

ARTICLE TYPE

Distributed and Deep Vertical Federated Learning with Big Data

Ji Liu¹ | Xuehai Zhou¹ | Lei Mo² | Shilei Ji¹ | Yuan Liao¹ | Zheng Li³ | Qin Gu⁴ | Dejing Dou¹¹Baidu Inc., Beijing, China²School of Automation, Southeast University, Nanjing, China³Research Center, Chengdu Medical Union Information Co.LTD, Chengdu, China⁴Research Center, Chengdu Big Data Group Co.LTD, Chengdu, China**Correspondence**

Ji Liu. Email: liuji04@baidu.com

Summary

In recent years, data are typically distributed in multiple organizations while the data security is becoming increasingly important. Federated Learning (FL), which enables multiple parties to collaboratively train a model without exchanging the raw data, has attracted more and more attention. Based on the distribution of data, FL can be realized in three scenarios, i.e., horizontal, vertical, and hybrid. In this paper, we propose to combine distributed machine learning techniques with Vertical FL and propose a Distributed Vertical Federated Learning (DVFL) approach. The DVFL approach exploits a fully distributed architecture within each party in order to accelerate the training process. In addition, we exploit Homomorphic Encryption (HE) to protect the data against honest-but-curious participants. We conduct extensive experimentation in a large-scale cluster environment and a cloud environment in order to show the efficiency and scalability of our proposed approach. The experiments demonstrate the good scalability of our approach and the significant efficiency advantage (up to 6.8 times with a single server and 15.1 times with multiple servers in terms of the training time) compared with baseline frameworks.

KEYWORDS:

Federated learning, Distributed system, Parallel computing.

1 | INTRODUCTION

Recent years have witnessed huge amounts of distributed data over Internet of Things (IoT) devices and organizations¹. However, data aggregation from distributed devices, multiple regions, or organizations, become almost impossible² due to the concerns of data security, stringent privacy and legal restrictions^{3,4,5,6,7}. In this case, training models with distributed data from various sources becomes critical for the collaboration among multiple parties. In order to address this issue, Federated Learning (FL)^{8,2} is proposed.

FL was first introduced to train a global model with unbalanced and non-Independent and Identically Distributed (non-IID) data distributed on mobile devices⁸, which is referred to Horizontal Federated Learning (HFL). The concept of FL is extended to three categories, i.e., horizontal, vertical, hybrid². The Vertical FL (VFL), which is also denoted as feature-based federated learning², handles the decentralized data of the same identifications with different features while the hybrid FL deals with the data of different identifications and different features. While many existing FL works focus on HFL, VFL remains a critical open problem^{9,10}. VFL is well-fit for the cases with similar sample space but differs in feature space. For instance, an investment bank and a commercial bank in the same area may share the same group of clients (similar sample space), but the investment bank records the investment behaviors of users while the commercial bank stores consuming habits (difference in the feature space).

FL is an on-the-rising research area of privacy computing because jointly training a model in a privacy-preserving fashion is in high demand. In the field of privacy computing, the Secure Multi-Party Computation (MPC)¹¹ and Trusted Execution Environment (TEE)¹² are two mainstream research directions. The nature of TEE itself needs to provide an isolated runtime environment and secure storage for data, which compromises much of the computation efficiency. The overlapping of FL and MPC studies is ubiquitous, while MPC techniques like Homomorphic Encryption (HE), Oblivious Transfer (OT) or Private Set Intersection (PSI) are regularly transited to FL settings. Nevertheless, because of the limitation of MPC algorithms, even when they have been optimized, the computational speed is an order of magnitude slower than plaintext computing. Hence, shortening algorithm runtime and simultaneously ensuring data security become critical in industrial applications. In this case, a distributed deep learning structure for FL is becoming promising to address this issue.

In this paper, we combine HE and PSI techniques from MPC studies to preserve data privacy. When the training data contains sensitive information, such as ages, facial features, health conditions, credit card numbers, etc., the direct usage of raw data becomes improper. As a widely used tool^{13,14}, HE can improve the data security in FL settings. Since the data is preferably not be transmitted explicitly, researchers naturally suggest passing in the ciphered data. While it allows computation over encrypted data without access to the secret key, the HE technique has been generally utilized in the FL settings¹⁵. Furthermore, before the training process, data from two parties need to be aligned using PSI. PSI methods allow two parties to obtain their data intersection without revealing additional information to another party^{16,17}.

Recent years have witnessed the thriving of VFL studies in multiple domains, e.g., privacy-preserving methods, performance optimization, and application scenario extension, etc. VFL was first proposed with a Logistic Regression (LR) method in Taylor approximation model with partially HE⁹. A stochastic kernel learning model is also adapted to FL¹⁸. In addition, multiparty situation of VFL is studied in¹⁹. Some studies apply Gaussian Differential Private (DP) perturbation on LR or neural networks, which broadens privacy-preserving approaches on VFL²⁰. Vertical FL is exploited to collaboratively train a global model while ensuring the privacy and security of data. When multiple data owners conduct cross-silo federated learning, in order to protect the privacy and security of data, vertical federated learning generally utilizes cryptographic operations, which incur frequent encryption and decryption. While cross-silo network bandwidth is limited, the cryptographic operations bring significant overhead, and training time can be prolonged to several times to millions of times. Especially in industrial scenarios, such as advertising and marketing with the collaboration from multiple companies, it may take hours or even days to collaboratively train a global model with big data with cryptographic operations for security, which cannot meet the time requirements for rapid business development. The efficiency of vertical FL has become a critical problem that restricts its large-scale application^{21,22}. Nonetheless, despite some of above approaches consider parallel execution, these approaches are not sufficient to handle industrial big data (generally over 10TB) in a deep learning setting. Although some frameworks, e.g., FATE²³ and PyVertical²⁴, enable the execution of VFL, they generally cannot exploit distributed resources, e.g., multiple workers, for efficient big data processing.

In this paper, With the consideration of both efficiency and data privacy, we propose the Distributed Vertical Federated Learning (DVFL) approach, which exploits a distributed architecture to accelerate the training process of VFL with big data while combining HE to ensure data security. The proposed architecture can accelerate the training process of the vertical FL with an almost linear performance improvement, which can be applied in joint marketing advertising, data federation²⁵ in government, joint risk control in financial companies and so on. We apply the parameter server and peer-to-peer communication architecture to accelerate the big data processing. The proposed architecture can be deployed in a large-scale cluster environment environment or a cloud environment²⁶, where multiple powerful servers can be offered for parallel computing²⁷.

We summarise our main contributions of this paper as follows:

- A fully Distributed structure for Vertical Federated Learning (DVFL) in order to handle big data in industrial applications.
- The combination of the DVFL and HE techniques to protect the security and privacy of data.
- We carry out extensive experimentation based on real-life deep learning models in a distributed environment, including the large-scale cluster environment and the cloud environment, which shows the advantages of DVFL over state-of-the-art platforms, i.e., FATE and PyVertical.

The rest of this paper is organized as follows. In Section 2, we give a brief overview of related works. In section 3, we list the preliminaries. In section 4, we present our distributed solution, i.e., DVFL. In section 5, we present the experiment details and results. In section 6, we draw our conclusion and shed light on possible future research work.

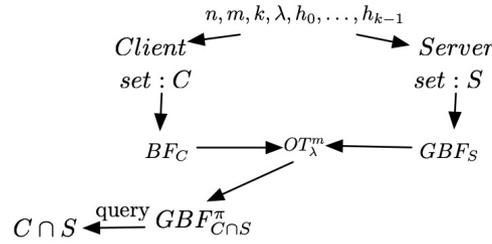


FIGURE 1 The basic PSI protocol⁵⁴

2 | RELATED WORK

In 2016, the concept of FL was first introduced by⁸. Their approach aims at training a Logistic Regression (LR) model based on distributed data stored in a huge amount of global mobile users without conflicting local regulations or laws. Their approach is classified into horizontal FL years later, because of the wide range of participants and similar data features, e.g., health care with the attention mechanism on graphs²⁸, 5G-empowered drone networks with reinforcement learning for smart grid or smart cities²⁹, frequent itemset mining³⁰, distributed medical data³¹ and education data³², purchase behaviour with an attention mechanism³³, and Internet of Things (IoT) devices^{34,31}. In addition, the horizontal FL can be carried out in the Cloud for extreme gradient boosting³⁵ or deep learning models with the combination of synchronous^{36,37,38,39,40} and asynchronous mechanisms^{41,42}. Later, the first FL on vertically partitioned data was proposed to train a LR model⁹, which exploits HE to protect data security. Lastly, A GELU-Net model is trained based on the combination of FL and HE, while the experimentation was carried out using the MNIST dataset, which shows a decent computation speed performance⁴³. In addition, the 128 MB MNIST dataset (MNIST) is far smaller than industrial big data. Hitherto, upon state-of-the-art VFL models, there is not a solution that can efficiently handle real big data with the VFL settings.

