.1, we can see that the most time-consuming operations are the matrix-related computations. These operations usually contain a large amount of regular dense computation, which shows the huge performance potential of applying GPUs. Third, the GPU memory usually provides much higher bandwidth. However, the GPU-based solution needs to consider the application characteristics, and reduce the inter-node and intra-node communication.

3.3 Challenges

Enabling two-party computation on GPUs requires to handle three challenges, as summarized below.

Challenge 1: Complex triplet multiplication based computation patterns. The two-party computation involves offline and online phases, and in each phase of real machine learning tasks, there are complex dependant steps. First, we need to allocate the steps where GPUs can be applied efficiently, and this requires us to well understand the two-party computation. Second, if the execution time of the step is small, the acceleration effects may not be large. Third, such GPU-based implementations require to distribute the original workloads to GPU threads in a fine-grained manner.

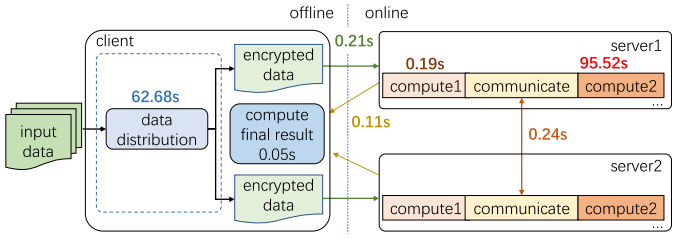


Fig. 2. Time breakdown for two-party computation. We put the MNIST dataset in one batch for illustration.

Challenge 2: Frequent intra-node data transmission between CPU and GPU. Distributing some steps in two-party computation to GPU while remaining the other steps to CPU could incur frequent GPU-CPU communication. Unfortunately, the GPU has its isolated memory, so using GPUs requires copying data from the host memory to the GPU memory via PCIe, which complicates the GPU application situation. In detail, if the PCIe data transmission overhead is larger than the GPU acceleration benefits, we cannot obtain overall performance benefits.

Challenge 3: Complicated inter-node data dependence. Compared to the original machine learning tasks without security, two-party computation involves complicated data dependency between two nodes, which causes inter-node communication overhead. Even worse, the location of the data to be transmitted affects the transmission efficiency. For example, if the needed data are stored in the GPU memory, the data need to be copied back to the host memory before transmission. Besides, the transmission step may interfere with the GPU computing fluency if it occurs during the computation. These factors make the GPU-based secure machine learning complicated in a distributed environment.

4 ParSecureML

In this section, we show our framework, ParSecureML, for secure GPU-based machine learning tasks. We first give an overview of our ParSecureML framework and then introduce the details of the framework from three aspects, including how GPUs are efficiently utilized to speedup computations, as well as the intra-node and inter-node designs which reduce the communication overhead.

4.1 ParSecureML Overview

ParSecureML consists of three major components: 1) profiling-guided adaptive GPU utilization, 2) double pipeline execution for overlapping intra-node data transmission and computation, and 3) compressed transmission for inter-node communication. These three components are integrated into the two-party computation process. Next, we show our basic idea in ParSecureML, including the solutions to the challenges mentioned in Section 3.3 and ParSecureML workflow.

Solutions to Challenges. ParSecureML can address all the challenges discussed in Section 3.3. For the first challenge, we can observe that the data distribution step in the offline phase and the last computation step in the online phase

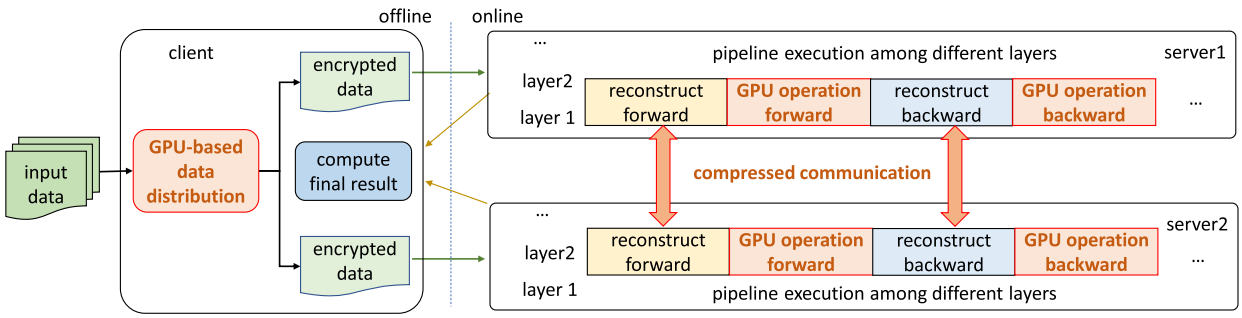


Fig. 3. ParSecureML overview.

take the most time. If possible, GPUs can be applied in these steps. Consequently, we develop the profiling-guided adaptive GPU offline computation module and GPU online computation module. For the second challenge of intra-node frequent data transmission in the online phase, as shown in Fig. 2, the *compute1* and *communicate* steps of two-party computation are short. Because these two steps take only a small proportion of the total execution time, the overall performance improvement with GPU acceleration for these steps is marginal; even worse, the extra PCIe data transmission cost and GPU warm-up overhead could degrade the overall performance. Hence, these two steps are not suitable for GPU acceleration, and we merge *compute1* and *communicate* steps as *reconstruct* phase conducted by CPUs, while leave the "compute2" step to the GPU denoted as *GPU operation*. A pipeline has been designed to overlap the computation and PCIe communication. Besides, since machine learning tasks usually involve several steps in each layer, some steps can also be overlapped in a pipeline across layers. Consequently, a double pipeline design has been proposed in machine learning tasks. For the third challenge of inter-node communication, we compress data before transmission, which increases the data transmission density and thus improves the transmission efficiency. Additionally, we adopt optimizations on CPUs and GPUs to further improve ParSecureML performance.

Difficulties in Integrating These Technologies. First, when porting compute-intensive parts to GPUs, we need to carefully design the cooperations with pipeline execution and compressed transmission, detailed in Section 4.2. Second, the double pipeline design involves CPU-GPU transmission, computation, and compressed transmission, which is more complicated than typical buffering for hiding latency, detailed in Section 4.3. Third, for compressed transmission, the data need to be transmitted could be stored in GPU memory, which relates to CPU-GPU communication and double pipelines, detailed in Section 4.4.

Workflow. Fig. 3 shows the overview of ParSecureML. First, for the input data, the client uses the GPU to generate the encrypted data, and transmits the encrypted data to *server1* and *server2*. Second, after the servers receive the data, the servers conduct operations to calculate results. The *reconstruct* phase involves compression in the communication between servers. The *GPU operation* phase involves the major computation, and it occupies most of the time. Note that current machine learning tasks usually contain several layers, and in each layer, there are forward propagation and backward propagation. Both forward and backward propagations have *reconstruct* and *GPU operation* phases. Pipeline execution among different layers has been developed to hide the communication overhead in *reconstruct* phases. Third, after the servers finish their processes, the servers transmit the results back to the client.

The rest of this section is organized as follows. Section 4.2 shows the profiling-guided adaptive GPU utilization. Section 4.3 shows the double pipeline design for intra-node data transmission. Section 4.4 shows the compression-based inter-node communication. These three components make up the ParSecureML. Further optimizations are shown in Section 5.

4.2 Profiling-Guided Adaptive GPU Utilization

In this part, we first analyze the two-party computation process. Then, we present details of our efforts on profiling-guided adaptive GPU utilization to accelerate two-party computation, including offline acceleration and online acceleration.

Analysis. Because two-party computation includes online and offline phases, GPU acceleration is also divided into online and offline parts. We first need to profile the two-party computation process to identify the most time consuming parts, and then accelerate them in an adaptive manner.

Offline Acceleration Design. The goal of the offline phase is to prepare the encrypted data for $server0$ and $server1$. For the triplet multiplication of A and B, we show the partitioning process in Fig. 4. Although this phase includes several steps, such as dividing A into A_0 and A_1, the most time-consuming step is the step of multiplying U by V to obtain Z, which accounts for more than 90 percent of the total time of the offline phase. This step is composed of compute-intensive matrix multiplication, which can be accelerated by GPUs. For the rest operations, we do not use the GPUs, because transferring extra data to GPU memory also takes time. In our experiments, distributing the rest operations on GPUs could cause extra 4.5 percent performance degradation. In the current state, we only need to copy the U and V to the GPU memory from the host memory, and copy the calculated Z back to the host memory.

