Improving Privacy-Preserving Vertical Federated Learning by Efficient Communication with ADMM

Chulin Xie
UIUC
chulinx2@illinois.edu

Pin-Yu Chen
IBM Research
pin-yu.chen@ibm.com

Qinbin Li
UC Berkeley
qinbin@berkeley.edu

Arash Nourian
Amazon Web Services & UC Berkeley
nouriara@amazon.com

Ce Zhang
University of Chicago
cez@uchicago.edu

Bo Li
University of Chicago & UIUC
lbo@illinois.edu

***Abstract*—Federated learning (FL) enables distributed resource-constrained devices to jointly train shared models while keeping the training data local for privacy purposes. Vertical FL (VFL), which allows each client to collect partial features, has attracted intensive research efforts recently. We identified the main challenges that existing VFL frameworks are facing: the server needs to communicate *gradients* with the clients for *each* training step, incurring high communication cost that leads to rapid consumption of privacy budgets. To address these challenges, in this paper, we introduce a VFL framework with multiple heads (`VIM`), which takes the separate contribution of each client into account, and enables an efficient decomposition of the VFL optimization objective to sub-objectives that can be iteratively tackled by the server and the clients *on their own*. In particular, we propose an Alternating Direction Method of Multipliers (ADMM)-based method to solve our optimization problem, which allows clients to conduct *multiple local updates* before communication, and thus reduces the communication cost and leads to better performance under differential privacy (DP). We provide the client-level DP mechanism for our framework to protect user privacy. Moreover, we show that a byproduct of `VIM` is that the weights of learned heads reflect the importance of local clients. We conduct extensive evaluations and show that on four vertical FL datasets, `VIM` achieves significantly higher performance and faster convergence compared with the state-of-the-art. We also explicitly evaluate the importance of local clients and show that `VIM` enables functionalities such as client-level explanation and client denoising. We hope this work will shed light on a new way of effective VFL training and understanding.** [1]

I. Introduction

Federated learning (FL) has enabled large-scale training with data privacy guarantees on distributed data for different applications [80, 8, 31, 79, 77]. In general, FL can be categorized into Horizontal FL (HFL) [52] where data samples are distributed across clients, and Vertical FL (VFL) [77] where features of the samples are partitioned across clients and the labels are usually owned by the server (or the active party in two-party setting [32]). In particular, VFL allows clients with partial information of the same dataset to jointly train the model, which leads to many real-world applications [36, 77, 31]. For instance, a patient may go to different types of healthcare providers, such as dental clinics and pharmacies for different purposes, and therefore it is important for different healthcare providers (i.e., VFL clients/data owners/organizations) to "share" their information about the same patient (i.e., partial features of the same sample) to better model the health condition of the patient. In addition, nowadays multimodal data has been ubiquitous, while usually, each client is only able to collect one or a few data modalities due to resource limitations. Therefore, VFL provides an effective way to allow such clients to train a model leveraging information from different data modalities jointly.

Despite the importance and practicality of VFL, the state-of-the-art (SOTA) VFL frameworks suffer from notable weaknesses: since the clients own the local features and the server holds the whole labels, the server needs to calculate training loss based on the labels and then send *gradients* to clients for *each* training step to update their local models [70, 12, 40], which incurs high communication cost and leads to potential rapid consumption of the privacy budget.

To solve the above challenges, in this work, we propose an efficient VFL optimization framework with multiple heads (`VIM`), where each head corresponds to one local client. `VIM` takes the individual contribution of clients into consideration and facilitates a thorough decomposition of the VFL optimization problem into multiple subproblems that can be iteratively solved by the server and the clients. In particular, we propose an Alternating Direction Method of Multipliers (ADMM) [7]-based method that splits the overall `VIM` optimization objective into smaller sub-objectives, and the clients can conduct multiple local updates w.r.t their local objectives at each communication round with the coordination of ADMM-related variables. This leads to faster model convergence and significantly reduces the communication cost, which is crucial to preserve privacy because the privacy cost of clients increases when the number of communication rounds increases [1, 53], due to the continuous transmission of sensitive local information. We consider two typical VFL settings: *with model splitting* (i.e., clients host partial models) and *without model splitting* (i.e., clients hold the entire model). Under with model splitting setting, we propose an ADMM-based algorithm `VIMADMM` under `VIM` framework. Compared to gradient-based methods, `VIMADMM` not only reduces communication frequency but also reduces the

[1] Our code is available at: https://github.com/AI-secure/VFL-ADMM

TABLE I: Comparison between our work and existing VFL studies.

VFL Setup	Method	Support DNN	Support $N > 2$ parties	Labels only held by one party	Support multiple local updates	Privacy guarantee
w/ model splitting	VAFL [12], VFL-PBM [68]	✓	✓	✓	×	✓
	Split Learning [70]	✓	✓	✓	×	×
	FedBCD [48]	✓	✓	✓	✓	×
	CELU-VFL	✓	×	✓	✓	×
	Flex-VFL [10]	✓	✓	×	✓	×
	VIMADMM (Ours)	✓	✓	✓	✓	✓
w/o model splitting	Fu et al. [25], FDML [36]	✓	✓	×	×	×
	AdaVFL [83], CAFE [39]	✓	✓	×	✓	×
	Linear-ADMM [35]	×	✓	✓	✓	✓
	VIMADMM-J (Ours)	✓	✓	✓	✓	✓

dimensionality by only exchanging ADMM-related variables. We provide convergence analysis for VIMADMM and prove that it can converge to stationary points with mild assumptions. With modifications of communication strategies and updating rules for servers and clients, we extend VIMADMM to the without model splitting setting and introduce VIMADMM-J. Under both settings, to further protect the privacy of the local features held by clients, we introduce privacy mechanisms that clip and perturb local outputs to satisfy *client-level* differential privacy (DP) [20, 19, 21, 52] and prove the DP guarantees. Moreover, we offer a basic solution to separately protect the privacy of labels owned by server, leveraging the established label-DP mechanism ALIBI [51] that perturbs the labels. Finally, we show that a byproduct of VIM is that the weights of learned heads reflect the importance of local clients, which enables functionalities such as client-level explanation, client denoising, and client summarization. Our main contributions are:

- We propose an efficient and effective VFL optimization framework with multiple heads (VIM). To solve our optimization problem, we propose an ADMM-based method, VIMADMM, which reduces communication costs by allowing multiple local updates at each step.
- We theoretically analyze the convergence of VIMADMM and prove that it can converge to stationary points.
- We introduce the client-level DP mechanism for our VIM framework and prove its privacy guarantees.
- We conduct extensive experiments on four diverse datasets (i.e., MNIST, CIFAR, NUS-WIDE, and ModelNet40), and show that ADMM-based algorithms under VIM converge faster, achieve higher accuracy, and remain higher utility under client-level DP and label DP than four existing VFL frameworks.
- We evaluate our client-level explanation under VIM based on the weights norm of the heads, and demonstrate the functionalities it enables such as clients denoising and summarization.

II. Related Work

a) Vertical Federated Learning: VFL has been well studied for simple models including trees [13, 74], kernel models [30], and linear and logistic regression [32, 78, 84, 24, 35, 46]. For instance, Hardy et al. [32] propose secure logistic regression for two-party VFL with homomorphic encryption [64, 28] and multiparty computation [4, 6]. However, a limitation of these methods is the performance constraint associated with the logistic regression. Subsequent research has expanded the scope of VFL to encompass Deep Neural Networks (DNNs), facilitating VFL training with a larger number of clients and on large-scale models and datasets. For DNNs, there are two popular VFL settings: with model splitting [70, 40, 12] and without model splitting [36, 39].

In the with model splitting setting, Split Learning [70] is the first related paradigm, where each client trains a partial network up to a cut layer, the server concatenates local activations and trains the rest of the network. VAFL [12] is proposed for asynchronous VFL where the server averages the local embeddings and sends gradients back to clients to update local models. However, such embedding averaging might lose the unique properties of each client. FedMVT [40] focuses on the semi-supervised VFL with multi-view learning. C-VFL [11] proposes embedding compression techniques to improve communication efficiency. However, we note that these methods [70, 12, 40, 11] still require the communication of gradients (w.r.t embeddings) from server to the client at *each* training step, leading to high communication frequency and communication cost before convergence. Recent research efforts have sought to reduce VFL communication frequency by allowing clients to make multiple local updates at each round. Particularly, in FedBCD [48], after obtaining gradients from the server, clients update local models using the same stale gradients for multiple steps. Building upon this, CELU-VFL [26] enhances the performance of FedBCD by caching stale gradients from earlier rounds and reusing them to estimate better model gradients at current round. Nonetheless, it is limited to supporting only two clients (party A and B, with B holding the labels) and cannot be directly extended to scenarios with more than two parties, as our study considers (specifically, it lacks a design to aggregate information from more parties). On another note, Flex-VFL [10] allows each party to undergo a different number of local updates constrained by a set timeout for every round. Yet, it assumes that clients possess copies of labels and receive local embeddings from other clients, enabling them to compute local gradients independently for multi-step local updates. In contrast, we propose an ADMM-based framework that enables multiple local updates and assumes that only the server possesses labels, which cannot be shared with other clients due to privacy restriction [25].