We have witnessed the growth of the distributed learning over past years. Distributed models like DistBelief⁴⁴, GraphLab⁴⁵, Petuum⁴⁶, Naiad⁴⁷, MLbase⁴⁸, etc, have been well explored in the early years. A parameter server architecture⁴⁹ can utilize a parameter server or several parameter servers and multiple workers to train a model using a dataset of 636 TB in around 1 hour^{50,51}. The parameter servers coordinates the execution while workers work in parallel to train the model. We were inspired by their parameter server architecture to propose the DVFL.

There has been a fairly long history of PSI study⁵². PSI has been widely used as the entity resolution protocol in VFL researches⁵³. An efficient PSI protocol that combines Bloom Filter (see details in Section 3.1) with secret share scheme⁵⁴ is used in DVFL. Using the PSI techniques, we can transform a hybrid FL problem to a VFL problem while ensuring the data security.

Lastly, HE has been widely used in VFL settings. Two main branches of HE exist, i.e., fully HE and partially HE. The fully HE, e.g., Brakerski-Gentry-Vaikuntanathan HE⁵⁵, supports both addition and multiplication on ciphertext, while partially HE, e.g., Paillier HE⁵⁶, supports either addition or multiplication operation on ciphertext. Both the fully and partially HE can be exploited with the VFL settings while the fully HE has the problem of high time complexity⁹. While it has better runtime efficiency, partially HE corresponds to less computational flexibility. The property of the Paillier cryptosystem allows addition on both parties of ciphertext and multiplication on one party. Paillier HE is widely utilized as the privacy-preserving strategy⁴³.

3 | PRELIMINARIES

In this section, we present the background of DVFL. We first introduce the concept of PSI and the idea of its protocol. Then, we explicate the detail of the parameter server. Afterward, we indicate Paillier HE and its homomorphic property. Finally, we specify the deep neural networks structure and how we apply the HE technique in DVFL.

3.1 | Private Set Intersection

Private Set Intersection (PSI) refers to a solution that calculates the intersection of two parties. Each of the two parties holds a set of items, and the two parties want to know the intersection of their sets without revealing any information outside of the

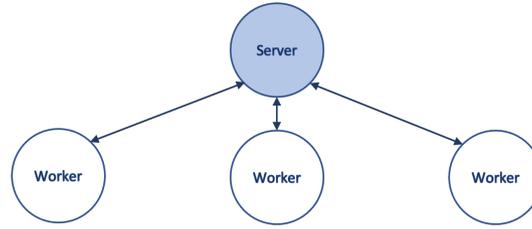


FIGURE 2 Parameter Server Architecture

intersection. Let us assume that one party has a set of items $A = \{a_1, a_2, \dots, a_n\}$ and another has a set of $B = \{b_1, b_2, \dots, b_m\}$. PSI is a protocol that acquires the intersection of the two parties, i.e., $A \cap B$, while it does not share any data excluded from the intersection of the two sets. If $a_i \notin B$, then party B should learn nothing about a_i with a PSI solution.

As shown in Figure 1, the PSI algorithm⁵⁴ we exploit is constructed using two Bloom Filters (BF) where the BF is designed to tell whether an element is present in a set⁵⁷. Let us assume that one party is a server with set S , and another party is a client with set C . The client computes a BF that encodes its set C and the server computes a Garbled Bloom Filter (GBF) that encodes its set S . Note that the GBF is conceptually a BF with a security parameter while GBF has better capacity to hide the information of the items in the union of the two sets but outside of the intersection of the two sets. Once the GBF and the BF are obtained, the server acts as a sender and the client acts as a receiver to execute an OT protocol⁵⁸. The OT protocol allows a receiver to obtain one out of two strings held by a sender, without revealing to the sender the identity of its selection⁵⁹. Then, the server and the client can build the intersection $GBF_{C \cap S}$ on GBF_S and BF_C . Thus, the client computes the $C \cap S$ set by querying all elements in its set filtered by the $GBF_{C \cap S}$. BF has an intrinsic flaw of false positive cases, while the false positive is negligible⁵⁴. In addition, since the data is independently hashed, it is a good single program multiple data (SPMD) scenario to run this algorithm in parallel.

3.2 | Parameter Server

The parameter server architecture⁶⁰ consists of a server and a group of workers, which moves data processing from one computing node to multiple computing nodes. Partitioned data are passed into paralleled workers. Each worker does not mutually communicate with other workers, but only communicates with the server node to retrieve and update gradients. Workers can run on GPUs or CPUs while the workers we utilized are CPU-oriented. Figure 2 gives the intuition of the parameter server architecture while the number of workers could increase or decrease in different application scenarios.

When there are many layers in a neural networks, the gradients are divided into multiple chunks to be communicated. These chunks of gradients need to be indexed, and the shared parameters are stored as key-value pairs. The key represents the ID, and the value represents the weight. Two operations are defined to bridge the server-worker communication:

- *push*: a worker sends its computed weights/gradients to the server.
- *pull*: a worker retrieves the aggregated weights/gradients from the server.

The distributed weights/gradients in workers are pushed and aggregated in the server, e.g., by calculating the average of the pushed weights or gradients. Then workers pull the combined gradient from the server. The server and the workers run these two operations until the model converges.

3.3 | Paillier Cryptosystem

HE allows data operates computation, sort, search, edit on the ciphertext. As mentioned earlier, fully HE and partially HE are two HE main branches. Fully HE supports multiple operations of unbounded depth on ciphertext, which is the strongest notion of HE. However, it is unpractical to run fully HE on big data because of its high time complexity. Even through partially HE supports only one operation (addition or multiplication, not both) on ciphertext, researchers tend to use partially HE to achieve a better runtime performance.

Paillier cryptosystem is a well-known partially HE, which is invented by and named after Pascal Paillier⁵⁶. Let us define plaintext numbers m_1 and m_2 and choose two large prime p and q . Then let n to be the multiplication of p and q . Paillier's

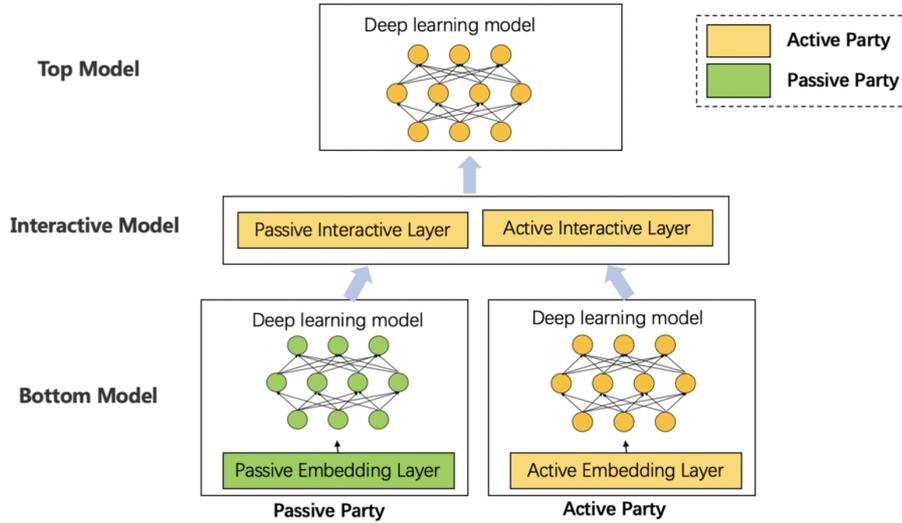


FIGURE 3 Deep Neural Networks for vertical FL⁴³

homomorphic property allows addition of two ciphertexts and multiplication of a ciphertext by a plaintext. When two ciphertexts are multiplied, the result decrypts to the sum of their plaintext as shown in Equation 1.