Online Acceleration Design. The online phase is used to process the uploaded data, which relates only to Equations (5) and (6). Equation (5) is lightweight, and Equation (6) causes the major time overhead. Therefore, we leave Equation (5) to the CPU, and use the GPU to accelerate Equation (6). To calculate Equation (6) with GPU, we need to transmit the matrices E, F, A_i, B_i, and Z_i to the GPU. Equation (6) can be expressed in Equation (7). Moreover, Equation (7) can be further expressed in Equation (8), which

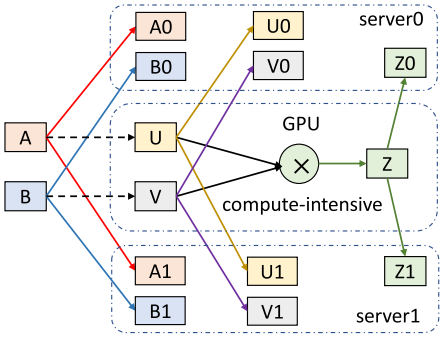


Fig. 4. Data processing in offline phase.

uses one addition operation to replace a multiplication operation, and thus reduces overhead

$$C_i = (\,(-i) \times E \quad A_i \quad E\,) \times \begin{pmatrix} F \\ F \\ B_i \end{pmatrix} + Z_i, i \in 0, 1 \tag{7}$$

$$C_i = (\,(-i) \times E + A_i \quad E\,) \times \begin{pmatrix} F \\ B_i \end{pmatrix} + Z_i, i \in 0, 1. \tag{8}$$

Activation Function Design. In the online phases, many machine learning algorithms, such as CNN and MLP, involve an activation function, such as *softmax* [38], which cannot be handled directly due to the nonlinear characteristics. One possible solution is to use Taylor Formula [39] to fit the nonlinear functions with linear functions, but the expansion has high complexities. We use the following Equation (9) to simulate the original nonlinear functions in GPUs. Note that we use Equation (9) by default but not ReLU [40], because ReLU does not have an upper limit which cannot be used in machine learning tasks such as Logistic regression. For common tasks such as CNN, the activation functions such as ReLU can be applied. Experiments show that such a replacement has little impact on accuracy. Similar methods have been used in other works such as [10]

$$f(x) = \begin{cases} 0, x < -\frac{1}{2} \\ x + \frac{1}{2}, -\frac{1}{2} < x < \frac{1}{2} \\ 1, x > \frac{1}{2} \end{cases}. \tag{9}$$

4.3 Double Pipeline for Intra-Node CPU-GPU Fine-Grained Cooperation

We first analyze common machine learning tasks and then present our double pipeline design in this part.

Analysis. Pipeline is used to overlap the computation and communication, which hides the PCIe data transmission overhead. However, pipeline in two-party computation for machine learning becomes complicated. After analyzing a series of common machine learning tasks [41], [42], we find that typical machine learning tasks, especially deep learning tasks, have two features: 1) there are usually several dependent neural network layers, and 2) the machine learning tasks involve forward and backward propagations. The number of layers is specified by the user based on experience, and is usually greater than two to capture the non-linear relationships. In the task, a forward propagation is used to process the data, and a backward propagation solves the parameter updating problem in neural networks. Even worse, data transmission happens in each layer, which complicates the computing procedure. This means that a fine-grained pipeline needs to be designed for efficient GPU utilization in both forward and backward propagation of each layer, instead of directly using a coarse-grained pipeline [43], [44]. Furthermore, since too many steps exist across all layers, a second pipeline can be constructed to overlap the possible steps in different layers to leverage available computing resources. Hence, double pipelines should be constructed to enhance fine-grained CPU-GPU cooperation.

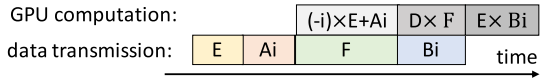


Fig. 5. Pipeline to overlap PCIe data transmission and GPU computation. D represents the result of "$(-i) \times E + A_i$".

Pipeline Design. Based on the analysis, we design a double pipeline processing. The first pipeline is to overlap GPU computation and PCIe data transmission in Equation (8). We show the major process for Equation (8) in Fig. 5. We can transmit E and A_i first. Then, the data transmission for F can be overlapped with "$(-i) \times E + A_i$" whose result is denoted as D, and the transmission for B_i can be overlapped with the computation of "$D \times F$". This pipeline can be applied to the GPU operations of forward and backward propagations in all layers.

For the second pipeline to overlap possible steps among different layers, we find that in applying the two-party computation, both the forward and backward propagations consist of a *reconstruct* step and a GPU operation step. The processing of the successive layers depends on the forward propagation of the current layer, so we cannot overlap the *reconstruct* step of forward propagation in the next layer with the GPU operation step of forward propagation in the current layer. However, the *reconstruct* step in the backward propagation does not need to wait for the successive layers, and hence can be conducted in pipeline with the propagation in the next layer. We use Fig. 6 for illustration. In each layer, there are four steps: 1) forward *reconstruct*, 2) forward GPU operation, 3) backward *reconstruct*, and 4) backward GPU operation. Fig. 6a shows the original execution without pipeline. We find that for backward *reconstruct*, partial computation depends on the results from the forward GPU operation in the same layer, and we thereby remain these computations in the backward *reconstruct*; the other computation depends on the results from the successive layer, and we thus distribute the computation to the backward GPU operation, as shown in Fig. 6b. Therefore, the *reconstruct* can be pipelined, and we can save a *reconstruct* step in each layer. Furthermore, in the execution of many epochs, similar optimization can also be applied to forward *reconstruct*.

Detailed Design. We show the detailed design of double pipelines for common machine learning tasks in this part. The first step is forward *reconstruct*, and its input are A_i, B_i, U_i, V_i, and Z_i. For A_i, it usually represents the input, and in the first layer, the A_i is from the client; in the successive layers, the A_i is from the previous layer. For B_i, it usually represents the model; the client usually input the B_i for each layer, and so does to U_i, V_i, and Z_i. The *reconstruct* step communicates with the other server to calculate E and F for the forward operation step. The second step is forward

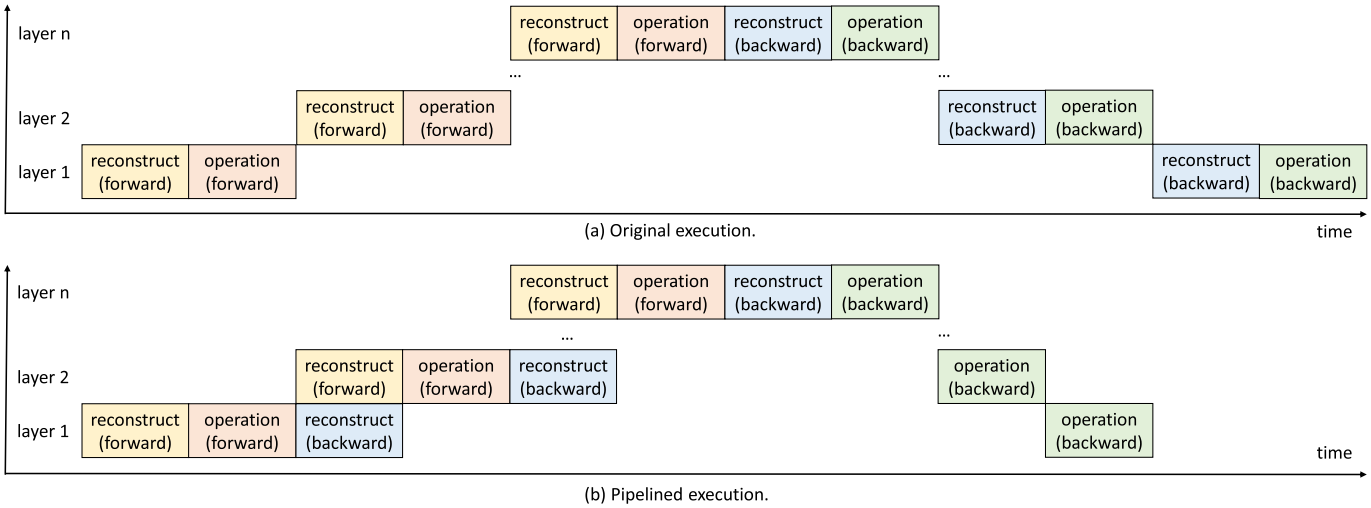


Fig. 6. Pipeline execution in ParSecureML. The operation (forward) and operation (backward) are conducted on GPUs.

GPU operation. Its input is from the forward *reconstruct* step, and its output C_i is used for the input A_i of the forward *reconstruct* step in the successive layer. The third step is backward *reconstruct*. We only remain the computation for F_i in this step where B_i is usually the C_i in the forward operation, and we merge the computation for E_i to the operation step due to that its A_i depends on the C_i of the next layer. Note that for the layer n, its A_i is input from the client; for the U_i, V_i, and Z_i, they are also input from the client. The fourth step is GPU operation for C_i. After the first layer finishes the computation for C_i, an epoch in the machine learning task finishes execution. Note that usually B_i represents the model, which shall be transmitted back to the client, and the pipeline to overlap PCIe data transmission and GPU computation is conducted in the second step of forward GPU operation and the fourth step of backward GPU operation. Unified memory could be a potential optimization. However, current studies show that it causes performance degradation [45], [46], so we do not involve this technique.