For VFL without model splitting setting, each client submits local logits to the server, who then averages over the logits and send gradients w.r.t logits back to clients, as detailed in Fu et al. [25]. Several other approaches assume that the server shares both labels and aggregated logits with the clients, enabling them to locally compute the gradient [36, 83]. FDML [36] performs one step of local update at each round for asynchronous and

distributed SGD. Considering that certain clients might have slower local computation speeds, AdaVFL [83] optimizes the number of local updates for each client at each round to minimize overall time. Meanwhile, CAFE [39] directly applies FedAvg [52] from Horizonta FL to VFL where all clients possess the labels and can exchange the model parameters with others for model aggregation. This deviates from the standard VFL setup where only the server retains the label and local models cannot be shared owing to privacy implications [25].

b) Differentially Private VFL: In existing VFL frameworks, VAFL [12] provides Gaussian DP guarantee [17] and VFL-PBM [68] quantizes local embeddings into DP integer vectors. However, they do not calculate the exact privacy budget in the evaluation. FDML [36] evaluate their framework under different levels of empirical noises, yet without offering detailed DP mechanisms or DP guarantee. The ADMM-based linear VFL framework (abbreviated to Linear-ADMM) [35] provides (ϵ, δ)-DP guarantee for linear models by calculating the closed-form sensitivity of each sample and perturbing the linear model parameters, which is not directly applicable to DNNs whose sensitivity is hard to estimate due to the nonconvexity. Instead, we propose to perturb local outputs and provide formal client-level (ϵ, δ)-DP theoretical guarantee in Section V.

We provide an overall comparison between our work and existing studies in Table I.

III. VFL with Multiple Heads (VIM)

In this section, we start with the VFL background in Section III-A, and then discuss VFL with model splitting setting and introduce our framework VIM and ADMM-based method VIMADMM in Section III-B. Finally, we show that our ADMM-based method can be easily extended to VFL without model splitting setting with slight modifications on communication strategies and update rules, yielding VIMADMM-J in Section III-C.

A. VFL Background

Typically in VFL, there are M clients who hold *different feature sets of the same training samples* and jointly train the machine learning models. We consider the classification task and denote d_c as the number of classes. Suppose there is a training dataset $D = \{x_j, y_j\}_{j=1}^N$ containing N samples, the server owns the labels $\{y_j\}_{j=1}^N$, and each client k has a local feature set $X_k = \{x_j^k\}_{j=1}^N$, where the vector $x_j^k \in \mathbb{R}^{d^k}$ denotes the local (partial) features of sample j. The overall feature $x_j \in \mathbb{R}^d$ of sample j is the concatenation of all local features $\{x_j^1, x_j^2, \ldots, x_j^M\}$, with $d = \sum_{k=1}^M d^k$.

Due to the privacy protection requirement of VFL, each client k does not share raw local feature set X_k with other clients or the server. Instead, VFL consists of two steps: (1) *local processing step*: each client learns a local model that maps the local features to local outputs and sends them to the server. (2) *server aggregation step*: the server aggregates the local outputs from all clients to compute the final prediction for each sample as well as the corresponding losses. Depending on whether or not the server holds a model, there are two popular VFL settings [25]: VFL *with model splitting* [12, 70] and VFL *without model splitting* [36]: (i) In the model splitting setting, each client trains a feature extractor as the local model that outputs *local embeddings*, and the server owns a model which predicts the final results based on the aggregated embeddings. (ii) In the VFL without model splitting setting, the clients host the entire model that outputs the *local logits*, and the server simply performs the logits aggregation operation without hosting any model.

In both settings, the local model is updated based on SGD with federated backward propagation [25]: a) server first computes the gradients w.r.t the local output (either embeddings or logits) from each client separately and sends the gradients back to clients; b) each client calculates the gradients of local output w.r.t the local model parameters and updates the local model using the chain rule.

B. VFL with Model Splitting

a) Setup: Let f parameterized by θ_k be the local model (i.e., feature extractor) of client k, which outputs a local embedding vector $h_j^k = f(x_j^k; \theta_k) \in \mathbb{R}^{d_f}$ for each local feature x_j^k. We denote the parameters of the model on the server-side as θ_0. Overall, the clients and the server aim to collaboratively solve the Empirical Risk Minimization (ERM) objective:

$$\min_{\{\theta_k\}, \theta_0} \frac{1}{N} \sum_{j=1}^N \ell(\{h_j^1, \ldots, h_j^M\}, y_j; \theta_0) + \sum_{k=1}^M \beta_k \mathcal{R}(\theta_k) + \beta \mathcal{R}(\theta_0) \quad (1)$$

where ℓ is a loss function (e.g., cross-entropy loss with softmax), $\mathcal{R}$ is a regularizer on model parameters, and $\beta_k \in \mathbb{R}$ is the regularization weight for client k, and β is the weight for server. The local embeddings for each sample j can be either concatenated together $h_j = [h_j^1, \ldots, h_j^M]$ as in Split Learning [70] or averaged $h_j = \sum_{k=1}^M \alpha_k h_j^k$ with aggregation weights $\alpha_k \in \mathbb{R}$ as in VAFL [12]. Then h_j is used as the input for server model θ_0 to calculate the loss. For more detailed description of the training algorithm Split Learning under VFL with model splitting, please refer to Algorithm 2 in Appendix A1.

However, as outlined in Section III-A, these VFL methods are based on SGD and depend on the server model θ_0 to complete the loss and gradient calculation using server labels for updating local models $\{\theta_k\}$. Consequently, the server needs to send the gradient w.r.t embeddings back to clients at *every* training step of the local models. Such (1) frequent communication and (2) high dimensionality of gradients (i.e., bd_f for b samples) lead to high communication costs.

b) VIM Formulation: To address these challenges, we propose the VIM framework where the server learns a model with multiple heads corresponding to multiple local clients. It takes the separate contribution of each client into account and facilitates the breakdown of the VFL optimization into several sub-problems to be solved by clients and the server independently via ADMM without communicating gradients, as we will elaborate on later. Specifically, the server's model θ_0 consists of M heads $W_1, W_2, \ldots, W_M$ where $W_k \in \mathbb{R}^{d_f \times d_c}, k \in [M]$. For the sake of simplicity, we consider each W_k to be a linear head here, and our formulation can be easily extended to the non-linear heads by viewing each

W_k as a non-linear model (see the end of Section III-B for more details). This is motivated by the recent studies in representation learning, which have shown that learning a linear classifier is sufficient to accurately predicting the labels on top of embedding representations [60, 41], given the expressive power of the local feature extractor that captures essential information from raw feature sets. For sample j, the server's model outputs $\hat{y}_j = \sum_{k=1}^{M} h_j^k W_k$ as the prediction, yielding our VIM objective:

$$\min_{\{W_k\},\{\theta_k\}} \mathcal{L}_{\text{VIM}}(\{W_k\},\{\theta_k\}) := \frac{1}{N}\sum_{j=1}^{N} \ell\left(\sum_{k=1}^{M} f(x_j^k;\theta_k)W_k, y_j\right) + \sum_{k=1}^{M}\beta_k\mathcal{R}_k(\theta_k) + \sum_{k=1}^{M}\beta_k\mathcal{R}_k(W_k) \tag{2}$$

c) VIMADMM: Based on the VIM formulation, we propose an ADMM-based method, reducing the communication frequency by allowing the clients to perform multiple local updates w.r.t their ADMM objectives at each round, and reducing the dimensionality by only exchanging ADMM-related variables (i.e., $(2b + d_f)d_c$ for b samples where $d_c \ll d_f, b$ for most VFL settings today [12, 36]). Specifically, we note that Eq. 2 can be viewed as the *sharing problem* [7] involving each client adjusting its variable to minimize the *shared* cost term $\ell(\sum_{k=1}^{M} h_j^k W_k, y_j)$ as well as its *individual* cost $\mathcal{R}(\theta_k) + \mathcal{R}(W_k)$. Moreover, the multiple heads in VIM enable the application of ADMM via a special decomposition into simpler sub-problems that can be solved in a distributed manner. We begin by rewriting Eq. 2 to an equivalent constrained optimization problem by introducing auxiliary variables $z_1, z_2, \ldots, z_N \in \mathbb{R}^{d_c}$:

$$\min_{\{W_k\},\{\theta_k\},\{z_j\}} \frac{1}{N}\sum_{j=1}^{N}\ell(z_j, y_j) + \sum_{k=1}^{M}\beta_k\mathcal{R}_k(\theta_k) + \sum_{k=1}^{M}\beta_k\mathcal{R}_k(W_k)$$
$$\text{s.t.} \sum_{k=1}^{M} f(x_j^k;\theta_k)W_k - z_j = 0, \forall j \in [N]. \tag{3}$$

Notably, each constraint implies a consensus between the server's output $\sum_{k=1}^{M} h_j^k W_k$ and the auxiliary variable z_j for each sample j. The augmented Lagrangian, which adds a quadratic term to the Lagrangian of Eq. 3, is given by:

$$\begin{aligned}
&\min_{\{W_k\},\{\theta_k\},\{z_j\},\{\lambda_j\}} \mathcal{L}_{\text{ADMM}}(\{W_k\},\{\theta_k\},\{z_j\},\{\lambda_j\}) \\
&:= \frac{1}{N}\sum_{j=1}^{N}\ell(z_j, y_j) + \sum_{k=1}^{M}\beta_k\mathcal{R}_k(\theta_k) + \sum_{k=1}^{M}\beta_k\mathcal{R}_k(W_k) \\
&+ \frac{1}{N}\sum_{j=1}^{N}\lambda_j^\top\left(\sum_{k=1}^{M} f(x_j^k;\theta_k)W_k - z_j\right) \\
&+ \frac{\rho}{2N}\sum_{j=1}^{N}\left\|\sum_{k=1}^{M} f(x_j^k;\theta_k)W_k - z_j\right\|_F^2,
\end{aligned} \tag{4}$$

where $\lambda_j \in \mathbb{R}^{d_c}$ is the dual variable for sample j, and $\rho \in \mathbb{R}^+$ is a constant penalty factor. Recall that $\hat{y}_j = \sum_{k=1}^{M} f(x_j^k, \theta_k)W_k$ is the server output (i.e., prediction) for sample x_j. VIMADMM essentially aims to minimize the loss between z_j and ground-truth label y_j, as well as the difference between z_j and $\hat{y}_j$ during training. Specifically, as shown in the ADMM loss (Eq. 4), $l(z_j, y_j)$ is the loss between z_j and y_j, while $\hat{y}_j - z_j = \sum_{k=1}^{M} f(x_j^k, \theta_k)W_k - z_j$ appears in the linear constraint and quadratic constraint terms. The auxiliary variables $\{z_j\}$ and dual variables $\{\lambda_j\}$ are used to facilitate the training of server heads $\{W_k\}$ and local models $\{\theta_k\}$.