$$D_{priv}(E_{pub}(m_1) \times E_{pub}(m_2) \mod n^2) = m_1 + m_2 \mod n, \quad (1)$$

where D_{priv} represents the decryption operation, and E_{pub} represents the encryption operation. When a ciphertext is raised to the power of plaintext, the result decrypts to the product of the two plaintexts as shown in Equation 2.

$$D_{priv}(E_{pub}(m_1)^{m_2} \mod n^2) = m_1 \times m_2 \mod n \quad (2)$$

3.4 | Deep Neural Networks

The deep learning structure shown in Figure 3 is constructed by three levels: bottom neural networks, interactive layer, and top neural networks⁴³. A bottom neural network is given to any party that provides feature data. It brings more scalability to the architecture when more parties may participate in. Note that the bottom model can be a neural network with an arbitrary number of layers and an arbitrary number of neurons per layer. Then bottom networks are joined by a fully connected interactive layer.

Let the party that holds the label matrix be the active party, while the party that only holds a feature matrix be the passive party. Inside the interactive layer, both passive and active parties generate noise and encrypt their data by HE. Then the merged output are passed to the top model. The top model calculates the loss of the model and propagates the error back. The merit of this structure is that all cross-institutional data communications are processed within the interactive layer, which significantly reduces the risk of data leakage.

4 | DVFL SOLUTION

In this section, we illuminate how the approaches mentioned in Section 3 are combined in DVFL. Moreover, we propose our solution, i.e., Distributed Vertical Federated Learning (DVFL).

We propose DVFL to solve the problem of excessive runtime given by existing models on vertically partitioned data when it comes to big data. We exploit the distributed architecture to address this issue. We exploit the parameter server architecture explained in Section 3.2 for the parallelism.

We adopt the deep learning architecture discussed in Section 3.4. DVFL supports joint training with two parties, i.e., an active party and a passive party. The active party holds the feature matrix and the label matrix, and dominates the computation in training. The passive party holds only the feature matrix. We denote the active party Party A and the passive party Party P. Figure 4 shows the distributed architecture of DVFL. Data from Party A and Party P are partitioned to the number of n workers, $\{(X_1,$

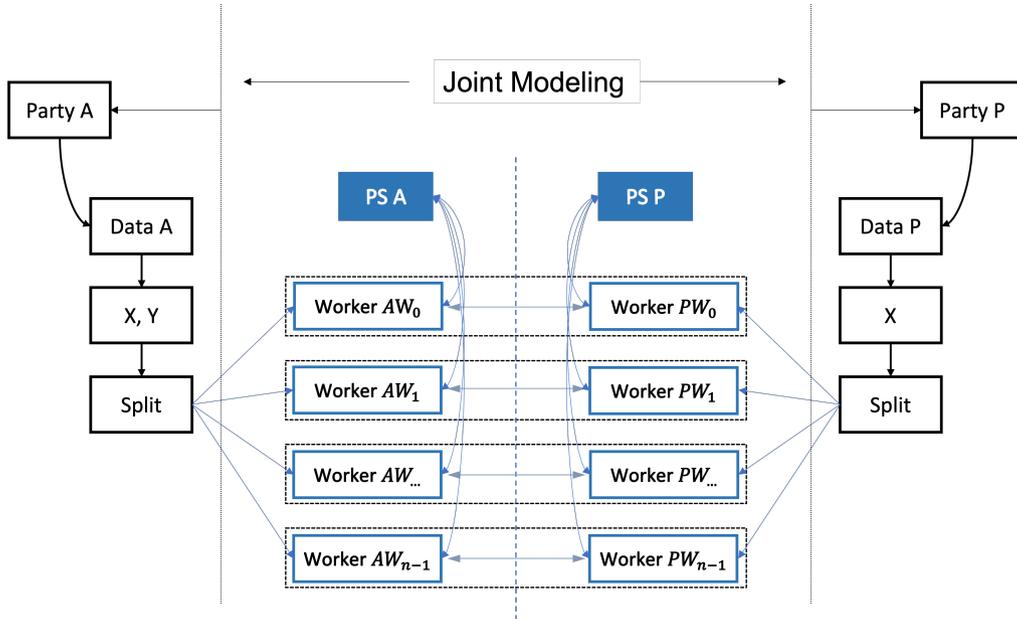


FIGURE 4 Distributed architecture with parameter servers for DVFL.

Algorithm 1 DVFL

Input:

- (X_A, Y_A) : active dataset;
- X'_P : passive dataset;
- n : the number of workers in a party

Output:

Net: a trained network

- 1: $(X_{A \cap P}, Y_{A \cap P}), X'_{A \cap P} \leftarrow \text{DistributedPSI}((X_A, Y_A), X'_P)$
 - 2: $\{(X_1, Y_1), (X_2, Y_2), \dots, (X_n, Y_n)\}, \{X'_1, X'_2, \dots, X'_n\} \leftarrow \text{sequentialPartition}((X_{A \cap P}, Y_{A \cap P}), X'_{A \cap P})$ ▷ partition the data into n similar length subsets
 - 3: $Net \leftarrow \text{VerticalFL}(\{(X_1, Y_1), (X_2, Y_2), \dots, (X_n, Y_n)\}, \{X'_1, X'_2, \dots, X'_n\})$
-

$Y_1), (X_2, Y_2), \dots, (X_n, Y_n)$ and $\{X'_1, X'_2, \dots, X'_n\}$. A parameter server is applied to each party. Workers under the same server do not communicate with each others. But they communicate with their corresponding workers across parties with a peer-to-peer manner for the PSI process and distributed training process of vertical FL.

The procedure of DVFL is demonstrated in Algorithm 1. (X_A, Y_A) represents the active dataset with X_A representing the features of the data set with identification set A in the sample space of the active party and Y_A representing the corresponding label for each data item. X'_P represents the passive dataset with identification set P in the sample space of the passive party. Once the dataset components are clarified, there are two preparation jobs before the training process of vertical FL. The first job is that two parties securely apply a Distributed PSI protocol to their data in order to obtain the common sample space $(A \cap P)$ (Line 1, see details in Algorithm 2). The second job is that two parties utilize a sequential data partitioning strategy to partition the big data into small chunks and send them to the workers of the parameter server architecture (Line 2). The items in each pair of data chunk with the same index in the active party and the passive party share the same identifications. Finally, a neural network is trained using the distributed vertical FL setting (Line 3) as explained in Algorithm 3.

Algorithm 2 illustrates the distributed PSI. First, the identification sets of active party (Line 1) and passive party (Line 2) are partitioned to n buckets of identifications with n representing the number of workers in each party using the same hash function⁶¹. Then, each bucket of identifications is handed by a corresponding worker, e.g., A_i is handled by Work i in the active party and P_i is handled by Work i in the passive party. As the identifications are partitioned using the same hash function, the same identifications in active dataset and passive dataset should be partitioned into the buckets of the same index. Then, the PSI

Algorithm 2 Distributed PSI**Input:**

- A : identification set in the active dataset;
- P : identification set in the passive dataset;
- n : the number of workers in a party

Output:

- $A \cap P$: the intersection of A and P
- 1: $\{A_1, A_2, \dots, A_n\} \leftarrow \text{hashPartition}(A)$ \triangleright Partition the identification set in the active party into n similar length subsets using a hash function
- 2: $\{P_1, P_2, \dots, P_n\} \leftarrow \text{hashPartition}(P)$ \triangleright Partition the identification set in the passive party into n similar length subsets using the same hash function
- 3: **for** $i \in \{1, 2, \dots, n\}$ **do** \triangleright execution in parallel
- 4: $A_i \cap P_i \leftarrow \text{PSI}(\{A_i, P_i\})$
- 5: **end for**
- 6: $A \cap P \leftarrow (A_1 \cap P_1) \cup (A_2 \cap P_2) \cup \dots \cup (A_n \cap P_n)$

