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

Fangcheng Fu
Peking University
cchengff@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.

1 School of Computer Science & Key Lab of High Confidence Software Technologies (MOE), Peking University
2 School of Computer Science, Wuhan University
3 Institute of Computational Social Science, Peking University (Qingdao), China

This work is licensed under the Creative Commons BY-NC-ND 4.0 International License. Visit https://creativecommons.org/licenses/by-nc-nd/4.0/ to view a copy of this license. For any use beyond those covered by this license, obtain permission by emailing info@vldb.org. Copyright is held by the owner/author(s). Publication rights licensed to the VLDB Endowment.

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

1 In Practice, two or more Party 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 A’s easily. Therefore, we only consider one Party A in this work for simplicity.

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 A’s that merely provide features; and (ii) one Party 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 A can be an Internet company that is able to generate rich and precise user profiling, whilst Party B can be an e-commercial company that hopes to build a better recommender system. As a result, VFL is gaining more popularity in both academia and industry [8, 17, 44, 49].
Worse still, since most companies have network restrictions to where (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. This 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., $(\nabla Z_i^{(j)}, \nabla Z_i^{(j)}_A)$, Party A treats $\nabla Z_A$ as an approximation and estimates the model gradients as $\tilde{\nabla}_{\theta_{Bot, A}} = \nabla Z_A \frac{\partial z_A}{\partial \theta_{Bot, A}}$; symmetrically, Party B feeds the stale values $Z_B^{(j)}$ and the ad hoc values $Z_B^{(j)}$ 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.

(Communication Bottleneck of VFL) Denote y as the labels owned by Party B, and $X_A, X_B$ as the features of Party A and Party B, respectively, where the k-th rows $X_{Ak}, X_{Bk}$ 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 P is associated with a bottom model that works as a feature extractor. It takes $X_P$ as input and outputs $Z_P = \text{Bot}_{top}(X_P; \theta_{Bot, P})$, where $\theta_{Bot, P}$ are the bottom model weights. Second, Party B is further associated with a top model parameterized by $\theta_{top}$ that makes predictions via $\hat{y} = \text{Top}(Z_A, Z_B; \theta_{top})$. The goal of VFL training is to find the model weights minimizing the empirical risk:

$$\min_{\theta_{Bot, A}, \theta_{bottom}} \mathbb{E}_l \left[ l(y_k, \hat{y}_k) \right]$$

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 $\nabla \theta = \frac{\partial L}{\partial \theta}$ for model updates. However, since Party 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 $\theta_{Bot, A}$ are calculated via $\nabla_{\theta_{Bot, A}} = \nabla Z_A \frac{\partial z_A}{\partial \theta_{Bot, A}}$, where $\nabla Z_A = \frac{\partial L}{\partial Z_A} \partial Z_A$ are provided by Party B.

As shown in the Gantt chart in Figure 1, in each step, two kinds of intermediate data are transmitted, i.e., forward activations $Z_A$ and backward derivatives $\nabla Z_A$. 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 $\nabla Z_A$. Thus, approximating $Z_A$ or $\nabla Z_A$ with stale statistics is needed. Specifically, Fed-BCD [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., $(\nabla Z_i^{(j)}, \nabla Z_i^{(j)}_A)$, Party A treats $\nabla Z_A$ as an approximation and estimates the model gradients as $\tilde{\nabla}_{\theta_{Bot, A}} = \nabla Z_A \frac{\partial z_A}{\partial \theta_{Bot, A}}$; symmetrically, Party B feeds the stale values $Z_B^{(j)}$ and the ad hoc values $Z_B^{(j)}$ to the top model.

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 weight-
ing 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 2.65-6.27X.

2 PRELIMINARIES

2.1 Vertical Federated Learning
Owing to the establishment of lawful regulation on privacy protec-
tion [1, 2, 41], how to train ML models with privacy preservation is
becoming a hot topic. Notably, VFL has become a prevailing para-
digm that unites the features from different parties for collabora-
tive 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 inter-
mediate 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 commu-
nication 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 \nabla 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);
- \epsilon: the threshold in our instance weighting mechanism;
- Z_A^{(i)}, \nabla Z_A^{(i)}: the forward activations and backward derivatives of
Party A of the i-th batch;
- Z_A^{(i)}, \nabla Z_A^{(i)}: the forward activations and backward derivatives
of Party 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 trans-
mission between parties. In the beginning, both parties agree on the
ordering of mini-batches. For each batch, the communication work-
ers execute the conventional forward and backward propagation
to exchange Z_A and \nabla Z_A. Then, both Z_A and \nabla 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 statist-
tics. For each local step, the local worker of each party chooses
one cached entry \(i, Z_A^{(i)}, \nabla Z_A^{(i)}, j\) 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 de-

depicted in Figure 2, Party 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.
Local Comm

\[ W = \text{workers.} \]