To solve Eq. 4, we follow standard ADMM [7] and update the primal variables $\{W_k\}$, $\{\theta_k\}$, $\{z_j\}$ and the dual variables $\{\lambda_j\}$ *alternatively*, which decomposes the problem in Eq. 3 into four sets of sub-problems over $\{W_k\}$, $\{\theta_k\}$, $\{z_j\}$, $\{\lambda_j\}$, and the parameters in each sub-problem can be solved *in parallel*. In practice, we propose the following strategy for the alternative updating in the server and clients: (i) updating $\{z_j\}$, $\{\lambda_j\}$ and $\{W_k\}$ at server-side, (ii) updating $\{\theta_k\}$ at the client-side in parallel. Moreover, we consider the realistic setting of stochastic ADMM with mini-batches. Concretely, at communication round t, the server samples a set of data indices, $B(t)$, with batch size $|B(t)| = b$. Then we describe the key steps of VIMADMM as follows:

(1) *Communication from client to server.* Each client k sends a batch of embeddings $\{h_j^{k(t)}\}_{j\in B(t)}$ to the server, where $h_j^{k(t)} = f(x_j^k; \theta_k^{(t)})$, $\forall j \in B(t)$.

(2) *Sever updates auxiliary variables* $\{z_j\}$. After receiving the local embeddings from all clients, the server updates the auxiliary variable for each sample $j \in B(t)$ as:

$$z_j^{(t)} = \underset{z_j}{\operatorname{argmin}} \quad \ell(z_j, y_j) - \lambda_j^{(t-1)\top} z_j + \frac{\rho}{2}\left\|\sum_{k=1}^{M} h_j^{k(t)} W_k^{(t)} - z_j\right\|_F^2. \tag{5}$$

Since the optimization problem in Eq. 5 is convex and differentiable with respect to z_j, we use the L-BFGS-B algorithm [85] to solve the minimization problem.

(3) *Sever updates dual variables* $\{\lambda_j\}$. The server updates dual variable for each sample $j \in B(t)$:

$$\lambda_j^{(t)} = \lambda_j^{(t-1)} + \rho\left(\sum_{k=1}^{M} h_j^{k(t)} W_k^{(t)} - z_j^{(t)}\right). \tag{6}$$

(4) *Sever updates the heads* $\{W_k\}$. Each head $W_k, \forall k \in [M]$ of the server is then updated:

$$\begin{aligned}
W_k^{(t+1)} = \underset{W_k}{\operatorname{argmin}} \quad & \beta_k\mathcal{R}_k(W_k) + \frac{1}{b}\sum_{j\in B(t)} \lambda_j^{(t)\top} h_j^{k(t)} W_k \\
& + \sum_{j\in B(t)} \frac{\rho}{2b}\left\|\sum_{i\in[M], i\neq k} h_j^{i(t)} W_i^{(t)} + h_j^{k(t)} W_k - z_j^{(t)}\right\|_F^2.
\end{aligned} \tag{7}$$

For squared ℓ_2 regularizer $\mathcal{R}$, we can solve $W_k^{(t+1)}$ in an inexact way to save the computation by *one* step of SGD with the objective of Eq. 7.

(5) *Communication from server to client.* After the updates in Eq. 7, we define a residual variable $s_j^{k(t+1)}$ for each sample $j \in B(t)$ of k-th client, which provides supervision for updating local model:

$$s_j^{k(t)} \triangleq z_j^{(t)} - \sum_{i\in[M], i\neq k} h_j^{i(t)} W_i^{(t+1)} \tag{8}$$

Algorithm 1 VIMADMM (with user-level differential privacy)

Input: number of communication rounds T, number of clients M, number of training samples N, batch size b , input features $\{\{x_j^1\}_{j=1}^N, \{x_j^2\}_{j=1}^N, \ldots, \{x_j^M\}_{j=1}^N\}$, the labels $\{y_j\}_{j=1}^N$, local model $\{\theta_k\}_{k=1}^M$; linear heads $\{W_k\}_{k=1}^M$; auxiliary variables $\{z_j\}_{j=1}^N$; dual variables $\{\lambda_j\}_{j=1}^N$; noise parameter σ, clipping constant C
for communication round $t \in [T]$ **do**
  Server samples a set of data indices $B(t)$ with $|B(t)| = b$
  for client $k \in [M]$ **do**
    generates a local training batch $\{x_j^k\}_{j\in B(t)}$
    computes local embeddings $\{h_j^{k(t)} \leftarrow f(x_j^k;\theta_k)\}_{j\in B(t)}$
    clips and perturbs local embedding matrix
    $\{h_j^{k(t)}\}_{j\in B(t)} \leftarrow \texttt{Clip}\left(\{h_j^{k(t)}\}_{j\in B(t)}, C\right) + \mathcal{N}\left(0, \sigma^2 C^2\right)$
    sends local embeddings $\{h_j^{k(t)}\}_{j\in B(t)}$ to the server
  Server **updates** auxiliary variables $\{z_j^{(t)}\}_{j\in B(t)}$ by Eq. 5
  Server **updates** dual variables $\{\lambda_j^{(t)}\}_{j\in B(t)}$ by Eq. 6
  Server **updates** linear heads $\{W_k^{(t+1)}\}_{k\in[M]}$ by Eq. 7
  Server **computes** residual variables $\{s_j^{k(t)}\}_{j\in B(t),k\in[M]}$ by Eq. 8
  Server **sends** $\{\lambda_j^{(t)}\}_{j\in B(t)}$, $\{s_j^{k(t)}\}_{j\in B(t)}$ and corresponding $W_k^{(t+1)}$ to each client $k, \forall k \in [M]$
  for client $k \in [M]$ **do**
    updates local model $\theta_k^{(t+1)}$ for τ steps by Eq. 9 via SGD

The server sends the dual variables $\{\lambda_j^{(t)}\}_{j\in B(t)}$ and the residual variables $\{s_j^{k(t)}\}_{j\in B(t)}$ of all samples, as well as the *corresponding* head $W_k^{(t+1)}$ to each client k.

(6) *Client updates local model parameters* θ_k. Finally, every client k locally updates the model parameters θ_k as follows:

$$\theta_k^{(t+1)} = \underset{\theta_k}{\operatorname{argmin}} \quad \beta_k \mathcal{R}_k(\theta_k) + \frac{1}{b}\sum_{j\in B(t)} \lambda_j^{(t)\top} f(x_j^k;\theta_k) W_k^{(t+1)} + \frac{\rho}{2b}\sum_{j\in B(t)} \left\| s_j^{k(t)} - f(x_j^k;\theta_k) W_k^{(t+1)} \right\|_F^2 . \tag{9}$$

Due to the nonconvexity of the loss function of DNNs, we use τ local steps of SGD to update the local model at each round with the objective of Eq. 9. We note that multiple local updates of Eq. 9 enabled by ADMM lead to better local models at each communication round compared to gradient-based methods, thus VIMADMM requires fewer communication rounds to converge as we will show in Section VI-A. These six steps of VIMADMM are summarized in Algorithm 1.

Note that ADMM auxiliary variables $\{z_j\}$ and dual variables $\{\lambda_j\}$ are only used during the training phase to help update server heads and local models. Therefore, in the test phase, for any sample x, the server directly uses the trained multiple heads to make prediction $\hat{y} = \sum_{k=1}^M h^k W_k$.

d) Extending VIMADMM to multiple non-linear heads: The server can learn non-linear transformation from the collected embeddings to uxiliary variables $\{z_j\}$ by employing multiple non-linear heads. To achieve this, we rewrite all $f(x_j^k, \theta_k)W_k$ as a more generalized form $g(f(x_j^k, \theta_k), W_k)$ from Eq. 2 to Eq. 9. Here, g can be a non-linear function parameterized by W_k. Consequently, the prediction for each sample j becomes $\hat{y}_j = \sum_{k=1}^M g(f(x_j^k, \theta_k), W_k)$. In this context, VIMADMM still aims to minimize the loss between z_j and ground-truth label y_j, as well as the difference between z_j and $\hat{y}_j$ during training in Eq. 4.