Algorithm 3 Vertical FL**Input:**

- $\{(X_1, Y_1), (X_2, Y_2), \dots, (X_n, Y_n)\}$: dataset in the active party;
- $\{X'_1, X'_2, \dots, X'_n\}$: dataset in the passive party;
- Net_o : an initial neural network

Output:

- Net : a trained neural network
- 1: $Net \leftarrow Net_o$
- 2: **while** Stop condition is not met **do**
- 3: **for** $i \in \{1, 2, \dots, n\}$ **do** \triangleright execution in parallel
- 4: $w_i \leftarrow \text{getWeight}(X_i, Y_i, X'_i)$
- 5: **end for**
- 6: $Net \leftarrow \text{update}(Net, \{w_1, w_2, \dots, w_n\})$
- 7: **end while**

protocol can be performed between the corresponding workers of the same index in active and passive parties. Then, the PSI protocol is realized within each pair of workers in parallel, i.e., the function PSI (Line 1) in the loop (Lines 3 - 5). Finally, the intersection between A and P is obtained as the union of the intersection results of each pair of identification buckets within the parameter servers in the active and passive parties.

The training process of the vertical FL is shown in Algorithm 3. First, the neural network (model) is randomly initialized (Line 1). Second, when the stop condition, e.g., required accuracy or a predefined number of iterations, is not met (Line 2), e.g., more iterations should be executed, the model continues to be updated. Then, each pair of workers in the active and passive parties calculates the weights of the model (Lines 3 and 4), which can be realized in parallel. Afterward, the workers send the weights to the parameter servers in each party, and the parameter servers update the model using a model aggregation method, such as BSP⁶². During the training process, each worker computes a local gradient on its given data, and the server aggregate these gradients to update the global parameters in each iteration. Workers pull the updated global parameter from the server and start the next round of iteration. The parameter server executes a great number of gradient exchanges to update the parameter until the model converges. The details of distributed training at each worker of the active or passive party are explained in Algorithms 4 and 5, respectively.

The training process of the distributed vertical FL in the active party is illustrated in Algorithms 4. First, the global weights are pulled from the parameter server in the active party (Line 1). Then, forward propagation is carried out with the bottom model and the intermediate result is generated (Line 2). Afterward, after receiving the result from the worker in the passive party (Line 4), and encrypting the intermediate result using the paillier cryptosystem as explained in Section 3.3 (Line 3), the

Algorithm 4 Distributed Training in the Active Party**Input:**

- PW_i : Worker i in the passive party;
 X_i : the features of dataset i in AW_i ;
 Y_i : the labels of dataset i in AW_i
- 1: $aw_{global} \leftarrow pull()$
 - 2: $activeResult \leftarrow forwardPropagation(aw_{global}, X_i, activeBottomModel)$
 - 3: $securedActiveResult \leftarrow HE(activeResult)$
 - 4: $passiveResult \leftarrow receiveResult(PW_i)$
 - 5: $result \leftarrow forwardPropagation(aw_{global}, securedActiveResult, passiveResult, activeTopModel)$
 - 6: $activeTopLoss, activeTopGradient \leftarrow backPropagation(result, activeTopModel, aw_{global}, Y_i)$
 - 7: $sendLoss(activeTopLoss, PW_i)$
 - 8: $activeBottomGradient \leftarrow backPropagation(activeTopLoss, activeBottomModel, aw_{global})$
 - 9: $updatedWeights \leftarrow gradientDescent(activeTopGradient, activeBottomGradient, ag_{global})$
 - 10: $sendWeightsToActivePS(updatedWeights)$

Algorithm 5 Distributed Training in the Passive Party**Input:**

- AW_i : Worker i in the active party;
 X'_i : the features of dataset i in PW_i
- 1: $pw_{global} \leftarrow pull()$
 - 2: $activeResult \leftarrow forwardPropagation(pw_{global}, X'_i, passiveBottomModel)$
 - 3: $securedActiveResult \leftarrow HE(activeResult)$
 - 4: $sendResult(AW_i, securedActiveResult)$
 - 5: $pg \leftarrow receiveLoss(AW_i)$
 - 6: $passiveBottomGradient \leftarrow backPropagation(pg, passiveBottomModel, pw_{global})$
 - 7: $updatedWeights \leftarrow gradientDescent(passiveBottomGradient, pg_{global})$
 - 8: $sendWeightsToPassivePS(updatedWeights)$

forward propagation is carried out with the top model (Line 5) and the result of the whole model is produced. We exploit the encryption and decryption detailed in⁴³ for the interaction between any pair of active worker and passive worker. The loss and gradients can be calculated using the result and the labels in the active party with the top model (Line 6). The intermediate loss is sent to the worker in the passive party (Line 7). In addition, the gradient for the bottom model can be generated through the back propagation of the bottom model (Line 8). The model is updated with the gradient using gradient descent (Line 9). Finally, the updated weights are sent to the parameter server in the active party.

The training process of the distributed vertical FL in the passive party is shown in Algorithms 5. First, the global weights are pulled from the parameter server in the active party (Line 1). Then, forward propagation is carried out with the bottom model and the intermediate result is generated (Line 2). The result is encrypted using the paillier cryptosystem as explained in Section 3.3 (Line 3) before being sent to the counterparty worker in the active party (Line 4). When receiving the gradients from the worker in the active party (Line 4), afterward, after receiving the loss from the worker in the passive party (Line 5), backward propagation is carried out to calculate the gradients for the bottom model in the passive party (Line 6). The model is updated with the generated gradients (Line 7), and the updated weights are sent to the parameter server in the passive party.

As we exploit the hash function to split the data, the overhead of the data split is $\mathcal{O}(1)$. In addition, in Line 6 of Algorithm 3, the models should be aggregated for the model update, thus, the overhead to calculate the averaged model is $\mathcal{O}(n)$ with n representing the number of cores in each party. Finally, the overall complexity of DVFL is $\mathcal{O}(n)$.

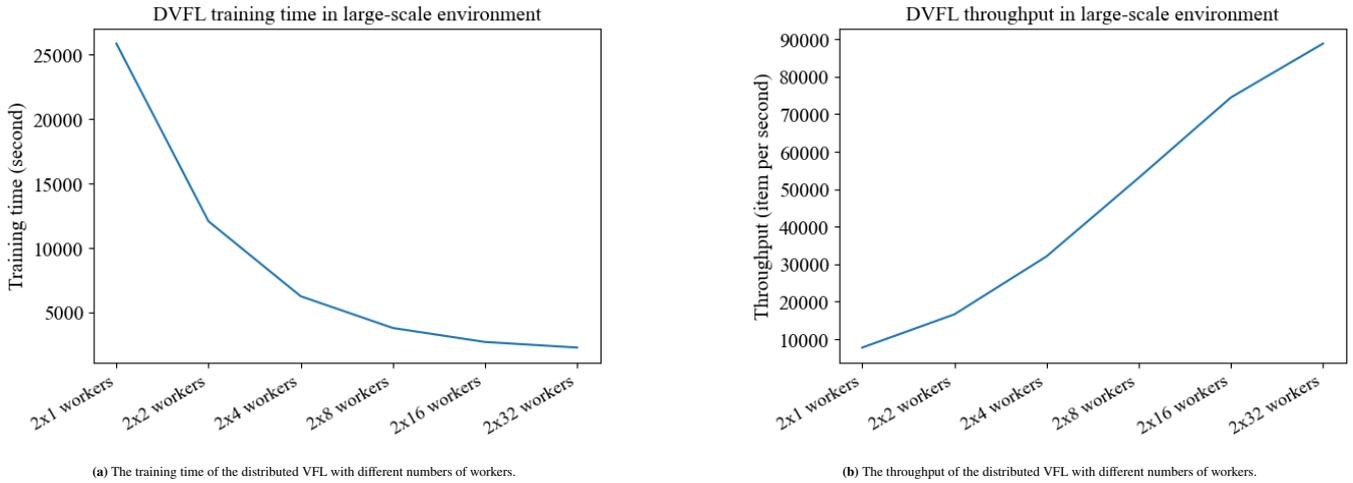


FIGURE 5 The performance of distributed VFL in the large-scale cluster environment. As the number of worker nodes increases, the execution time of the distributed VFL is reduced, and the throughput is improved.

