Towards Communication-efficient Vertical Federated Learning Training via Cache-enabled Local Updates

Fangcheng Fu*
Peking University
ccchengff@pku.edu.cn

Xupeng Miao*
Peking University
xupeng.miao@pku.edu.cn

Jiawei Jiang†
Wuhan University
jiawei.jiang@whu.edu.cn

Huanran Xue
Tencent Inc.
huanranxue@tencent.com

Bin Cui*‡
Peking University
bin.cui@pku.edu.cn

ABSTRACT

Vertical federated learning (VFL) is an emerging paradigm that allows different parties (e.g., organizations or enterprises) to collaboratively build machine learning models with privacy protection. In the training phase, VFL only exchanges the intermediate statistics, i.e., forward activations and backward derivatives, across parties to compute model gradients. Nevertheless, due to its geo-distributed nature, VFL training usually suffers from the low WAN bandwidth.

In this paper, we introduce CELU-VFL, a novel and efficient VFL training framework that exploits the local update technique to reduce the cross-party communication rounds. CELU-VFL caches the stale statistics and reuses them to estimate model gradients without exchanging the ad hoc statistics. Significant techniques are proposed to improve the convergence performance. First, to handle the stochastic variance problem, we propose a uniform sampling strategy to fairly choose the stale statistics for local updates. Second, to harness the errors brought by the staleness, we devise an instance weighting mechanism that measures the reliability of the estimated gradients. Theoretical analysis proves that CELU-VFL achieves a similar sub-linear convergence rate as vanilla VFL training but requires much fewer communication rounds. Empirical results on both public and real-world workloads validate that CELU-VFL can be up to six times faster than the existing works.

PVLDB Reference Format:
Fangcheng Fu, Xupeng Miao, Jiawei Jiang, Huanran Xue, and Bin Cui. Towards Communication-efficient Vertical Federated Learning Training via Cache-enabled Local Updates. PVLDB, 15(10): 2111 - 2120, 2022.
doi:10.14778/3547305.3547316

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://github.com/ccchengff/FDL/tree/main/playground/celu_vfl.

*School of Computer Science & Key Lab of High Confidence Software Technologies (MOE), Peking University
†School of Computer Science, Wuhan University
‡Institute of Computational Social Science, Peking University (Qingdao), China

Proceedings of the VLDB Endowment, Vol. 15, No. 10 ISSN 2150-8097.
doi:10.14778/3547305.3547316

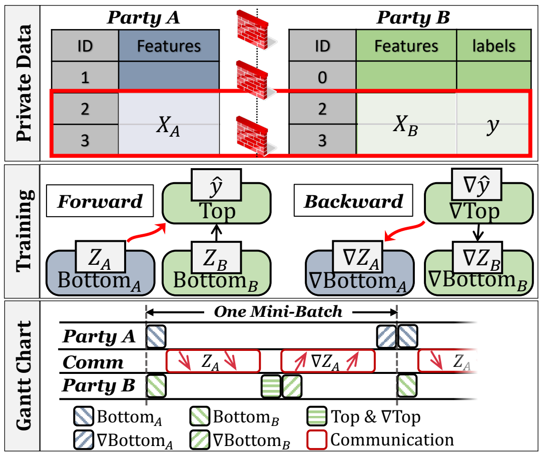


Figure 1: An illustration of VFL. The cross-party communication (indicated by red arrows) dominates the time cost of training.

1 INTRODUCTION

The past few years have witnessed the explosion of federated learning (FL) research and applications [27, 28, 30, 48], where two or more parties (participants) collaboratively train a machine learning (ML) model using their own private data. In particular, vertical FL (VFL) illustrates such a scenario where different parties hold disjoint features but overlap on a set of instances. As shown in Figure 1, the overlapping instances are regarded as a virtual dataset that is vertically partitioned. There are two types of parties: (i) one or more *Party $\mathcal{A}$*'s that merely provide features; and (ii) one *Party $\mathcal{B}$* that holds both features and the ground truth labels for the learning tasks. The goal of VFL is to fit the labels by the features of all parties, whilst keeping the private data (including both features and labels) of each party secret from the others. VFL is suitable for a variety of cross-enterprise collaborations. For instance, *Party $\mathcal{A}$* can be an Internet company that is able to generate rich and precise user profiling, whilst *Party $\mathcal{B}$* can be an e-commercial company that hopes to build a better recommender system[1]. As a result, VFL is gaining more popularity in both academia and industry [8, 17, 44, 49].

[1]In Practice, two or more *Party $\mathcal{A}$*'s would be possible. However, such a two-party setting is the most widely seen circumstance for cross-enterprise collaborations. Furthermore, our work can be generalized to two or more *Party $\mathcal{A}$*'s easily. Therefore, we only consider one *Party $\mathcal{A}$* in this work for simplicity.

Algorithm 1: VFL training with local updates (FedBCD [32]) for *Party $\mathcal{A}$*. Line 4-5 can be executed in background to overlap the communication and local updates.

```
1 foreach mini-batch $i$ do
2     Compute and send $Z_A^{(i)}$; Recv $\nabla Z_A^{(i)}$;
3     Update with $\nabla\theta_{Bot\_A} = \nabla Z_A^{(i)} \frac{\partial Z_A^{(i)}}{\partial \theta_{Bot\_A}}$;
4     for $j \leftarrow 1$ until $R$ do // local updates
5         Compute $Z_A^{(i,j)}$; Update with $\widetilde{\nabla}\theta_{Bot\_A} = \nabla Z_A^{(i)} \frac{\partial Z_A^{(i,j)}}{\partial \theta_{Bot\_A}}$;
```

(Communication Bottleneck of VFL) Denote y as the labels owned by *Party $\mathcal{B}$*, and X_A, X_B as the features of *Party $\mathcal{A}$* and *Party $\mathcal{B}$*, respectively, where the k-th rows $X_{A,k}, X_{B,k}$ together form the features of the k-th instances. As shown in Figure 1, VFL models typically follow a bottom-to-top architecture. First, each *Party $\mathcal{P}$* is associated with a bottom model that works as a feature extractor. It takes X_P as input and outputs $Z_P = \mathtt{Bottom}_P(X_P; \theta_{Bot\_P})$, where θ_{Bot_P} are the bottom model weights. Second, *Party $\mathcal{B}$* is further associated with a top model parameterized by θ_{Top} that makes predictions via $\hat{y} = \mathtt{Top}(Z_A, Z_B; \theta_{Top})$. The goal of VFL training is to find the model weights minimizing the empirical risk:

$$\underset{\{\theta_{Bot\_A}, \theta_{Bot\_B}, \theta_{Top}\}}{\arg\min} \quad L := \sum_k l(y_k, \hat{y}_k), \tag{1}$$

where l is the loss function (e.g., the logistic loss).

The most common avenue to solve such a supervised ML problem is to adopt the mini-batch stochastic gradient descent (SGD) algorithm or its variants. For each step, a batch of instances is sampled to compute the model gradients $\nabla\theta = \partial L / \partial \theta$ for model updates. However, since *Party $\mathcal{A}$* must not get access to the labels or the top model due to the privacy consideration, it cannot accomplish the backward propagation process alone. To overcome this problem, existing VFL frameworks adopt a two-phase propagation routine — the gradients for θ_{Bot_A} are calculated via $\nabla\theta_{Bot\_A} = \nabla Z_A \frac{\partial Z_A}{\partial \theta_{Bot\_A}}$, where $\nabla Z_A = \partial L / \partial Z_A$ are provided by *Party $\mathcal{B}$*.

As shown in the Gantt chart of Figure 1, in each step, two kinds of intermediate data are transmitted, i.e., forward activations Z_A and backward derivatives ∇Z_A. In practice, the communication cost is expensive due to the limited network bandwidth. Since the data centers of different parties are usually physically distributed, VFL tasks have to run in a geo-distributed manner where Wide Area Network (WAN) bandwidth is very low. For instance, as reported by Qian et al. [37] and verified by our empirical benchmark, the bandwidth between different data centers are usually smaller than 300 Mbps. Worse still, since most companies have network restrictions to avoid external attacks, the servers that carry out the VFL tasks are forbidden from connecting to WAN directly. Instead, messages are proxied by some gateway machines, leading to even slower communication. Eventually, in real-world VFL applications, we observe that over 90% of the total training time is spent on communication, leading to extremely low efficiency and a huge waste of computational resources. Consequently, how to tackle the communication bottleneck in VFL is a timely and important problem.