C. VFL without Model Splitting

a) Setup: Recall the VFL without model splitting setting described in § III-A. Let g parameterized by ψ_k be the local model (i.e., whole model) of client k, which outputs local logits $o_j^k = g(x_j^k;\psi_k) \in \mathbb{R}^{d_c}$ for each local feature x_j^k. The clients and the server aim to jointly solve the problem

$$\min_{\{\psi_k\}_{k=1}^M} \frac{1}{N}\sum_{j=1}^N \ell(\{o_j^1, \ldots, o_j^M\}, y_j) + \beta_k \sum_{k=1}^M \mathcal{R}_k(\psi_k), \forall k \in [M] \tag{10}$$

b) VIMADMM-J: In exisiting VFL frameworks, the server averages the local logits as final prediction $\sum_{k=i}^M o_j^k$, but these methods also suffers from the high communication cost by sending the gradients w.r.t. local logits to each client at *each* training step of the local model [25]. To solve this problem with our VIM framework, we adapt VIMADMM to the without model splitting setting and propose VIMADMM-J, where each feature extractor θ_k and each head W_k are held by the corresponding client k, and are always updated locally. The corresponding Algorithm 3 and detailed description are in Appendix A.

IV. Convergence Analysis for VIMADMM

In this section, we provide the convergence guarantee for VIMADMM, which is non-trivial due to the complexity of the *alternative optimization* between four sets of parameters $\{W_k\}, \{\theta_k\}, \{z_j\}, \{\lambda_j\}$. To convey the salient ideas of convergence analysis, we consider full batch, i.e., $B(t) = [N]$ and use the exact minimization solutions during training (Eq. 5, 6, 7) following [34].

We present our main results below and defer formal proofs to Appendix B due to space constraints.

Theorem 1. *Assume that $\mathcal{L}_{VIM}$ is bounded from below, that is $\underline{e} := \min_{\{\theta_k\},\{W_k\}} \mathcal{L}_{VIM}(\{\theta_k\},\{W_k\}) > -\infty$. Assume that $\ell(z;\cdot)$ is L-Lipschitz smooth w.r.t z and $\mathcal{L}_{\text{ADMM}}$ loss is strongly convex w.r.t $\{z_j\}, \{W_k\}, \{\theta_k\}$ with constant μ_z, μ_W, μ_θ respectively. Assume that the norm of W_k is bounded $\|W_k\| \le \sigma_W$, the local model $f(\cdot;\theta)$ has bounded gradient $\|\nabla f(\cdot;\theta)\| \le L_\theta$ and bounded output norm $\|f(\cdot;\theta)\| \le \sigma_\theta$. If Algorithm 1 is run, and there exists a ρ satisfying $\max\{L, \frac{2L^2}{\mu_z}\} < \rho < \min\{\frac{\mu_\theta}{L_\theta^2\sigma_W^2}, \frac{\mu_W}{\sigma_\theta^2}\}$, then we have the following:*

(A) *$\mathcal{L}_{\text{ADMM}}$ loss is monotonically decreasing and lower-bounded:*

$$\mathcal{L}_{\text{ADMM}}(\{W_k^{(t+1)}\},\{\theta_k^{(t+1)}\},\{z_j^{(t+1)}\},\{\lambda_j^{(t+1)}\}) < \mathcal{L}_{\text{ADMM}}(\{W_k^{(t)}\},\{\theta_k^{(t)}\},\{z_j^{(t)}\},\{\lambda_j^{(t)}\}) \tag{11}$$

$$\lim_{t\to\infty} \mathcal{L}_{\text{ADMM}}(\{W_k^{(t)}\},\{\theta_k^{(t)}\},\{z_j^{(t)}\},\{\lambda_j^{(t)}\}) \ge \underline{e} \tag{12}$$

(B) *Let $(\{W_k^*\},\{\theta_k^*\},\{z_j^*\},\{\lambda_j^*\})$ denote any limit points of the sequence $(\{W_k^{(t+1)}\},\{\theta_k^{(t+1)}\},\{z_j^{(t+1)}\},\{\lambda_j^{(t+1)}\})$ generated*

by Algorithm 1, then it is stationary:

$$z_j^* \in \arg\min_{z_j} \ell\left(z_j; y_j\right) + \lambda_j^{*\top}\left(\sum_{k=1}^{M} f(x_j^k;\theta_k^*)W_k^* - z_j\right) \text{ and}$$

$$\sum_{k=1}^{M} f(x_j^k;\theta_k^*)W_k^* = z_j^*, \forall j \in [N], \text{ and}$$

$$\beta_k \nabla \mathcal{R}_k(W_k^*) + \frac{1}{N}\sum_{j=1}^{N} \lambda_j^{*\top} f(x_j^k;\theta_k^*) = 0 \text{ and}$$

$$\beta_k \nabla \mathcal{R}_k(\theta_k^*) + \frac{1}{N}\sum_{j=1}^{N} \lambda_j^{*\top} \nabla f(x_j^k;\theta_k^*)W_k^* = 0, k \in [M]. \tag{13}$$

Proof Sketch. We obtain Eq. (11) by breaking down the changes of loss $\mathcal{L}_{\text{ADMM}}$ at each round t into the alternatively updates of four components: $\{\lambda_j^{(t+1)}\}$, $\{z_j^{(t+1)}\}$, $\{W_k^{(t+1)}\}$, and $\{\theta_k^{(t+1)}\}$, respectively. Through our assumptions and the optimality of the minimizers, we demonstrate that the combined loss decreases at each round. Next, to derive Eq. (12), we leverage the Lipschitz continuity of ℓ, the condition $\rho \geq L$, the lower bound of $\mathcal{L}_{\texttt{VIM}}$, and the fact that the quadratic loss term in $\mathcal{L}_{\text{ADMM}}$ is non-negative. Finally, by letting $t \to \infty$ and examining the optimality conditions of the minimizers, we drive Eq. (13). □

Remark. Theorem 1 (A) shows that VIMADMM converges, measured by the monotonically decreasing and convergent loss, and (B) establishes that any limit point is a stationary solution to the problem 4. Note that we make several assumptions in Theorem 1 to derive the above guarantees, as often made in ADMM analysis [34] for alternative optimization of multiple sets of variables. Specifically, we follow Hong et al. [34] to assume convexity, Lipschitz smoothness, and the bounded loss for convergence analysis of VIMADMM. Furthermore, we acknowledge that analyzing the local model can be challenging, given the complexity of DNNs, so we introduce an additional assumption that bounds the norm of the gradient and the output of local models, which could be practical when the model training exhibits stability. Similarly, we assume a bounded norm for the server model. By incorporating these assumptions, we aim to offer a more comprehensive understanding of the convergence behavior of VIMADMM.

V. CLIENT-LEVEL DIFFERENTIALLY PRIVATE VIM

While the raw features and local models are kept locally without sharing in VFL, sharing the model outputs such as local embeddings or predictions during the training process might also leak sensitive client information [50, 57]. Therefore, we aim to further protect the privacy of the local feature set X_k of each client k against potential adversaries such as honest-but-curious server and clients, and external attackers.

a) Threat Model: We consider different types of *potential adversaries based on their capabilities*: (1) Honest-but-curious server and clients: they follow the VFL protocol correctly but might try to infer private client information from information exchanged between the clients and server [68]. (2) External attackers: they are not directly involved in the VFL process but may observe the predicted results from the server and the communicated information during training, trying to extract private client information. Regarding *attack scenarios*, these attackers may conduct membership inference attacks [65] to determine whether the data of a specific VFL client was included during training. Our goal is to protect the local data of each client against potential attackers so that the attacker cannot make significant inferences about any single client's data. Next, we provide privacy-preserving mechanisms to satisfy client-level differential privacy (DP) guarantees.

b) Client-level DP: We begin with the (ϵ, δ)-DP definition, which guarantees that the change in a randomized algorithm's output distribution caused by an input difference is bounded.

Definition 1 ((ϵ, δ)-DP [21])**.** A randomized algorithm $\mathcal{M} : \mathcal{X}^n \mapsto \Theta$ is (ϵ, δ)-DP if for every pair of neighboring datasets $X, X' \in \mathcal{X}^n$ (i.e., differing only by one sample), and every possible (measurable) output set $E \subseteq \Theta$ the following inequality holds: $\Pr[\mathcal{M}(X) \in E] \leq e^{\epsilon} \Pr\left[\mathcal{M}\left(X'\right) \in E\right] + \delta$.

Next, we introduce client-level (ϵ, δ)-DP [54], which guarantees that the algorithm's output would not be changed much by differing one client.

Definition 2 (Client-level (ϵ, δ)-DP [54])**.** Let X and X' be adjacent datasets if they differ by all samples associated with a single client[2]. The mechanism $\mathcal{M}$ satisfies client-level (ϵ, δ)-DP if it meets Definition 1 with X and X' as adjacent datasets.

