Flexible Vertical Federated Learning With Heterogeneous Parties

Timothy Castiglia, Shiqiang Wang, Senior Member, IEEE, and Stacy Patterson

Abstract—We propose flexible vertical federated learning (Flex-VFL), a distributed machine algorithm that trains a smooth, nonconvex function in a distributed system with vertically partitioned data. We consider a system with several parties that wish to collaboratively learn a global function. Each party holds a local dataset; the datasets have different features but share the same sample ID space. The parties are heterogeneous in nature: the parties' operating speeds, local model architectures, and optimizers may be different from one another and, further, they may change over time. To train a global model in such a system, Flex-VFL utilizes a form of parallel block coordinate descent (P-BCD), where parties train a partition of the global model via stochastic coordinate descent. We provide theoretical convergence analysis for Flex-VFL and show that the convergence rate is constrained by the party speeds and local optimizer parameters. We apply this analysis and extend our algorithm to adapt party learning rates in response to changing speeds and local optimizer parameters. Finally, we compare the convergence time of Flex-VFL against synchronous and asynchronous VFL algorithms, as well as illustrate the effectiveness of our adaptive extension.

Index Terms—Algorithms, computational and artificial intelligence, convergence, distributed algorithms, federated learning, machine learning, mathematics, neural networks, nonconvex optimization, stochastic gradient descent (SGD).

NOMENCLATURE

Notation Definitions
\( K \) Number of parties/number of vertical partitions.
\( x_k^i \) Local features of data sample \( i \) belonging to party \( k \).
\( X_k \) Local features belonging to party \( k \) of all data samples.
\( y^i \) Label for data sample \( i \).
\( y \) Labels for all data samples.
\( \Theta \) Global model parameters.
\( \theta_k \) Party \( k \)'s local partition of the global model parameters.
\( h_k(\cdot) \) Embedding function for party \( k \).
\( l_i(\cdot) \) Loss function on data sample \( i \).
\( F(\cdot) \) Objective function.

\( g_k(\cdot) \) Stochastic partial derivative of the objective function with respect to \( \theta_k \).
\( B \) Mini-batch of data samples and their associated labels.
\( L, L_k \) Smoothness parameters for \( \nabla_{\theta l} l(\cdot) \) and \( \nabla_{\theta l} l_i(\cdot) \), respectively.
\( \sigma_k \) Variance of party \( k \)'s stochastic partial derivatives.
\( \tau_k^r \) Number of local iterations taken by party \( k \) in round \( r \).
\( \eta_k^r \) Learning rate for party \( k \) in round \( r \).
\( \Phi_k \) Set of embeddings from all parties in round \( r \) except party \( k \).
\( w_k^t \) Weights on party \( k \)'s stochastic partial derivatives at local iteration \( t \) and round \( r \).

I. INTRODUCTION

In modern distributed systems, data are often generated by multiple parties and must remain on-premises to follow regulations (e.g., general data protection regulation (GDPR) [1], Health Insurance Portability and Accountability Act (HIPAA) [2]) and protect sensitive personal information. Federated learning algorithms [3], [4], [5] were introduced to provide methods for training machine-learning models in distributed systems without the need to share raw data between parties. In these algorithms, data-owning parties train models locally and share intermediate information with a parameter server to update the global model. Federated learning has many important applications including personalized healthcare, smart transportation, and predictive energy systems [4], [6].

Vertical Federated Learning (VFL) is an important class of federated learning. In VFL, parties’ local datasets share a common sample ID space but have different feature sets [5], [7], [8]. This is in contrast to horizontal federated learning (HFL), where all parties’ datasets share the same feature space, but each party’s data corresponds to a distinct set of sample IDs [3], [9], [10], [11], [12]. For example, consider a case where a healthcare provider, insurance company, and wearable device manufacturer wish to collaboratively train a model to identify diseases without directly sharing raw user information with one another [13]. These parties store information about the same people, but each party has a distinct set of information for each individual. In VFL, each party typically trains a local feature extractor, while a central server trains a fusion model. The parties periodically exchange intermediate information for updating their local models. We provide an example of a VFL model setup in Fig. 1.

Many VFL algorithms assume parties have identical local training; each party uses the same local optimizer to update its model, typically standard stochastic gradient descent.
supports a variety of local optimizers. In Flex-VFL, rather than the server waiting for all parties to complete a given number of local iterations, each party completes as many iterations as possible within a specified timeout. The parties synchronize with the server after this set amount of time has passed. Our approach serves as a middle-ground between fully synchronous and asynchronous methods: training is not slowed down by stragglers, but parties still synchronize regularly to avoid training with stale information for long periods of time. Furthermore, unlike previously proposed VFL algorithms, Flex-VFL allows parties to customize their local optimizers based on their local data and feature extractor architecture.

Flex-VFL is the first theoretically-verified VFL algorithm that has convergence guarantees when parties use different local optimizers. To represent these optimizers in our analysis, we apply arbitrary weights to party gradients at each local iteration. Our work also provides an adaptive extension to Flex-VFL known as adaptive Flex-VFL: a meta-optimization algorithm that improves the convergence rate. In systems where other jobs may be colocated on the participating devices, party operating speeds may vary over the course of training. In these cases, it is a challenge to choose hyperparameters to accommodate such heterogeneity over time. Based on our theoretical results on convergence in Flex-VFL, the server can gather party information in each global round to optimize the party learning rates. Adaptive Flex-VFL is designed to be robust to heterogeneous and time-varying party speeds and optimizer parameters.

Our main contributions are as follows.

1) We propose Flex-VFL, a VFL algorithm that is robust to heterogeneous, time-varying parties. Flex-VFL supports a large class of SGD variants such as SGD with local momentum, proximal steps, and variable learning rates.

2) We provide convergence analysis for Flex-VFL and show that the error incurred by heterogeneous parties can be offset with the proper choice of learning rates in each round.

3) We propose adaptive Flex-VFL, an extension of Flex-VFL where each party’s learning rate is tailored to its speed and optimizer parameters at each round.

4) We experimentally compare Flex-VFL with other VFL algorithms using both simulated and real-world party operating speeds. We find that Flex-VFL outperforms purely synchronous and asynchronous VFL algorithms, reaching target accuracy up to 4× as fast. We also compare adaptive Flex-VFL and Flex-VFL using real-world time-varying party operating speeds and show up to a 30% time-to-target improvement.

The rest of the article is structured as follows. In Section II, we discuss related work. Section III introduces our sys-
tem model and problem formulation. We present Flex-VFL in Section IV. We analyze the convergence of our algorithm in Section V, and present an optimization method to improve convergence speed by adapting party learning rates in Section VI. In Section VII, we present our experiments. Finally, we conclude in Section VIII.

II. RELATED WORK

Many works in HFL tackle the challenges of high-latency communication with the use of local iterations. These works analyze the effect of local iterations on convergence [9], [12], [37]. Castiglia et al. [38] and Wang et al. [35] both proposed HFL algorithms that support heterogeneous party operating speeds, multi-level local (MLL)-SGD [38] and FedNova [35]. FedNova also supports several common local optimizers, such as SGD with proximal steps and local momentum. Both these works provide an analysis that gives insight into the benefit of supporting these features in federated learning algorithms. However, these previous works in HFL cannot be applied to the VFL case. HFL algorithms rely on distributed gradient descent methods and share model parameter updates, while most VFL algorithms utilize distributed coordinate descent methods and share the output of feature extractors. Thus, the algorithms and analyses for VFL algorithms are fundamentally different.

VFL algorithms are typically variations of coordinate descent methods. Parallel and distributed coordinate descent methods have been proposed [39], [40], [41], but these works depend on a shared memory structure or data sharing between parties, which is not applicable to the VFL setting. Several works have proposed variants of distributed coordinate descent methods for VFL. Many early works do not include support for multiple local iterations [42], [43], [44]. Without support for multiple local iterations, progress in optimization is limited by communication time with the server, which can be costly in cases of high communication latency. Some works proposed synchronous VFL algorithms that support multiple local iterations [7], [8], [15], [17], [18]. However, these algorithms require all parties to use the same standard SGD local optimizer. Xie et al. [34] proposed a synchronous VFL algorithm with multiple local iterations using an ADMM-based optimizer. However, all parties use the same local optimizer in their method, and the fusion network is limited to a linear model, reducing the types of model architectures supported. In addition, all of these algorithms require that all parties run the same number of local iterations, allowing stragglers to create a bottleneck in training time.

Several works propose asynchronous VFL algorithm [14], [16], [26], [27], but the algorithms do not support multiple local iterations. In addition, these algorithms only support SGD local updates. Gu et al. [32] and Zhang et al. [33] proposed several asynchronous VFL algorithms that support local iterations and common optimizers other than SGD. However, the schemes employed require that each party uses a linear model, which limits the use cases of the proposed algorithms. Their algorithms also do not support parties using different local optimizers in the same algorithm execution.

In contrast with previous work, our work jointly provides support and analyzes the effects of party heterogeneity, time-varying speeds, and limited bandwidth in a VFL setting. Specifically, each party can execute a different number of local iterations in each round, and this number of local iterations can change over time. Furthermore, we provide an analysis of our algorithm that includes the impact of this heterogeneity on the convergence rate and convergence error. These features in our algorithm and analysis allow us to model a more realistic VFL scenario and gives us insight into how to adapt party learning rates to mitigate the error introduced by party heterogeneity.

III. SYSTEM MODEL AND PROBLEM FORMULATION

In this section, we present our system model and problem formulation. We consider a system with a set of parties \( K = \{1, \ldots, K\} \). The parties communicate via a central server, forming a hub-and-spoke architecture. The parties and the server may have different operating speeds, and these rates may change over time. We will formalize party operating speeds in Section IV.

Each party \( k \) has a local dataset \( \mathbf{X}_k \in \mathbb{R}^{N \times D_k} \). We let the \( i \)th row of \( \mathbf{X}_k \) be denoted by \( x^i_k \). We assume that these local datasets are aligned, i.e., \( x^i_k \) and \( x^i_j \) for all parties \( k \neq j \) are different features of the same data sample with sample ID \( i \). We let \( \mathbf{X} \in \mathbb{R}^{N \times D} = [\mathbf{X}_1, \ldots, \mathbf{X}_K] \) where \( D = \sum_k D_k \). We can see each \( \mathbf{X}_k \) as a vertical partition of \( \mathbf{X} \). Let the \( i \)th data sample be the \( i \)th row in \( \mathbf{X} \), which we denote as \( x^i \). Let \( y \in \mathbb{R}^{N \times 1} \) be the corresponding labels for the data samples, and let \( y^i \) be the label of the \( i \)th data sample. We assume that the parties and server have a copy of the labels \( y \). We discuss cases where labels are private and only present at a single party in Section IV.