For instance, the top row of Figure 4 depicts a case where divergence would be slowed down as the maximum staleness of \( W \) far smaller than the total number of communication rounds.

\[ W \]

guaranteeing the uniformity. Although bubbles are incurred in the strategy, the training worker samples the mini-batches one by one, in the workset cannot be sampled again in the next \( W \) 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.

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.

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. \textbf{Function} \textsc{InsWeight}(ad hoc V\(_{\text{A}}\), stale V\(_{\text{B}}\)):
   1. \textsc{weights} \leftarrow \cos(V_{\text{A}}^{(i)}, V_{\text{B}}^{(i)}, \text{axis} = 1);
   2. \textsc{weights} \leftarrow \cos(V_{\text{B}}^{(i)}, V_{\text{A}}^{(i)}, \text{axis} = 1);
   3. return \textsc{weights}

2. \textbf{Function} \textsc{LocalUpdatePartyA}(i, Z\(_A^{(i)}\), \nabla Z\(_A^{(i)}\), f):
   1. \(Z_{\text{A}}^{(i)} \leftarrow \textsc{ForwardProp}(X_{\text{A}}^{(i)});\)
   2. \textsc{ins_weights} \leftarrow \textsc{InsWeight}(Z_{\text{A}}^{(i)}, Z_{\text{A}}^{(i)});
   3. \textsc{BackwardProp}(\textsc{ins_weights} \odot \nabla Z_{\text{A}}^{(i)});

3. \textbf{Function} \textsc{LocalUpdatePartyB}(i, Z\(_B^{(i)}\), \nabla Z\(_B^{(i)}\), f):
   1. \(Z_{\text{B}}^{(i)} \leftarrow \textsc{ForwardProp}(X_{\text{B}}^{(i)});\)
   2. \textsc{ins_weights} \leftarrow \textsc{InsWeight}(Z_{\text{B}}^{(i)}, Z_{\text{B}}^{(i)});
   3. \textsc{BackwardProp}(\textsc{ins_weights} \odot \nabla Z_{\text{B}}^{(i)});

4. rob tu certain noises in gradients, the model performance will be harmed inevitably when the staleness is large. To tackle this problem, we desire 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 \(\text{A}\), since the ad hoc forward activations are computed for each local update, we measure the cosine similarities by \(\cos(Z_{\text{A}}^{(i)}, Z_{\text{B}}^{(i)})\), where the cosine operation applies on every row for each instance individually. In symmetric, Party \(\text{B}\) measures the cosine similarities via the backward derivatives\(^2\), i.e., \(\cos(\nabla Z_{\text{A}}^{(i)}, \nabla Z_{\text{B}}^{(i)})\). 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.

\textbf{Discussion.} To help readers better understand the rationality of such a heuristic method, we use a fully connected layer \(z_{\text{out}} = z_{\text{in}} \theta\) as an example, where \(\theta \in \mathbb{R}^{d_{\text{in}} \times d_{\text{out}}}\) denotes the model weights and \(z_{\text{in}} \in \mathbb{R}^{d_{\text{in}}}, z_{\text{out}} \in \mathbb{R}^{d_{\text{out}}}\) 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., \(\theta = z_{\text{in}} \odot \nabla z_{\text{out}}\). For Party \(\text{A}\), suppose the stale derivatives are denoted as \(\tilde{\nabla} z_{\text{out}}\), it is easy to know that \(\cos(\nabla \theta, \tilde{\nabla} \theta) = \cos(\nabla z_{\text{out}}, \tilde{\nabla} z_{\text{out}})^3\), where \(\tilde{\nabla} \theta = z_{\text{in}} \odot \tilde{\nabla} z_{\text{out}}\). Similarly, we have the same property for Party \(\text{B}\). 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.,

\[
\min_{\theta \in \Theta} f(\theta) = \frac{1}{N} \sum_{k \in D} f_k(\theta), \quad \theta_{t+1} = \theta_t - \eta g_t,
\]

where \(\eta\) 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:

- \(f(\theta_t) = \frac{1}{N} \sum_{k \in D} f_k(\theta_t)\) is the full-batch gradient.
- \(g_t = \frac{1}{B} \sum_{k \in B} \nabla f_k(\theta_t)\) is the mini-batch gradient of the batch \(B\) subsampled from the workset table \(\mathcal{W}\), i.e., \(B \subseteq \mathcal{W} \subseteq D\).
- \(\tilde{g}_t\) 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., \(\mathbb{E}[\tilde{g}_t] \neq \mathbb{E}[g(\theta_t)]\). Consequently, the expected errors can be dissected into two terms:

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

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.