Remark. (1) *Client-level DP* protects the privacy of all local samples of each client [54]. The neighboring datasets in client-level DP are defined between client-adjacent datasets, denoted by $X = \{X_1, \ldots, X_k, \ldots, X_M\}$ and $X' = \{X_1, \ldots, X_k', \ldots, X_M\}$ for some client k. The algorithm's output should not change significantly if a single client's entire dataset is changed. (2) *User-level DP* is another prevalent privacy notion in FL literature, and its definition depends on how "user" is interpreted. If a "user" denotes a client/data owner in FL, then user-level DP aligns with client-level DP [27, 54, 2]. Additionally, a "user" in VFL might refer to an entity contributing different samples with partial features, where M VFL clients hold disjoint partial features $\{x_j^1, x_j^2, \ldots, x_j^M\}$ about the same user j [15, 62]. For example, different healthcare providers (VFL clients such as dental clinics and pharmacies) can hold different features about the same patient (user). In such cases, neighboring datasets are defined as those differing by all local samples associated with one user across all VFL client datasets. In this work, we focus on client-level DP due to its widespread adoption in FL [54].

Since the only shared information from clients is their local outputs, denoted as $\mathcal{A}_k$ for k-th client, we leverage the following DP mechanisms to perturb the local outputs

[2]We consider the "zero-out" notion for the neighboring dataset, following [59]: datasets are adjacent if any one client's local data is replaced with the special "zero" data (exactly zero for numeric data).

of each client k at every round t: (1) *clip* the whole local output matrix (either embeddings ${\mathcal{A}_k}^{(t)} = \{h_j^{k(t)}\}_{j\in B(t)}$ or logits ${\mathcal{A}_k}^{(t)} = \{o_j^{k(t)}\}_{j\in B(t)}$) with threshold C such that the ℓ_2*-sensitivity for each client is upper bounded* by C. That is, $\texttt{Clip}\left(\mathcal{A}_k, C\right) = \mathcal{A}_k \cdot \min\left(1, \frac{C}{\|\mathcal{A}_k\|_F}\right)$ where $\|\cdot\|_F$ is the Frobenius norm[3]. (2) Then we add scalar *Gaussian noise* independently to each cell of the matrix. The noise is sampled from $\mathcal{N}(0, \sigma^2 C^2)$, which is proportional to C and can *randomize the local output matrix of each client*: $\mathcal{A}_k \leftarrow \texttt{Clip}\left(\mathcal{A}_k, C\right) + \mathcal{N}\left(0, \sigma^2 C^2\right)$. Based on the above modification to Algorithm 1 and 3, we now provide their privacy guarantee in Theorem 2.

Theorem 2. *Given a total of M clients, T communication rounds, clipping threshold C and noise level σ, DP versions of Algorithm 1, 3 satisfy client-level* $(\frac{T\alpha}{2\sigma^2} + \log\frac{\alpha-1}{\alpha} - \frac{\log\delta+\log\alpha}{\alpha-1}, \delta)$*-DP for any $\alpha > 1$ and $0 < \delta < 1$.*

Proof Sketch. We derive the privacy guarantee using Rényi Differential Privacy (RDP) [56] as a bridge. We first leverage the RDP guarantee for the Gaussian mechanism [56] to analyze the privacy cost for one communication round under local output perturbation. Then we use RDP Composition property [56] to accumulate the privacy costs over T communication rounds. Finally, we convert client-level RDP guarantee into client-level DP guarantee [3]. Detailed proofs are deferred to Appendix C.

Remark. Since DP mechanisms (i.e., clipping and noise addition), are applied to each client's outputs (i.e., embedding or logits matrix) *locally*, these local outputs satisfy client-level local DP, protecting against privacy attacks from other clients, server or external attackers. That is, by observing the local outputs matrix of one client, other parties cannot determine the presence of that client's actual training data. The concatenated output matrix from all clients satisfies the same client-level DP guarantee based on DP parallel composition [55], due to non-overlapping nature of local data among clients.

Note that the aforementioned DP mechanisms do not protect the privacy of labels held by server. Therefore, we separately use state-of-the-art label DP mechanism [51] to protects server's label privacy via label perturbing, and conduct empirical evaluations of our method under label DP in Section VI-B2.

VI. Experiments

We conduct extensive experiments on four VFL datasets. We show that our proposed framework `VIM` achieves significantly faster convergence and higher accuracy than SOTA (Section VI-A), maintains higher utility under client-level DP and label DP (Section VI-B), and enables client-level explainability (Section VI-C).

1) Data and Models: We consider classification tasks on four datasets: MNIST [43], CIFAR [42], multi-modality dataset NUS-WIDE with image and textual features [14], and multi-view dataset ModelNet40 [66].

- MNIST [43] contains images with handwritten digits. We create the VFL scenario by splitting the input features evenly by rows for 14 clients. We use a fully connected model of two linear layers with ReLU activations as the local model.
- CIFAR [42] contains colour images. We split each image into patches for 9 clients. We use a standard CNN architecture from the PyTorch library [4] as the local model.
- NUS-WIDE [14] is a multi-modality dataset with 634 low-level image features and 1000 textual tag features. We distribute image features to 2 clients (300 dim and 334 dim), and text features to 2 clients (500 dim and 500 dim). We use a fully connected model of two linear layers with ReLU activations as the local model.
- ModelNet40 [66] is a multi-view image dataset, containing the shaded images from 12 views for the same objects. We use 4 views and distribute them to 4 clients respectively. We use ResNet-18 [33] as the local model.

We split each dataset into the train, validation, and test sets. See Table II for more details about the number of samples and the number of classes for each dataset.

TABLE II: Dataset description.

Dataset	# features	# classes d_c	# clients M	# samples train	validation	test
MNIST	28×28	10	14	54000	6000	10000
CIFAR	$32 \times 32 \times 3$	10	9	45000	5000	10000
NUS-WIDE	1634	5	4	54000	6000	10000
ModelNet40	$224 \times 224 \times 3 \times 12$	40	4	8877	966	2468

To prevent over-fitting (due to the potential over-parameterization with the large number of model parameters from all clients and server as a global model), we adopt standard stopping criteria, i.e., stop training when the model converges or the validation accuracy starts to drop more than 2%. More details about setups and hyperparameters are in Appendix D.

2) Baselines: We (1) compare `VIMADMM` with `VAFL` [12] `Split Learning` [70], and `FedBCD` [48] under *w/ model splitting* setting; (2) compare `VIMADMM-J` with `FDML` [36] under *w/o model splitting* setting. Particularly, in `VAFL`, the server aggregates local embeddings using their linear combination with learnable aggregation weights, and subsequently use these aggregated embeddings as input for the server model. Both `Split Learning` and `FedBCD` utilize concatenated local embeddings as server model input. Notably, in `VAFL` and `Split Learning`, the clients only perform one step of local update based the partial gradients from the server. Conversely, `FedBCD` employs the same (stale) partial gradients for τ local updates. In `FDML`, the server averages local logits, and sends aggregated logits back to clients at eatch communication round. The clients, who owns the copies of labels, can calculate the local gradient and execute one step of local update. Our empirical findings suggest that our ADMM-based methods outperform the aforementioned methods, due to the multiple local updates that utilize ADMM-related variables.

For fair comparisons, we use the same local models for all methods. Under w/ model splitting setting, owing to the strong

[3]The Frobenius norm for a $m \times n$ matrix A is $\|A\|_F = \sqrt{\sum_{i=1}^{m}\sum_{j=1}^{n}|a_{ij}|^2}$

[4]https://github.com/pytorch/opacus

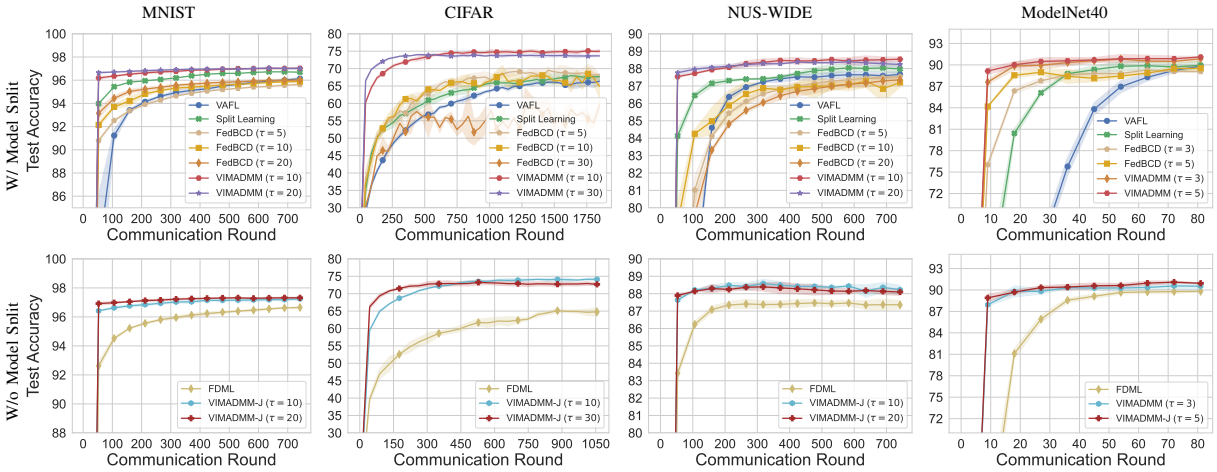

Fig. 1: Test accuracy of VFL methods under with model (first row) and without splitting (second row) settings on four datasets. Our methods (`VIMADMM` and `VIMADMM-J`) outperforms baselines due to multiple local updates enabled by ADMM ($\tau > 1$). Compared with `FedBCD` under different number of local steps τ, `VIMADMM` also achieves faster convergence and higher accuracy, which shows that the strategic utilization of ADMM-related variables in `VIMADMM` is more effective than the stale partial gradient in `FedBCD` for local updates.

feature extraction power of local DNN models, we utilize the linear model as server model by default. Additionally, we evaluate all methods with the non-linear server model, as detailed in Section VI-A2.