(Motivation) To address the communication bottleneck, an intuitive idea is to amortize the network cost by performing multiple local updates between two communication rounds. In fact, such a local update technique (a.k.a. local SGD) has been widely studied in conventional (non-federated) distributed ML and horizontal FL [22, 27, 28, 40]. However, applying this technique in VFL is non-trivial since each party cannot accomplish the forward or backward propagation individually due to the lack of Z_A or ∇Z_A. Thus, approximating Z_A or ∇Z_A with stale statistics is needed. Specifically, FedBCD [32] proposes to execute R updates for each mini-batch, which is shown in Algorithm 1. During the j-th update of the i-th mini-batch, instead of exchanging the ad hoc values, i.e., $\langle Z_A^{(i,j)}, \nabla Z_A^{(i,j)} \rangle$, *Party $\mathcal{A}$* treats $\nabla Z_A^{(i)}$ as an approximation and estimates the model gradients as $\widetilde{\nabla}\theta_{Bot\_A} = \nabla Z_A^{(i)} \frac{\partial Z_A^{(i,j)}}{\partial \theta_{Bot\_A}}$; symmetrically, *Party $\mathcal{B}$* feeds the stale values $Z_A^{(i)}$ and the ad hoc values $Z_B^{(i,j)}$ to the top model.

Nevertheless, there are two drawbacks of local updates in VFL: (1. variance problem) using the same stale mini-batch for multiple consecutive steps would enlarge the stochastic variance in model gradients (similar phenomenon can be observed in SGD algorithms without data shuffling [15, 35, 47]); (2. approximation errors) because approximation via stale statistics inevitably introduces errors to the model gradients, it would slow down the convergence or even lead to divergence if we directly update the models with the approximated gradients. According to the above analysis, there exists a discrepancy in the state of VFL with local updates — doing more local steps brings a better improvement on the *system efficiency*, but can easily harm the *statistical efficiency*.

(Summary of Contributions) Motivated by this, our work proposes a novel VFL training framework that improves both system efficiency and statistical efficiency at the same time. To summarize:

- We develop CELU-VFL, an efficient VFL training framework that exploits a cache-enabled local update technique for better system efficiency. To reduce the communication cost, CELU-VFL introduces an abstraction of workset table that caches the forward activations and backward derivatives to enable the local update technique. CELU-VFL further overlaps the reads and writes of the workset table triggered by local computation and cross-party communication to improve resource utilization.
- To improve the statistical efficiency, CELU-VFL addresses the two drawbacks of local updates in VFL based on the workset table abstraction. (1) First, to tackle the variance problem, we introduce a uniform local sampling strategy — instead of repetitively using the same mini-batch, the cached statistics are fairly sampled from the workset table to execute the local updates. (2) Second, to mitigate the impact of approximation errors, we design a staleness-aware instance weighting mechanism. The essential idea is to measure the staleness using cosine similarities and control the contribution of each instance according to its staleness.
- Furthermore, we theoretically analyze the convergence property of CELU-VFL, which shows that our work achieves a similar sub-linear convergence rate as vanilla VFL training but requires much fewer communication rounds. The analysis also interprets why our uniform local sampling strategy and instance weighting mechanism improve the statistical efficiency.
- We empirically evaluate the performance of CELU-VFL through extensive experiments. First, we investigate the statistical efficiency of CELU-VFL along three dimensions, i.e., the number of

local updates, the strategy of local sampling, and the weighting mechanism of staleness. Second, end-to-end evaluation on both public datasets and real-world workloads demonstrates that CELU-VFL is able to outperform the existing works by 2.65-6.27×.

2 PRELIMINARIES

2.1 Vertical Federated Learning

Owing to the establishment of lawful regulation on privacy protection [1, 2, 41], how to train ML models with privacy preservation is becoming a hot topic. Notably, VFL has become a prevailing paradigm that unites the features from different parties for collaborative applications [8, 17, 18, 23, 29, 42, 44, 48, 49].

Privacy Preservation. Unlike conventional distributed ML, raw data are not collected together in VFL, and model weights or updates are not transferred between parties, either. Instead, only the intermediate statistics, i.e., forward activations and backward derivatives, are exchanged between parties to protect the data and models. Many existing works have theoretically analyzed the privacy guarantees of VFL under the honest-but-curious assumption [7, 29, 50, 51].

Data Management. Under the VFL setting, the data in both parties are assumed to be vertically aligned. In other words, the overlapping instances (e.g., instances 2 and 3 in Figure 1) have been extracted and aligned in the same ordering before training. This can be done by the private set intersection (PSI) technique [6, 12, 16]. Then, both parties can sample the mini-batches using the same random seed to ensure that each mini-batch is also aligned.

The Cross-party Communication Bottleneck. As introduced in Section 1, since different parties are usually geo-distributed and the WAN bandwidth is limited in practice, the cross-party communication cost inevitably becomes the major bottleneck in VFL. Take a real case as an example, where the output dimensionality of Z_A is 256 and the batch size is 4096. The size of Z_A or ∇Z_A will be 4MB using single-precision floating-point numbers. Eventually, each communication round (including two transmissions) will take 213 milliseconds given a 300Mbps network bandwidth. In contrast, the computation time is much shorter thanks to the explosive growth of modern accelerators (e.g., GPUs) in the past decade. As a result, how to reduce the communication cost and improve the utilization of computational resources has become an urgent issue in VFL.

2.2 Local Update

To address the communication deficiency in distributed training, umpteenth works are developed to reduce the communication rounds by conducting local updates [22, 27, 28, 31, 40]. However, they mainly focus on distributed ML or horizontal FL (HFL), where the datasets are horizontally partitioned among different machines (training workers or clients). Under such a setting, the label and features for the same instance are co-located so that each machine can accomplish the forward and backward propagation alone.

However, in VFL, parties need to fetch the forward activations or backward derivatives in each step. To apply the local update technique in VFL, we have to approximate the intermediate statistics rather than fetching from the other party. Since ML models are trained iteratively, there is a common sense that the models do not change significantly in a few iterations and are robust to certain noises. Thus, Liu et al. [32] propose to approximate the ad hoc statistics with the stale ones. Specifically, they reuse the statistics of the last batch for multiple iterations and update the models with the estimated gradients directly. However, as discussed in Section 1, the convergence would be harmed due to the enlarged stochastic variance and the approximation errors in local updates.

3 CELU-VFL

In this section, we first introduce CELU-VFL, a novel VFL framework that enables local updates via cached statistics. Then, we propose two techniques to address the two drawbacks of local updates in VFL accordingly. We first present the frequently used notations:

- B: size of mini-batches;
- R: maximum number of updates for each mini-batch;
- W: size of the workset table (number of cached mini-batches);
- ξ: the threshold in our instance weighting mechanism;
- $Z_A^{(i)}, \nabla Z_A^{(i)}$: the forward activations and backward derivatives of *Party* $\mathcal{A}$ of the i-th batch;
- $Z_A^{(i,j)}, \nabla Z_A^{(i,j)}$: the forward activations and backward derivatives of *Party* $\mathcal{A}$ of the i-th batch at the j-th updates.

3.1 Overview

Figure 2 illustrates the overview of CELU-VFL. There are three major components, as introduced below.

The workset table is in charge of the maintenance of the cached, stale statistics. It records two "clocks" for each cached batch: (i) the timestamp when this batch is inserted, and (ii) the current number of updates done by this batch. The capacity of the table is W, i.e., during the insertion at time i, we discard cached batches inserted before time $i - W + 1$ to control the maximum staleness. Whenever a cached batch is sampled for one local update, we increment the second clock correspondingly. Cached batches that reach the maximum number of updates, i.e., R, will be dropped as well.