Each party \( k \) holds a local model characterized by an embedding function \( h_k(\cdot) \) and parameterized by model parameters \( \theta_k \in \mathbb{R}^{V_k} \). An embedding function \( h_k \) maps the raw features \( \mathbf{x}_k \) to a representation space, typically of lower dimension. For example, \( h_k \) can be a neural network. Each party may have a different model architecture. We let the \( k \)th embedding of a data sample \( x^i \) be \( h_k(\theta_k; x^i_k) \), the output of party \( k \)’s feature extractor. If \( h_k \) is a neural network, then an embedding is the output of the last layer of the network for a single sample. The server stores a server model with parameters \( \theta_0 \in \mathbb{R}^{V_0} \). The server model is a function of the embeddings from each party and its output is a predicted label \( \hat{y} \). We define the global model parameters as \( \Theta = [\theta_0, \ldots, \theta_K] \in \mathbb{R}^W \) where \( V = \sum_k V_k \); each \( \theta_k \) is a coordinate partition of \( \Theta \). The goal of the parties is to train \( \Theta \). We provide an example of the VFL model structure in Fig. 1. A benefit of the structure of \( \Theta \) is that parties can compute partial derivatives of the loss function by exchanging embeddings rather than exchanging their \( \theta_k \) parameters. Since the size of the embeddings is often much smaller than \( \theta_k \), message sizes can be greatly reduced with this structure. Going forward, for simplicity of notation, we may drop \( x^i \) from \( h_k(\cdot) \) when the context is clear.

The VFL objective is to minimize the following function:

\[
F(\Theta; \mathbf{X}; y) := \frac{1}{N} \sum_{i=1}^N l_i(\theta_0; h_1(\theta_1; x^i_1); \ldots; h_K(\theta_K; x^i_K); y^i)
\]

where \( l_i(\cdot) \) is the loss function for a data sample \( x^i \) and its corresponding label \( y^i \). The loss function \( l_i(\cdot) \) measures the...
error in predicting a label \( y' \) and can be a nonlinear function, such as cross-entropy loss, a support vector machine, or a deep neural network, as shown in Fig. 1.

Let the partial derivative associated with the coordinate partition \( \Theta_k \) be

\[
\nabla_k F (\Theta; \mathbf{x}; y) := \frac{1}{N} \sum_{i=1}^{N} \nabla_{\Theta_k} l_i (\theta_0; h_1 (\theta_1; \mathbf{x}_i'); \ldots; h_K (\theta_K; \mathbf{x}_K'); y').
\]

Let \( B \) be a mini-batch of indices of size \( B \) corresponding to a subset of rows in \( \mathbf{X} \). We let \( \mathbf{X}^B \) be the rows of \( \mathbf{X} \) that correspond to a mini-batch \( B \). Similarly, we let \( y^B \) be the labels that correspond to \( B \). With some abuse of notation, we define \( h_k (\theta_k; \mathbf{X}_k^B) \) to be the set of embeddings for \( B \) for party \( k \). We denote the stochastic partial derivative of the coordinate partition \( \Theta_k \) as

\[
g_k (\theta_0; h_1 (\theta_1; \mathbf{X}_1^B); \ldots; h_K (\theta_K; \mathbf{X}_K^B); y^B) := \frac{1}{B} \sum_{i \in B} \nabla_{\theta_k} l_i (\theta_0; h_1 (\theta_1; \mathbf{x}_i'); \ldots; h_K (\theta_K; \mathbf{x}_K'); y').
\]

With a slight abuse of notation, we let the partial derivatives \( g_k (\theta_0; h_1 (\theta_1; \mathbf{X}_1^B); \ldots; h_K (\theta_K; \mathbf{X}_K^B); y^B) \) be equivalently denoted as \( g_k (\Theta; B) \). We may drop \( \mathbf{X} \) and \( y \) from \( F (\cdot) \) and \( B \) from \( g_k (\cdot) \) when the context is clear.

We make the following standard assumptions for \( l (\cdot) \), \( F (\cdot) \), and \( g_k (\cdot) \) [45], [46], [47].

**Assumption 1: Smoothness:** There exists positive constants \( L < \infty \) and \( L_k < \infty \) for \( k = 0, \ldots, K \) such that for all \( \Theta_1 \in \mathbb{R}^V \), \( \Theta_2 \in \mathbb{R}^V \)

\begin{align}
\| \nabla_{\Theta_1} l_i (\Theta_1) - \nabla_{\Theta_1} l_i (\Theta_2) \| &\leq L \| \Theta_1 - \Theta_2 \| \tag{2} \\
\| \nabla_{\Theta_1} l_i (\Theta_1) - \nabla_{\Theta_1} l_i (\Theta_2) \| &\leq L_k \| \Theta_1 - \Theta_2 \|. \tag{3}
\end{align}

**Assumption 2: Unbiased gradients:** For a mini-batch \( B \), for \( k = 0, \ldots, K \), the stochastic partial derivatives are unbiased, i.e., for all \( \Theta \in \mathbb{R}^V \)

\[
E_B [ g_k (\Theta) ] = \nabla_k F (\Theta). \tag{4}
\]

**Assumption 3: Bounded variance:** There exists constants \( \sigma_k < \infty \) for \( k = 0, \ldots, K \) such that the variances of the stochastic partial derivatives are bounded as

\[
E_B \| \nabla_k F (\Theta) - g_k (\Theta) \|^2 \leq \sigma_k^2 \tag{5}
\]

for a mini-batch \( B \) and for all \( \Theta \in \mathbb{R}^V \).

Assumption 1 bounds how fast the gradient and partial derivatives can change. Assumption 2 requires that the stochastic partial derivatives computed by each party and the server are unbiased estimates of the full-batch partial derivatives. Assumption 2 can be satisfied in practice by ensuring that sample IDs for a mini-batch are chosen at random. Note, that we make no assumption about the distribution of the full dataset \( \mathbf{X} \). Finally, Assumption 3 bounds the variance between the stochastic partial derivatives and full-batch partial derivatives by some constant.

### IV. Algorithm

We now present Flex-VFL, our algorithm for training a global model with distributed, vertically partitioned data in a system with heterogeneous parties. In each **global round** of Flex-VFL, we employ a type of parallel stochastic block coordinate descent. Each party and the server updates its coordinate partition using its local optimizer for one or more **local iterations**. The parties complete as many of these local iterations as possible before a specified timeout. This differs from synchronous VFL algorithms that wait for all parties to complete the same number of local iterations [7], which can lead to a bottleneck when slow parties are present. We assume that all parties participate in each global round and run at least one local iteration before the given timeout. Flex-VFL runs a system with heterogeneous parties. In each global round, the local iteration begins. We let \( \tau_k \) be the number of local gradient descent steps that party \( k \) completes in round \( r \), with \( 1 \leq \tau_k < \infty \). This number of local iterations \( \tau_k \) depends on party \( k \)'s operating speed in global round \( r \) and the local training timeout. Slower parties have smaller \( \tau_k \) values, while faster parties have larger \( \tau_k \) values. Since computation loads

Authorized licensed use limited to the terms of the applicable license agreement with IEEE. Restrictions apply.
Algorithm 1 Flexible Vertical Federated Learning

1: Initialize: $\theta_k^{0,0}$ for all parties $k$ and server model $\theta_0^{0,0}$
2: for $r \leftarrow 0, \ldots, R - 1$ do
3: Select a mini-batch of sample IDs $B' \in \{X, y\}$
4: for $k \leftarrow 1, \ldots, K$ in parallel do
5: Sample features $X_k^{B'}$ corresponding to IDs in $B'$
6: Send $h_k(\theta_k^{r,0}, X_k^{B'})$ to server
7: end for
8: $\Phi' = \{\theta_0^{r,0}, h_1(\theta_1^{r,0}, X_1^{B'}), \ldots, h_K(\theta_K^{r,0}, X_K^{B'})\}$
9: Server sends $\Phi'$ to all parties
10: for $k \leftarrow 0, \ldots, K$ in parallel do
11: \[\w_{k,t} \leftarrow \text{Parties and server run until local training timeout} \]
12: for $t \leftarrow 0, \ldots, t'_k - 1$ do
13: $\theta_k^{r+1,t} \leftarrow \theta_k^{r,t} - \eta_k^t D_k(\Phi'_{k} - \theta_k^{r,t}, \theta_k^{r,t}, y_k^{B'})$
14: end for
15: $\theta_k^{r+1,0} \leftarrow \theta_k^{r,t}$
16: end for
17: end for

1) Classical SGD: In classical SGD, our update rule is

$$D_k(\Phi'_{k}; \theta_k^{0,0}, \ldots, \theta_k^{r,t}; y_k^{B'}) := g_k(\Phi'_{k}; h_k(\theta_k^{r,t}); y_k^{B'})$$

By setting $w_{k,t}^{r,t} = 1$ for all parties, rounds, and local iterations in (8), our update rule becomes classical SGD.

2) SGD With Momentum: SGD with local momentum is where a party resets its momentum buffer to zero at the start of each global round. The update rule for SGD with local momentum is defined as follows:

$$D_k(\Phi'_{k}; \theta_k^{0,0}, \ldots, \theta_k^{r,t}; y_k^{B'}) := u_k^{r,t}$$

where

$$u_k^{r,0} = g_k(\Phi'_{k}; h_k(\theta_k^{0,0}))$$

$u_k^{r,t} = \rho u_k^{r,t-1} + g_k(\Phi'_{k}; h_k(\theta_k^{r,t}))$

where $\rho$ is a tunable parameter. The updates $u_k^{r,t}$ at each local iteration $t$ can be defined as follows:

$$u_k^{r,t} = \rho u_k^{r,t-1} + g_k(\Phi'_{k}; h_k(\theta_k^{r,t-1}))$$

Plugging this into (6), we have

$$\theta_k^{r+1,t} = \theta_k^{r,t} - \eta_k^t \sum_{r=0}^{t} \rho^{t-2} g_k(\Phi'_{k}; h_k(\theta_k^{r,t}))$$

Applying this recursion to our update rule in (7), we have

$$\theta_k^{r+1,t} = \theta_k^{r,t} - \eta_k^t \sum_{r=0}^{t} \rho^{t-2} g_k(\Phi'_{k}; h_k(\theta_k^{r,t}))$$

Thus, in order to represent local momentum, the weights in (8) can be set as follows:

$$w_k^{r,t} = 1 + \rho + \rho^2 + \ldots + \rho^{t-1} = \frac{1 - \rho^{t-1}}{1 - \rho}$$

3) Proximal Updates: As first proposed in FedProx [36], and first applied to a VFL algorithm by Liu et al. [7], one can apply a proximal term by defining a party’s update rule as follows:

$$D_k(\Phi'_{k}; \theta_k^{0,0}, \ldots, \theta_k^{r,t}; y_k^{B'}) := g_k(\Phi'_{k}; h_k(\theta_k^{r,t})) + \mu (\theta_k^{r,t} - \theta_k^{0,0})$$

where $\mu$ is a tunable parameter. Plugging this into (6) we have

$$\theta_k^{r+1,t} = \theta_k^{r,t} - \eta_k^t (g_k(\Phi'_{k}; h_k(\theta_k^{r,t})) + \mu (\theta_k^{r,t} - \theta_k^{0,0}))$$

may change over time, a party’s operating speed, and thus its $t'_k$, may change between each global round. Note that during local iterations, each party only updates its own embedding $h_k(\theta_k^{r,t})$, and uses stale versions of $h_j(\theta_j^{0})$ for all $j \neq k$. Each party also reuses the same mini-batch for $t'_k$ iterations, saving on the communication of a new set of embeddings at each local iteration. We show in Section V that the model converges, even though this stale information is used.