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

\textbf{Assumption 2.} For any \(v_k \in \{1, 2, ..., N\}, 0, 0' \in \mathbb{R}^d\), 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. \(f(\theta^*)\) is the global minimum.

Assumption 1 ensures the unbiasedness of \(\mathbb{E}[g_k] = \nabla f(\theta_t)\) 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 \(\nabla Z_{\text{A}}^{(i)}\), which is not needed in non-weighted local updates, the extra cost is negligible and worthwhile.

\(^3\)Here we assume 2-dimensional matrices (\(\nabla \theta, \tilde{\nabla} \theta\)) are flattened before the computation of cosine similarities.
Theorem 1. Under Assumption 1,2 and setting the step size appropriately, after T steps, select \( \theta_t \) randomly from \( (\theta_t)_{t=0}^{T-1} \). With probability at least \( 1 - \delta \), we have \( \|\nabla f(\theta_t)\|^2 \leq O(\sqrt{\Delta / T}) \), where \( \Delta = \frac{L^2 \log \left( \frac{2d / \delta}{B} \right)}{B} \left( 1 + \frac{1}{W} \right) + \sigma^2 (2 - \rho) \).

Remark. Note that the convergence analysis by Liu et al. [32] relies on a strong assumption on the approximated gradients, 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 other hand, if the staleness is small, the stochastic variance becomes smaller (i.e., smaller \( \Delta \)) and thus benefits the convergence. On the other hand, if the stale information is more reliable, which gives a higher \( \rho \), the convergence would be faster as well. Nevertheless, it is worthy to notice that there exists a tradeoff since \( \rho \) 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 \( \text{Party A} \) and provides user features, and \( \text{Party B} \) is an e-commerce 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. \textit{Criteo} and \textit{Avazu} are two large-scale public datasets. Whilst \textit{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 \( Z_0 \) 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).

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 \( R, W, \xi \), 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 \textit{Criteo} dataset and measure the number of communication rounds taken to reach 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 \( R \approx 8 \) 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

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/4.8M                | 14/8          |
| D3      | 32M/3.5M                | 25/18         |

\( ^4 \)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
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 $\xi$. 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 present 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 $2.47 \times$ 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 $2.35 \times$.

---

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

| Local Update   | No Local ($R = 1$) | $R = 3$ | $R = 5$ | $R = 8$ |
|---------------|-------------------|---------|---------|---------|
| $W = 5, \xi = 90^\circ$ | $13540 \pm 631.19$ | $14000 \pm 424.26$ | $12767 \pm 654.90$ | $12733 \pm 410.96$ |
| $W = 5, \xi = 60^\circ$ | $13540 \pm 631.19$ | $14000 \pm 424.26$ | $12767 \pm 654.90$ | $12733 \pm 410.96$ |
| $W = 3, R = 3$ | $13333 \pm 329.98$ | $12767 \pm 286.74$ | $12767 \pm 286.74$ | $12767 \pm 286.74$ |
| $W = 5, R = 5$ | $13333 \pm 249.44$ | $12767 \pm 204.04$ | $12667 \pm 419.00$ | $12667 \pm 419.00$ |

- **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.
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 1.31-1.34x 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 4.82-6.27x on the two training tasks. FedBCD has a better performance on the industrial dataset, however, it is still 2.05-2.65x 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.

(Asyncronous 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 Ai’s and allows different parties to train asynchronously [7, 23]. To be specific, Party B is treated as a server that holds the labels and stores the forward activations received from all Party Ai’s. During the training process, each Party Ai individually samples a mini-batch and sends ZAi to Party B. Upon receiving, Party B updates the local storage, computes and returns ∇ZAi. Finally, each Party Ai individually performs a backward propagation to update θAi without synchronizing with other parties. The spirit of this line of research lies with the asynchrony between Party Ai’s — different Party Ai’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 Ai 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 Ai 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.
[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.