The communication worker is responsible for the message transmission between parties. In the beginning, both parties agree on the ordering of mini-batches. For each batch, the communication workers execute the conventional forward and backward propagation to exchange Z_A and ∇Z_A. Then, both Z_A and ∇Z_A are inserted into the workset table in order to execute local updates. The timestamp i is set as the current number of communication rounds, i.e., the current number of processed batches.

The local worker performs local updates via the cached statistics. For each local step, the local worker of each party chooses one cached entry $\langle i, Z_A^{(i)}, \nabla Z_A^{(i)}, j\rangle$ from the workset table, where i identifies the corresponding mini-batch and j denotes how many times it has been used. Unlike FedBCD that trains on the same mini-batch for multiple consecutive steps, we exploit a round-robin local sampling strategy to reduce the stochastic variance, which will be described in Section 3.2. In addition to executing the local propagation, we develop a mechanism that weights the instances to mitigate the impact brought by staleness. For instance, as depicted in Figure 2, *Party* $\mathcal{A}$ feeds the stale derivatives $\nabla Z_A^{(i)}$ to the backward propagation process and utilizes the stale activations $Z_A^{(i)}$ to measure the weights of instances. We will introduce the weighting mechanism in depth in Section 3.3. Furthermore, we let the two types of workers run concurrently to make full use of both computation and communication resources.

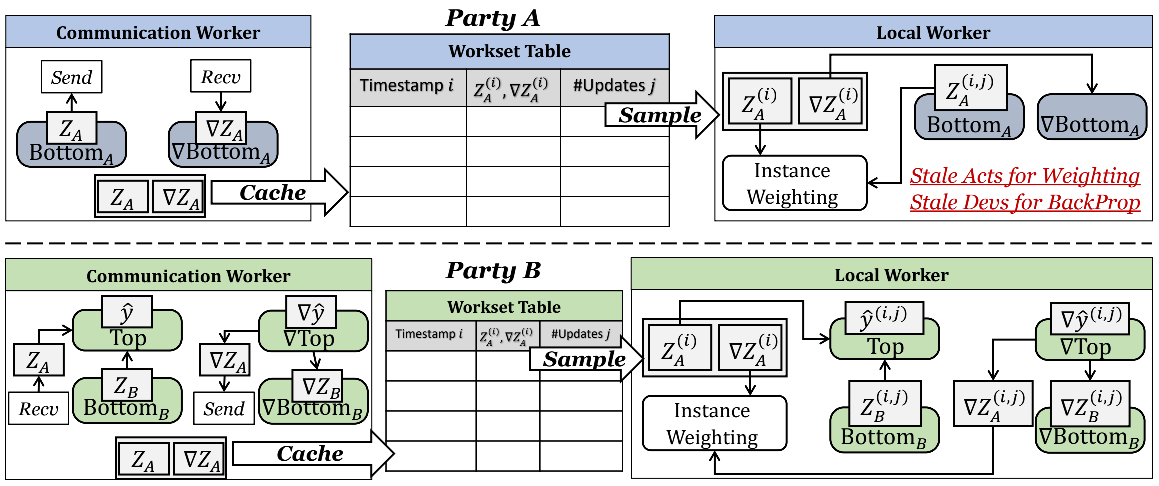


Figure 2: Overview of CELU-VFL.

3.2 Round-Robin Local Sampling

It is well known that stochastic optimization algorithms have inherent variance in the mini-batch gradients [5, 26, 38]. As shown in Figure 3, if each mini-batch is used for consecutive steps, the stochastic variance will be accumulated and hinder the convergence. Such stochastic variance can be mitigated if we apply the mini-batches uniformly, e.g., by alternating the mini-batches. Motivated by this, we introduce a workset table that stores $W > 1$ mini-batches to allow uniform local sampling. It is worthy to note that since FedBCD repetitively uses the same mini-batch for consecutive local updates, it is analogous to the simplest case of $W = 1$.

Nevertheless, we observe that simply letting $W > 1$ still cannot deal with the repetitive training issue. The reason is that the training workers are running too much faster than the communication workers. For instance, the top row of Figure 4 depicts a case where $W = R = 3$. Whenever a mini-batch is inserted into the workset, the training worker will immediately finish the three local updates before the next mini-batch is ready. As a result, the local sampling downgrades to the repetitive training pattern.

To solve this problem, we devise a simple but effective round-robin local sampling strategy, which enforces that each mini-batch in the workset cannot be sampled again in the next $W - 1$ steps. As depicted in the bottom row of Figure 4, with the round-robin strategy, the training worker samples the mini-batches one by one, guaranteeing the uniformity. Although bubbles are incurred in the first $W - 1$ communication rounds, they are negligible since W is far smaller than the total number of communication rounds.

Discussion. Readers might suspect that whether the convergence would be slowed down as the maximum staleness of CELU-VFL becomes $W \times R$, whilst that of FedBCD is only R. However, as we will analyze in Section 4.1 and evaluate in Section 5, although the round-robin strategy increases the maximum staleness, the fairness in local sampling is beneficial to convergence from two aspects. First, given the same R, training with $W > 1$ converges faster than $W = 1$. Second, CELU-VFL can support a larger R whilst FedBCD diverges, so the resource utilization can be improved as well.

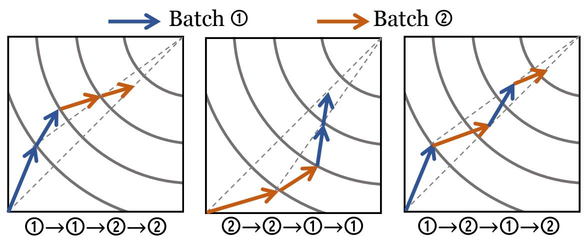
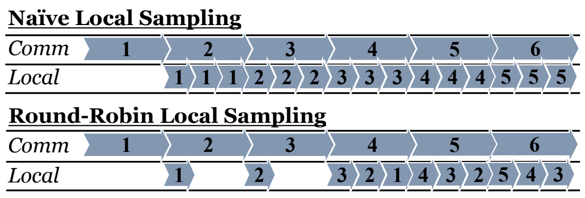


Figure 3: An illustration of the impact of mini-batches ordering. Both of the two mini-batches (indicated in blue and orange, respectively) are trained for two steps. Training in an alternating way (right-most) converges better than training in a repetitive way (left-most and middle).



Naïve Local Sampling
Comm 1 2 3 4 5 6
Local 1 1 1 2 2 2 3 3 3 4 4 4 5 5 5
Round-Robin Local Sampling
Comm 1 2 3 4 5 6
Local 1 2 3 2 1 4 3 2 5 4 3

Figure 4: Naïve vs. round-robin local sampling.

In addition, there are many alternatives to achieve uniform sampling, such as randomly drawing the instances inside the workset table. However, we treat each pre-determined mini-batch as a whole to be more implementation-friendly — since the mini-batches are inserted at different steps, it is easier to maintain the "clocks" of each instance. Furthermore, we can randomly shuffle the entire training dataset before training to ensure that the instances inside the workset table are also in random order.

3.3 Instance Weighting Mechanism

Undoubtedly, approximation with the stale statistics causes errors in the estimated model gradients. Although there have been various works discussing that gradient optimization algorithms are

Algorithm 2: Procedures of local update with our instance weighting mechanism. $\cos\xi$ is a pre-defined threshold and $\odot$ denotes element-wise multiplication.

```
1  Function InsWeight(ad hoc $V^{(i,j)}$, stale $V^{(i)}$):
2      $weights \leftarrow \cos(V^{(i,j)}, V^{(i)}, axis = 1)$;
3      $weights[weights < \cos\xi] \leftarrow 0$;
4      return $weights$
5  Function LocalUpdatePartyA($i, Z_A^{(i)}, \nabla Z_A^{(i)}, j$):
6      $Z_A^{(i,j)} \leftarrow$ ForwardProp($X_A^{(i)}$);
7      $ins\_weights \leftarrow$ InsWeight($Z_A^{(i,j)}, Z_A^{(i)}$);
8      BackwardProp($ins\_weights \odot \nabla Z_A^{(i)}$);
9  Function LocalUpdatePartyB($i, Z_A^{(i)}, \nabla Z_A^{(i)}, j$):
10     $\hat{y}^{(i,j)} \leftarrow$ ForwardProp($Z_A^{(i)}, X_B^{(i)}$);
11     $loss^{(i,j)} \leftarrow$ Loss($y^{(i)}, \hat{y}^{(i,j)}, axis = 1$);
12     $\nabla Z_A^{(i,j)} \leftarrow \partial loss^{(i,j)} / \partial Z_A^{(i)}$;
13     $ins\_weights \leftarrow$ InsWeight($\nabla Z_A^{(i,j)}, \nabla Z_A^{(i)}$);
14     BackwardProp($ins\_weights \odot loss^{(i,j)}$);
```

robust to certain noises in gradients, the model performance will be harmed inevitably when the staleness is large. To tackle this problem, we desiderate a fine-grained mechanism that measures the reliability of each cached instance according to its staleness.