A party $k$’s updates during each local iteration are defined as follows:

$$\theta_k^{r+1,t} = \theta_k^{r,t} - \eta_k^t D_k(\Phi'_{k} - \theta_k^{r,t}, \theta_k^{r,t}, y_k^{B'})$$

where $\Phi'_{k}$ is the set of all embeddings except those from party $k$, and $D_k(\cdot)$ is the local optimizer update rule based on the stochastic partial derivatives from the start of the global round to the current local iteration. The cumulative update to each local model over a set of local iterations is as follows:

$$\theta_k^{r,t'} = \theta_k^{r,0} - \eta_k^t \sum_{t=0}^{t'-1} D_k(\Phi'_{k} - \theta_k^{r,t}, \theta_k^{r,t}, y_k^{B'})$$

We assume that the cumulative update over all local iterations can be rewritten in the following form:

$$\sum_{t=0}^{t'-1} D_k(\Phi'_{k} - \theta_k^{r,t}, \theta_k^{r,t}, y_k^{B'}) := \sum_{t=0}^{t'-1} u_k^{r,t} g_k(\Phi'_{k}; h_k(\theta_k^{r,t}); y_k^{B'})$$

where $w_{k,t}^{r,t} \geq 1$ are weights applied to each gradient update. Note that we do not assume that each index of these sums in (8) are equivalent. By rewriting party update rules in this way, we can analyze a variety of common local optimizers. Below, we present some examples of common local optimizers whose updates can be written in the form of (8).
Subtracting \( \theta_k^{r+1} \) from both sides, we have
\[
\theta_k^{r+1} - \theta_k^0 = (1 - \eta_k^r \mu_k) (\theta_k^r - \theta_k^0) - \eta_k^r g_k (\Phi_k^r; h_k(\theta_k^r)).
\]

Repeatedly applying the recursion on \( \theta_k^r \), we have
\[
\theta_k^{r+1} = \theta_k^0 - \eta_k^r \sum_{i=0}^{r-1} (1 - \eta_k^r \mu_k)^{r-1-i} g_k (\Phi_k^r; h_k(\theta_k^r)).
\]

Thus, the weights in (8) can be set as follows to represent proximal steps in VFL
\[
w_k^{r,t} = (1 - \eta_k^r \mu_k)^{r-1-t+1}.
\]

This method of generalizing to several local optimizers was first shown in the context of HFL by Wang et al. [35]. However, it has yet to be analyzed in the context of VFL, which provides its own unique challenges. We discuss this more in Section V.

A. Communication Cost

The size of messages in Flex-VFL is of note. For each party to compute its partial derivative, every party must exchange its embeddings for the current mini-batch, and the server must send its model \( \theta_0 \) to the parties. Let the size of the \( k \)th embedding for a single data sample be \( O_k \). The total amount of data sent per global round is then \( K (| \theta_0 | + B \sum_k O_k) \).

B. Privacy

HFL algorithms typically share model updates or gradient information in messages. Gradients can potentially leak raw data information, as shown in previous work [48], [49]. However, in Flex-VFL, messages only contain embeddings and each party can only calculate the partial derivatives associated with the server model and its local model. Thus, gradient attacks proposed for HFL cannot be performed on Flex-VFL. Embeddings may be vulnerable to model inversion attacks [50], though these attacks can be mitigated by applying homomorphic encryption [44], [51] or secure multiparty computation [32] to Flex-VFL. As mentioned in Section III, we assume that all parties have access to the labels. There are many practical scenarios where data samples are private between the parties, but the labels are not, such as predicting credit score. However, if labels are private and only present at a single party, Flex-VFL can be augmented using the method proposed by Liu et al. [7], allowing gradient calculation without the need for sharing labels. The analysis in Section V still holds in this case.

V. ANALYSIS

In this section, we provide a convergence analysis of Flex-VFL. To avoid cumbersome notation going forward, we define \( g_k^{r,t} := g_k (\Phi_k^r; h_k (\theta_k^r; X_k^F); y^F) \) and drop \( \Phi_k^r \) from \( D_k (\cdot) \) when the context is clear.

We start by defining a recurrence relation for updates to the global model. Let \( G' \) be the stack of gradient updates in a global round \( r \)
\[
G' = \sum_{t=0}^{r-1} D_0 (\theta_0^{0,t}, \ldots, \theta_0^{t,n}) \cdots \sum_{t=0}^{r-1} D_k (\theta_K^{0,t}, \ldots, \theta_K^{t,f_k}).
\]

We can define our updates to the global model during a global round with the following recurrence relation:
\[
\Theta_k^{r+1,0} = \Theta_k^{r,0} - \eta_k^r G'.
\]

With the help of (10), we can model Flex-VFL as a gradient coordinate descent algorithm and analyze the algorithm convergence in this vein.

We note that each party reuses stale embeddings of the same mini-batch from other parties for multiple local iterations in Algorithm 1: each party \( k \) takes \( \tau_k^r \) descent steps using mini-batch \( B' \) at a global round \( r \). This indicates that the stochastic gradients are not unbiased during local iterations \( t > 0 \). However, using conditional expectation, we can apply Assumption 2 to the gradient calculated at local iteration \( t = 0 \).

If we take expectation over \( B' \), conditioned on the previous models \( \Theta_k^{r,0} \) up to round \( r \), we obtain
\[
\mathbb{E}_{B'} [ g_k^{r,0} \mid \{ \Theta_k^{r,t} \}_{t=0}^{r} ] = \nabla_{\theta_k} F ( \Theta_k^{r-1}; h_k (\theta_k^{r-1}) ) (11)
\]

With the help of (11), we can prove convergence by bounding the difference between the gradient at the start of each global round and those calculated during local iterations.

In particular, we prove the following lemma.

**Lemma 1:** If \( \eta_k^r \leq \frac{1}{2 \tau_k^r L_k} \max_{0 \leq t \leq \tau_k^r} \| w_k^{r,t} \|^2 \), then under Assumptions 1–3 the weighted conditional expected squared norm difference of gradients \( g_k^{r,t} \) and \( g_k^{r,0} \) for a set of \( \tau_k^r \) local iterations is bounded as follows:
\[
\begin{align*}
& \sum_{t=0}^{\tau_k^r-1} w_k^{r,t} \mathbb{E} [ \| g_k^{r,t} - g_k^{r,0} \| ] \\
& \leq 8 (\tau_k^r)^3 \eta_k^r L_k^2 \max_{0 \leq t \leq \tau_k^r - 1} (w_k^{r,t})^3 (\| \nabla_{\theta_k} F (\Theta_k^{r,0}) \|^2 + \sigma_k^2) \quad (12)
\end{align*}
\]

and
\[
\begin{align*}
& \sum_{t=0}^{\tau_k^r-1} (w_k^{r,t})^2 \mathbb{E} [ \| g_k^{r,t} - g_k^{r,0} \| ] \\
& \leq 8 (\tau_k^r)^3 \eta_k^r L_k^2 \max_{0 \leq t \leq \tau_k^r - 1} (w_k^{r,t})^4 (\| \nabla_{\theta_k} F (\Theta_k^{r,0}) \|^2 + \sigma_k^2) \quad (13)
\end{align*}
\]

where \( \mathbb{E}' \) is the expectation taken on the mini-batch \( B' \) conditioned on \( \{ \Theta_k^{r,t} \}_{t=0}^{r} \).

The proof of Lemma 1 can be found in Appendix B. Lemma 1 bounds the error incurred at each party in the stochastic partial derivatives during a set of local iterations as a result of using stale embeddings from other parties, as well as from using the same mini-batch for all local iterations in a single round. The lemma also captures the effect of different local optimizers at each party on the partial derivatives. Using Lemma 1, we can now analyze how this error accumulates over all iterations. Applying our smoothness assumption along with (10) and Lemma 1, we can prove that Flex-VFL converges.
We present our convergence result in the following theorem.

**Theorem 1:** Under Assumptions 1–3, if \( \eta_k^r \) satisfies
\[
\eta_k^r \leq \frac{1}{16 \tau_k^r \max\{L, L_k\}} \max_{0 \leq t < \tau_k^r - 1} w_k^r
\]
then the weighted average squared gradient norm over all parties and \( R \) rounds of Algorithm 1 is bounded by
\[
\frac{1}{S} \sum_{r=0}^{R-1} \sum_{k=0}^{K} \eta_k^r W_k^r E[\|\nabla_k F(\Theta^{0,r})\|^2]
\leq \frac{4}{S} (F(\Theta^{0,0}) - F^{\text{inf}}) + \frac{4L}{S} \sum_{r=0}^{R-1} \sum_{k=0}^{K} (\eta_k^r)^2 W_k^r \max_{0 \leq t < \tau_k^r - 1} w_k^r \tau_k^r \sigma_k^2
\]

where \( F^{\text{inf}} \) is a lower bound on \( F(\cdot) \), \( S = \sum_{r=0}^{R-1} \sum_{k=0}^{K} \eta_k^r W_k^r \), and \( W_k^r = \sum_{t=0}^{\tau_k^r - 1} w_k^r \).

We provide the full proof of Theorem 1 in Appendix C.

The left-hand side of (15) is a weighted average of all the partial derivatives’ norms during the training. As this term approaches zero, Flex-VFL approaches a fixed point. The first term in the right-hand side of (15) is similar to that of distributed gradient descent [45], and is affected by the difference in the initial and final models of the algorithm. The first term is the convergence rate term: as the number of global rounds \( R \) approaches \( \infty \), this first term goes to zero, while the second term, the additive convergence error, remains.

The second term is the error associated with the variance in taking stochastic gradient steps, as well as the error incurred by running multiple local iterations.

If we let \( \eta_k^r = (1/(R \max_{t \leq t_k^r} \tau_k^r)^{1/2}) \) in (15), then we can see the convergence rate of Flex-VFL is \( O((1/(R \max_{t \leq t_k^r} \tau_k^r)^{1/2})) \), the same as other VFL algorithms [7, 14, 16]. This indicates that we can achieve a fast convergence speed despite the error introduced by heterogeneous party speeds and local optimizers.