4.4 Compressed Transmission for Inter-Node Communication

In this part, we show our analysis and design of compressed transmission for the inter-node communication.

Analysis. To compute E and F, servers need to communicate E_i and F_i with each other. E_i and F_i depend on A_i and B_i. In machine learning tasks, A and B can be model parameters, and we find that their iterative matrices are usually sparse. The reasons are as follows. First, for many activation functions, such as ReLU function [40], the results after activation contain a lot of zeros. Second, when the number of layers is large, the gradient of the loss function for the initial few layers is small. Third, when the training process is about to complete, the gradient is usually small. Besides, the input may be originally sparse.

Compression-Based Communication Design. We use $A_{i,j}$ and $B_{i,j}$ to represent the A and B at server i for the j epoch. Then, $A_{i,j+1}$ and $B_{i,j+1}$ can be represented in Equation (10), where $\Delta_{ij}^{A/B}$ represents the difference between two epochs. In practice, $\Delta_{ij}^{A/B}$ often represents the gradient

$$A_{i,j+1} = A_{i,j} + \Delta_{ij}^{A}, B_{i,j+1} = B_{i,j} + \Delta_{ij}^{B}. \tag{10}$$

After representing the $A_{i,j}$ and $B_{i,j}$, we can represent the $E_{i,j}$ and $F_{i,j}$ in Equations (11) and (12). Consequently, if we can judge in advance that $\Delta_{ij}^{A/B}$ is sparse, then we only need to transmit the difference $\Delta_{ij}^{A/B}$ between two servers for E and F

$$E_{i,j+1} = A_{i,j+1} - U_i = A_{ij} + \Delta_{ij}^{A} - U_i = Eij + \Delta_{ij}^{A} \tag{11}$$

$$F_{i,j+1} = B_{i,j+1} - V_i = B_{ij} + \Delta_{ij}^{B} - V_i = Fij + \Delta_{ij}^{B}. \tag{12}$$

Detailed Design. In the detailed design, before we transmit E_i and F_i, we check the Δ_{ij}^{A} and Δ_{ij}^{B}. If Δ_{ij}^{A} or Δ_{ij}^{B} is sparse (75 percent elements in the matrix are zero in our default settings), we use the compressed sparse row format (CSR) [47] to store it, and transmit the compressed Δ_{ij}^{A} or Δ_{ij}^{B} instead of the original data. Otherwise, we transmit the original version.

5 Optimization

In this section, we show our optimizations for futher improving the performance of ParSecureML.

5.1 CPU Optimization

In this part, we show our CPU optimizations for ParSecureML.

Analysis. As discussed in Section 4.2, we have ported the compute-intensive parts to GPUs while remainng the other parts to CPUs. The major computing tasks on CPU include 1) generation for random matrices A_0, B_0, U_0, V_0, U_1, and V_1, introduced in Section 2.2, and 2) computation for matrices A_1, B_1, U, and V through additions and subtractions. We optimize these operations in parallel to fully utilize the CPU computing resources.

Parallelism for Random Number Generation. ParSecureML involves random number generations. To generate random numbers concurrently in multiple threads, a thread-safe random number generator is required. The simple generator *rand()* in C standard library accesses and modifies internal state objects, which may cause data races and generate incorrect random numbers in multi-thread programs.

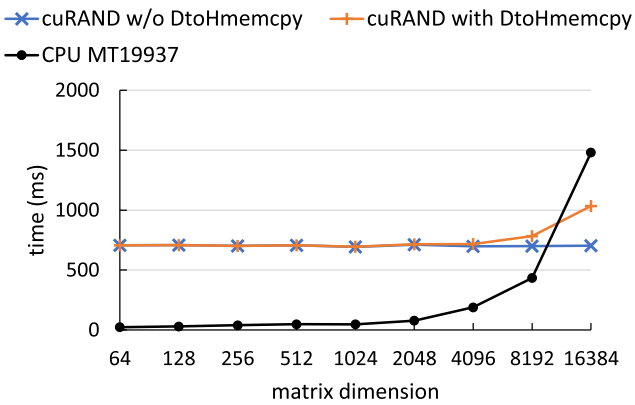


Fig. 7. The performance of cuRAND on the GPU and MT19937 on the CPU. Matrix dimension n represents that the matrix size of $n \times n$.

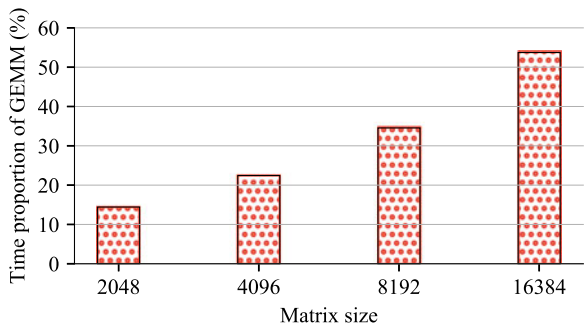


Fig. 8. Time proportion of GEMM.

A simple solution is to add thread lock in function *rand()*; however, it results in longer execution time of multi-thread program than that of single-thread program due to the overhead of thread lock. To solve this question, we apply a thread-safe random number generator, Mersenne Twister 19937 generator (MT19937) [48], from C++ 11 random library. MT19937 is a pseudo-random generator (PRNG) based on Mersenne Twister algorithm with Mersenne prime $2^{19937} - 1$, generating 32-bit numbers. The Mersenne Twister random generator is wildly used in software systems, including Microsoft Excel, MATLAB, Python, PHP, R, Ruby, Stata, and so on. Compared to function *rand()*, MT19937 needs 1.06× time to generate random numbers [49]. To fully realize CPU multi-thread parallelism, each thread has its own MT19937 generator, whose seed is the sum of the current time and the hash of the thread identifier. To avoid the overhead of repeated generation and deletion of MT19937 generators, they are declared as *static thread_local* variables and are released at the end of the program. Therefore, each thread only needs to generate and delete the MT19937 generator once in the entire program execution. Another potential solution is to use cuRAND function on GPUs. However, it brings performance benefits only when processing large matrices, as shown in Fig. 7, so we do not use this design.

Optimization for Matrix Additions and Subtractions. The matrix additions and subtractions mainly come from Equations (3) and (5), which are achieved by traversing two matrices through a for-loop and can be directly optimized into a multi-threaded for-loop in parallel. When ParSecureML writes the calculation result into the sum or difference matrix, the overhead caused by multiple threads writing the same cache line needs to be avoided. Each cache line stores 16 FP32, and the cache line writing races can be avoided by scheduling at least 16 cyclic tasks to each thread. In addition, to reduce the overhead of opening more than one parallel region, multiple parallel regions should be merged.

5.2 GPU Optimization

We in this part show our GPU optimizations in the system.

Analysis. To further improve the GPU performance, we first analyze the GPU utilization processes. We use the profiling tool nvprof [50], provided by NVIDIA, to collect and view GPU activities, including kernel execution, data transmission, and CUDA API calls. We collect GPU profiling data from both clients and servers, and observe that there are three major activities in ParSecureML: CUDA memory copy from host to device, general matrix multiply (GEMM) operations, and CUDA memory copy from device to host. Since the overhead of CUDA memory copy between host and devices is decided by the memory transmission channel, we mainly optimize the GEMM operations. We show the proportion of time consumed by GEMM operations with different matrix dimensions in Fig. 8. As the matrix size increases, the time proportion of GEMM becomes larger. For example, when the matrix dimension is 16,384, the GEMM computation time accounts for more than half of the total time. Hence, the GEMM optimization is the focus of GPU optimization. Theoretically, the effect of this optimization becomes more significant as the size of the matrix in GEMM increases.

Tensor Core Utilization. To optimize the GEMM operations of ParSecureML, we use NVIDIA Tensor Cores [18] to achieve higher throughput of GPUs without sacrificing accuracy. Tensor Cores are the defining features of NVIDIA Volta GPU Architecture and are wildly used in high-performance computing and deep learning. Popular GPU machine learning frameworks, including TensorFlow [35], PyTorch [36], MXNet [51], and Caffe2 [52], all utilize Tensor Cores. In detail, Tensor Cores are programmable matrix-multiply-and-accumulate units, which calculate $D = A \times B + C$, where A and B are FP16 4×4 matrices, while C and D may be FP16 or FP32 4×4 matrices. Fig. 9 shows the Tensor Core operation process. The NVIDIA Tesla V100 GPU contains 640 Tensor Cores that deliver a peak performance of 125 TFLOPS, resulting in a 12× increase in throughput with standard FP32 operations compared to the NVIDIA Pascal P100 GPU. Previous research [53] concludes that Tensor Cores provide a throughput increase of 2.5 to 12× with respect to the GEMM from cuBLAS library on NVIDIA Tesla V100 GPU. Therefore, Tensor Cores are suitable for accelerating ParSecureML framework since there are many matrix multiplication operations in ParSecureML. In our implementation, we first invoke *cublasSetMathMode()* to set math mode as *CUBLAS_TENSOR_OP_MATH*, and then use *cublasSgemmEx()* to do all GEMM operations in ParSecureML framework. In this way, the GEMM operations are all running on Tensor Cores of our platform.