As we will analyze in Section 4.1, the convergence speed is highly related to the angles between the approximated gradients calculated using stale statistics and the expected gradients calculated using ad hoc statistics. Therefore, it is a straightforward idea to leverage the cosine similarity to measure the staleness. Nevertheless, it is impossible to calculate the expected gradients for local updates since the ad hoc statistics are not transmitted between parties, making the cosine similarity unmeasurable. Consequently, we heuristically exploit the cosine similarity measurement to the forward activations and backward derivatives, rather than the model gradients.

To be specific, our instance weighting mechanism is shown in Algorithm 2. We measure the cosine similarities between the ad hoc statistics and the stale ones, and use the similarities as instance weights. For *Party $\mathcal{A}$*, since the ad hoc forward activations are computed for each local update, we measure the cosine similarities by $\cos(Z_A^{(i,j)}, Z_A^{(i)})$, where the cosine operation applies on every row for each instance individually. In symmetric, *Party $\mathcal{B}$* measures the cosine similarities via the backward derivatives[2], i.e., $\cos(\nabla Z_A^{(i,j)}, \nabla Z_A^{(i)})$. Furthermore, we introduce a lower bound $\cos\xi$ as the threshold, i.e., for instances whose similarities are smaller than $\cos\xi$, their weights will be set as zero. Finally, the backward propagation is executed in a weighted manner so that the model gradients will be computed in the weighted-averaged fashion.

Discussion. To help readers better understand the rationality of such a heuristic method, we use a fully connected layer $z_{out} = z_{in}\theta$ as an example, where $\theta \in \mathbb{R}^{d_{in} \times d_{out}}$ denotes the model weights and $z_{in} \in \mathbb{R}^{d_{in}}, z_{out} \in \mathbb{R}^{d_{out}}$ denote the inputs and outputs of one instance, respectively. During the backward propagation, the model gradients are computed via the outer product between forward activations and backward derivatives, i.e., $\nabla\theta = z_{in}^T \nabla z_{out}$. For *Party $\mathcal{A}$*, suppose the stale derivatives are denoted as $\widetilde{\nabla} z_{out}$, it is easy to know that $\cos(\nabla\theta, \widetilde{\nabla}\theta) = \cos(\nabla z_{out}, \widetilde{\nabla} z_{out})$[3], where $\widetilde{\nabla}\theta = z_{in}^T \widetilde{\nabla} z_{out}$. Similarly, we have the same property for *Party $\mathcal{B}$*. Although this is only valid for fully connected layers, it provides a hint that measuring the similarities via forward activations and backward derivatives is reasonable and achieves promising results.

4 ANALYSIS

4.1 Convergence Analysis

When applying local updates in VFL training, we are actually solving the problem defined in Equation 1 with approximated gradients. For simplicity, we re-write it as a more general (non-convex) empirical risk minimization (ERM) problem, i.e.,

$$\min_{\theta \in \mathbb{R}^d} f(\theta) = \tfrac{1}{N} \textstyle\sum_{k \in \mathcal{D}} f_k(\theta), \ \theta_{t+1} = \theta_t - \eta \tilde{g}_t,$$

where η is the step size and step t is denoted as the number of updates. It is worthy to note that since the models of different parties may not be updated for the same number of steps due to the local update technique, the learning task in CELU-VFL can be reckoned as solving two ERM problems separately. To analyze the problem, we denote three types of gradients:

- $\nabla f(\theta_t) = \frac{1}{N} \sum_{k \in \mathcal{D}} \nabla f_k(\theta_t)$ is the full-batch gradient.
- $g_t = \frac{1}{B} \sum_{k \in \mathcal{B}} \nabla f_k(\theta_t)$ is the mini-batch gradient of the batch $\mathcal{B}$ subsampled from the workset table $\mathcal{W}$, i.e., $\mathcal{B} \subseteq \mathcal{W} \subseteq \mathcal{D}$.
- $\tilde{g}_t$ is the approximated version of g_t using stale statistics.

Compared to vanilla SGD training, the analysis in our work is more challenging due to the two drawbacks of local updates in VFL: (1. variance problem) g_t is sampled over the workset table instead of the entire training data; (2. approximation errors) $\tilde{g}_t$ in local updates is approximated using the stale information, which cannot guarantee unbiasedness, i.e., $\mathbb{E}[\tilde{g}_t] \neq \nabla f(\theta_t)$. Consequently, the expected errors can be dissected into two terms:

$$\mathbb{E}\left[\|\tilde{g}_t - \nabla f(\theta_t)\|^2\right] \leq 2\mathbb{E}\left[\|g_t - \nabla f(\theta_t)\|^2\right] + 2\mathbb{E}\left[\|\tilde{g}_t - g_t\|^2\right],$$

where the first term depicts the stochastic variance due to mini-batch sampling whilst the second term comes from the approximation via stale information. Then, we make two assumptions.

Assumption 1. *For each step t, we make these two assumptions: (1. Uniform sampling) $\mathcal{W}$ is uniformly subsampled from $\mathcal{D}$ and $\mathcal{B}$ is uniformly subsampled from $\mathcal{W}$.*
(2. Correct direction) The angles between $\tilde{g}_t, g_t$ are bounded, i.e., there exists a constant $\rho \in (0, 1]$ s.t. $\cos(\tilde{g}_t, g_t) \geq \rho$.

Assumption 2. *For $\forall k \in \{1, 2, ..., N\}, \theta, \theta' \in \mathbb{R}^d$, we make these assumptions following Allen-Zhu [5]:*
(1. L-Lipschitz) $\|\nabla f_k(\theta) - \nabla f_k(\theta')\| \leq L\|\theta - \theta'\|$;
(2. Bounded moment) $\mathbb{E}[\|\nabla f_k(\theta)\|^2] \leq \sigma^2$;
(3. Existence of global minimum) $\exists\theta^*$ s.t. $f(\theta) \geq f(\theta^*)$.

Assumption 1 ensures the unbiasedness of $\mathbb{E}[g_t] = \nabla f(\theta_t)$ and guarantees that the descending directions of the approximated gradients are convincing. Assumption 2 consists of the standard assumptions widely used in the convergence analysis of stochastic optimization. We present our theoretical result in Theorem 1.

[2]Although the algorithm incurs the extra computation for $\nabla Z_A^{(i,j)}$, which is not needed in non-weighted local updates, the extra cost is negligible and worthwhile.

[3]Here we assume 2-dimensional matrices $(\nabla\theta, \widetilde{\nabla}\theta)$ are flattened before the computation of cosine similarities.