### A. Effect of Heterogeneous Speeds

We observe that \( S \) appears in the denominator of the first term in (15); a larger value of \( S \) improves the convergence rate of Flex-VFL. This quantity \( S \) depends on the party learning rates \( \eta_k^r \) and the constraint (14) requires that \( \eta_k^r \) be inversely proportional to \( \tau_k^r \), the number of local iterations party \( k \) takes in round \( r \). In addition, \( S \) increases with the sum of weights \( w_k^r \) over \( \tau_k^r \) local iterations. Consider the case where each party employs classical SGD, implying \( w_k^r = 1 \) for all parties over all iterations. In this case, \( S = \sum_{r=0}^{R-1} \sum_{k=0}^{K} \eta_k^r \tau_k^r \). If we let \( \eta_k^r = (1/\tau_k^r) \), then \( S = R(K + 1) \). If we instead consider a fixed learning rate \( \eta_k^r = \eta \) across all parties and iterations, then to satisfy (14), \( \eta = (1/\max_{t \leq t_k^r} \tau_k^r) \). In this case, \( S = \sum_{r=0}^{R-1} \sum_{k=0}^{K} (\tau_k^r / \max_{t \leq t_k^r} \tau_k^r) \leq R(K + 1) \). Thus, choosing the learning rates \( \eta_k^r \) according to each party’s number of local iterations in a round \( \tau_k^r \) can reduce the error in the first term of (15) and improve the convergence rate. However, \( \tau_k^r \) may not always be known in advance. We discuss these findings further in Section VI, where we introduce adaptive Flex-VFL.

### B. Effect of Heterogeneous Optimizers

From Theorem 1, we can see that the first term in (15) decreases with larger local optimizer weights, while the second term increases with larger weights. Note that proper tuning of \( \eta_k^r \) for each party \( k \) can help offset the error introduced by \( \max_{0 \leq t < \tau_k^r - 1} w_k^r \) in the second term. We also note that the constraint (14) requires \( \eta_k^r \) to be inversely proportional to \( \max_{0 \leq t < \tau_k^r - 1} w_k^r \). Thus, there is tension between choosing larger optimizer weights and choosing a larger step size. In cases where \( \eta_k^r \) remains constant while still satisfying (14), Theorem 1 indicates that weights larger than 1, such as when using momentum and proximal steps can improve the convergence rate by decreasing the first term in (15). However, it can also negatively affect the error as \( R \to \infty \) by increasing the second term in (15). If all else is constant, a large \( W_k^r \) is beneficial when \( \sigma_k \) is small. In other words, in cases where stochastic variance is small, our analysis shows a potential benefit of using local optimizers other than classical SGD.

We now introduce some corollaries to study the convergence rate of Flex-VFL. We first consider the case where each party has the same stochastic variance, runs the same number of local iterations, and uses SGD locally.

**Corollary 1:** Suppose \( \sigma_k = \sigma, \tau_k^r = \tau \), and \( \eta_k^r = \eta \) for all parties \( k \) and rounds \( r \). Let \( w_k^r = 1 \) for all rounds \( r \), local iterations \( t \), and parties \( k \). Under Assumptions 1-3, if \( \eta \) satisfies
\[
\eta \leq \frac{1}{16 \tau \max\{L, L_k\}}
\]
then the average squared gradient norm over all parties and \( R \) rounds of Algorithm 1 is bounded by
\[
\frac{1}{R} \sum_{r=0}^{R-1} E[\|\nabla F(\Theta^{0,r})\|^2] \leq \frac{4(F(\Theta^{0,0}) - F^{\text{inf}})}{R\eta \tau (K + 1)} + 4L\eta \tau \sigma^2.
\]

If we let \( \eta = (1/(\tau R)^{1/2}) \) in Corollary 1, then we can see our convergence rate is \( O((1/(\tau R)^{1/2})) \), which is the same as distributed SGD algorithms [9, 45, 52].

We consider a decaying learning rate in the following corollary.

**Corollary 2:** Suppose \( \eta_k^r \leq (1/16 \tau_k^r \max\{L, L_k\}) \max_{0 \leq t < \tau_k^r - 1} w_k^r \), and suppose that \( \sum_{t=0}^{\infty} (\eta_k^r)^2 < \infty \) for all parties \( k \). Then, under Assumptions 1-3, the left-hand side of (15) goes to zero as \( R \) approaches \( \infty \).

Corollary 2 states that given a sequence of learning rates that are not summable, but square summable, then Algorithm 1 achieves convergence to a fixed point. One possible choice for learning rates to satisfy these conditions is by diminishing \( \eta_k^r \) at a rate of \( O((1/r)) \) where \( r \) is the current global round. This is a standard step size requirement of SGD algorithms for nonconvex objectives [45].

### VI. Adaptive Extension

In this section, we present an adaptive extension to Flex-VFL, which we call adaptive Flex-VFL. One can think of adaptive Flex-VFL as a meta-optimization algorithm where the server keeps track of party operating speeds and local
Algorithm 2 Adaptive Flexible Vertical Federated Learning

1: Initialize: $\theta^{1,0}_k$ for all parties $k$ and server model $\theta^{1,0}_0$
2: Initialize: $A_k$ and $\eta^1_k = \frac{A_k}{\overline{v}_k}$ for all parties
3: for $r \leftarrow 1, \ldots, R$ do
4: \hspace{0.5cm} Sample a mini-batch $B' \in \{X, y\}$
5: \hspace{0.5cm} for $k \leftarrow 1, \ldots, K$ in parallel do
6: \hspace{1.0cm} Sample features $X^B_k$ corresponding to IDs in $B'$
7: \hspace{1.0cm} Send $h_k(\theta^{r,0}_k, X^B_k)$ to server
8: \hspace{0.5cm} end for
9: $\Phi' := \{\theta^{1,0}_0, h_1(\theta^{r,0}_1, \ldots, h_K(\theta^{r,0}_K)\}$
10: Server sends $\Phi'$ to all parties
11: for $k \leftarrow 0, \ldots, K$ in parallel do
12: \hspace{0.5cm} for $t \leftarrow 0, \ldots, \tau^t_k - 1$ do
13: \hspace{1.0cm} $\theta^{r,t+1}_k = \theta^{r,t}_k - \eta^t_k \frac{D_k(\Phi^{r,t}_k; h_k(\theta^{r,t}_k), y^B)}{\overline{v}_k}$
14: \hspace{0.5cm} end for
15: \hspace{0.5cm} Send $\tau^t_k$ and $\overline{v}_k$ to server
16: \hspace{0.5cm} end for
17: Server sets $\eta^{r+1}_k = \alpha \frac{A_k}{\tau^r_k \overline{v}_k}$ for all parties
18: end for

We observe that learning rate $\eta^t_k$ can be determined during each global round in order to choose the best learning rates for convergence speed.

Let $\overline{w}^t_k := \max_{0 \leq t < \tau^t_k - 1} w^{r,t}_k$. According to Theorem 1, the learning rate $\eta^t_k$ at a global round $r$ for party $k$ must be inversely proportional to the number of local iterations $\tau^t_k$ and the largest weight applied during local iterations $\overline{w}^t_k$. We also know that the first term of (15) is the main contributor to the convergence floor, mostly affected by the stochastic variance. If the bound (14) on the learning rates $\eta^t_k$ holds, the second term's effect is minimal. Therefore, based on our analysis, the natural improvement to Flex-VFL is to maximize $\overline{w}^t_k$ subject to (14) in each global round, tailoring each party's learning rate to their specific operating rate and local optimizer parameters.

Adaptive Flex-VFL is outlined in Algorithm 2. For the first global round, $\tau^0_k$ for each party is estimated, either with prior knowledge of the party operating speeds or by running a dummy global round. Each party can communicate $\overline{w}^0_k$ to the server before the start of training. All $\eta^1_k$ are initialized to some $(A_k / \tau^0_k \overline{w}^0_k)$, where $A_k$ is chosen small enough to satisfy (14). At the end of each global round, the server gathers information about how many local iterations each party took, $\tau^t_k$, and their maximum weight applied to gradients during local iterations, $\overline{w}^t_k$. We let $\eta^{r+1}_k = (A_k / \tau^r_k \overline{w}^r_k)$. If we assume that for each party $k$, $\tau^t_k$ and $\overline{w}^t_k$ do not change too rapidly across global rounds, the server can accurately estimate the appropriate learning rate to assign to each party that will maximize the convergence rate.

We now define more formal conditions under which Adaptive Flex-VFL is guaranteed to converge according to Theorem 1. Suppose the maximum rate of change of $\tau^t_k \overline{w}^t_k$ from a round $r$ to $r + 1$ is $\alpha$

$$\frac{\tau^r \overline{w}^r}{\tau^{r+1} \overline{w}^{r+1}} \leq \alpha.$$  (18)

Note that $\eta^{r+1}_k = (A_k / \tau^r \overline{w}^r_k)$. In order to satisfy constraint (14), we need $\eta^{r+1}_k \leq (1/16 \max\{L_k, L_k\} \tau^{r+1} \overline{w}^{r+1}$. Therefore, we need $A_k$ to be chosen such that

$$A_k \leq \frac{1}{16 \max\{L_k, L_k\}}$$  (19)

and

$$A_k \leq \alpha \left(\frac{1}{16 \max\{L_k, L_k\}}\right).$$  (20)

Thus, in order for adaptive Flex-VFL to satisfy (14) in all rounds of training, $A_k$ must satisfy (20). We note that in practice, as we show in Section VII, adaptive Flex-VFL can show a clear improvement over Flex-VFL with a fixed learning rate.

VII. EXPERIMENTS

Next, we present experiments to compare Flex-VFL with synchronous and asynchronous VFL algorithms, and to observe the effect of the adaptive extension to Flex-VFL in practice.\(^1\)

A. Datasets and Experiment Setup

We utilize three datasets for our experiments: the MOSEI dataset [53], the ImageNet dataset [54], and the ModelNet40 dataset [55]. For each dataset and VFL algorithm, we performed a grid search to choose the best learning rate and regularization parameters (where applicable); we trained each algorithm with different hyperparameters for 100 epochs and chose the hyperparameters with the lowest training loss.