6 Implementation

We implement the ParSecureML framework, in the hope to ease the burden of developers in making their machine learning applications secure. ParSecureML includes a set of typical machine learning tasks, which can be used in many

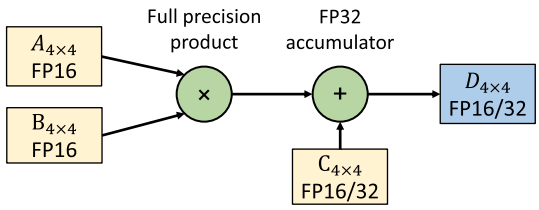


Fig. 9. Tensor core operation.

existing applications. Based on the implementation of ParSecureML, more accelerators, such as FPGAs and TPUs [54], can be involved in the current framework.

ParSecureML is written using C++, MPI, and CUDA. The main framework is written in C++, including the APIs, connections to the CUDA and MPI modules, and data preprocessing operations. We write the multi-node control module in MPI, which is used for the data transmission between client and servers. We use CUDA to implement the GPU triplet multiplication functions. Note that users do not need to know the implementation details of ParSecureML. We have defined a set of APIs. The users only need to understand the program interfaces and use these interfaces to write programs.

7 Evaluation

Focusing on the secure GPU-based machine learning tasks, we evaluate the performance of ParSecureML in a distributed environment, and compare ParSecureML with the state-of-the-art method, SecureML.

7.1 Experimental Setup

In this part, we exhibit our evaluation methods, benchmarks, and datasets used in the evaluation, and our experiment platform.

Evaluated Methods. The baseline method used in our evaluation is SecureML [10]. SecureML is the state-of-the-art machine learning engine that utilizes two-party computation for security. However, SecureML is not open source, so we implement SecureML according to [10].

Benchmarks. We use six typical machine learning tasks to evaluate the overall performance of ParSecureML and SecureML. The batch size is 128 in our experiment, and for the other settings, the performance behaviors are similar. The details of the six machine learning tasks are shown as follows.

- Convolution neural network (CNN). CNN [19] is a deep neural network that is commonly used in matrix-based applications, such as image recognition. As its name indicates, CNN employs convolution operations, which involves a large amount of triplet multiplications. In our evaluation, CNN contains one convolutional layer and two fully connected layers with *relu* activation function. The size of the convolutional layer depends on the input while the size of the convolution kernel is 5-by-5. The number of output neurons in the middle layer is 64, and the size of the output layer is 10.
- Multilayer Perceptron (MLP). MLP [20] is a neural network that maps input vectors to output vectors. In our evaluation, MLP consists of three layers: an input layer with 128 neurons, a middle layer with 64 neurons, and an output layer with 10 neurons. The activation function is *ReLU*. MLP also uses a backward propagation for model correction.
- Recurrent neural network (RNN). RNN [23] is a neural network that focuses on temporal sequence data. Different from traditional neural networks, RNN passes the states in its own network, and thus can capture the characteristics of time series events.
- Linear regression. Linear regression [21] is a machine learning approach to model the relationship between two variables. The linear regression model involves multiplication and the input data are expressed as matrices.
- Logistic regression. Logistic regression [22] is a generalized regression model, which involves a logistic function. Consequently, logistic regression can be used to describe some nonlinear relations.
- Support Vector Machine (SVM). SVM [24] is a supervised learning model for classification, regression, and outlier detection. SVM constructs a hyperplane with the largest distance to the nearest training point in a high-dimensional space by the sequential minimal optimization (SMO) algorithm [55].

Datasets. We use five datasets along with the benchmarks in our evaluation. These datasets have been commonly used in previous research [56], [57], [58]. Note that RNN does not apply to images, so we only use our generated dataset, SYNTHETIC, to evaluate RNN.

- VGGFace2. VGGFace2 is a large scale face detection dataset [26]. VGGFace2 contains a set of face images, and we randomly select 40,000 images. Because the size of each image is different, we process each image to a 200-by-200 matrix.
- NIST. NIST comes from NIST 8-Bit Gray Scale Images of Fingerprint Image Groups(FIGS) [27]. NIST contains a set of fingerprint images, and the size of each image is 512 by 512. We randomly select 4,000 images.
- CIFAR-10. CIFAR-10 is a color image dataset used in computer vision [28]. It consists of ten classes with 6,000 images per class. The ten classes are airplanes, cars, birds, cats, deer, dogs, frogs, horses, ships, and trucks. It has 50,000 training images and 10,000 test images.
- SYNTHETIC. We generate a synthetic dataset used to explore the performance under different workload scales. In the performance evaluation, we use 640,000 matrices for validation, and each matrix size is 32 by 64. In Section 7.5, we analyze the influence of workload size by changing the number of matrices from 128 to 524,288.
- MNIST. MNIST comes from the MNIST database of handwritten digits [25]. MNIST includes 60,000 samples for training and 10,000 samples for testing. Each sample is a pixel matrix and the element in the matrix ranges from zero to 255.

Platform. We conduct our experiments on a three node cluster. Each node is equipped with two Intel(R) Xeon(R) CPU E5-2670 v3 CPUs and 128GB memory. Their operating systems are Ubuntu 16.04.4. Each node is also equipped with an Nvidia Tesla V100 GPU. We use one node to simulate the

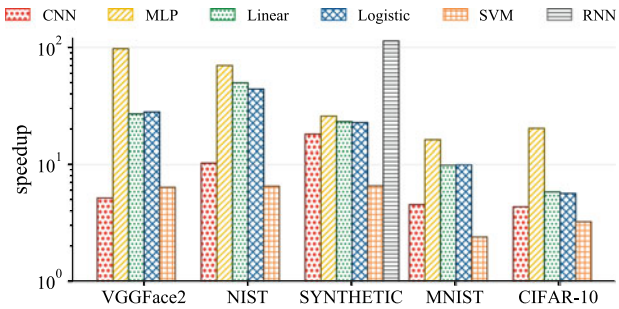


Fig. 10. Overall performance speedups.

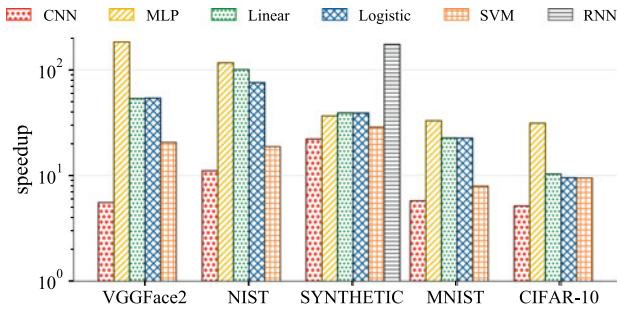


Fig. 11. Online performance speedups.

client and two nodes to simulate servers. The three nodes are connected by 100Gbps 4×EDR Infiniband, so the experimental environment is greatly improved compared to [10].

7.2 Performance Speedup

We use the performance speedup of ParSecureML over SecureML as the metric to quantify the performance benefits. We show the overall performance speedups, along with the offline and online speedups.

Overall Speedups. The overall performance speedups are shown in Fig. 10. On average, ParSecureML achieves an average speedup of 33.8× over the SecureML. In detail, we have the following observations. First, ParSecureML exhibits a significant performance improvement in most cases, which shows that GPUs can successfully accelerate the two-party computation processing. Second, for the small dataset, MNIST, the workload is not large enough to fully release the GPU capacity, but ParSecureML still achieves clear performance speedups. Third, CPUs are more appropriate to process small datasets, while GPUs are more appropriate to process large datasets.

Online Speedups. The online performance speedups are shown in Fig. 11. Online procedure starts after the server receives the data, and ends when the data processing is completed. The average online performance speedup is 64.5× (even higher than the overall speedup), which implies that the overall performance improvement mainly comes from the online part. In our CNN implementation, we use point-to-point multiplication (inner product) for CNN, which has lower computation density compared to other tasks that involve a large amount of matrix-matrix multiplication. Architecture-specific optimizations, such as Tensor Cores on Volta microarchitecture [18], have been involved, as discussed in Section 5.