Theorem 1. *Under Assumption 1,2 and setting the step size appropriately, after T steps, select $\bar{\theta}_T$ randomly from $\{\theta_t\}_{t=0}^{T-1}$. With probability at least $1-\delta$, we have $\mathbb{E}\left[\|\nabla f(\bar{\theta}_T)\|^2\right] \leq O(\sqrt{\Delta/T})$, where*

$$\Delta = \frac{L^2 \log(2d/\delta)}{B}\left(1+\frac{1}{W}\right) + \sigma^2(2-\rho).$$

Remark. Note that the convergence analysis by Liu et al. [32] relies on a strong assumption on the approximated gradients[4], which fails to reveal the negative impacts brought by local updates. In contrast, our analysis relies on more relaxed assumptions and uncovers the variance problem and approximation errors in local updates. This also sheds light on how the local sampling and instance weighting influence the convergence. On the one hand, with a larger W, the stochastic variance becomes smaller (i.e., smaller Δ) and thus benefits the convergence. On the other hand, if the stale information is more reliable, which gives a higher ρ, the convergence would be faster as well. Nevertheless, it is worthy to notice that there exists a tradeoff since ρ decreases w.r.t. W and R due to the larger staleness.

Communication Complexity. Compared with the sub-linear rate $O(1/\sqrt{T})$ of vanilla SGD training, our framework introduces an extra factor $O(\sqrt{\Delta})$. In other words, vanilla SGD training needs $O(T)$ steps whilst we need $O(\Delta \times T)$ steps to achieve similar convergence. However, the communication complexity of our framework can be reduced to $O(\Delta/R)$ times of that of vanilla SGD training. In Section 5, we will empirically show that `CELU-VFL` requires much fewer communication rounds to achieve the same model quality.

4.2 Security Analysis

Obviously, our framework shares the same security guarantees as vanilla VFL training as introduced in Section 2, since we do not need to exchange values other than activations and derivatives. We refer interested readers to the previous studies for the detailed analysis [7, 21, 32, 51]. Moreover, because our framework can effectively reduce the number of communication rounds as shown in our experiments, the number of messages transmitted between parties would become smaller, which conveys stronger privacy preservation.

5 IMPLEMENTATION AND EVALUATION

5.1 Setup

Implementation. We implement `CELU-VFL` on top of TensorFlow [3] and gRPC. We first develop lightweight message queues to support the `send` and `recv` primitives for cross-party communication. Then, an `FLGraph` abstraction is proposed to automatically generate the backward computation graph and the local update operation after the users have defined the bottom and top models. `CELU-VFL` has been deployed to the collaborative recommendation tasks of Tencent Inc. To be specific, Tencent works as *Party $\mathcal{A}$* and provides user features, and *Party $\mathcal{B}$* is an e-commercial company that owns item features and labels (such as clicks or conversions), but lacks a rich set of user features. In the preprocessing phase, the private data of both companies are aligned as user-item pairs. Then, `CELU-VFL` trains VFL models to fit the labels by the features of both parties.

Experimental Setup. All experiments are carried out on two servers where each server acts as one party. For each party, we run the training tasks with one NVIDIA V100 GPU. The WAN bandwidth between the two servers is 300Mbps.

Models. We focus on the deep learning recommendation models (DLRMs) since it is one of the most common neural network architectures in real-world VFL collaborations. To be specific, we consider two widely-used DLRMs, which are Wide and Deep (WDL) and Deep Structured Semantic Model (DSSM).

Datasets. Three datasets are used in our experiments, as listed in Table 1. *Criteo*[5] and *Avazu*[6] are two large-scale public datasets. Whilst *D3* is a productive dataset in Tencent Inc. As our major focus is the training phase, we assume all datasets have been aligned.

Protocols. We train the models using the AdaGrad optimizer [13] with a batch size of 4096. The output dimensionality of Z_A is set as 256. To achieve a fair comparison, we tune the learning rate for vanilla VFL training and apply the same to our work directly. For each experiment, we run three trials and report the mean and standard deviation (stddev).

Table 1: Description of datasets for end-to-end evaluation.

Dataset	#Instances (train/test)	#Fields (A/B)
Criteo	41M/4.5M	26/13
Avazu	36M/4.0M	14/8
D3	32M/3.5M	25/18

5.2 Ablation Study and Sensitivity

Compared with the vanilla VFL training, `CELU-VFL` contains three additional techniques, i.e., the local update technique, the uniform local sampling strategy, and the instance weighting mechanism (parameterized by R, W, ξ, respectively). We first investigate the impact of each technique and assess the sensitivity w.r.t. the hyperparameters. To be specific, we train WDL on the *Criteo* dataset and measure the number of communication rounds taken to the same target AUC metric. The convergence curves are shown in Figure 5 and the required communication rounds are listed in Table 2.

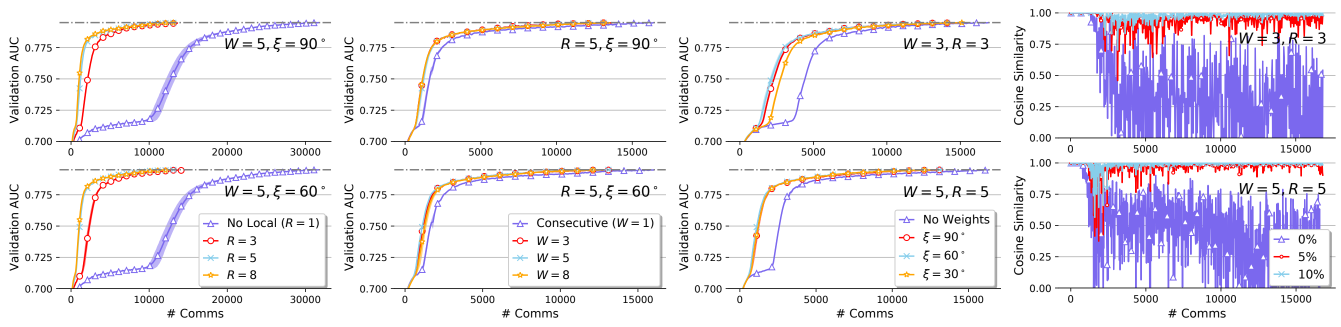
Impact of Local Update. We first assess the effect of the local update technique by varying R. The results are shown in Figure 5(a). In general, compared with the vanilla VFL training, applying the local update technique can significantly reduce the number of communication rounds required to reach the same model performance. For instance, when $R = 3$, the number of communication rounds is reduced by around 55%. When we increase R to 5, the improvement increases to around 60% accordingly. Letting $R = 8$ provides a faster convergence speed than that of $R = 5$ in the early phase. However, due to the larger staleness, it takes a similar number of communication rounds to attain the same validation AUC.

Impact of Local Sampling. To assess the impact of the local sampling strategy, we conduct experiments with various W. When $W = 1$, each mini-batch will be utilized for multiple consecutive local steps, whilst we apply the round-robin local sampling strategy when $W > 1$ to achieve uniform sampling. As shown in Figure 5(b), the round-robin local sampling strategy consistently outperforms the consecutive counterpart — the numbers of required communication rounds are reduced by 18-22%. This matches our analysis

[4]Liu et al. [32] assume the approximated gradients are unbiased to analyze the convergence of FedBCD, whilst our analysis does not need such a strong assumption.

[5]http://labs.criteo.com/2014/02/kaggle-display-advertising-challenge-dataset/

[6]https://www.kaggle.com/c/avazu-ctr-prediction/data



(a) Impact of local update. (b) Impact of local sampling. (c) Impact of instance weighting. (d) Quantiles of cosine similarities.

Figure 5: Validation AUC metrics in terms of the number of communication rounds. For each technique, we conduct experiments with and without the technique and vary the corresponding hyper-parameter under two different settings.

Table 2: Number of communication rounds (mean and stddev) required to reach the same model performance in Figure 5.