1) MOSEI: CMU-MOSEI is a multimodal dataset for sentiment analysis. The dataset consists of 23 and 453 sentences from YouTube videos giving opinions on various topics. The dataset includes video, audio, and text data, and each sentence is labeled with sentiment values, scoring the positivity or negativity of the sentence. For our experiment setup, we consider a case with three parties, where each party stores one type of data. The parties with video and audio train local LSTMs, and the party with text trains a BERT model. The server model consists of a three-layer fully-connected neural network. We use an L1 loss function with L2 regularization. The parties train using SGD with a batch size of 50. For the video and audio parties the learning rate was selected from $[0.01, 0.005, 0.001, 0.0005]$. For the text party the learning rate was selected from $[5 \times 10^{-5}, 1 \times 10^{-5}, 5 \times 10^{-6}]$. The regularization coefficient was selected from $[0, 10^{-3}, 10^{-4}, 10^{-5}]$.

2) ImageNet100: The ImageNet dataset consists of images from several classes of objects. In our experiments, we randomly choose 100 classes from the ImageNet dataset (ImageNet100), consisting of about 130,000 images. We consider a case with four parties, each storing a quadrant of each image. Each party trains ResNet50 locally and the server trains a single fully connected layer. We use a cross-entropy loss function with L2 regularization. The parties train using a batch size of 256. Each party trains using SGD with local momentum with $\rho = 0.9$. The initial learning rate for each party was selected from $[0.3, 0.08, 0.03, 0.008, 0.003]$ and the regularization coefficient was selected from $[0, 10^{-3}, 10^{-4}, 10^{-5}]$. The learning rate decays by a factor of 10 every $\sim 75,000$ local iterations.

\(^1\)Code for experiments available at https://github.com/rpi-ual/flex-vfl.
3) **ModelNet40**: The ModelNet40 dataset are images of CAD models with 40 classes of objects, each with 12 different camera views. In our experiments, we consider a setup with 12 parties, each with a single view of each CAD model. Each party trains ResNet50 locally and the server trains a single fully connected layer. We use a cross-entropy loss function for training. The parties train using a batch size of 64. Each party trains using SGD with local momentum with $\rho = 0.9$. The learning rate for each party was selected from \{10^{-3}, 5 \times 10^{-4}, 1 \times 10^{-4}, 5 \times 10^{-5}\}.

4) **Baselines**: We compare Flex-VFL with synchronous and asynchronous VFL methods. We limit our comparisons to those that support arbitrary party feature extractors and arbitrary server fusion networks, as well as multiple local iterations \[7, 8, 14\].

1) **Sync-VFL** Synchronous VFL is a special case of Flex-VFL where all parties run the same number of local iterations, regardless of party operating speeds. When all parties use standard SGD and the server model has no trainable parameters, Sync-VFL is equivalent to the VFL algorithm proposed by Liu et al. \[7\]. For our experiments, we consider two cases of Sync-VFL: Sync-Min-VFL and Sync-Max-VFL.

   a) **Sync-Min-VFL**: Flex-VFL and Sync-VFL use the same choice of local training timeout, meaning each party in Sync-VFL runs $\tau' = \min_k \tau_k^r$ descent steps before synchronizing.

   b) **Sync-Max-VFL**: Sync-VFL waits for all parties to run $\tau' = \max_k \tau_k^r$ descent steps before synchronizing. This means that Sync-Max-VFL ensures that all parties run the same number of iterations as the fastest party in Flex-VFL, which extends the duration of a global round to accommodate the slowest party.

2) **V AFL**: In V AFL \[14\], each party calculates its embeddings for a randomly selected mini-batch. The party then immediately exchanges information with the server to update both the server model and the party’s model parameters. Parties may take different lengths of time to execute their individual training steps, and so the model updates are asynchronous.

3) **P-BCD**: We also include a baseline for Flex-VFL where we set $\tau_k^r = 1$ for all $k$ and $r$. This is equivalent to parallel block coordinate descent (P-BCD) \[56\].

5) **Time Units**: In each of our experiments, we measure time-to-target in terms of simulated time units. We define the communication time with the server, the computation time for each party, and the local training timeout to be used by Flex-VFL and Sync-VFL in terms of these time units. The time units taken to complete a local iteration and the timeout chosen informs how many local iterations each party will perform in Flex-VFL and Sync-VFL. For example, suppose two parties take 5 and 10 time units to complete a local iteration, respectively, and the timeout is set to 20. In Sync-VFL, both parties will complete two local iterations. In Flex-VFL, the first party will complete four local iterations while the second party completes two.

### Table I

| Video/Audio Optimizers | Text Optimizer | Time units ($\times 10^3$) to reach target |
|------------------------|----------------|------------------------------------------|
| SGD                    | SGD            | 64.55 ± 11.74                           |
| Local Momentum         | SGD            | 35.21 ± 11.74                           |
| SGD                    | Local Momentum | 29.34 ± 0.00                            |
| Local Momentum         | Local Momentum | 41.08 ± 14.37                           |

**B. Heterogeneous Optimizers**

We first study the effect of training with heterogeneous optimizers. We run Flex-VFL and train on the MOSEI dataset with parties either using standard SGD or SGD with local momentum. We consider a case where all parties and the server have the same operating rate. We let the computation time for each party be 1 time unit and set the timeout for each global round to be 20 time units. Thus, each party runs 20 local iterations in each global round. We let the communication latency be 10 time units. We measure the average time taken to achieve a target mean-square error (MAE) over 5 runs. The results are given in Table I. The best optimizer combination over the five runs is using standard SGD at the video and audio parties that train LSTMs, while the text party uses SGD with local momentum to train the BERT model. Thus, we can see a benefit from choosing different optimizers at each party depending on the local model architecture. We use this combination of optimizers for the rest of the experiments with the MOSEI dataset.

**C. Fixed Operating Speeds**

We next study the setting where party operating speeds are heterogeneous but remain fixed throughout training. For each dataset, we define the speed of a party by how many time units it takes for it to complete a local iteration, as well as a local training timeout to be used by Flex-VFL and Sync-VFL.

For MOSEI, we set the timeout to 20 time units, and set the operating rates such that the parties storing video, audio, and text take 5, 10, and 15 local iterations before the timeout, respectively, while the server completes 20 local iterations before the timeout. For ImageNet100, we set the timeout to 10 time units, and set the operating rates such that the four parties take 2, 4, 6, and 8 local iterations before the timeout, respectively, while the server completes 10 local iterations. For ModelNet40, we set the timeout to 20 time units, and set the operating rates such that the server takes 20 local iterations before the timeout, and let groups of 3 parties each take 5, 10, 15, and 20 local iterations, respectively. These operating rates are chosen such that there are an equal number of stragglers, medium speed parties, and fast parties in the system.

We simulate three communication network settings, representing cases of different ratios of computation time versus communication time. We denote the round-trip message latency with the server as $t_{comm}$ and we let the time for the fastest party to complete a local iteration be 1 time unit. In the first setting, we assume communication latency with
the server is very low, equal to the computation time of a single local iteration, and let \( t_{\text{comm}} = 1 \) unit. In the second setting, we consider a case where communication time starts to outweigh the computation time of a single local iteration and set \( t_{\text{comm}} = 10 \) units. In this case, communication with the server takes 10 local iterations at the fastest party. For the final setting, we consider the case where there is very high communication latency, setting \( t_{\text{comm}} = 50 \) units. Such high communication latency can occur when parties are globally distributed.

In Table II, we show the time units it takes to reach a target test accuracy. For the MOSEI dataset, we let the target be 0.65 MAE, a common measure of performance for the dataset. We can see that for the MOSEI dataset, P-BCD and VFL perform well when communication latency is low. However, as communication latency increases, Flex-VFL is able to reach the target MAE twice as fast as Sync-Min-VFL and P-BCD, and four times as fast as VFL. For the ImageNet100 dataset, the target is set to 60% top-5 accuracy. In this case, P-BCD performs well when communication latency is low. As the communication latency increases, however, Flex-VFL outperforms P-BCD, benefitting from local iterations and saving on overall communication. Finally, for the ModelNet40 dataset, the target is set to 70%. Here, Flex-VFL always performs better than the other algorithms, regardless of communication latency. In our experiments with ModelNet40, there are a larger number of parties than in the other datasets. This causes the heterogeneity of party operating speeds to have a greater effect on time-to-target accuracy. With the chosen distribution of party operating speeds, Flex-VFL reaches a target accuracy up to four times as fast as other VFL algorithms. In Section VII-D, we consider how different operating speed distributions can affect the performance of each VFL algorithm.

### D. Uniform Versus Average Operating Rates

For our next experiment, we again consider fixed operating speeds, and now focus on the effect of operating rate distribution. Specifically, we compare an evenly distributed case of party operating speeds to a typical case, and show how Flex-VFL, Sync-VFL, VFL, and P-BCD perform. For these experiments, we use the ModelNet40 dataset and consider two distributions of party operating speeds. For Flex-VFL and Sync-VFL, we use a local training timeout of 20 time units. The first distribution is the same as the previous experiment: the timeout is set to 20 time units, and the operating rates are set such that the server takes 20 local iterations per round, and groups of three parties each take 5, 10, 15, and 20 local iterations, respectively. For determining a realistic typical case of party operating speeds, we use Google’s Cluster Data [57], a dataset of workload traces running on Google compute cells. We choose 13 random traces from the dataset and take the average CPU usage for each to determine the operating speed of parties for these experiments. For a party \( k \), we let the time to complete a local iteration be \( (1 - c_k)^{-1} \) for all \( r \) rounds, where \( c_k \) is the \( k \)th machine’s average CPU utilization. Since the timeout is 20 time units, \( \tau_r^k = 20 \cdot (1 - c_k) \) for all \( r \) rounds.

In Fig. 2, we present the results of training with the different operating rate distributions. We plot the top-5 test accuracy plotted against time units for ModelNet40 dataset with uniform and average distribution of party operating speeds. The solid lines are the mean of 5 runs, while the shaded region represents the standard deviation. (a) \( t_{\text{comm}} = 1 \) with evenly distributed operating speeds. (b) \( t_{\text{comm}} = 50 \) with evenly distributed operating speeds. (c) \( t_{\text{comm}} = 1 \) with Google cluster operating rate distribution. (d) \( t_{\text{comm}} = 50 \) with Google cluster operating rate distribution.
A we use a timeout of 10 time units and set on the ModelNet40 and MOSEI datasets. For ModelNet40, we use a timeout of 10 time units and set \( \tau \) for each party. We set \( \tau \) to be the shortest time to complete a local iteration be \( k \) and \( \tau \) to be the shortest time over a short time interval. For a party \( F \). Variable Operating Speeds

Table III shows the time taken for Sync-VFL and Flex-VFL to achieve 70% top-5 accuracy on the ModelNet40 dataset for different timeouts. We see this same trend with Sync-Max-VFL when increasing the number of local iterations. This is in line with previous works that explore the effect of local iterations on the error incurred by local iterations outweighs the benefits of convergence rate [3], [7], [58]. Increasing the number of local iterations can outperform both synchronous and asynchronous VFL algorithms, reaching a target accuracy up to 4\( \times \) faster than other VFL algorithms. Our experiments also indicate that Flex-VFL can outperform both synchronous and asynchronous VFL algorithms, reaching a target accuracy up to 4\( \times \) faster than other VFL algorithms.