5 | EXPERIMENT

In this section, we present the results of experiments in a large-scale cluster environment and the Baidu AI cloud environment. We illustrate our data source as well as data features and labels. We account for how much impact to run not only the training process of vertical FL but also the PSI protocol parallel with different number of workers. Then, we discuss the bottleneck of the experiment between CPU and memory usage under Baidu AI Cloud environment. Afterward, we compare the efficiency of DVFL and state-of-the-art platforms, i.e., FATE²³ and PyVertical²⁴, with different amounts of data and numbers of workers on the Baidu AI cloud environment. We fine-tuned the parameters for both FATE and DVFL.

We implemented a prototype based on the architecture of DVFL. The data communication in DVFL between any two of the parameter servers and workers is implemented by a Remote Procedure Call (RPC), i.e., gRPC⁶³, and the data communication is realized using Socket⁶⁴. In the following figures, we denote n parties with m workers at each party by $n * m$. We exploit the "a9a" dataset from the LIBSVM library⁶⁵ in the experimentation.

5.1 | DVFL

The large-scale cluster environment is given by Hygon C86 7185 thirty-two 6000-core processors, 128 GB memory, four 6000-core Distributed Control Unit (DCU), 64-bit CentOS 7.6 Linux with x86_64 architecture. To run DVFL, Party A and Party P each contains 10^6 rows of data. To evaluate the system performance, we record and compare the time consumption of two parties running DVFL under 1, 2, 4, 8, 16, 32 workers respectively. As shown in Figure 5, with the number of workers increase, the overall runtime drops while the total data processed per second increases. The execution time given by non-distributed system (1-worker per party) is 25865 seconds, while the execution time given by 32-worker per party is 2252 seconds. And the data processed per second is increased from 7732 rows (1-worker) to 88810 (32-worker). The assumption of the viability of DVFL is confirmed by noticing the tremendous execution time drop and the immense data processing capacity by the multi-worker system.

We also observed the training time of distributed vertical FL decreases while worker nodes increase. Because of the inevitable communication cost, the throughput per worker of 1-worker-system is higher than a 32-worker-system. The training time dropped significantly when the number of workers increases from one to two (25865 seconds to 12056 second). With the workers continually increasing, the training time decreasing becomes inert. When the number of workers increases, the overall throughput of the distributed vertical FL is also augmented. However, because of the communication overhead, the increment becomes less significant when the number of workers becomes big enough.

Additionally, we examined the execution time of the distributed PSI. In this experiment, we put 5×10^8 rows of data in Party A and 2×10^7 rows of data in Party P in the large-scale cluster environment. As shown in Figure 6, we obtain similar tendency

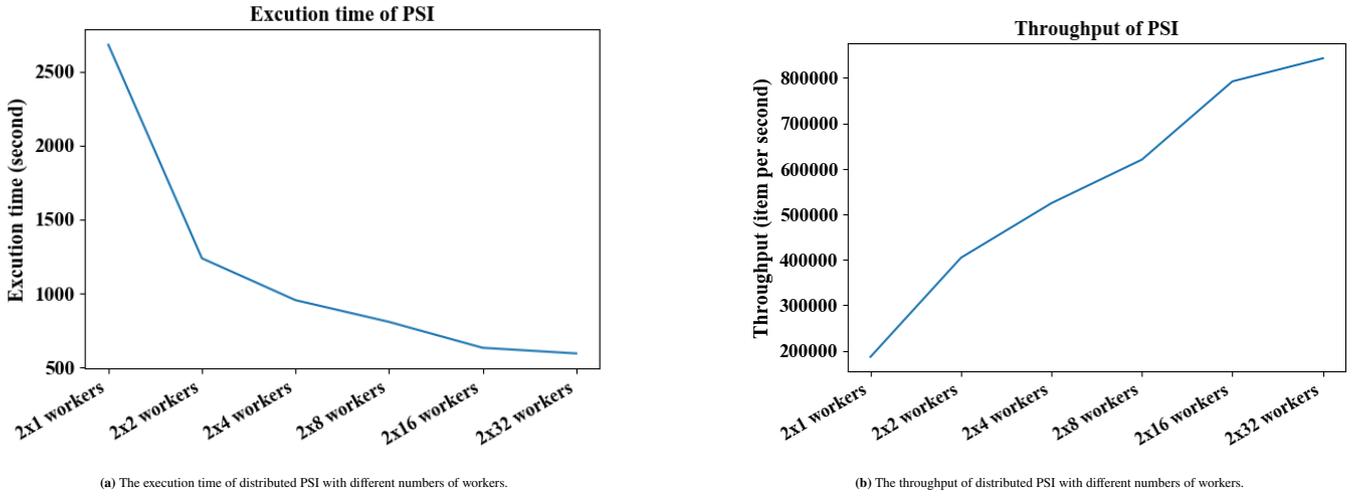


FIGURE 6 The performance of Distributed PSI in the large-scale cluster environment. As the number of worker nodes increases, the execution time of the distributed PSI is reduced, while the throughput is improved

	workers per party	1 worker	2 worker	4 worker	8 worker
Machine A	peak CPU spike (%)	98.91	98.59	97.09	93.48
	memory usage (G)	61.95	51.3	48	45.37
	Throughput	1105	1505	1101	772
Machine B	peak CPU spike (%)	60.39	39.48	27.91	15.98
	memory usage (G)	18.78	16.82	15.83	14.82
	Throughput	809	324	248	175
Machine C	peak CPU spike (%)	Null	98.65	97.81	96.74
	memory usage (G)	Null	23.29	18	15.14
	Throughput	Null	1843	1828	1921
Machine D	peak CPU spike (%)	Null	34.8	18.37	17.03
	memory usage (G)	Null	12.15	11.53	10.5
	Throughput	Null	400	262	165

TABLE 1 Hardware usage of DVFL

in terms of the execution time and the throughput as that of DVFL. The execution time decreases from 2680 seconds to 593 seconds while the throughput increases from 186567 items per second to 843170 items per second.

We carry out another experiment to verify the performance of DVFL on the Baidu AI Cloud environment. In this experiment, Party A holds 4×10^8 rows of data and Party P holds 2×10^7 rows of data. The environment is given by four 32-core, 64 GB memory, and 160 GB storage machines in version 7.4 CentOS Linux. Figure 7 shows the execution time of 1, 2, 4, 8 workers per party in DVFL on above datasets and environment.

Besides drawing a similar conclusion as the previous experiment in the large-scale cluster, we also monitor the peak CPU spike, memory usage, and throughput (TPS) as shown in Table 1. In 1-worker per party case, Machine A handles Party A and Machine B handles Party P. In multi-worker cases, Machines A and B are responsible for Party A, while Machines C and D are responsible for Party P. We notice the CPU is overloaded when handling the magnitude of 10^8 dataset in 1-worker system. Even the number of workers increases, the peak CPU spikes decrease lethargically. However, the memory usage is also close to full-filled to handle Party A in the 1-worker system (61.95 G out of 64 G). But the memory usage is effectively reduced while the number of workers is increased. The throughput drops as expected when the overall I/O remains, excepting few outliers. Hence, the limitation of CPU in a cloud environment could be a common bottleneck of DVFL when dealing with big data.