We further compare the utility of various VFL methods under differential privacy. Existing VFL frameworks (see Table I) focus on sample-level DP [12, 36, 35, 68, 15, 62], where neighboring datasets are defined as those differing by a single sample in a client's local dataset. In particular, `VAFL` [12] adds random noise to the output of each local embedding convolutional layer; VFL-PBM [68] quantizes local embeddings into differentially private integer vectors; `FDML` [36] and `Linear-ADMM` [35] add noise to local outputs. However, these methods lack exact privacy budget evaluations, providing only empirical utility under different levels of noise. Additionally, [62] perturbs local model weights to satisfy DP. However, it requires bounding the sensitivity of each layer's weights in the local model. To enable a fair comparison of VFL methods under DP guarantees, we evaluate all methods through our proposed DP mechanisms with perturbed local outputs to satisfy client-level (ϵ, δ)-DP guarantee. Notably, a mechanism satisfying (ϵ, δ) client-level DP also satisfies (ϵ, δ) sample-level DP based on their definitions. Since client-level DP offers stronger privacy protection, it has gained widespread adoption in FL [27, 54, 2, 5, 75]. Furthermore, we evaluate all VFL methods using label DP mechanism `ALIBI` [51] to separately satisfy the ϵ-label DP guarantee. We report the averaged results of three times of experiments with different random seeds.

A. Evaluation on Vanilla VFL

In this section, we evaluate the ADMM-based methods and baselines in terms of convergence rate, model performance and communication costs. Also, we show the generality of `VIMADMM` under non-linear server heads, and study `VIMADMM` performance under different ADMM penalty factor ρ.

1) Convergence rates and model performance: Figure 1 shows the convergence rates of all methods, where two `VIM` algorithms consistently outperform baselines. Concretely, **(1)** our ADMM-based methods converge faster and achieve higher accuracy than gradient-based baselines, especially on CIFAR. This is because the multiple local updates enabled by ADMM lead to higher-quality local models at each round, thereby speeding up the convergence. **(2)** `VIMADMM` outperforms `FedBCD` under various local steps. This superiority can be attributed to the use of ADMM-related variables for local updates τ in `VIMADMM`, which is more effective than stale partial gradients in `FedBCD`. **(3)** When # of local steps τ is larger, ADMM-based methods converge faster as the local models can be trained better with more local updates at each round.

Moreover, we empirically compare `VIMADMM` with `Linear-ADMM` [35]. While both VFL methods are rooted in ADMM, we propose new VFL optimization objective and algorithm with multiple heads that enable the ADMM decomposition for practical DNN training under model splitting. Results in Table III show that VIMADMM consistently outperforms `Linear-ADMM` on MNIST and NUS-WIDE. Compared to DNNs enabled by `VIMADMM`, the limitations of logistic regression in `Linear-ADMM` would be more evident when applied to more complex datasets like CIFAR and ModelNet40.

TABLE III: Performance comparison between `VIMADMM` and `Linear-ADMM` [35]. `VIMADMM` achieves higher accuracy.

	MNIST	NUS-WIDE
`VIMADMM`	97.13	88.51
`Linear-ADMM`	91.65	84.63

2) Non-linear server heads: To demonstrate the generality and applicability of `VIMADMM`, we evaluate `VIMADMM` when the server model is non-linear. Specifically, the head consists of multiple fully-connected layers accompanied by Dropout layers with 0.25 dropout rate and ReLu activation functions. For a fair comparison, we also use MLP server model architecture for other baseline methods. We use 3 layered MLP for NUS-WIDE

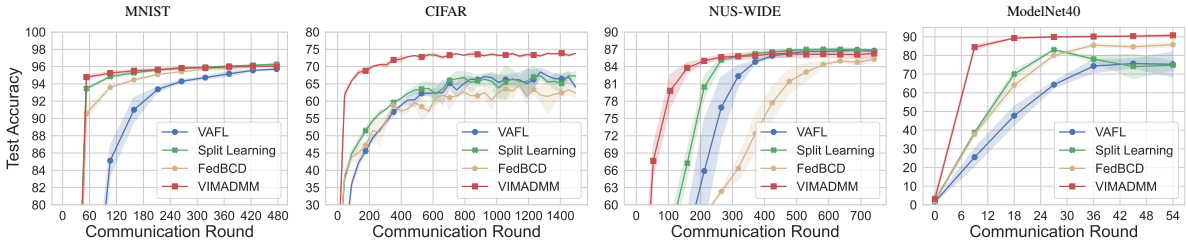

Fig. 2: Performance comparison when the server has the non-linear MLP model. ADMM-based method still outperforms other baselines under general architectures with the non-linear server model.

TABLE IV: Communication costs (in megabytes) comparison. VIMADMM requires lower communication costs per round than baselines under w/ model splitting setting. ADMM-based methods require lower communication costs to achieve the same target accuracy performance.

VFL setup	Method	Comm. costs per round			Comm. costs to reach target accuracy performance			
		Each client to server	Server to each client	Total	MNIST ($\geq 96.0\%$)	CIFAR ($\geq 65.0\%$)	NUS-WIDE ($\geq 85.0\%$)	ModelNet40 ($\geq 89.0\%$)
w/ model splitting	VAFL	0.23	0.23	0.46	4520.12	5381.40	397.37	134.96
	Split Learning	0.23	0.23	0.46	1738.51	4082.44	198.69	84.35
	FedBCD	0.23	0.23	0.46	4867.82	2597.92	397.37	118.09
	VIMADMM	0.23	**0.08**	**0.31**	**233.36**	**124.54**	**66.67**	**11.32**
w/o model splitting	FDML	0.039	**0.039**	**0.078**	405.13	617.76	33.07	89.13
	VIMADMM-J	0.039	0.078	0.117	**86.81**	**46.33**	**24.8**	**8.42**

and 2 layered MLP for other datasets. The evaluation results in Figure 2 show that our method still outperforms other baselines under general architectures with the non-linear server model.

3) Communication costs: Here we report the memory of parameters communicated between clients and the server to evaluate communication cost in Table IV. We use batch size 1024 and local embedding size 60 for all datasets. The overall embedding size scales with the number of clients. From Table IV, we observer that **(1)** for each round, all methods under w/ model splitting setting have the same number of parameters sent from each client to the server (i.e., 0.23 MB for a batch of embeddings), and VIMADMM has a smaller number of parameters sent from server to each client (i.e., 0.08 MB in total for a batch of dual variables, residual variables as well as one corresponding linear head) than VAFL, Split Learning and FedBCD (i.e., 0.23 MB for a batch of gradients w.r.t. embeddings). **(2)** With smaller # of communicated parameters at each round and faster convergence (i.e., smaller # of communicated rounds to achieve a target accuracy), VIMADMM requires significantly lower communication costs than baselines. For example, to achieve 65.0% accuracy on CIFAR, VAFL needs 5381.4 MB while VIMADMM only requires 124.54 MB, which is about 43x lower costs. Here we use $\tau = 20, 30, 20, 5$ for the four datasets respectively. **(3)** The results under w/o model splitting setting demonstrates that VIMADMM-J incurs lower communication costs than FDML to achieve the same accuracy, due to faster convergence with multiple local updates. **(4)** We note that the communication cost under w/o model splitting setting is generally lower than w/ model splitting setting, which is because the local logits have a lower dimension than local embeddings, i.e., $d_c < d_f$.

4) Effect of penalty factor ρ: In ADMM-based methods, we introduce one hyper-parameter – penalty factor ρ. Here we study the test accuracy of VIMADMM with different penalty factor ρ. The results in Figure 5 of Appendix D show that VIMADMM is not sensitive to ρ on four datasets, and we suggest that the practitioners choose the optimal ρ from 0.5 to 2, which does not influence the test accuracy significantly.

5) Evaluation on long-tail datasets: Long-tail datasets are characterized by a significant imbalance, where minority classes have far fewer samples than majority ones. This horizontal imbalance is distinct from the challenges addressed by VFL, where *the same sample (whether it belongs to a majority or minority class) is vertically split across multiple clients.* We compared the VIMADMM model, which consists of M local models followed by a server model, with a reference model in a centralized setting. This reference model has the same model size as one local model coupled with a server model. The results in Table V demonstrate that VIMADMM is still effective on challenging long-tail training datasets, yielding results comparable to those of the reference model in a centralized setting. We defer more discussion and detailed experimental setups to Appendix D.

TABLE V: Accuracy and fairness (measured by Standard Deviation of class-wise accuracy) on balanced data and long-tail data.

	balanced MNIST	long-tail MNIST	balanced CIFAR	long-tail CIFAR
VIMADMM	97.13 (0.76)	95.69 (1.58)	75.25 (9.17)	62.81 (15.27)
Reference model in centralized setting	98.19 (0.45)	95.02 (2.70)	77.61 (9.20)	66.11 (15.29)

6) Fairness implication: A common fairness definition is to enforce accuracy parity between protected groups [82]. Here we study the fairness implications of VIMADMM on achieving accuracy parity, at both the class and client levels: **(1)** when considering class-level accuracy parity, a fair model should exhibit equalized accuracy for each class [67, 76], indicating that the model's accuracy is statistically independent of the ground truth label. We use the Standard Deviation of class-wise accuracy [76] to evaluate fairness, where a lower value indicates higher fairness. The results in Table V show that VIMADMM performs comparably or even better in fairness than the reference model in a centralized setting, across MNIST and CIFAR10 datasets with both balanced and long-tail distributions. **(2)** Furthermore, client-level accuracy parity is a prevalent criterion for fairness in FL [44, 45], measuring the degree of

TABLE VI: Utility of VFL methods under *user-level DP*. ADMM-based methods maintain higher utility.