### Table III

| Local Training Period Timeout | Flex-VFL | Sync-VFL |
|-------------------------------|----------|----------|
| Time units (\( \times 10^3 \))  | Per Round | Time units (\( \times 10^3 \)) |
| 20 units                      | 36.04 ± 3.46 | 20 iterations | 94.25 ± 3.54 |
| 25 units                      | 30.49 ± 3.70 | 25 iterations | 72.97 ± 3.54 |
| 30 units                      | 24.02 ± 1.85 | 30 iterations | 60.98 ± 3.79 |
| 35 units                      | 22.18 ± 1.85 | 35 iterations | 58.21 ± 3.54 |
| 40 units                      | 33.26 ± 1.85 | 40 iterations | 85.93 ± 3.54 |

### Fig. 3.

CPU utilization over time of four machines from the Google cluster workload dataset.

### Fig. 4.

Top-5 test accuracy and test MAE of adaptive and static methods of Flex-VFL training on (a) ModelNet40 and (b) MOSEI datasets, respectively.

- Adaptive Flex-VFL, on the other hand, chooses \( \eta_k \) in each global round based on the previous rounds values of \( \tau_k \) and \( w_k \), as described in Algorithm 2.

E. Effect of Local Training Timeout

In this section, we explore the effect of different timeouts with fixed operating rates on the time-to-target accuracy of Flex-VFL. For these experiments, we use the ModelNet40 dataset and the even distribution of operating rates from previous experiments. We let the communication time be 10 time units. We also include results from Sync-Max-VFL as a baseline. Recall that Sync-Max-VFL ensures all parties run the same number of local iterations as the fastest party does in Flex-VFL. For example, if the fastest party in Flex-VFL runs 20 local iterations, then all parties in Sync-Max-VFL run 20 local iterations.

Table III shows the time taken for Sync-VFL and Flex-VFL to achieve 70% top-5 accuracy on the ModelNet40 dataset for different timeouts. We see that for Flex-VFL, increasing the timeout improves the time-to-target, up until a timeout of 40 time units. We see this same trend with Sync-VFL when changing the number of local iterations. This is in line with previous works that explore the effect of local iterations on convergence rate [3], [7], [58]. Increasing the number of local iterations improves time-to-target up to a certain point, where the error incurred by local iterations outweighs the benefits of the convergence rate. We can also see in Table III that Flex-VFL outperforms Sync-VFL regardless of the timeout.

### F. Variable Operating Speeds

We next investigate the setting where party operating speeds change during training. To model realistic changes in these rates, we again use Google’s Cluster Data [57]. In Fig. 3, we plot the CPU utilization of 4 randomly chosen machine traces over a short time interval. For a party \( k \), we let the time to complete a local iteration be \( (1 - c_k) \) and \( c_k \) is the \( k \)th machine’s CPU utilization at global round \( r \). We set \( c_k^r = t_o \cdot (1 - c_k) \) for round \( r \), where \( t_o \) is the local training.

We compare Flex-VFL and Adaptive Flex-VFL and train on the ModelNet40 and MOSEI datasets. For ModelNet40, we use a timeout of 10 time units and set \( A_k = 0.0001 \), and for MOSEI, we use a timeout of 20 time units and set

### VIII. Conclusion

We proposed Flex-VFL, a VFL algorithm that learns on distributed, vertically-partitioned data in a system with heterogeneous parties. We analyzed Flex-VFL and showed the benefit of optimizers that utilize momentum or proximal steps in VFL settings. We also showed that convergence requires that each party’s learning rate is tailored to its operating speed and local optimizer. Based on this observation, we proposed adaptive Flex-VFL, which optimizes party learning rates at each global round based on party operating speeds and optimizer parameters. In our experiments, we demonstrated that Flex-VFL can outperform both synchronous and asynchronous VFL algorithms, reaching a target accuracy up to 4\( \times \) faster than other VFL algorithms. Our experiments also indicate that
Flex-VFL is often the best overall choice of VFL algorithms when it’s necessary to be flexible with high communication latency and different party operating rate distribution. We also provided experiment results comparing adaptive Flex-VFL and Flex-VFL using real-world party operating speeds. We found that adaptive Flex-VFL can improve time-to-target accuracy by 30% over Flex-VFL. In future work, we will explore the impact of partial participation of VFL parties on algorithm performance.

**APPENDIX**

In this section, we provide our full proof of Theorem 1. We start by introducing some additional notation and providing the proof of Lemma 1.

**A. Additional Notation**

We define

$$\gamma_{k,j}^{r,t} = \begin{cases} \theta_{j}^{r,t} & k = j \\ \theta_{j}^{r,0} & \text{otherwise} \end{cases}$$  \hspace{1cm} (21)



to represent party k’s view of party j’s model in round r and iteration t. We define the column vector $\Gamma_{k}^{r,t} = [\gamma_{k,0}^{r,t} , \ldots , \gamma_{k,k}^{r,t}]$ to be party k’s view of the system model in round r and iteration t.

**B. Proof of Lemma 1**

We now prove Lemma 1, stated in Section V.

**Proof:** We start by bounding the expected squared norm difference between gradients at the start of the global round r and local iteration t

$$E^t[\|g_k^{r,t} - g_k^{r,0}\|^2]\leq (1+n)E^t[\|g_k^{r,t} - g_k^{r,0}\|^2] + \left(1 + \frac{1}{n}\right)\|g_k^{r,0}\|^2 - \|g_k^{r,t}\|^2 \hspace{1cm} (23)$$

where (23) follows from the fact that $(X+Y)^2 \leq (1+n)X^2 + (1+(1/n))Y^2$ for some positive n.

Applying Assumption 1 to the first term in (23), we have

$$E^t[\|g_k^{r,t} - g_k^{r,0}\|^2] \leq (1+n)L_k^2E^t[\|\Gamma_k^{r,t} - \Gamma_k^{r,0}\|^2] + \left(1 + \frac{1}{n}\right)\|g_k^{r,0}\|^2 \hspace{1cm} (24)$$

$$= (1+n)(\eta_k^t)^2L_k^2\left(w_k^{r,t}\right)^2E^t[\|g_k^{r,t-1}\|^2] + \left(1 + \frac{1}{n}\right)\|g_k^{r,0}\|^2 \hspace{1cm} (25)$$

where (25) follows from the update rule $\theta_k^{r,t} = \theta_k^{r,t-1} - \eta_k^tw_k^{r,t-1}g_k^{r,t-1}$.

We now add and subtract $g_k^{r,0}$ to the first term in (25)

$$E^t[\|g_k^{r,t} - g_k^{r,0}\|^2] \leq (1+n)(\eta_k^t)^2L_k^2\left(w_k^{r,t}\right)^2E^t[\|g_k^{r,t-1} + g_k^{r,0}\|^2] + \left(1 + \frac{1}{n}\right)\|g_k^{r,0}\|^2 \hspace{1cm} (26)$$

$$\leq 2(1+n)(\eta_k^t)^2L_k^2\left(w_k^{r,t}\right)^2E^t[\|g_k^{r,t-1} - g_k^{r,0}\|^2] + 2(1+n)(\eta_k^t)^2L_k^2\left(w_k^{r,t}\right)^2E^t[\|g_k^{r,0}\|^2] + \left(1 + \frac{1}{n}\right)\|g_k^{r,0}\|^2 \hspace{1cm} (27)$$

If we let $n = \tau_k^t$, we can bound (27) further

$$E^t[\|g_k^{r,t} - g_k^{r,0}\|^2] \leq 2(1+\tau_k^t)(\eta_k^t)^2L_k^2\left(w_k^{r,t}\right)^2E^t[\|g_k^{r,t-1} - g_k^{r,0}\|^2] + 2(1+\tau_k^t)(\eta_k^t)^2L_k^2\left(w_k^{r,t}\right)^2E^t[\|g_k^{r,0}\|^2] + \left(1 + \frac{1}{\tau_k^t}\right)\|g_k^{r,0}\|^2 \hspace{1cm} (28)$$

Let $\eta_k^t \leq (1/2\tau_k^t)L_k \max_{0 \leq \tau \leq \tau - 1} w_k^{r,t}$. We bound (28) as follows:

$$E^t[\|g_k^{r,t} - g_k^{r,0}\|^2] \leq \left(1 + \frac{2}{\tau_k^t}\right)\|g_k^{r,t-1} - g_k^{r,0}\|^2 + 2(1+\tau_k^t)(\eta_k^t)^2L_k^2 \max_{0 \leq \tau \leq \tau - 1} w_k^{r,t}E^t[\|g_k^{r,0}\|^2] \hspace{1cm} (29)$$

We define the following notation for simplicity:

$$A^{r,t} := E^t[\|g_k^{r,t} - g_k^{r,0}\|^2] \hspace{1cm} (30)$$

$$B := 2(1+\tau_k^t)(\eta_k^t)^2L_k^2 \max_{0 \leq \tau \leq \tau - 1} w_k^{r,t}E^t[\|g_k^{r,0}\|^2] \hspace{1cm} (31)$$

$$C := \left(1 + \frac{2}{\tau_k^t}\right) \hspace{1cm} (32)$$

Note that we have shown that $A^{r,t} \leq CA^{r,t-1} + B$. Utilizing this bound, we can also show that

$$A^{r,1} \leq CA^{r,0} + B \hspace{1cm} (33)$$

$$A^{r,2} \leq C^2A^{r,0} + CB + B \hspace{1cm} (34)$$

$$A^{r,3} \leq C^3A^{r,0} + C^2B + CB + B \hspace{1cm} (35)$$

$$\vdots$$

$$A^{r,t} \leq C^tA^{r,0} + B \sum_{\tau_i = 0}^{t-1} C^{t-\tau_i} \hspace{1cm} (36)$$

Note that $A^{r,0} = E^t[\|g_k^{r,0} - g_k^{r,0}\|^2] = 0$. It is left to bound the second term in (36) over the set of local iterations

$$\sum_{t=0}^{\tau_k^t-1} w_k^{r,t}B \sum_{\tau_i = 0}^{t-1} C^{t-\tau_i} \hspace{1cm} (37)$$

$$= B \sum_{t=0}^{\tau_k^t-1} w_k^{r,t}\left(\frac{C^t - 1}{C - 1}\right) \hspace{1cm} (38)$$

$$= \frac{B}{C - 1} \sum_{t=0}^{\tau_k^t-1} w_k^{r,t}\left(\frac{C^t - 1}{C - 1}\right) \hspace{1cm} (39)$$

$$= \frac{B}{C - 1} \max_{0 \leq \tau \leq \tau - 1} w_k^{r,t}\left(\frac{C^\tau - 1}{C - 1} - \frac{1}{C - 1}\right) \hspace{1cm} (40)$$

Authorized licensed use limited to the terms of the applicable license agreement with IEEE. Restrictions apply.
where \( \tau \) follows from the fact that

\[
C. \text{ Proof of Theorem 1}
\]

Applying our smoothness assumption, given in Assumption 1

\[
F(\Theta^{t+1,0}) - F(\Theta^{t,0})
\]

\[
\leq (\nabla F(\Theta^{t,0}), \Theta^{t+1,0} - \Theta^{t,0}) + \frac{L}{2} ||\Theta^{t+1,0} - \Theta^{t,0}||^2
\]

\[
\leq -\sum_{k=0}^{K} \eta_k^t \sum_{i=0}^{\tau_k^t-1} w_{k}^{t,i} (\nabla_k F(\Theta^{t,0}), g_k^{t,i})
\]

\[
+ \frac{L}{2} \sum_{k=0}^{K} \eta_k^t \sum_{i=0}^{\tau_k^t-1} (w_{k}^{t,i})^2 ||g_k^{t,i}||^2
\]

where (48) follows from the fact that \( (\sum_{n=1}^{N} x_n)^2 \leq N \sum_{n=1}^{N} x_n^2 \).