Offline Speedups. The offline performance speedups are shown in Fig. 12. Applying GPUs in the offline phase brings 1.3 × performance benefits, which is moderate. The reason is that the computation in this part is relatively low. Consequently, the speedups of different benchmarks are similar. However, the offline phase only accounts for a small proportion of the whole execution time (detailed in Section 7.5), and does not affect the overall performance improvement.

Inference Speedups. As secure inference is a domain of immense interest to the privacy preserving machine learning community, we study the speedups of inference process by ParSecureML in this part. Inference process is essentially a sub-process of the training protocol (the forward pass). These benchmarks involve forward and backward propagations.

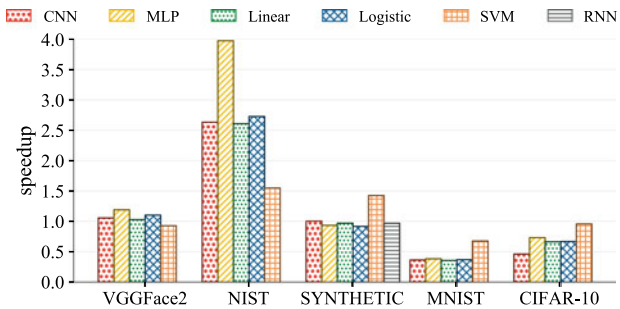


Fig. 12. Offline performance speedups.

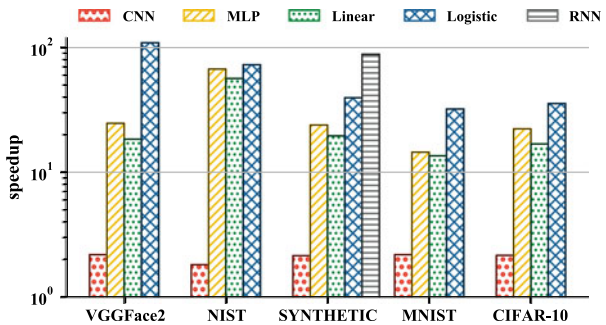


Fig. 13. Inference performance speedups.

The inference results of both linear regression and SVM are calculated by $w^T x + b$, so we only show the result of linear regression. We show the inference speedups of these benchmarks in Fig. 13. The average inference performance speedup is 31.7×, which implies that the inference process of machine learning tasks can benefit a lot from ParSecureML.

7.3 Optimization Benefits

We introduce several optimizations in Section 5. In this part, we analyze these optimizations to verify their effectiveness in ParSecureML. The baseline is the ParSecureML without these optimizations. The overall average benefit is 13.82 percent performance improvement compared to the original ParSecureML without the optimizations in Section 5. Most of these benefits come from CPU optimization.

CPU Parallelism. Our preliminary work [17] does not involve parallelism in the CPU part of ParSecureML. According to Section 5, there are random number generations and matrix operations left on CPUs, so we parallel these operations and verify the effectiveness of our CPU parallelism in this part. The acceleration results are shown in Fig. 14, and we have the following observations. First, the CPU parallelism

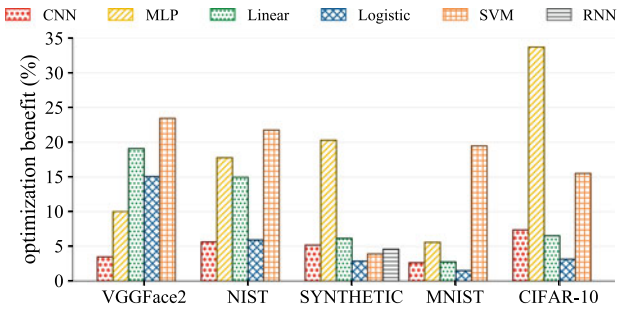


Fig. 14. CPU optimization benefit.

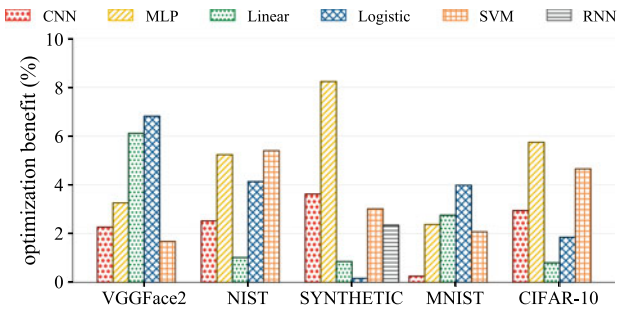


Fig. 15. GPU optimization benefit.

brings clear performance improvements; on average, the CPU parallelism improves the performance by 10.71 percent. Second, the performance on different datasets varies greatly. For example, ParSecureML obtains 17.60 percent improvement on VGGFace2 while 8.69 percent benefits on MNIST. The reason is that the image size of VGGFace2 is 200-by-200, which is much larger than the size of MNIST, 28-by-28. The CPU threads in VGGFace2 tasks can be better scheduled, avoiding cache line writing races. Third, even for the same program, the performance varies. For example, MLP achieves 33.70 percent performance improvement on CIFAR-10, but only 5.58 percent improvement on MNIST.

GPU Tensor Cores. We use the GPU Tensor Cores to improve the performance of ParSecureML. The acceleration results are shown in Fig. 15. First, this optimization achieves a clear performance improvement of 3.11 percent, which is less than that of the CPU optimizations. Second, the GPU benefits in Fig. 15 are different from the CPU benefits in Fig. 14. The reason relates to the difference between CPU tasks and GPU tasks. For example, a program may use the CPU to generate several small-scale random matrices in a short time, but then perform many GEMM operations on the GPU for a long time. In contrast, another program may generate many large-scale random matrices on CPU but conduct few GEMM operations on the GPU. Third, the programs that have many large-scale GEMM operations benefit the most from Tensor Cores. The reason is that the GEMM operation's time proportion of this type of program is higher than that of the other programs; therefore, the speedup brought by Tensor Cores is more significant.

7.4 Comparison With Original Machine Learning Tasks on GPUs

As discussed in Section 4.3, due to the significant slowdown compared to the original machine learning tasks, previous secure machine learning frameworks are difficult to put into practice. ParSecureML shortens this gap, and in this part, we analyze the comparison between ParSecureML and non-secure machine learning tasks on GPUs.

TABLE 2
Comparison With Original Non-Secure GPU Machine Learning Tasks

Dataset	Model	GPU time (s)	SecureML slowdown (×)	ParSecureML slowdown (×)
VGGFace2	CNN	15.26	181.06	34.92
	MLP	8.65	964.51	9.91
	linear	7.33	320.95	11.94
	logistic	7.30	321.23	11.51
	SVM	109.79	90.38	14.10
NIST	CNN	10.88	343.62	33.26
	MLP	6.40	941.60	13.48
	linear	5.33	575.13	11.57
	logistic	5.44	562.95	12.81
	SVM	76.09	43.83	6.69
SYNTHETIC	CNN	16.11	197.30	10.93
	MLP	13.58	226.72	8.80
	linear	9.76	196.97	8.53
	logistic	10.27	189.37	8.36
	SVM	82.33	43.39	6.58
	RNN	15.28	772.29	6.81
MNIST	CNN	2.05	36.53	8.01
	MLP	2.22	51.77	3.19
	linear	1.77	35.47	3.57
	logistic	1.83	34.50	3.44
	SVM	4.47	13.24	5.51
CIFAR-10	CNN	2.85	83.13	19.07
	MLP	2.51	142.37	7.03
	linear	2.13	35.83	6.13
	logistic	2.16	36.51	6.42
	SVM	4.51	42.09	12.93
Average	All	16.40	249.34	10.98

The GPU time refers to the execution time of non-secure machine learning tasks on GPUs.

We show the comparison of both the previous SecureML and our ParSecureML to previous non-secure machine learning tasks on GPUs in Table 2. First, SecureML exhibits a 249.34× slowdown compared to the non-secure machine learning tasks, while our ParSecureML significantly shortens this performance gap to 10.98×, which greatly advances the practical application of two-party computation. Second, CNN has the highest performance gap in most situations; the reason is that each sliding window reads matrices E, F, and Z, which incurs more overhead of two-party computation. Third, tasks on MNIST exhibit the lowest performance gap, which is due to the small image size of MNIST.

7.5 Detailed Analysis

In this part, we conduct a detailed analysis for ParSecureML, including time breakdowns, communication benefits, and influence from workload size.