In order to analyze the overhead of Paillier HE, we carry out an experimentation to compare the training time with vanilla vertical FL and that with Paillier HE. We set the number of rounds to 10, the learning rate to 0.05 and the batch size to 16. We

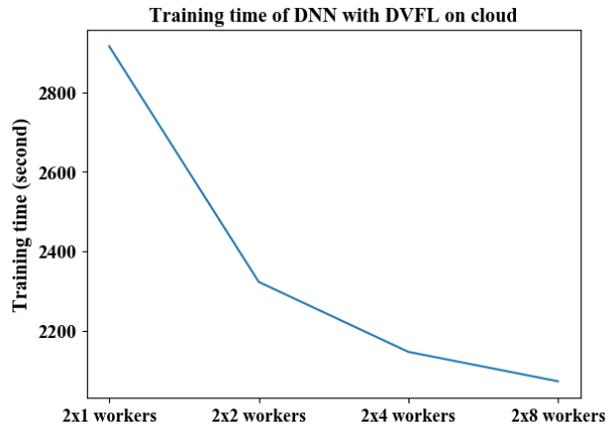


FIGURE 7 The training time of Deep Neural Networks (DNN) with DVFL and different numbers of workers on Baidu AI Cloud

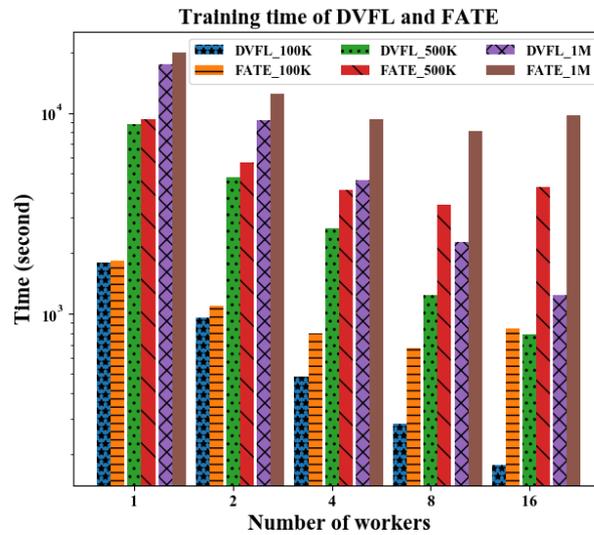


FIGURE 8 The training time of DVFL and FATE with a single server at each party.

vary the length of the public key in Paillier HE. As shown in Table 2, the overhead of Paillier HE cannot be ignored. When we utilize the public key of 128 bits, the training is 8.9 times longer while the inference time is slightly longer (less than 1.4%). When the length becomes 1024, the overhead can be much more significant (213 times longer). In practice, we exploit the public key of 128 bits to balance the security and the efficiency.

Type	Vanilla	HE (128)	HE (1024)
Training	89	878	19021
Inference	72	73	74

TABLE 2 Execution time for training and inference. The time unit is second. “128” and “1024” represent the length of public key.

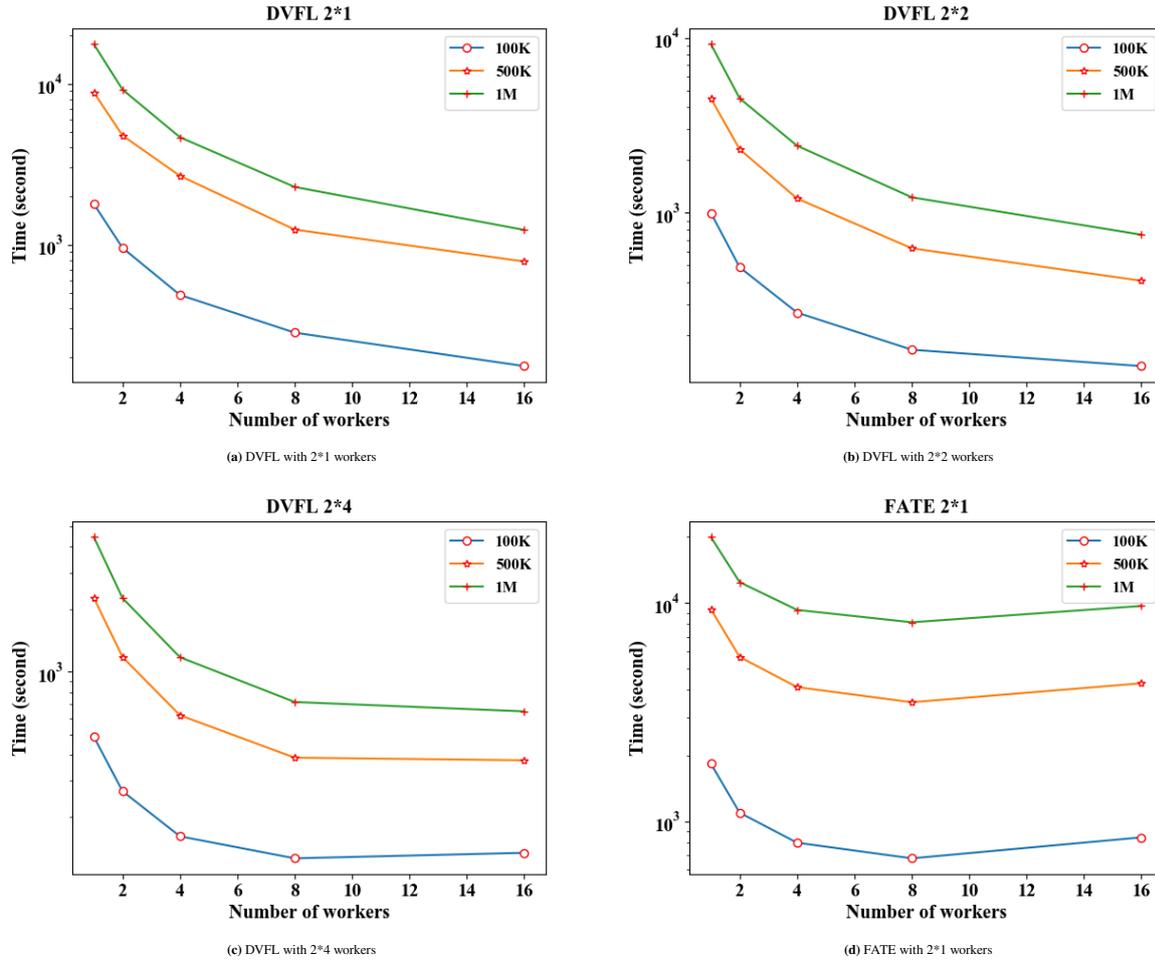


FIGURE 9 The execution time of DVFL and FATE with different numbers of workers.

5.2 | Comparison of DVFL and Baseline Frameworks

In order to show the efficiency and scalability of DVFL, we compare the performance of DVFL with baseline frameworks, i.e., FATE and PyVertical. The major difference between DVFL and baseline frameworks is the implementation and the communication strategy. In DVFL, we exploit a peer to peer method. FATE utilizes a centralized strategy and only the parameter server can communicate with each other between the two parties. In addition, PyVertical does not provide support for multiple cores or workers for vertical federated learning.

The experiment is given by 128 GB memory, 16-core, 64-bit CentOS 8.2 Linux with x86_64 architecture. To run DVFL and FATE, Party A and Party P each contains 50 thousand, 250 thousand and 500 thousand rows of data. To evaluate the performance, we record and compare the execution time of two parties running DVFL and FATE, while each party has 1, 2, 4 servers and each server has 1, 2, 4, 8, 16 virtual nodes respectively.

As shown in Figure 9, every time when the number of workers is doubled, the execution time of DVFL is reduced to nearly half of the original execution. Figure 9a shows the execution with one worker in each party with DVFL, Figure 9b shows the execution with two workers in each party with DVFL and Figure 9c shows the execution with four worker in each party with DVFL. When the number of workers increases, the overall execution time is significantly reduced. However, as shown in Figure 9c, the execution time decreases slowly when number of workers increases from 8 to 16, the execution time is not reduced due to the overhead of communication and synchronization.