We bound the first term in (48)

\[
- (\nabla_k F(\Theta^{t,0}), g_k^{t,i})
\]

\[
= - \left( (\nabla_k F(\Theta^{t,0}), g_k^{t,i} - g_k^{t,0}) + (\nabla_k F(\Theta^{t,0}), g_k^{t,0}) \right)
\]

\[
\leq \frac{1}{2} ||\nabla_k F(\Theta^{t,0})||^2 + \frac{1}{2} ||g_k^{t,i} - g_k^{t,0}||^2
\]

\[+ (\nabla_k F(\Theta^{t,0}), g_k^{t,0})
\]

where (50) follows from the fact that \( A \cdot B = (1/2)A^2 + (1/2)B^2 - (1/2)(A - B)^2 \leq (1/2)A^2 + (1/2)B^2 \).

We bound the second term in (48)

\[
||g_k^{t,i}||^2 = ||g_k^{t,i} - g_k^{t,0} + g_k^{t,0}||^2
\]

\[
\leq 2 ||g_k^{t,i} - g_k^{t,0}||^2 + ||g_k^{t,0}||^2
\]

Plugging (50) and (52) into (48), and applying the expectation \( E'^t \) to both sides

\[
E'[F(\Theta^{t+1,0})] - F(\Theta^{t,0})
\]

\[
\leq -\frac{1}{2} \sum_{k=0}^{K} \eta_k^t \sum_{i=0}^{\tau_k^t-1} w_{k}^{t,i} (1 - 2\eta_k^t L \tau_k^t w_{k}^{t,i}) ||\nabla_k F(\Theta^{t,0})||^2
\]

\[
+ \frac{1}{2} \sum_{k=0}^{K} \eta_k^t \sum_{i=0}^{\tau_k^t-1} (w_{k}^{t,i})^2 (E'[||g_k^{t,i} - g_k^{t,0}||^2] + \sigma_k^t)
\]

\[
= -\frac{1}{2} \sum_{k=0}^{K} \eta_k^t \sum_{i=0}^{\tau_k^t-1} w_{k}^{t,i} (1 - 2\eta_k^t L \tau_k^t w_{k}^{t,i}) ||\nabla_k F(\Theta^{t,0})||^2
\]

\[
+ \frac{1}{2} \sum_{k=0}^{K} \eta_k^t \sum_{i=0}^{\tau_k^t-1} (w_{k}^{t,i})^2 (E'[||g_k^{t,i} - g_k^{t,0}||^2] + \sigma_k^t)
\]

\[
+ L \sum_{k=0}^{K} (\eta_k^t)^3 \tau_k^t \sum_{i=0}^{\tau_k^t-1} (w_{k}^{t,i})^2 \sigma_k^t.
\]

Applying Lemma 1 to (54)

\[
E'[F(\Theta^{t+1,0})] - F(\Theta^{t,0})
\]

\[
\leq -\frac{1}{2} \sum_{k=0}^{K} \eta_k^t \sum_{i=0}^{\tau_k^t-1} w_{k}^{t,i} (1 - \frac{1}{4} L \tau_k^t w_{k}^{t,i}) ||\nabla_k F(\Theta^{t,0})||^2
\]

\[
+ \frac{1}{2} \sum_{k=0}^{K} \eta_k^t \sum_{i=0}^{\tau_k^t-1} (w_{k}^{t,i})^2 (E'[||g_k^{t,i} - g_k^{t,0}||^2] + \sigma_k^t)
\]

\[
+ L \sum_{k=0}^{K} (\eta_k^t)^3 \tau_k^t \sum_{i=0}^{\tau_k^t-1} (w_{k}^{t,i})^2 \sigma_k^t.
\]
After some rearranging of terms

\[
\sum_{k=0}^{K} \sum_{r=0}^{t-1} u_k^{r+1} \| \nabla_k F(\Theta^{(k)}) \|^2 \\
\leq 4(F(\Theta^{(0)}) - E[F(\Theta^{(R)})]) + 4L \sum_{k=0}^{K} \max_{0 \leq t \leq t-1} u_k^{r+1} \sum_{r=0}^{t-1} u_k^{r+1}.
\]

(58)

Next, we average over all global rounds \( r = 0, \ldots, R - 1 \) and take total expectation

\[
\frac{1}{R} \sum_{r=0}^{R-1} \sum_{k=0}^{K} u_k^{r+1} \max_{0 \leq t \leq t-1} u_k^{r+1} \sum_{r=0}^{t-1} u_k^{r+1}.
\]

(59)

In order to get our weighted average on the left-hand side, we divide through by \( \sum_{k=0}^{K} \sum_{r=0}^{t-1} u_k^{r+1} \), which completes the proof of Theorem 1.

REFERENCES

[1] P. Voigt and A. Von dem Bussche, “The EU general data protection regulation (GDPR),” in A Practical Guide, vol. 10, 1st ed. Cham, Switzerland: Springer, 2017.

[2] Health Insurance Portability and Accountability Act. Accessed: Jun. 12, 2020. [Online]. Available: https://www.hhs.gov/hipaa/index.html

[3] B. McMahen, E. Moore, D. Ramage, S. Hampson, and B. A. Y. Arcas, “Communication-efficient learning of deep networks from decentralized data,” in Proc. 20th Int. Conf. Artif. Intell., 2017, pp. 1273–1282.

[4] P. Kairouz et al., “Advances and open problems in federated learning,” Found. Trends® Mach. Learn., vol. 14, nos. 1–2, pp. 1–210, 2019.

[5] Q. Yang, Y. Liu, T. Chen, and Y. Tong, “Federated machine learning: Concept and applications,” ACM Trans. Intell. Syst. Technol., vol. 10, no. 2, pp. 1–19, 2019.

[6] J. Zhou et al., “A survey on federated learning and its applications for accelerating industrial Internet of Things,” 2021, arXiv:2104.10501.

[7] Y. Liu et al., “FedBCD: A communication-efficient collaborative learning framework for distributed features,” IEEE Trans. Signal Process., vol. 70, pp. 4277–4290, Aug. 2022.

[8] T. Castiglia, A. Das, S. Wang, and S. Patterson, “Compressed-VFL: Communication-efficient learning with vertically partitioned data,” in Proc. 39th Int. Conf. Mach. Learn., 2022, pp. 2738–2766.

[9] F. Zhou and G. Cong, “On the convergence properties of a K-step averaging stochastic gradient descent algorithm for nonconvex optimization,” in Proc. 27th Int. Conf. Conf. Artif. Intell., Jul. 2018, pp. 3219–3227.

[10] F. Satlter, S. Wiedemann, K.-R. Müller, and W. Samek, “Robust and communication-efficient federated learning from non-i.i.d. Data,” IEEE Trans. Neural Netw. Learn. Syst., vol. 31, no. 9, pp. 3400–3413, Sep. 2020.

[11] Y. Chen, X. Sun, and Y. Jin, “Communication-efficient federated deep learning with layerwise asynchronous model update and temporally weighted aggregation,” IEEE Trans. Neural Netw. Learn. Syst., vol. 31, no. 10, pp. 4229–4238, Oct. 2020.

[12] H. Jamali-Rad, M. Abdizadeh, and A. Singh, “Federated learning with taskonomy for non-IID data,” IEEE Trans. Neural Netw. Learn. Syst., early access, Mar. 22, 2022. doi: 10.1109/TNNLS.2022.3152581.

[13] C. Sun et al., “A privacy-preserving infrastructure for analyzing personal health data in a vertically partitioned scenario,” Stud. Health Technol. Inform., vol. 264, 2019, pp. 573–577.

[14] T. Chen, X. Yin, S. Sun, and W. Yin, “VFL: A method of vertical asynchronous federated learning,” 2020, arXiv:2007.06081.

[15] A. Das, T. Castiglia, S. Wang, and S. Patterson, “Cross-silo federated learning for multi-tier networks with vertical and horizontal data partitioning,” ACM Trans. Intell. Syst. Technol., vol. 13, no. 6, pp. 1–27, Dec. 2022.

[16] Y. Hu, D. Niu, J. Yang, and S. Zhou, “FDMIL: A collaborative machine learning framework for distributed features,” in Proc. 25th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining, Jul. 2019, pp. 2232–2240.

[17] F. Fu, X. Miao, J. Jiang, H. Xue, and B. Cui, “Towards communication-efficient vertical federated learning training via cache-enabled local updates,” 2022, arXiv:2207.14628.

[18] J. Zhang et al., “Adaptive vertical federated learning on unbalanced features,” IEEE Trans. Parallel Distrib. Syst., vol. 33, no. 12, pp. 4006–4018, Dec. 2022.

[19] D. Hazarika, R. Zimmermann, and S. Poria, “MISA: Modality-invariant and Specific representations for multimodal sentiment analysis,” in Proc. 28th ACM Int. Conf. Multimedia, Oct. 2020, pp. 1122–1131.

[20] W. Han, H. Chen, and S. Poria, “Improving multimodal fusion with hierarchical mutual information maximization for multimodal sentiment analysis,” in Proc. Conf. Empirical Methods Natural Lang. Process., 2021, pp. 9180–9192.

[21] W. Lin et al., “ViBERTgrid: A jointly trained multi-modal 2D document representation for key information extraction from documents,” in Proc. 16th Int. Conf. Document Anal. Recognit., vol. 12821, 2021, pp. 548–563.