Time Breakdowns. The detailed time breakdowns are shown in Table 3, and we have the following observations. First, for SecureML, the online part accounts for more than 90 percent of the machine learning tasks, which should be

TABLE 3
Time Breakdown for SecureML and ParSecureML

Dataset	Model	Online Time (s)		Total Time (s)		Occupancy (%)	
		SecureML	ParSecureML	SecureML	ParSecureML	SecureML	ParSecureML
VGGFace2	CNN	2718.13	490.24	2763.09	532.84	98.37	92.01
	MLP	8296.76	45.23	8344.94	85.73	99.42	52.76
	linear	2308.54	43.25	2354.11	87.56	98.06	49.39
	logistic	2300.79	42.62	2346.45	84.05	98.05	50.71
	SVM	8886.77	120.22	9922.26	431.38	89.56	27.87
NIST	CNN	3653.78	330.00	3737.47	361.80	97.76	91.21
	MLP	5882.10	26.45	6024.19	80.84	97.64	32.71
	linear	2979.88	29.59	3063.35	61.60	97.28	48.03
	logistic	2979.58	39.25	3062.55	69.69	97.29	56.32
	SVM	2774.92	42.84	3334.53	147.69	83.22	29.01
SYNTHETIC	CNN	3143.54	142.23	3177.50	176.03	98.93	80.80
	MLP	3045.21	83.17	3079.19	119.54	98.90	69.57
	linear	1888.29	48.24	1922.20	83.21	98.24	57.98
	logistic	1911.45	48.94	1945.30	85.84	98.26	57.01
	SVM	2946.99	19.46	3572.34	102.66	82.49	18.96
	RNN	11767.90	67.48	11803.32	104.06	99.70	64.85
MNIST	CNN	73.45	12.76	74.77	16.39	98.23	77.87
	MLP	113.45	3.44	114.85	7.09	98.78	48.53
	linear	61.65	2.72	62.94	6.34	97.95	42.84
	logistic	61.97	2.74	63.29	6.31	97.91	43.40
	SVM	46.50	1.40	59.18	5.87	78.56	23.86
CIFAR-10	CNN	232.59	45.23	236.79	54.32	98.22	83.27
	MLP	352.11	11.27	356.75	17.61	98.70	63.97
	linear	72.38	7.01	76.39	13.06	94.74	53.65
	logistic	74.91	7.87	78.93	13.88	94.91	56.69
	SVM	149.17	4.26	189.93	15.77	78.54	27.02

Occupancy refers to the ratio of online time to total time.

the major focus for GPU acceleration. Second, the offline part only accounts for a small portion of the whole process, which is less than 10 percent in our experiments. Third, after the GPU acceleration, the occupancy is reduced to 54.2 percent on average, which proves the effectiveness of ParSecureML.

Communication Benefits. We develop a compression-based communication technique between servers in ParSecureML to reduce communication overhead. In this part, we analyze the benefits of this optimization. The communication benefits are shown in Fig. 16. On average, ParSecureML reduces 22.9 percent communication overhead, which proves the effectiveness of this technique.

Influence of Workload Size. We study the influence from workload size in this part, since GPU parallelism cannot be fully utilized when the workload size is small. From Section 7.2, we can see that the workload size influences the performance speedups of ParSecureML: small workloads should be processed on CPUs while large workloads should be scheduled on GPUs. We quantitatively analyze this influence, as shown in Fig. 17. We use the SYNTHETIC dataset for validation, and change the workload size from 1MB to 4GB. We can see that the performance improvement increases along with the workload size.

7.6 Insight

Based on the detailed analysis, we have the following insights. First, GPU can significantly accelerate the process of secure machine learning, so it is essential to use GPUs in secure machine learning. Second, simply using GPUs cannot fully release the system computing power. Other techniques such double pipeline, compressed transmission, and architecture optimizations are also necessary. Third, the performance of ParSecureML relates to the workload size: the system is more suitable for processing large workloads.

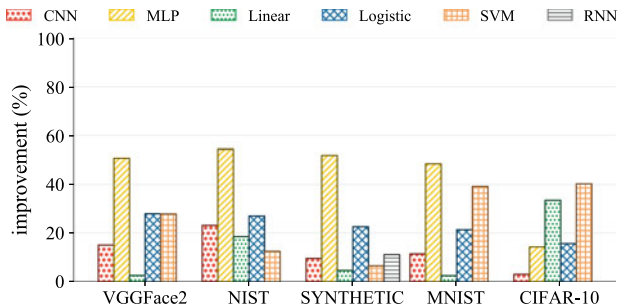


Fig. 16. Communication benefits.

7.7 Discussion

We discuss some common concerns about ParSecureML in this part, including comparison with the original versions, application scope, and limitation.

Possible Application Scenarios. First, ParsecureML accelerates two-party computation, so secure machine learning applications, which are based on two-party computation, can benefit

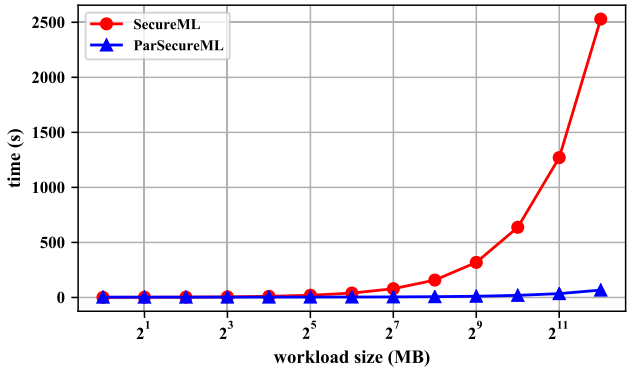


Fig. 17. Performance of different workload size.

from ParSecureML. Second, machine learning tasks based on convolution can be protected by the system. Third, Although ParSecureML targets machine learning tasks, ParSecureML can also be used in other matrix-based computing tasks. The core functionality in ParSecureML is to protect the triplet multiplication operations. When a program includes such matrix operations, the program can be protected and accelerated by ParSecureML. Moreover, for more advanced machine learning models, like ResNet [59], they have not substantially changed the use of convolution. For example, most layers in ResNet are convolution layers, which are similar to the CNN in our evaluation. Therefore, ParSecureML still can be used.

Limitation. The ParSecureML does not gain much performance benefits to the small scale machine learning tasks. First, when the input matrix is not large enough, the GPU resources may not be fully utilized. Second, for small scale application, the PCIe transmission overhead is usually obvious, which diminishes the GPU computation benefits. Third, CPUs are good at solving small scale problems.

Accuracy Loss. Our work focuses on improving the performance of SecureML [10], but not the accuracy. ParSecureML achieves the same accuracy as SecureML [10] does, which incurs marginal accuracy loss (less than 1 percent).

8 Related Work

As far as we know, this is the first work to enable GPUs in secure machine learning with two-party computation. In recent years, many works have been conducted to apply GPUs in machine learning [29], [60], [61], [62], [63]. The work closest to our work, ParSecureML, is SecureML [10], which enables two-party computation in common machine learning applications. Compared to SecureML, ParSecureML advances the practicality of two-party computation to an unprecedented level. First, SecureML is extremely slow; according to our analysis, SecureML is about 2× slower of the original machine learning tasks, which limits its applicability in practice. Second, SecureML uses small datasets for validation, including MNIST [25] Gisette dataset [64], and Arcene dataset [65]. These datasets are too small and cannot represent the data size of current popular machine learning tasks. Third, the experimental platform used in [10] cannot represent the computing power of current hardware. SecureML is only validated on Amazon EC2 with c4.8xlarge machines, while this paper verifies the performance in the latest HPC environments.

Two-party computation is a representative of multiparty computation [66]. Multiparty computation splits data across multiple servers, and each server contains only encrypted partial data, which protects the data from being leaked. Unfortunately, the more the number of parties are used, the more obvious the performance degradation, due to the increased computation and communication overheads. Consequently, many researchers are trying to improve the efficiency of multiparty computation [13], [67], [68], [69]. For example, Bogdanov and others [69] studied and designed efficient protocols for multiparty computation. Agrawal and others [67] developed a customized secure two-party protocol for discretized training of DNNs. Moreover, two-party computation is the lowest cost case in multiparty computation, and thus Mohassel and others [10] used two-party computation as the basic algorithm in SecureML. However, GPUs, as the most extensive accelerators for machine learning tasks, have not been used in two-party computation for performance acceleration. This work is by far the first work to use GPUs to accelerate secure machine learning applications of two-party computation.

Differential privacy [14] is a similar concept that is easily confused with data security. It adds some disturbing data in the original data, so that an intruder cannot guarantee the authenticity of the data. However, because a large amount of randomization is added to the original, the applicability of the data decreases. Even worse, if the data in the server are leaked, how much protection differential privacy can provide is unknown. Currently, differential privacy is mainly used to make the trained model contain private data as less as possible [70]. For example, in the field of medical analysis, the purpose of differential privacy is to extract features without exposing the user's personal privacy information [71]. Different from differential privacy, multiparty computation protects the data itself, which is much safer. Homomorphic encryption [72], [73] is also an important data protection method, but its high computational complexity limits its scope of application.