Figure 9d shows the execution time of FATE with a server in each party. When the number of workers increases, the execution time is reduced when the number of workers is smaller or equal to 8. When the number of workers increases from 8 to 16, the execution time is augmented as well. This is because the bottleneck of the centralized parameter servers that realize the

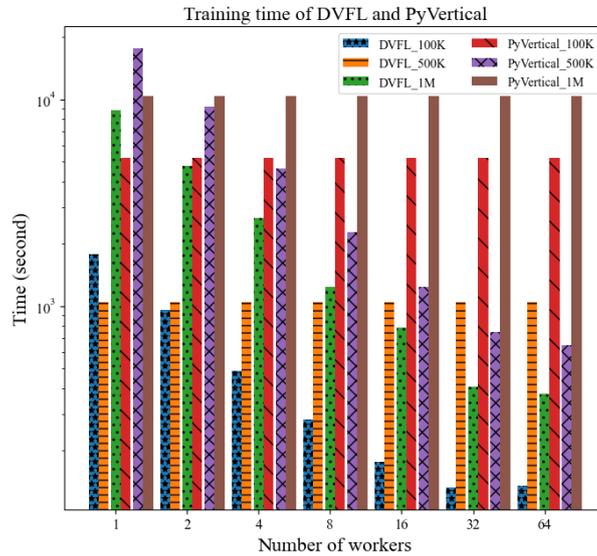


FIGURE 10 The training time of DVFL and PyVertical with multiple cores or workers at each party.

communication between two parties. In addition, to the best of our knowledge, FATE cannot support the execution with multiple servers at each party.

The comparison of DVFL and FATE is shown in Figure 8, when the amount of data and the number of workers are the same, the execution time of FATE is significantly longer than that of DVFL (up to 6.8 times longer). In addition, the difference becomes more significant with more data to process. In addition, the execution of DVFL can be up to 11.6 times faster than the best performance of FATE with the consideration of multiple servers in each party.

As shown in Figure 10, when the amount of data is the same, the execution of PyVertical²⁴ is shorter (up to 41.4%) than DVFL. Please note that this difference is brought by the implementation of Paillier HE in DVFL while PyVertical only exploits DP. Paillier HE can well protect the security of the data while DP is much faster than Paillier HE in practice. However, when exploiting multiple cores at both party, DVFL can significantly outperform PyVertical (up to 15.1 times faster) when the data is partitioned and processed in parallel.

6 | CONCLUSION AND PERSPECTIVE

In this paper, we propose a Distributed Vertical Federated Learning (DVFL) approach, which exploit distributed architecture for the execution of PSI and vertical federated learning. We exploit a peer-to-peer communication strategy, which corresponds to efficient distributed execution. We carried out extensive experiments to show the efficiency and scalability of DVFL with different scales of data, diverse numbers of servers in two environment, i.e., a large-scale cluster environment and a cloud environment. We examined the performance with one party holding the order of 10^8 and another holding the order of 10^7 datasets in both the large-scale cluster and the cloud environments. We demonstrated the acceleration of multiple servers with DVFL against non-distributed deep VFL in two parties scenario. In addition, we monitored the hardware usage and evaluated the possible performance bottleneck of DVFL. Furthermore, we compare the execution time of DVFL with state-of-the-art frameworks, i.e., FATE and PyVertical. The results shown that the advantage of DVFL can be up to 6.8 faster with single server at each party and 15.1 times faster with multiple servers.

In the future, we plan to extend DVFL to multiple parties and meanwhile to improve its performance. In addition, the interpretability^{66,67} is also of much importance for federated learning, we intend to enable interpretability within DVFL in the future.

References

1. Liu J, Huang J, Zhou Y, et al. From distributed machine learning to federated learning: a survey. *Knowledge and Information Systems* 2022; 64(4): 885-917.
2. Yang Q, Liu Y, Chen T, Tong Y. Federated machine learning: Concept and applications. *ACM Trans. on Intelligent Systems and Technology (TIST)* 2019; 10(2): 1-19.
3. Standing Committee of the National People's Congress . Cybersecurity Law of the People's Republic of China. <https://www.newamerica.org/cybersecurity-initiative/digichina/blog/translation-cybersecurity-law-peoples-republic-china/>; . Online; accessed 22/02/2021.
4. Official Journal of the European Union . General data protection regulation. <https://eur-lex.europa.eu/legal-content/EN/TXT/PDF/?uri=CELEX:32016R0679>; . Online; accessed 12/02/2021.
5. Chik WB. The Singapore Personal Data Protection Act and an assessment of future trends in data privacy reform. *Computer Law & Security Review* 2013; 29(5): 554–575.
6. California Consumer Privacy Act Home Page. <https://www.caprivacy.org/>; . Online; accessed 14/02/2021.
7. Gaff BM, Sussman HE, Geetter J. Privacy and Big Data. *Computer* 2014; 47(6): 7-9.
8. McMahan B, Moore E, Ramage D, Hampson S, Arcas yBA. Communication-efficient learning of deep networks from decentralized data. In: *Int. Conf. on Artificial Intelligence and Statistics (AISTATS)*. ; 2017: 1273-1282.
9. Hardy S, Henecka W, Ivey-Law H, et al. Private federated learning on vertically partitioned data via entity resolution and additively homomorphic encryption. *arXiv preprint arXiv:1711.10677* 2017.
10. Nock R, Hardy S, Henecka W, et al. Entity resolution and federated learning get a federated resolution. *arXiv preprint arXiv:1803.04035* 2018.
11. Liu Y, Huang A, Luo Y, et al. Fedvision: An online visual object detection platform powered by federated learning. In: . 34. AAAI Conf. on Artificial Intelligence. ; 2020: 13172-13179.
12. Sabt M, Achemlal M, Bouabdallah A. Trusted Execution Environment: What It is, and What It is Not. In: . 1. IEEE Int. Conf. on Trust, Security and Privacy in Computing and Communications. ; 2015: 57-64.
13. Armknecht F, Boyd C, Carr C, et al. A Guide to Fully Homomorphic Encryption. Cryptology ePrint Archive, Report 2015/1192; 2015. <https://eprint.iacr.org/2015/1192>.
14. Li B, Micciancio D. On the security of homomorphic encryption on approximate numbers. In: *Annual Int. Conf. on the Theory and Applications of Cryptographic Techniques*. ; 2021: 648–677.
15. Zhang C, Li S, Xia J, Wang W, Yan F, Liu Y. BatchCrypt: Efficient Homomorphic Encryption for Cross-Silo Federated Learning. In: ; 2020: 493–506.
16. Pinkas B, Schneider T, Zohner M. Scalable private set intersection based on OT extension. *ACM Trans. on Privacy and Security (TOPS)* 2018; 21(2): 1–35.
17. Chen H, Laine K, Rindal P. Fast private set intersection from homomorphic encryption. In: *ACM SIGSAC Conf. on Computer and Communications Security*. ; 2017: 1243-1255.
18. Gu B, Dang Z, Li X, Huang H. Federated doubly stochastic kernel learning for vertically partitioned data. In: *ACM SIGKDD Int. Conf. on Knowledge Discovery & Data Mining*. ; 2020: 2483–2493.
19. Feng S, Yu H. Multi-participant multi-class vertical federated learning. *arXiv preprint arXiv:2001.11154* 2020.
20. Chen T, Jin X, Sun Y, Yin W. Vaff: a method of vertical asynchronous federated learning. *arXiv preprint arXiv:2007.06081* 2020.