Local Update	No Local ($R = 1$)	$R = 3$	$R = 5$	$R = 8$
$W = 5, \xi = 90^\circ$	31540 ± 631.19	14000 ± 424.26(↓ 55.61%)	12767 ± 654.90(↓ 59.52%)	12733 ± 410.96(↓ 59.63%)
$W = 5, \xi = 60^\circ$	31540 ± 631.19	14333 ± 329.98(↓ 54.56%)	12767 ± 286.74(↓ 59.52%)	12500 ± 424.26(↓ 60.37%)
Local Sampling	Consecutive ($W = 1$)	$W = 3$	$W = 5$	$W = 8$
$R = 5, \xi = 90^\circ$	16400 ± 244.95	13333 ± 339.94(↓ 18.70%)	12767 ± 654.90(↓ 22.15%)	13333 ± 249.44(↓ 18.70%)
$R = 5, \xi = 60^\circ$	15967 ± 339.94	12700 ± 141.42(↓ 20.46%)	12767 ± 286.74(↓ 20.04%)	12667 ± 419.00(↓ 20.67%)
Instance Weighting	No Weights	$\xi = 90^\circ$	$\xi = 60^\circ$	$\xi = 30^\circ$
$W = 3, R = 3$	16767 ± 492.16	14233 ± 235.70(↓ 15.11%)	15200 ± 163.30(↓ 9.36%)	14667 ± 94.28(↓ 12.52%)
$W = 5, R = 5$	16467 ± 492.16	12767 ± 286.74(↓ 22.47%)	12567 ± 205.48(↓ 23.68%)	12767 ± 654.90(↓ 22.47%)

that uniform sampling can help alleviate the stochastic variance in mini-batch gradients and therefore improve the convergence. Moreover, it is worthy to note that the convergence speed does not change significantly when $W \in \{3, 5, 8\}$, which verifies the robustness of CELU-VFL against the choice of workset size.

Impact of Instance Weighting. Next, we explore the benefit brought by the instance weighting mechanism with different ξ. The results are presented in Figure 5(c). In short, the instance weighting mechanism accelerates the convergence under both settings since it restricts the approximation errors incurred by the stale values. Under the first setting ($W = 3, R = 3$), we can save 9-15% of the communication rounds. Under the second setting ($W = 5, R = 5$), the improvement bumps to around 23%. This is a reasonable phenomenon — with larger W and/or R, the staleness of cached values increases and the approximation errors are enlarged as well, so the contribution of our weighting mechanism is more significant.

We further plot the cosine similarities for better interpretability. To be specific, for each local update, we compute the quantiles of all similarities in the current batch (0% quantiles correspond to the lowest similarities). The results in Figure 5(d) show two facts: (1) Most of the stale statistics are reliable. For instance, over 90% of the cosine similarities are greater than 0.5 even in the fast converging period, which supports our motivation that the model does not change significantly and the stale statistics can be re-used in local updates. (2) Our instance weighting mechanism is able to filter or penalize the outliers to accelerate the convergence.

Summary. The experiments demonstrate the power of each technique individually. Overall, all techniques have significant contributions to reducing the number of communication rounds compared with the counterparts without the techniques. Furthermore, our work is robust to different configurations (the choices of the hyper-parameters), which is vital to the applicability in practice.

5.3 End-to-End Evaluation

Then, we evaluate the end-to-end training performance of CELU-VFL. To be specific, we compare CELU-VFL with two competitors, which are FedBCD [32] and the vanilla VFL training (denoted as Vanilla). Following the results in Section 5.2, we fix $W = 5, \xi = 60^\circ$ and let $R \in \{5, 8\}$ for all experiments. We presents the convergence curves in Figure 6 and discuss the results for different datasets below.

Results on Criteo. We first train WDL on the *Criteo* dataset. Thanks to the reduction in communication, CELU-VFL achieves 2.47× of speedup compared with Vanilla. Whilst the performance of FedBCD is unsatisfactory. For $R = 5$, although FedBCD runs faster than Vanilla in the short-term convergence, it slows down in the long-term convergence. Eventually, the gap between FedBCD and Vanilla is very small. The reason is that when the model approaches the (local) optimum point, the gradients become smaller in scale and therefore are more sensitive to the errors caused by local sampling and stale information. Worst still, FedBCD fails to converge for $R = 8$ due to the large staleness. In contrast, CELU-VFL has a better statistical efficiency owing to our techniques. As a result, CELU-VFL can support a larger R and outperform FedBCD by 2.35×.

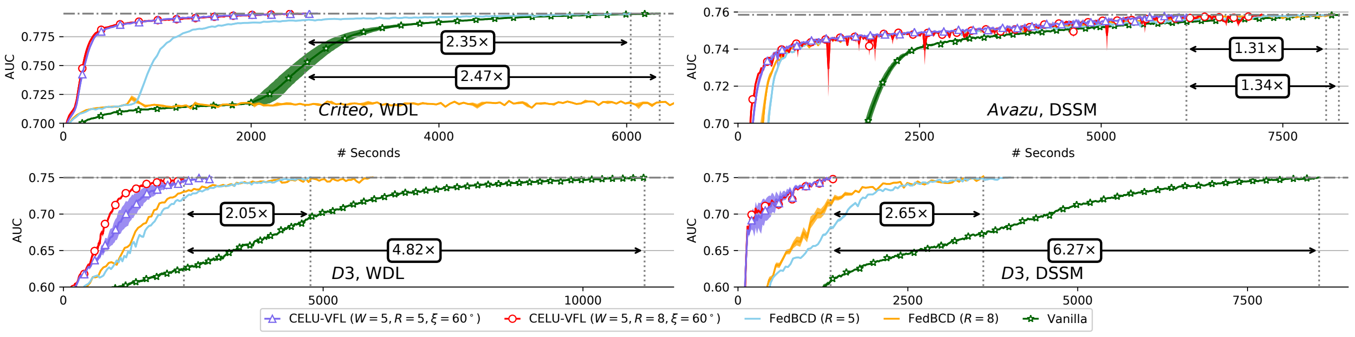


Figure 6: Validation AUC metrics in terms of running time (in seconds)

Results on Avazu. We then evaluate the performance by training DSSM on the *Avazu* dataset. First, in the early phase, both CELU-VFL and FedBCD accelerate the convergence by a large extent compared with Vanilla with the help of local updates. However, similar to the results on *Criteo*, FedBCD suffers from a slow convergence later, whilst CELU-VFL converges properly and achieves 1.31-1.34× of improvement compared with the other competitors.

Results on Industrial Dataset. Finally, we conduct experiments on the industrial dataset *D3* to see how CELU-VFL performs in real-world workloads. Overall, CELU-VFL outperforms Vanilla by 4.82-6.27× on the two training tasks. FedBCD has a better performance on the industrial dataset, however, it is still 2.05-2.65× slower than CELU-VFL. These empirical results demonstrate the superiority of our work in practical VFL applications.

6 RELATED WORKS

Federated learning. In recent years, FL has aroused the interests from both academia and industry. Notably, VFL conveys a possibility to unite the data of different corporations and jointly build the models, which fits numerous real-world cross-enterprise collaborations [8, 17, 23, 29, 49]. Besides VFL, another important field of FL is the horizontal FL (HFL), where the datasets are horizontally partitioned [27, 28, 31, 48]. These works are orthogonal to ours since we focus on the VFL scenario.

Asynchronous VFL. In addition to the local update technique, various works address the cross-party communication cost in VFL via asynchronous learning. However, these works cannot be applied in our scenario. Below we categorize them into two lines.

(Asynchronous Workers) The first kind of works tries to overlap computation and communication by using multiple training workers (threads) for each party [21, 51]. The models are stored in a shared-memory and each worker individually reads and updates the models in a Hogwild! manner [39]. By doing so, when some workers are waiting for the message transmission, the other workers can concurrently do the computation.

Nevertheless, these methods mainly focus on linear models where the output dimensionality of the exchanged statistics is very low (e.g., only one for logistic regression and support vector machine). Due to the small message size, sending one message cannot make full use of the bandwidth, so their major bottleneck is actually the high network latency rather than the low network bandwidth. In contrast, the output dimensionality is much higher for many modern deep learning models, so the messages are much larger in size and the network bandwidth would be easily occupied when sending one message. Thus, letting multiple workers communicate concurrently would cause severe network congestion and even lead to failure. Consequently, these methods do not fit our scenario.