9 Conclusion

This paper proposes the first GPU-based two-party computation machine learning method, to enable GPUs in secure machine learning tasks. By enabling efficient profiling-guided adaptive GPU acceleration, along with the optimizations of double pipelines and compression-based transmission, common machine learning tasks achieve an average of 33.8× speedup over the state-of-the-art method. This paper presents how the method can be materialized, and discusses the major challenges in applying GPUs in two-party computation. Additionally, we provide a framework called ParSecureML to help ease the burden for developers using GPUs in secure machine learning applications. The experiments demonstrate the huge potential of applying GPUs in secure machine learning applications.

Reproducibility

We support reproducible science. iMLBench is available as a free open-source benchmark on GitHub (https://github.

com/ZhengChenCS/ParSecureML), Mulan Open Source Community (https://toscode.gitee.com/ZhengChenCS/ParSecureML), and Code Ocean.

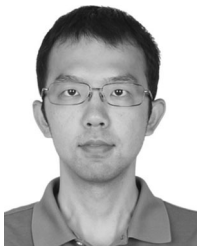
Acknowledgments

This work was supported by the National R&D Program of China under Grant 2020AAA0105200, in part by the National Natural Science Foundation of China under Grant U20A20226, Grant 61802412, Grant 61802260, Grant 61972403, and Grant 61732014, in part by the Beijing Natural Science Foundation under Grant 4202031 and Grant L192027, in part by the Beijing Academy of Artificial Intelligence (BAAI), and in part by the Tsinghua University-Peking Union Medical College Hospital Initiative Scientific Research Program. The work of Amelie Chi Zhou was also supported by the Shenzhen Science and Technology Foundation under Grant JCYJ20180305125737520 and a Tencent "Rhinoceros Birds" project of Scientific Research Foundation for Young Teachers of Shenzhen University.

References

[1] D. W. Archer *et al.*, "From keys to databases—Real-world applications of secure multi-party computation," *Comput. J.*, vol. 61, pp. 1749–1771, 2018.
[2] Jana: Private data as a service, 2020. [Online]. Available: https://galois.com/project/jana-private-data-as-a-service/
[3] The next generation of data-driven services with end-to-end data protection and accountability, 2020. [Online]. Available: https://sharemind.cyber.ee/
[4] A. C. Yao, "Protocols for secure computations," in *Proc. 23rd Annu. Symp. Found. Comput. Sci.*, 1982, pp. 160–164.
[5] Y. Zhang *et al.*, "A provable-secure and practical two-party distributed signing protocol for SM2 signature algorithm," *Front. Comput. Sci.*, vol. 14, no. 3, 2020, Art. no. 143803.
[6] Transforming Trust, 2020. [Online]. Available: https://www.unboundtech.com/
[7] A. Gascón *et al.*, "Secure linear regression on vertically partitioned datasets," *IACR Cryptol. ePrint Arch.*, vol. 2016, 2016, Art. no. 892.
[8] V. Nikolaenko, U. Weinsberg, S. Ioannidis, M. Joye, D. Boneh, and N. Taft, "Privacy-preserving ridge regression on hundreds of millions of records," in *Proc. IEEE Symp. Secur. Privacy*, 2013, pp. 334–348.
[9] P. Mohassel and P. Rindal, "ABY 3: A mixed protocol framework for machine learning," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, 2018, pp. 35–52.
[10] P. Mohassel and Y. Zhang, "SecureML: A system for scalable privacy-preserving machine learning," in *Proc. IEEE Symp. Secur. Privacy*, 2017, pp. 19–38.
[11] P. Schoppmann, A. Gascón, and B. Balle, "Private nearest neighbors classification in federated databases," *IACR Cryptol. ePrint Arch.*, vol. 2018, 2018, Art. no. 289.
[12] E. M. Songhori, S. U. Hussain, A.-R. Sadeghi, and F. Koushanfar, "Compacting privacy-preserving k-nearest neighbor search using logic synthesis," in *Proc. 52nd ACM/EDAC/IEEE Des. Autom. Conf.*, 2015, pp. 1–6.
[13] C. Hong *et al.*, "Privacy-preserving collaborative machine learning on genomic data using tensorflow," in *Proc. ACM Turing Celebration Conf.-China*, 2020.
[14] C. Dwork, "Differential privacy: A survey of results," in *Proc. Int. Conf. Theory Appl. Models Comput.*, 2008, pp. 1–19.
[15] S. R. Upadhyaya, "Parallel approaches to machine learning—A comprehensive survey," *J. Parallel Distrib. Comput.*, vol. 73, pp. 284–292, 2013.
[16] A. Parvat, J. Chavan, S. Kadam, S. Dev, and V. Pathak, "A survey of deep-learning frameworks," in *Proc. Int. Conf. Inventive Syst. Control*, 2017, pp. 1–7.
[17] Z. Chen *et al.*, "ParSecureML: An efficient parallel secure machine learning framework on GPUs," in *Proc. 49th Int. Conf. Parallel Process.*, 2020, Art. no. 22.
[18] S. Markidis, S. W. D. Chien, E. Laure, I. B. Peng, and J. S. Vetter, "NVIDIA tensor core programmability, performance precision," in *Proc. IEEE Int. Parallel Distrib. Process. Symp. Workshops*, 2018, pp. 522–531.
[19] S. Lawrence, C. L. Giles, A. C. Tsoi, and A. D. Back, "Face recognition: A convolutional neural-network approach," *IEEE Trans. Neural Netw.*, vol. 8, no. 1, pp. 98–113, Jan. 1997.
[20] S. K. Pal and S. Mitra, "Multilayer perceptron, fuzzy sets, classifiaction," *IEEE Trans. Neural Netw.*, vol. 3, no. 5, pp. 683–697, Sep. 1992.
[21] G. A. Seber and A. J. Lee, *Linear Regression Analysis*, Hoboken, NJ, USA, vol. 936, 2012.
[22] D. G. Kleinbaum *et al.*, *Logistic Regression*. Berlin, Germany: Springer, 2002.
[23] T. Mikolov *et al.*, "Recurrent neural network based language model," in *Proc. 11th Annu. Conf. Int. Speech Commun. Assoc.*, 2010, pp. 1045–1048.
[24] J. A. Suykens and J. Vandewalle, "Least squares support vector machine classifiers," *Neural Process. Lett.*, vol. 9, pp. 293–300, 1999.
[25] Y. LeCun, C. Cortes, and C. J. Burges, "The MNIST database of handwritten digits," 1998. [Online]. Available: http://yann.lecun.com/exdb/mnist
[26] Q. Cao *et al.*, "VGGFace2: A dataset for recognising faces across pose and age," in *Proc. Int. Conf. Autom. Face Gesture Recognit.*, 2018, pp. 67–74.
[27] C. I. Watson and C. Wilson, "Nist special database 4," *Fingerprint Database Nat. Inst. Standards Technol.*, vol. 17, no. 77, p. 5, 1992.
[28] A. Krizhevsky *et al.*, "Learning multiple layers of features from tiny images," 2009.
[29] S. Tang, B. He, C. Yu, Y. Li, and K. Li, "A survey on spark ecosystem: Big data processing infrastructure, machine learning, and applications," *IEEE Trans. Knowl. Data Eng.*, to be published, doi: 10.1109/TKDE.2020.2975652.
[30] S. Tang, C. Yu, and Y. Li, "Fairness-efficiency scheduling for cloud computing with soft fairness guarantees," *IEEE Trans. Cloud Comput.*, to be published, doi: 10.1109/TCC.2020.3021084.
[31] D. Malkhi *et al.*, "Fairplay-secure two-party computation system," in *Proc. USENIX Secur. Symp.*, 2004, Art. no. 20.
[32] K.-S. Oh and K. Jung, "GPU implementation of neural networks," *Pattern Recognit.*, vol. 37, pp. 1311–1314, 2004.
[33] L. Liu, S. Yang, L. Peng, and X. Li, "Hierarchical hybrid memory management in OS for tiered memory systems," *IEEE Trans. Parallel Distrib. Syst.*, vol. 30, no. 10, pp. 2223–2236, Oct. 2019.
[34] X. Li *et al.*, "Thinking about a new mechanism for huge page management," in *Proc. 10th ACM SIGOPS Asia-Pacific Workshop Syst.*, 2019, pp. 40–46.
[35] M. Abadi *et al.*, "TensorFlow: A system for large-scale machine learning," in *Proc. 12th USENIX Conf. Operating Syst. Des. Implementation*, 2016, pp. 265–283.
[36] N. Ketkar, "Introduction to Pytorch," in, *Deep Learning With Python*. Shelter Island, NY, USA: Manning Publications, 2017.
[37] Y. Jia *et al.*, "Caffe: Convolutional architecture for fast feature embedding," in *Proc. 22nd ACM Int. Conf. Multimedia*, 2014, pp. 675–678.
[38] C. M. Bishop, *Pattern Recognition and Machine Learning*. Berlin, Germany: Springer, 2006.
[39] P. Hummel and C. Seebeck Jr, "A generalization of Taylor's expansion," *Amer. Math. Monthly*, vol. 56, pp. 243–247, 1949.
[40] A. F. Agarap, "Deep learning using rectified linear units (ReLU)," 2018, *arXiv: 1803.08375*.
[41] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, pp. 436–444, 2015.
[42] D. Michie *et al.*, Machine Learning, Neural and Statistical Classification, 1994.
[43] F. Zhang, J. Zhai, B. Wu, B. He, W. Chen, and X. Du, "Automatic irregularity-aware fine-grained workload partitioning on integrated architectures," *IEEE Trans. Knowl. Data Eng.*, vol. 33, no. 3, pp. 867–881, Mar. 2021.
[44] F. Zhang, J. Zhai, B. He, S. Zhang, and W. Chen, "Understanding co-running behaviors on integrated CPU/GPU architectures," *IEEE Trans. Parallel Distrib. Syst.*, vol. 28, no. 3, pp. 905–918, Mar. 2017.
[45] R. Landaverde, T. Zhang, A. K. Coskun, and M. Herbordt, "An investigation of unified memory access performance in CUDA," in *Proc. IEEE High Perform. Extreme Comput. Conf.*, 2014, pp. 1–6.
[46] W. Li *et al.*, "An evaluation of unified memory technology on NVIDIA GPUs," in *Proc. 15th IEEE/ACM Int. Symp. Cluster Cloud Grid Comput.*, 2015, pp. 1092–1098.