[22] P. Narkhede, R. Walambe, S. Mandaokar, P. Chandel, K. Kotecha, and G. Ghinea, “Gas detection and identification using multimodal artificial intelligence based sensor fusion,” Appl. Syst. Innov., vol. 4, no. 1, p. 3, Jan. 2021.

[23] A. Reisizadeh, H. Taheri, A. Mokhtari, H. Hassanii, and R. Pedarsani, “Robust and communication-efficient collaborative learning,” in Proc. Adv. Neural Inf. Process. Syst., 2019, pp. 8386–8397.

[24] Z. Xu, F. Yu, J. Xiong, and X. Chen, “Helios: Heterogeneity-aware federated learning with dynamically balanced collaboration,” 2019, arXiv:1912.01684.

[25] A. Reisizadeh, I. Tziotis, H. Hassanii, A. Mokhtari, and R. Pedarsani, “Straggler-resilient federated learning: Leveraging the interplay between statistical accuracy and system heterogeneity,” 2020, arXiv:2012.14453.

[26] Q. Zhang, B. Gu, C. Deng, and H. Huang, “Secure bilevel asynchronous vertical federated learning with backward updating,” in Proc. Conf. Artif. Intell., 2021, pp. 10896–10904.

[27] H. Shi, Y. Xu, Y. Jiang, H. Yu, and L. Cui, “ Efficient asynchronous multi-participant vertical federated learning,” IEEE Trans. Big Data, early access, Aug. 25, 2022, doi: 10.1109/TBDA.2022.3201729.

[28] H. Cui, H. Zhang, G. R. Ganger, P. B. Gibbons, and E. P. Xing, “GeoPS: Scalable deep learning on distributed GPUs with a GPU-specialized parameter server,” in Proc. 11th Eur. Conf. Comput. Syst., Apr. 2016, pp. 4–1:4–16.

[29] W. Dai, “Learning with staleness,” Ph.D. dissertation, Dept. Mach. Learn., Carnegie Mellon Univ., Pittsburgh, PA, USA, 2018.

[30] M. Gugunathan, P. Goyal, and L. Kagal, “Multi-VFL: A vertical federated learning system for multiple data and label owners,” 2021, arXiv:2106.05468.

[31] K. Yang, Z. Song, Y. Zhang, Y. Zhou, X. Sun, and J. Wang, “Model optimization method based on vertical federated learning,” in Proc. IEEE Int. Symp. Circuits Syst. (ISCAS), May 2021, pp. 1–5.

[32] B. Gu, A. Xu, Z. Hao, C. Deng, and H. Huang, “Privacy-preserving asynchronous vertical federated learning algorithms for multiparty collaborative learning,” IEEE Trans. Neural Netw. Learn. Syst., vol. 33, no. 11, pp. 6103–6115, Nov. 2022.

[33] Q. Zhang et al., “AsySQR: Faster vertical federated learning algorithms with better computation resource utilization,” in Proc. 27th ACM SIGKDD Conf. Knowl. Discovery Data Mining, Aug. 2021, pp. 3917–3927.

[34] C. Xie, P.-Y. Chen, C. Zhang, and B. Li, “Improving privacy-preserving vertical federated learning by efficient communication with ADMM,” 2022, arXiv:2207.10226.

[35] J. Wang, Q. Liu, H. Liang, G. Joshi, and H. V. Poor, “A novel framework for the analysis and design of heterogeneous federated learning,” IEEE Trans. Signal Process., vol. 69, pp. 5234–5249, 2021.

[36] T. Li, A. K. Sahu, M. Zaheer, M. Sanjabi, A. Talwalkar, and V. Smith, “Federated optimization in heterogeneous networks,” in Proc. Mach. Learn. Syst., 2020. Accessed: Apr. 12, 2023. [Online]. Available: https://proceedings.mlsys.org/book/316.pdf
J. Bradley, A. Kyrola, D. Bickson, and C. Guestrin, “A unified theory of decentralized SGD with changing topology and local updates,” in *Proc. 37th Int. Conf. Mach. Learn.*, vol. 119, Jul. 2020, pp. 5381–5393.

T. Castiglione, A. Das, and S. Patterson, “Multi-level local SGD: Distributed SGD for heterogeneous hierarchical networks,” in *Proc. Int. Conf. Learn. Represent.*, 2021. Accessed: Apr. 12, 2023. [Online]. Available: https://openreview.net/forum?id=C70cp4Cn32

J. Bradley, A. Kyrola, D. Bickson, and C. Guestrin, “Parallel coordinate descent for L1-regularized loss minimization,” in *Proc. Int. Conf. Mach. Learn.*, 2011, pp. 321–328.

J. Liu, S. Wright, C. Ré, V. Bittorf, and S. Sridhar, “An asynchronous parallel stochastic coordinate descent algorithm,” in *Proc. Int. Conf. Mach. Learn.*, 2014, pp. 469–477.

P. Richtárik and M. Takáč, “Distributed coordinate descent method for learning with big data,” *J. Mach. Learn. Res.*, vol. 17, no. 1, pp. 2657–2681, Jan. 2016.

L. Wan, W. K. Ng, S. Han, and V. C. S. Lee, “Privacy-preservation for gradient descent methods,” in *Proc. 13th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, Aug. 2007, pp. 775–783.

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

S. Hardy et al., “Private federated learning on vertically partitioned data via entity resolution and additively homomorphic encryption,” 2017, arXiv:1711.10677.

L. Bottou, F. E. Curtis, and J. Nocedal, “Optimization methods for large-scale machine learning,” *SIAM Rev.*, vol. 60, no. 2, pp. 223–311, Jan. 2018.

J. Tsitsiklis, D. Bertsekas, and M. Athans, “Distributed asynchronous deterministic and stochastic gradient optimization algorithms,” *IEEE Trans. Autom. Control*, vol. AC-31, no. 9, pp. 803–812, Sep. 1986.

L. M. Nguyen, P. H. Nguyen, M. van Dijk, P. Richtárik, K. Scheinberg, and M. Takáč, “SGD and Hogwild!: Convergence without the bounded gradients assumption,” in *Proc. 35th Int. Conf. Mach. Learn.*, vol. 80, 2018, pp. 3747–3755.

L. T. Phong, Y. Aono, T. Hayashi, L. Wang, and S. Moriai, “Privacy-preserving deep learning via additively homomorphic encryption,” *IEEE Trans. Inf. Forensics Security*, vol. 13, no. 5, pp. 1333–1345, May 2018.

J. Geiping, H. Bauermeister, H. Dröge, and M. Moeller, “Inverting gradients—How easy is it to break privacy in federated learning?” in *Proc. Adv. Neural Info. Process. Syst.*, 2020. Accessed: Apr. 12, 2023. [Online]. Available: https://proceedings.neurips.cc/paper/2020/hash/c6c4e365bb89b9a6e112b0ac6b1f45-Abstract.html

A. Mahendran and A. Vedaldi, “Understanding deep image representations by inverting them,” in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2015, pp. 5188–5196.

K. Cheng et al., “SecureBoost: A lossless federated learning framework,” *IEEE Trans. Syst.*, vol. 36, no. 6, pp. 87–98, Nov. 2021.

S. Ghadimi and G. Lan, “Stochastic first- and zeroth-order methods for nonconvex stochastic programming,” *SIAM J. Optim.*, vol. 23, no. 4, pp. 2341–2368, Jan. 2013.

A. Bagher Zadeh, P. P. Liang, S. Poria, E. Cambria, and L.-P. Morency, “Multimodal language analysis in the wild: CMU-MOSEI dataset and interpretable dynamic fusion graph,” in *Proc. 56th Annu. Meeting Assoc. Comput. Linguistics*, 2018, pp. 2236–2246.

J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, “ImageNet: A large-scale hierarchical image database,” in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Jun. 2009, pp. 248–255.

Z. Wu et al., “3D ShapeNets: A deep representation for volumetric shapes,” in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2015, pp. 1912–1920.

V. Smith, S. Forte, M. Chenxin, M. Takeda, M. J. Jordan, and M. Jaggi, “CoCoA: A general framework for communication-efficient distributed optimization,” *J. Mach. Learn. Res.*, vol. 18, p. 230, pp. 1–12, Jul. 2018.

C. Reiss, J. Wilkes, and J. L. Hellerstein, “Google cluster-usage traces: Format + schema,” Google Inc., Mountain View, CA, USA, Tech. Rep., 2011.

T. Castiglione, A. Das, and S. Patterson, “Multi-level local SGD: Distributed SGD for heterogeneous hierarchical networks,” in *Proc. Int. Conf. Learn. Represent.*, 2021. Accessed: Apr. 12, 2023. [Online]. Available: https://openreview.net/forum?id=B1eyOlBFPr

Shiqiang Wang (Senior Member, IEEE) received the Ph.D. degree from the Department of Electrical and Electronic Engineering, Imperial College London, London, U.K., in 2015.

He is currently a Staff Research Scientist at the IBM T. J. Watson Research Center, Yorktown Heights, NY, USA. His current research focuses on the intersection of distributed computing, machine learning, networking, and optimization, with a broad range of applications including data analytics, edge-based artificial intelligence (edge AI), the Internet of Things (IoT), and future wireless systems. He has made foundational contributions to edge computing and federated learning that generated both academic and industrial impact.

Dr. Wang received the IEEE Communications Society (ComSoc) Leonard G. Abraham Prize in 2021, the IEEE ComSoc Best Young Professional Award in Industry in 2021, the IBM Outstanding Technical Achievement Awards (OTAA) in 2019, 2021, 2022, and 2023, the Multiple Invention Achievement Awards from IBM in 2016, the Best Paper Finalist from the IEEE International Conference on Image Processing (ICIP) in 2019, and the Best Student Paper Award of the Network and Information Sciences International Technology Alliance (NIS-ITA) in 2015. He serves as an Associate Editor for IEEE TRANSACTIONS ON MOBILE COMPUTING and IEEE TRANSACTIONS ON PARALLEL AND DISTRIBUTED SYSTEMS.

Stacy Patterson received the M.S. and Ph.D. degrees in computer science from UC Santa Barbara, Santa Barbara, CA, USA, in 2003 and 2009, respectively.

From 2009 to 2011, she was a Post-Doctoral Scholar at the Center for Control, Dynamical Systems and Computation, UC Santa Barbara. From 2011 to 2013, she was a Post-Doctoral Fellow with the Department of Electrical Engineering, Technion—Israel Institute of Technology, Haifa, Israel. She is an Associate Professor with the Department of Computer Science, Rensselaer Polytechnic Institute, Troy, NY, USA. Her research interests include distributed systems, cloud computing, and the Internet of Things.

Dr. Patterson was a recipient of the Viterbi Post-Doctoral Fellowship, the IEEE CSS Axelby Outstanding Paper Award, and the NSF CAREER Award.

Authorized licensed use limited to the terms of the applicable license agreement with IEEE. Restrictions apply.