(Asynchronous Parties) The second category assumes that there are two or more *Party* $\mathcal{A}$'s and allows different parties to train asynchronously [7, 23]. To be specific, *Party* $\mathcal{B}$ is treated as a server that holds the labels and stores the forward activations received from all *Party* $\mathcal{A}$'s. During the training process, each *Party* $\mathcal{A}_i$ individually samples a mini-batch and sends Z_{A_i} to *Party* $\mathcal{B}$. Upon receiving, *Party* $\mathcal{B}$ updates the local storage, computes and returns ∇Z_{A_i}. Finally, each *Party* $\mathcal{A}_i$ individually performs a backward propagation to update $\theta_{Bot\_A_i}$ without synchronizing with other parties. The spirit of this line of research lies with the asynchrony between *Party* $\mathcal{A}$'s — different *Party* $\mathcal{A}$'s can train on different mini-batches to achieve asynchronous learning.

However, such methods are orthogonal to our work since we focus on the case where only one *Party* $\mathcal{A}$ is involved, which is more commonly seen in real-world applications. Moreover, since these works still require to exchange forward activations and backward derivatives in every step, our work can be integrated with their methods easily by applying the local update technique for each *Party* $\mathcal{A}_i$ individually. We would like to leave the extension to multi-party VFL training as our future work.

7 CONCLUSION

This paper focuses on the deficiency caused by heavy cross-party communication in VFL training. Specifically, we propose CELU-VFL, an efficient VFL training framework with cached-enable local updates. To improve the statistical efficiency, we develop two novel techniques to achieve fairness in stale statistics sampling and measure the fine-grained reliability of the local updates, respectively. Finally, both theoretical analysis and empirical experiments verify the effectiveness of CELU-VFL.

ACKNOWLEDGMENTS

This work is supported by National Natural Science Foundation of China (NSFC) (No. 61832001), PKU-Tencent joint research Lab, and Beijing Academy of Artificial Intelligence (BAAI). Bin Cui is the corresponding author.

REFERENCES

[1] 2018. California Consumer Privacy Act (CCPA). https://oag.ca.gov/privacy/ccpa.

[2] 2021. Virginia Consumer Data Protection Act (CDPA). https://lis.virginia.gov/cgi-bin/legp604.exe?212+sum+HB2307.

[3] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, Manjunath Kudlur, Josh Levenberg, Rajat Monga, Sherry Moore, Derek Gordon Murray, Benoit Steiner, Paul A. Tucker, Vijay Vasudevan, Pete Warden, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. 2016. TensorFlow: A System for Large-Scale Machine Learning. In *12th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2016*. USENIX Association, 265–283.

[4] Martín Abadi, Andy Chu, Ian J. Goodfellow, H. Brendan McMahan, Ilya Mironov, Kunal Talwar, and Li Zhang. 2016. Deep Learning with Differential Privacy. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, CCS 2016*. ACM, 308–318.

[5] Zeyuan Allen-Zhu. 2017. Natasha: Faster Non-Convex Stochastic Optimization via Strongly Non-Convex Parameter. In *Proceedings of the 34th International Conference on Machine Learning, ICML 2017 (Proceedings of Machine Learning Research)*, Vol. 70. PMLR, 89–97.

[6] Hao Chen, Kim Laine, and Peter Rindal. 2017. Fast Private Set Intersection from Homomorphic Encryption. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, CCS 2017*. ACM, 1243–1255.

[7] Tianyi Chen, Xiao Jin, Yuejiao Sun, and Wotao Yin. 2020. VAFL: a Method of Vertical Asynchronous Federated Learning. *CoRR* abs/2007.06081 (2020).

[8] Kewei Cheng, Tao Fan, Yilun Jin, Yang Liu, Tianjian Chen, Dimitrios Papadopoulos, and Qiang Yang. 2021. SecureBoost: A Lossless Federated Learning Framework. *IEEE Intell. Syst.* 36, 6 (2021), 87–98.

[9] William L. Croft, Jörg-Rüdiger Sack, and Wei Shi. 2022. Differential Privacy via a Truncated and Normalized Laplace Mechanism. *J. Comput. Sci. Technol.* 37, 2 (2022), 369–388.

[10] Wei Dai, Jinliang Wei, Xun Zheng, Jin Kyu Kim, Seunghak Lee, Junming Yin, Qirong Ho, and Eric P. Xing. 2013. Petuum: A Framework for Iterative-Convergent Distributed ML. *CoRR* abs/1312.7651 (2013).

[11] Jeffrey Dean, Greg Corrado, Rajat Monga, Kai Chen, Matthieu Devin, Quoc V. Le, Mark Z. Mao, Marc'Aurelio Ranzato, Andrew W. Senior, Paul A. Tucker, Ke Yang, and Andrew Y. Ng. 2012. Large Scale Distributed Deep Networks. In *Annual Conference on Neural Information Processing Systems 2012, NeurIPS 2012*. 1232–1240.

[12] Changyu Dong, Liqun Chen, and Zikai Wen. 2013. When private set intersection meets big data: an efficient and scalable protocol. In *Proceedings of the 2013 ACM SIGSAC Conference on Computer and Communications Security, CCS 2013*. ACM, 789–800.

[13] John C. Duchi, Elad Hazan, and Yoram Singer. 2011. Adaptive Subgradient Methods for Online Learning and Stochastic Optimization. *J. Mach. Learn. Res.* 12 (2011), 2121–2159.

[14] Cynthia Dwork. 2008. Differential Privacy: A Survey of Results. In *Theory and Applications of Models of Computation, 5th International Conference, TAMC 2008, Xi'an, China, April 25-29, 2008. Proceedings*, Vol. 4978. Springer, 1–19.

[15] Xixuan Feng, Arun Kumar, Benjamin Recht, and Christopher Ré. 2012. Towards a unified architecture for in-RDBMS analytics. In *Proceedings of the 2012 International Conference on Management of Data, SIGMOD 2012*. ACM, 325–336.

[16] Michael J. Freedman, Kobbi Nissim, and Benny Pinkas. 2004. Efficient Private Matching and Set Intersection. In *International Conference on the Theory and Applications of Cryptographic Techniques, EUROCRYPT 2004*, Vol. 3027. Springer, 1–19.

[17] Fangcheng Fu, Yingxia Shao, Lele Yu, Jiawei Jiang, Huanran Xue, Yangyu Tao, and Bin Cui. 2021. VF^{2}Boost: Very Fast Vertical Federated Gradient Boosting for Cross-Enterprise Learning. In *Proceedings of the 2021 International Conference on Management of Data, SIGMOD 2021*. ACM, 563–576.

[18] Fangcheng Fu, Huanran Xue, Yong Cheng, Yangyu Tao, and Bin Cui. 2022. BlindFL: Vertical Federated Machine Learning without Peeking into Your Data. In *Proceedings of the 2022 International Conference on Management of Data, SIGMOD 2022*. ACM, 1316–1330.

[19] Zhihe Gao, Xiaoming Chen, and Xiaodan Shao. 2022. Robust federated learning for edge-intelligent networks. *Sci. China Inf. Sci.* 65, 3 (2022).

[20] Yong-Feng Ge, Jinli Cao, Hua Wang, Zhenxiang Chen, and Yanchun Zhang. 2021. Set-Based Adaptive Distributed Differential Evolution for Anonymity-Driven Database Fragmentation. *Data Sci. Eng.* 6, 4 (2021), 380–391.

[21] Bin Gu, An Xu, Zhouyuan Huo, Cheng Deng, and Heng Huang. 2020. Privacy-Preserving Asynchronous Federated Learning Algorithms for Multi-Party Vertically Collaborative Learning. *CoRR* abs/2008.06233 (2020). arXiv:2008.06233

[22] Farzin Haddadpour, Mohammad Mahdi Kamani, Mehrdad Mahdavi, and Viveck R. Cadambe. 2019. Local SGD with Periodic Averaging: Tighter Analysis and Adaptive Synchronization. In *Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019*. 11080–11092.

[23] Yaochen Hu, Di Niu, Jianming Yang, and Shengping Zhou. 2019. FDML: A Collaborative Machine Learning Framework for Distributed Features. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, KDD 2019*. ACM, 2232–2240.

[24] Jiawei Jiang, Fangcheng Fu, Tong Yang, and Bin Cui. 2018. SketchML: Accelerating Distributed Machine Learning with Data Sketches. In *Proceedings of the 2018 International Conference on Management of Data, SIGMOD 2018*. ACM, 1269–1284.