[47] N. Bell and M. Garland, "Efficient sparse matrix-vector multiplication on CUDA," Nvidia Corporation, Santa Clara, CA, USA, Nvidia Tech. Rep. NVR-2008–004, 2008.
[48] S. Konuma and S. Ichikawa, "Design and evaluation of hardware pseudo-random number generator MT19937," *IEICE Trans. Inf. Syst.*, vol. 88, no. 12, pp. 2876–2879, 2005.
[49] M. Matsumoto and T. Nishimura, "Mersenne twister: A 623-dimensionally equidistributed uniform pseudo-random number generator," *ACM Trans. Model. Comput. Simul.*, vol. 8, no. 1, pp. 3–30, 1998.
[50] Profiler: CUDA Toolkit Documentation, 2020. [Online]. Available: https://docs.nvidia.com/cuda/profiler-users-guide/index.html
[51] T. Chen *et al.*, "MXNet: A flexible and efficient machine learning library for heterogeneous distributed systems," 2015, *arXiv:1512.01274*.
[52] A. Markham and Y. Jia, "Caffe2: Portable high-performance deep learning framework from Facebook," *NVIDIA Corporation*, 2017.
[53] S. Markidis, S. W. Der Chien, E. Laure, I. B. Peng, and J. S. Vetter, "NVIDIA tensor core programmability, performance & precision," in *Proc. IEEE Int. Parallel Distrib. Process. Symp. Workshops*, 2018, pp. 522–531.
[54] N. P. Jouppi *et al.*, "In-datacenter performance analysis of a tensor processing unit," in *Proc. 44th Annu. Int. Symp. Comput. Archit.*, 2017, pp. 1–12.
[55] Z.-Q. Zeng, H.-B. Yu, H.-R. Xu, Y.-Q. Xie, and J. Gao, "Fast training support vector machines using parallel sequential minimal optimization," in *Proc. 3rd Int. Conf. Intell. Syst. Knowl. Eng.*, 2008, pp. 997–1001.
[56] F. V. Massoli *et al.*, "Improving multi-scale face recognition using VGGFace2," in *Proc. Int. Conf. Image Anal. Process.*, 2019, pp. 21–29.
[57] Khanna and W. Shen, "Automated fingerprint identification system (AFIS) benchmarking using the national institute of standards and technology (NIST) special database 4," in *Proc. IEEE Int. Carnahan Conf. Secur. Technol.*, 1994, pp. 188–194.
[58] L. Schott *et al.*, "Towards the first adversarially robust neural network model on MNIST," 2018, *arXiv: 1805.09190*.
[59] H. Mikami *et al.*, "ImageNet/ResNet-50 training in 224 seconds," 2018, *arXiv: 1811.05233*.
[60] T. Z. Islam *et al.*, "A machine learning framework for performance coverage analysis of proxy applications," in *Proc. Int. Conf. High Perform. Comput. Netw. Storage Anal.*, 2016, Art. no. 46.
[61] E. Park *et al.*, "Predictive modeling in a polyhedral optimization space," *Int. J. Parallel Program.*, vol. 41, pp. 704–750, 2013.
[62] G. Ma *et al.*, "Similarity learning with higher-order proximity for brain network analysis," 2018, *arXiv: 1811.02662*.
[63] C. Xie *et al.*, "OO-VR: NUMA friendly object-oriented VR rendering framework for future NUMA-based multi-GPU systems," in *Proc. 46th Int. Symp. Comput. Archit.*, 2019, pp. 53–65.
[64] Gisette data set, 2008. [Online]. Available: https://archive.ics.uci.edu/ml/datasets/Gisette
[65] I. Guyon *et al.*, "Arcene data set," Access mode, 2004. [Online]. Available: http://archive.ics.uci.edu/ml/datasets/Arcene
[66] O. Goldreich, "Secure multi-party computation," Manuscript. Preliminary version, 1998.
[67] N. Agrawal *et al.*, "QUOTIENT: Two-party secure neural network training and prediction," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, 2019, pp. 1231–1247.
[68] M. Hirt, U. Maurer, and B. Przydatek, "Efficient secure multi-party computation," in *Proc. Int. Conf. Theory Appl. Cryptol. Inf. Secur.*, 2000, pp. 143–161.
[69] D. Bogdanov *et al.*, "High-performance secure multi-party computation for data mining applications," *Int. J. Inf. Secur.*, vol. 11, pp. 403–418, 2012.
[70] M. Abadi *et al.*, "Deep learning with differential privacy," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, 2016, pp. 308–318.
[71] C. Lin *et al.*, "Differential privacy preserving in big data analytics for connected health," *J. Med. Syst.*, vol. 40, 2016, Art. no. 97.
[72] C. Gentry and D. Boneh, *A Fully Homomorphic Encryption Scheme*. Stanford, CA, USA: Stanford Univ., 2009.
[73] A. Rodrigo, M. Dayarathna, and S. Jayasena, "Latency-aware secure elastic stream processing with homomorphic encryption," *Data Sci. Eng.*, vol. 4, pp. 223–239, 2019.



Feng Zhang received the bachelor's degree from Xidian University, Xi'an, China, in 2012, and the PhD degree in computer science from Tsinghua University, Beijing, China, in 2017. He is an associate professor with the Key Laboratory of Data Engineering and Knowledge Engineer (MOE), Renmin University of China. His major research interests include high performance computing, heterogeneous computing, and parallel and distributed systems.

Zheng Chen received the bachelor's degree from the School of Information, Renmin University of China, Beijing, China, in 2020. He is currently working toward the PhD degree with the School of Information, Renmin University of China, Beijing, China. He joined the Key Laboratory of Data Engineering and Knowledge Engineer (MOE) in 2019, and now works as a research assistant. His research interests include high performance computing, distributed and parallel big data systems, and machine learning.

Chenyang Zhang received the undergraduate degree from the School of Information, Renmin University of China, Beijing, China. She joined the Key Laboratory of Data Engineering and Knowledge Engineer (MOE) in 2019, and now works as a research assistant. Her research interests include high performance computing, heterogeneous computing, and parallel accelerating.

Amelie Chi Zhou received the PhD degree from the School of Computer Engineering, Nanyang Technological University, Singapore, in 2016. She is currently an assistant professor with Shenzhen University, China. Before joining Shenzhen University, she was a postdoc fellow with Inria-Bretagne Research Center, France. Her research interests include cloud computing, high performance computing, big data processing, and resource management.

Jidong Zhai received the BS degree in computer science from the University of Electronic Science and Technology of China, Chengdu, China, in 2003, and the PhD degree in computer science from Tsinghua University, Beijing, China, in 2010. He is an associate professor with the Department of Computer Science and Technology, Tsinghua University. His research interests include performance evaluation for high performance computers, performance analysis, and modeling of parallel applications.

Xiaoyong Du received the BS degree from Hangzhou University, Zhengjiang, China, in 1983, the ME degree from the Renmin University of China, Beijing, China, in 1988, and the PhD degree from the Nagoya Institute of Technology, Nagoya, Japan, in 1997. He is currently a professor with the School of Information, Renmin University of China. His current research interests include databases and intelligent information retrieval.