21. Guo Y, Polychroniadou A, Shi E, Byrd D, Balch T. MicroFedML: Privacy Preserving Federated Learning for Small Weights. *Cryptology ePrint Archive* 2022.
22. Kairouz P, McMahan HB, Avent B, et al. Advances and open problems in federated learning. *Foundations and Trends in Machine Learning* 2021; 14(1-2): 1-210.
23. WeBank . Federated AI Technology Enabler (FATE). <https://github.com/FederatedAI/FATE>; . Online; accessed 16/02/2021.
24. Romanini D, Hall AJ, Papadopoulos P, et al. Pyvertical: A vertical federated learning framework for multi-headed splitnn. *arXiv preprint arXiv:2104.00489* 2021.
25. Liu J, Mo L, Yang S, et al. Data Placement for Multi-Tenant Data Federation on the Cloud. *IEEE Transactions on Cloud Computing* 2021.
26. De Oliveira DC, Liu J, Pacitti E. Data-intensive workflow management: for clouds and data-intensive and scalable computing environments. *Synthesis Lectures on Data Management* 2019; 14(4): 1–179.
27. Liu J, Dong D, Wang X, et al. Large-scale knowledge distillation with elastic heterogeneous computing resources. *Concurrency and Computation: Practice and Experience* 2022; e7272.
28. Ahmed U, Lin JCW, Srivastava G. Hyper-Graph Attention Based Federated Learning Method For Mental Health Detection. *IEEE Journal of Biomedical and Health Informatics* 2022.
29. Ahmed U, Lin JCW, Srivastava G. 5G-Empowered Drone Networks in Federated and Deep Reinforcement Learning Environments. *IEEE Communications Standards Magazine* 2021; 5(4): 55-61.
30. Ahmed U, Srivastava G, Lin JCW. A federated learning approach to frequent itemset mining in cyber-physical systems. *Journal of Network and Systems Management* 2021; 29(4): 1–17.
31. Gaurav A, Psannis K, Peraković D. Security of cloud-based medical internet of things (miots): a survey. *Int. Journal of Software Science and Computational Intelligence (IJSSCI)* 2022; 14(1): 1-16.
32. Rajabi E, Greller W. Exposing social data as linked data in education. *Int. Journal on Semantic Web and Information Systems (IJSWIS)* 2019; 15(2): 92-106.
33. Ahmed U, Srivastava G, Lin JCW. Reliable customer analysis using federated learning and exploring deep-attention edge intelligence. *Future Generation Computer Systems* 2022; 127: 70–79.
34. Sejdiu B, Ismaili F, Ahmedi L. Integration of semantics into sensor data for the IoT: a systematic literature review. *Int. Journal on Semantic Web and Information Systems (IJSWIS)* 2020; 16(4): 1-25.
35. Wang Z, Yang Y, Liu Y, Liu X, Gupta BB, Ma J. Cloud-based federated boosting for mobile crowdsensing. *arXiv preprint arXiv:2005.05304* 2020.
36. Zhou C, Liu J, Jia J, et al. Efficient Device Scheduling with Multi-Job Federated Learning. *AAAI Conf. on Artificial Intelligence* 2022; 36(9): 9971-9979.
37. Liu J, Jia J, Ma B, et al. Multi-Job Intelligent Scheduling With Cross-Device Federated Learning. *IEEE Transactions on Parallel and Distributed Systems (TPDS)* 2022; 34(2): 535-551.
38. Zhang H, Liu J, Jia J, Zhou Y, Dai H. FedDUAP: Federated Learning with Dynamic Update and Adaptive Pruning Using Shared Data on the Server. In: *Int. Joint Conf. on Artificial Intelligence (IJCAI)*. ; 2022: 1-7. To appear.
39. Che T, Zhang Z, Zhou Y, et al. Federated Fingerprint Learning with Heterogeneous Architectures. In: *IEEE Int. Conf. on Data Mining (ICDM)*. ; 2022: 31–40.
40. Jin J, Ren J, Zhou Y, Lyu L, Liu J, Dou D. Accelerated federated learning with decoupled adaptive optimization. In: *Int. Conf. on Machine Learning (ICML)*. ; 2022: 10298–10322.

41. Li G, Hu Y, Zhang M, et al. FedHiSyn: A Hierarchical Synchronous Federated Learning Framework for Resource and Data Heterogeneity. In: *Int. Conf. on Parallel Processing (ICPP)*. ; 2022: 1-10. To appear.
42. Stergiou CL, Psannis KE, Gupta BB. InFeMo: flexible big data management through a federated cloud system. *ACM Transactions on Internet Technology (TOIT)* 2021; 22(2): 1-22.
43. Zhang Q, Wang C, Wu H, Xin C, Phuong TV. GELU-Net: A Globally Encrypted, Locally Unencrypted Deep Neural Network for Privacy-Preserved Learning.. In: *Int. Joint Conf. on Artificial Intelligence (IJCAI)*. ; 2018: 3933–3939.
44. Dean J, Corrado GS, Monga R, et al. Large scale distributed deep networks. *Advances in Neural Information Processing Systems (NeurIPS)* 2012: 1232-1240.
45. Low Y, Gonzalez JE, Kyrola A, Bickson D, Guestrin CE, Hellerstein J. Graphlab: A new framework for parallel machine learning. *arXiv preprint arXiv:1408.2041* 2014.
46. Xing EP, Ho Q, Dai W, et al. Petuum: A new platform for distributed machine learning on big data. *IEEE Trans. on Big Data* 2015; 1(2): 49–67.
47. Murray DG, McSherry F, Isaacs R, Isard M, Barham P, Abadi M. Naiad: a timely dataflow system. In: *ACM Symposium on Operating Systems Principles (SOSP)*. ; 2013: 439–455.
48. Kraska T, Talwalkar A, Duchi JC, Griffith R, Franklin MJ, Jordan MI. MLbase: A Distributed Machine-learning System. In: . 1. *Conf. on Innovative Data Systems Research (CIDR)*. ; 2013: 2–1.
49. Liu J, Wu Z, Yu D, et al. Heterps: Distributed deep learning with reinforcement learning based scheduling in heterogeneous environments. *arXiv preprint arXiv:2111.10635* 2021.
50. Li M, Zhou L, Yang Z, et al. Parameter server for distributed machine learning. In: . 6. *Advances in Neural Information Processing Systems (NeurIPS) Workshop*. ; 2013: 2.
51. Li M, Andersen DG, Smola AJ, Yu K. Communication efficient distributed machine learning with the parameter server. *Advances in Neural Information Processing Systems (NeurIPS)* 2014; 27: 19–27.
52. Freedman MJ, Nissim K, Pinkas B. Efficient Private Matching and Set Intersection. In: Cachin C, Camenisch JL., eds. *Advances in Cryptology - EUROCRYPT* Advances in Cryptology - EUROCRYPT. ; 2004: 1–19.
53. Lu L, Ding N. Multi-Party Private Set Intersection in Vertical Federated Learning. In: *IEEE Int. Conf. on Trust, Security and Privacy in Computing and Communications (TrustCom)*. ; 2020: 707-714.
54. Dong C, Chen L, Wen Z. When private set intersection meets big data: an efficient and scalable protocol. In: *ACM SIGSAC Conf. on Computer & communications security*. ; 2013: 789–800.
55. Gentry C. Fully homomorphic encryption using ideal lattices. In: *Annual ACM symposium on Theory of computing*. ; 2009: 169-178.
56. Paillier P. Public-key cryptosystems based on composite degree residuosity classes. In: *Int. Conf. on the theory and applications of cryptographic techniques*. ; 1999: 223–238.
57. Broder A, Mitzenmacher M. Network applications of bloom filters: A survey. *Internet mathematics* 2004; 1(4): 485–509.
58. Naor M, Pinkas B. Efficient oblivious transfer protocols.. In: . 1. *Symposium on Discrete Algorithms (SODA)*. ; 2001: 448–457.
59. Ishai Y, Prabhakaran M, Sahai A. Founding cryptography on oblivious transfer-efficiently. In: *Annual Int. cryptology Conf.* ; 2008: 572-591.
60. Smola A, Narayanamurthy S. An architecture for parallel topic models. *Very Large Data Base (VLDB) Endowment* 2010; 3(1-2): 703–710.

61. Shasha D, Wang TL. Optimizing equijoin queries in distributed databases where relations are hash partitioned. *ACM Transactions on Database Systems (TODS)* 1991; 16(2): 279-308.
62. Zhao X, Papagelis M, An A, Chen BX, Liu J, Hu Y. Elastic bulk synchronous parallel model for distributed deep learning. In: *IEEE Int. Conf. on Data Mining (ICDM)*. ; 2019: 1504-1509.
63. Open Source High Performance Remote Procedure Call. <https://grpc.io/>; . Online; accessed 7/05/2021.
64. Kalita L. Socket programming. *Int. Journal of Computer Science and Information Technologies* 2014; 5(3): 4802-4807.
65. Chang CC, Lin CJ. LIBSVM: a library for support vector machines. *ACM transactions on intelligent systems and technology (TIST)* 2011; 2(3): 1-27.
66. Li X, Xiong H, Li X, et al. Interpretable Deep Learning: Interpretation, Interpretability, Trustworthiness, and Beyond. *arXiv preprint arXiv:2103.10689* 2021.
67. Liu J, Pacitti E, Valduriez P, Mattoso M. A survey of data-intensive scientific workflow management. *Journal of Grid Computing* 2015; 13(4): 457–493.