[25] Jiawei Jiang, Fangcheng Fu, Tong Yang, Yingxia Shao, and Bin Cui. 2020. SKCompress: compressing sparse and nonuniform gradient in distributed machine learning. *VLDB J.* 29, 5 (2020), 945–972.

[26] Rie Johnson and Tong Zhang. 2013. Accelerating Stochastic Gradient Descent using Predictive Variance Reduction. In *Annual Conference on Neural Information Processing Systems 2013, NeurIPS 2013*. 315–323.

[27] Jakub Konečný, H. Brendan McMahan, Daniel Ramage, and Peter Richtárik. 2016. Federated Optimization: Distributed Machine Learning for On-Device Intelligence. *CoRR* abs/1610.02527 (2016). arXiv:1610.02527

[28] Jakub Konečný, H. Brendan McMahan, Felix X. Yu, Peter Richtárik, Ananda Theertha Suresh, and Dave Bacon. 2016. Federated Learning: Strategies for Improving Communication Efficiency. *CoRR* abs/1610.05492 (2016). arXiv:1610.05492

[29] Oscar Li, Jiankai Sun, Xin Yang, Weihao Gao, Hongyi Zhang, Junyuan Xie, Virginia Smith, and Chong Wang. 2022. Label Leakage and Protection in Two-party Split Learning. In *International Conference on Learning Representations, ICLR 2022*.

[30] Tian Li, Anit Kumar Sahu, Ameet Talwalkar, and Virginia Smith. 2020. Federated Learning: Challenges, Methods, and Future Directions. *IEEE Signal Process. Mag.* 37, 3 (2020), 50–60.

[31] Tian Li, Anit Kumar Sahu, Manzil Zaheer, Maziar Sanjabi, Ameet Talwalkar, and Virginia Smith. 2020. Federated Optimization in Heterogeneous Networks. In *Proceedings of Machine Learning and Systems 2020, MLSys 2020*. mlsys.org.

[32] Yang Liu, Yan Kang, Xinwei Zhang, Liping Li, Yong Cheng, Tianjian Chen, Mingyi Hong, and Qiang Yang. 2019. A Communication Efficient Vertical Federated Learning Framework. *CoRR* abs/1912.11187 (2019).

[33] Xupeng Miao, Yining Shi, Hailin Zhang, Xin Zhang, Xiaonan Nie, Zhi Yang, and Bin Cui. 2022. HET-GMP: A Graph-based System Approach to Scaling Large Embedding Model Training. In *Proceedings of the 2022 International Conference on Management of Data, SIGMOD 2022*. ACM, 470–480.

[34] Xupeng Miao, Hailin Zhang, Yining Shi, Xiaonan Nie, Zhi Yang, Yangyu Tao, and Bin Cui. 2021. HET: Scaling out Huge Embedding Model Training via Cache-enabled Distributed Framework. *Proc. VLDB Endow.* 15, 2 (2021), 312–320.

[35] Konstantin Mishchenko, Ahmed Khaled, and Peter Richtárik. 2020. Random Reshuffling: Simple Analysis with Vast Improvements. In *Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020*.

[36] Mukti Padhya and Devesh C. Jinwala. 2020. R-OO-KASE: Revocable Online/Offline Key Aggregate Searchable Encryption. *Data Sci. Eng.* 5, 4 (2020), 391–418.

[37] Zhengping Qian, Chenqiang Min, Longbin Lai, Yong Fang, Gaofeng Li, Youyang Yao, Bingqing Lyu, Xiaoli Zhou, Zhimin Chen, and Jingren Zhou. 2021. GAIA: A System for Interactive Analysis on Distributed Graphs Using a High-Level Language. In *18th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2021*. USENIX Association, 321–335.

[38] Morteza Ramezani, Weilin Cong, Mehrdad Mahdavi, Anand Sivasubramaniam, and Mahmut T. Kandemir. 2020. GCN meets GPU: Decoupling "When to Sample" from "How to Sample". In *Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020*.

[39] Benjamin Recht, Christopher Ré, Stephen J. Wright, and Feng Niu. 2011. Hogwild: A Lock-Free Approach to Parallelizing Stochastic Gradient Descent. In *Annual Conference on Neural Information Processing Systems 2011, NeurIPS 2013*. 693–701.

[40] Sebastian U. Stich. 2019. Local SGD Converges Fast and Communicates Little. In *International Conference on Learning Representations, ICLR 2019*.

[41] Paul Voigt and Axel Von dem Bussche. 2017. The eu general data protection regulation (gdpr). *A Practical Guide, 1st Ed., Cham: Springer International Publishing* (2017).

[42] Li Wan, Wee Keong Ng, Shuguo Han, and Vincent C. S. Lee. 2007. Privacy-preservation for gradient descent methods. In *Proceedings of the 13th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, KDD 2007*. ACM, 775–783.

[43] Shiwen Wu, Fei Sun, Wentao Zhang, Xu Xie, and Bin Cui. 2022. Graph Neural Networks in Recommender Systems: A Survey. *ACM Comput. Surv.* (mar 2022). Just Accepted.

[44] Yuncheng Wu, Shaofeng Cai, Xiaokui Xiao, Gang Chen, and Beng Chin Ooi. 2020. Privacy Preserving Vertical Federated Learning for Tree-based Models. *Proc. VLDB Endow.* 13, 11 (2020), 2090–2103.

[45] Xu Xie, Fei Sun, Zhaoyang Liu, Shiwen Wu, Jinyang Gao, Bolin Ding, and Bin Cui. 2020. Contrastive Learning for Sequential Recommendation. *CoRR* abs/2010.14395 (2020). arXiv:2010.14395

[46] Xu Xie, Fei Sun, Xiaoyong Yang, Zhao Yang, Jinyang Gao, Wenwu Ou, and Bin Cui. 2021. Explore User Neighborhood for Real-time E-commerce Recommendation. In *37th IEEE International Conference on Data Engineering, ICDE 2021*. IEEE, 2464–2475.

[47] Lijie Xu, Shuang Qiu, Binhang Yuan, Jiawei Jiang, Cédric Renggli, Shaoduo Gan, Kaan Kara, Guoliang Li, Ji Liu, Wentao Wu, Jieping Ye, and Ce Zhang. 2022. In-Database Machine Learning with CorgiPile: Stochastic Gradient Descent without Full Data Shuffle. In *Proceedings of the 2022 International Conference on Management of Data, SIGMOD 2022*. ACM, 1286–1300.

[48] Qiang Yang, Yang Liu, Tianjian Chen, and Yongxin Tong. 2019. Federated Machine Learning: Concept and Applications. *ACM Trans. Intell. Syst. Technol.* 10, 2 (2019), 12:1–12:19.

[49] Shengwen Yang, Bing Ren, Xuhui Zhou, and Liping Liu. 2019. Parallel Distributed Logistic Regression for Vertical Federated Learning without Third-Party Coordinator. *CoRR* abs/1911.09824 (2019). arXiv:1911.09824

[50] Qingsong Zhang, Bin Gu, Cheng Deng, Songxiang Gu, Liefeng Bo, Jian Pei, and Heng Huang. 2021. AsySQN: Faster Vertical Federated Learning Algorithms with Better Computation Resource Utilization. In *Proceedings of the 27th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, KDD 2021*. ACM, 3917–3927.

[51] Qingsong Zhang, Bin Gu, Cheng Deng, and Heng Huang. 2021. Secure Bilevel Asynchronous Vertical Federated Learning with Backward Updating. In *Thirty-Fifth AAAI Conference on Artificial Intelligence, AAAI 2021*. AAAI Press, 10896–10904.

[52] Sixin Zhang, Anna Choromanska, and Yann LeCun. 2015. Deep learning with Elastic Averaging SGD. In *Annual Conference on Neural Information Processing Systems 2015, NeurIPS 2015*. 685–693.

[53] Wentao Zhang, Jiawei Jiang, Yingxia Shao, and Bin Cui. 2020. Snapshot boosting: a fast ensemble framework for deep neural networks. *Sci. China Inf. Sci.* 63, 1 (2020), 112102.