VFL setup	Method	MNIST			CIFAR			NUS-WIDE			ModelNet40		
		$\epsilon=\infty$	$\epsilon=8$	$\epsilon=1$	$\epsilon=\infty$	$\epsilon=8$	$\epsilon=1$	$\epsilon=\infty$	$\epsilon=8$	$\epsilon=1$	$\epsilon=\infty$	$\epsilon=8$	$\epsilon=1$
w/ model splitting	VAFL	96.86	22.29	11.31	66.39	16.82	14.91	87.81	38.27	38.19	90.07	4.66	4.29
	Split Learning	96.92	56.53	16.77	68.32	21.09	15.8	88.25	38.29	33.05	89.98	18.19	6.28
	FedBCD	96.59	66.07	65.05	71.2	70.67	55.42	87.59	42.95	41.02	89.87	88.3	87.02
	VIMADMM	97.13	92.35	92.09	**75.25**	**73.83**	**61.65**	**88.51**	83.77	83.51	**91.32**	**91.29**	**91.18**
w/o model splitting	FDML	97.06	92.02	85.01	66.8	41.07	35.25	87.67	79.58	67.38	89.86	54.7	43.4
	VIMADMM-J	**97.37**	**92.71**	**92.33**	74.48	72.36	58.64	88.46	**84.94**	**84.88**	91.13	90.13	89.37

TABLE VII: Utility of VFL methods under *label-level DP*. ADMM-based methods maintain higher utility.

VFL setup	Method	MNIST			CIFAR			NUS-WIDE			ModelNet40		
		$\epsilon=\infty$	$\epsilon=2.8$	$\epsilon=1.4$	$\epsilon=\infty$	$\epsilon=2.8$	$\epsilon=1.4$	$\epsilon=\infty$	$\epsilon=2.8$	$\epsilon=1.4$	$\epsilon=\infty$	$\epsilon=2.8$	$\epsilon=1.4$
w/ model splitting	VAFL	96.86	94.27	51.68	66.39	54.6	38.44	87.81	85.77	60.41	90.07	45.26	2.59
	Split Learning	96.92	94.93	91.75	68.32	57.12	49.71	88.25	85.86	82.3	89.98	65.68	33.79
	FedBCD	96.59	94.47	87.95	71.2	61.05	46.14	87.59	85.62	64.01	89.87	65.92	43.15
	VIMADMM	97.13	95.48	92.8	**75.25**	**65.07**	52.97	**88.51**	86.62	82.43	**91.32**	76.70	**46.39**
w/o model splitting	FDML	97.06	94.97	91.87	66.8	58.78	49.83	87.67	85.79	82.37	89.86	64.99	29.74
	VIMADMM-J	**97.37**	**95.80**	**93.25**	74.48	64.04	**53.49**	88.46	**86.74**	**82.71**	91.13	**77.15**	45.22

uniformity in performance across clients. Notably, in VFL, all clients share the same prediction for each sample, where each of them contributes partial features. Consequently, all clients inherently achieve the same accuracy, fulfilling client-level accuracy parity by the nature of VFL.

B. Evaluation on Differentially Private VFL

We evaluate the utility of ADMM-based methods and baselines under client-level DP and label DP, which protect the privacy of local features and server labels, respectively.

1) Utility under client-level DP (privacy of client data): We report the utility under $\epsilon = 8$ and $\epsilon = 1$ client-level DP. To ensure fair comparison, we perform a grid search for the combination of hyperparameters, including noise scale σ, clipping threshold C, and learning rate η, for all methods (details are deferred to Appendix D). Table VI shows that **(1)** the accuracy of ADMM-based methods under DP is on par with the non-private accuracy ($\epsilon = \infty$) on MNIST, NUS-WIDE and ModelNet40. Nevertheless, there is a discernible decrease of 13.6% for VIMADMM on CIFAR when $\epsilon = 1$, which underscores the inherent privacy-utility trade-off for algorithms with formal DP privacy guarantees [1]. **(2)** Our ADMM-based methods reach significantly higher utility than gradient-based methods, especially under small ϵ. We attribute this to the fact that ADMM-based methods converge in fewer rounds than gradient-based methods at each round, which is also evident in the non-DP setting as shown in Figure 1. *This rapid convergence is critical for DP, since the privacy budget ϵ is consumed quickly as communication rounds increase.* The fast convergence and high utility of VIMADMM under DP compared to other baselines can be interpreted through two lenses. First, **multiple local updates** lead to a more effectively trained local model at each round. As a consequence, both FedBCD and VIMADMM demonstrate a markedly better DP-utility tradeoff compared to VAFL and Split Learning, as illustrated in Table VI. Furthermore, we explicitly investigate the influence of τ on the utility of VIMADMM under $\epsilon = 1$ in Table VIII. The results show that opting for a $\tau > 1$ yields substantially enhanced accuracy than when $\tau = 1$ (e.g., 14.68% improvement on CIFAR). Second, **update mechanism of ADMM** empowers clients to *independently* update their local models w.r.t the ADMM sub-objective (Eq. 9). It is worth noting that during this local forward/backward computation based on Eq. 9, clients do *not* add noise locally, since local models always remain in their possession without sharing. Clients only need to perturb local embeddings that are sent to the server (i.e., output perturbation). Consequently, even though the server leverages these perturbed embeddings to derive ADMM-related variables, the clients will re-calculate clean embeddings during forward pass of Eq. 9 based on the received ADMM-related variables for local model updates. This updating mechanism potentially facilitate convergence under DP. In contrast, gradients-based methods solely rely on the partial gradients, which are derived from perturbed embeddings, for local update, leading to compromised utility.

TABLE VIII: A larger number of local steps τ leads to better utility of VIMADMM under client-level DP $\epsilon = 1$.

MNIST		CIFAR		NUS-WIDE	
τ	$\epsilon=1$	τ	$\epsilon=1$	τ	$\epsilon=1$
1	90.63	1	48.19	1	79.38
5	90.84	10	61.65	3	82.58
20	92.09	30	62.87	10	83.51

Methods w/o model splitting (FDML, VIMADMM-J) generally performs better than methods w/ model splitting. This is mainly because the logits have a smaller dimension than the embeddings, and the total amount of noise added to the logits output is smaller than the embedding output; thus VFL w/o model splitting methods retain higher utility under DP.

Additionally, the utility under client-level DP VFL is not directly comparable to sample-level DP in centralized ML [1] or client-level DP in standard (horizontal) FL [54] due to the *unique properties* of VFL. For instances, **(1)** the *dimension of DP-perturbed information* in VFL can be smaller (e.g., a batch of local embeddings or local logits) than the existing centralized learning or FL (e.g., gradients or model updates of a large model), which could lead to the higher utility under DP noise. **(2)** The private local training set of VFL for each user has *a smaller raw feature dimension* (i.e., $1/M$ if features are divided evenly among M clients) than the entire dataset (or local dataset) in the central setting (or horizontal FL) and it *does not contain the labels*, which leads to a different dataset

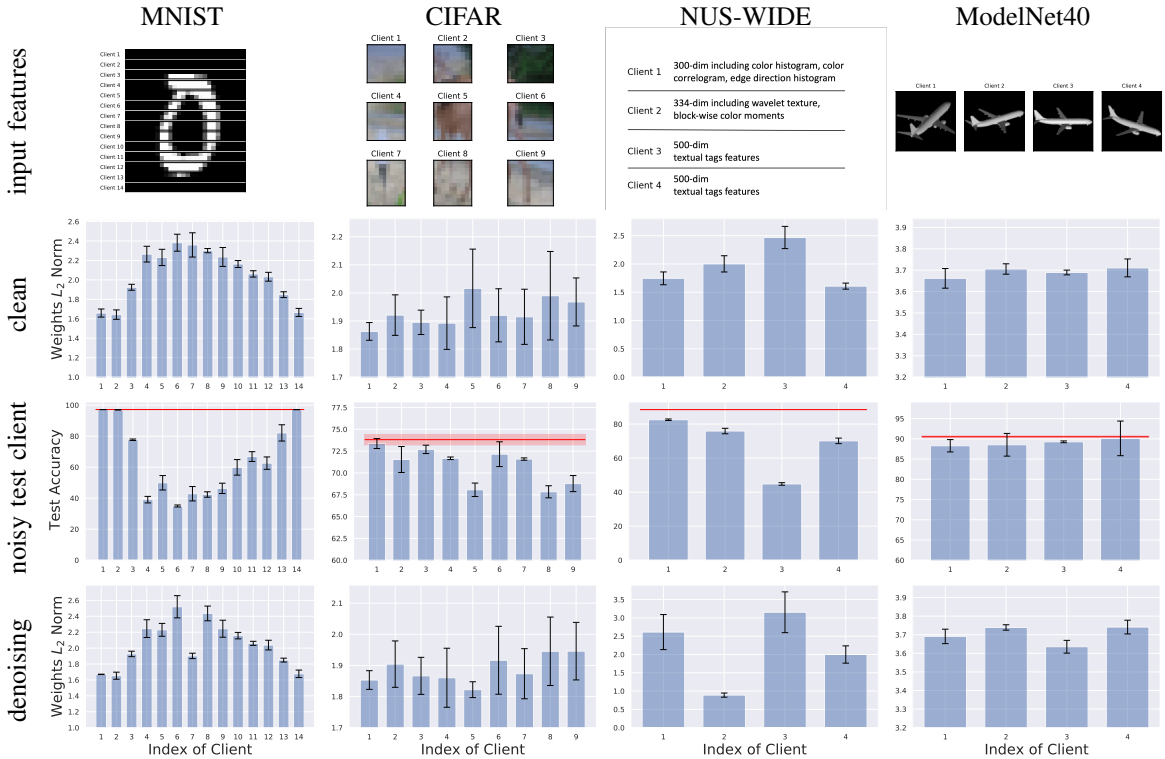

Fig. 3: Client-level explainability of VIM. Row 1 visualizes the input features. Row 2 shows the weights norm of linear heads. Row 3 shows the test accuracy when each client's test input features are perturbed (red line denotes the clean test accuracy). Row 4 shows the weights norm of linear heads under only one noisy client.

notion in DP definition. In our work, we follow existing privacy notions in VFL to protect each user's local training set [12, 36] with proposed client-level DP mechanisms.

2) Utility under label DP (privacy of server labels): To protect the privacy of the labels in the server with formal privacy guarantee, we utilize the existing state-of-the-art label DP mechanism ALIBI [51], which is originally proposed in centralized learning. We evaluate all methods under label DP with a privacy budget $\epsilon = 2.8$ and $\epsilon = 1.4$, which are obtained by adding Laplacian noise with noise parameter $\lambda_{\text{Lap}} = 1$ and $\lambda_{\text{Lap}} = 2$, respectively, on the labels *once* before VFL training, and we use randomized labels for training based on ALIBI. In particular, ALIBI post-processes the model predictions through Bayesian inference to improve the model utility under noisy labels [51]. The results on Table VII show that ADMM-based methods retain higher utility than gradient-based methods under the label DP. This could be due to two potential reasons: **(1)** the additional variables introduced by ADMM (i.e., auxiliary variables $\{z_j\}$ and dual variables $\{\lambda_j\}$) are dynamically adjusted during training, which might contribute to a more robust optimization [16] for VFL models (i.e., $\{W_k\}, \{\theta_k\}$) against label noises, and **(2)** multiple updates in each round could result in improved local models. As shown in Table IX, more local steps τ can significantly enhance the utility of VIMADMM under label-level DP $\epsilon = 1.4$ ($\sim$10% and $\sim$13% improvement for CIFAR and NUS-WIDE, respectively).

C. Client-level Explainability of VIM

In this section, we first visualize the local embeddings of clients, which are diverse, stemming from the distinct input features of clients. This also justifies the multi-head design of VIM that can reweight the embeddings based on their importance. Then, we show that the weights norm of learned linear heads can indeed reflect the importance of local clients, which enables functionalities such as test-time noise validation, client denoising, and summarization.

TABLE IX: A larger number of local steps τ leads to better utility of VIMADMM under label-level DP $\epsilon = 1.4$.

MNIST		CIFAR		NUS-WIDE	
τ	$\epsilon = 1.4$	τ	$\epsilon = 1.4$	τ	$\epsilon = 1.4$
1	92.28	1	46.08	1	69.41
5	92.51	10	52.97	3	81.43
20	92.8	30	56.13	10	82.43

1) T-SNE of Local Embeddings: In row 1 of Figure 3, we show the raw feautures of different clients on four datasets. The quality of features can vary among clients. For instance, in MNIST, since the digit always occupies the center, clients hold black background pixels might not provide useful information for the classification task, and thus are less important. The T-SNE [69] visualizations in Figure 4 reveal that important clients learn better local embeddings than unimportant clients on MNIST, CIFAR and NUS-WIDE. Specifically, in NUS-WIDE, client #3 produces linear separable local embeddings (left), which are better than client #4's embeddings (right) that overlap different classes. For ModelNet40, since clients with multi-view data are of similar importance, their local embeddings exhibit similarities and demonstrate linear separability. A scrutiny of these local embeddings confirms that the unique characteristic of input features in each client lead to varied local embeddings.

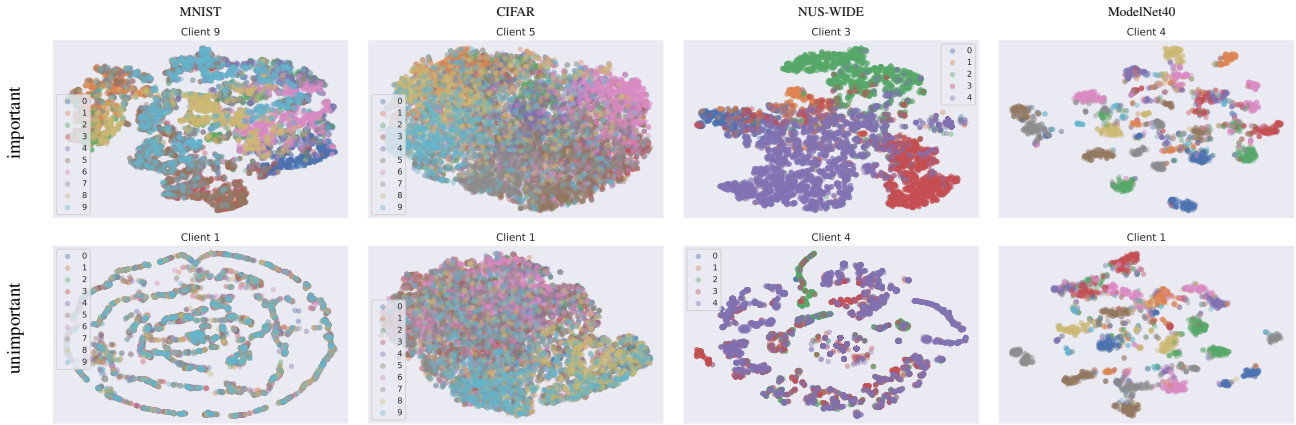

Fig. 4: T-SNE visualizations of local embeddings from important client and unimportant client for VIMADMM.

Consequently, we employ multiple heads as the server model, allowing us to account for the diverse feature quality across clients and aptly reweight the local embeddings.

2) Client Importance: Given a trained VIMADMM model, we plot the weights norm of each client's corresponding linear heads in Figure 3 row 2. Combining it with row 1, we find that *the client with important local features indeed results in high weights*[5]. For example, clients #6, #7, #8 in MNIST holding middle rows of images that contain the center of digits, have high weights, while clients #1, #14 holding the black background pixels have low weights. A similar phenomenon is observed on CIFAR for client #5 (center) and client #1 (corner). On CIFAR, clients #8, #9 also have high weights, which is because the objects on CIFAR also appear on the right bottom corner. On ModelNet40, clients have complementary views of the same objects, so their features have similar importance, leading to similar weights norms. Based on our observation, we conclude that *the weights of linear heads can reflect the importance of local clients.* We use this principle to infer that, for NUS-WIDE, the first 500 dim. of textual features have higher importance than other multimodality features, resulting in the high weights norm of client 3.

3) Client Importance Validation via Noisy Test Client: Given a trained VIMADMM model, we add Gaussian noise to the test local features to verify the client-level importance indicated by the linear heads. For each time, we only perturb the features of one client and keep other clients' features unchanged. The results in Figure 3 row 3 show that *perturbing the client with high weights affects more for the test accuracy*, which verifies that clients with higher weights are more important.

4) Client Denoising: We study the denoising ability of VIM under training-time noisy clients. We construct one noisy client (i.e., client #7, #5, #2, #3 for MNIST, CIFAR, NUS-WIDE, ModelNet40 respectively) by adding Gaussian noise to its local features and re-train the VIMADMM model. The obtained weights norm in Figure 3 row 4 shows that VIMADMM can *automatically detect the noisy client and lower its weights* (compared to the clean one in row 2). Table XIII in Appendix D shows that VIMADMM outperforms baselines with faster convergence and higher accuracy under noisy clients.

[5]Here the weights of clients refer to the weights of the client's corresponding linear head owned by the server.

5) Client Summarization: Regarding client summarization, **(1)** we first rank the importance of clients according to their weights norm (Figure 3 row 2), then we select $u\%$ proportion of the most "important" clients to re-train the VIMADMM model. We find that *its performance is close to the one trained by all clients*. Table X shows that the test accuracy-drop of training with 50% of the most important clients is less than 1% on MNIST and NUS-WIDE, and less than 4% on CIFAR; the accuracy-drop of training with 20% of the most important clients is less than 10% on all datasets. **(2)** We select $u\%$ proportion of the least important clients to re-train the model, and we find that its performance is significantly lower than the one trained with important clients, which indicates the effectiveness of VIM for client selection. **(3)** For the multi-view dataset ModelNet40, we find that the test accuracy of models trained with 12, 8, and 4 clients are similar, i.e., 91.04%, 90.69%, and 90.64%, suggesting that a few views can already provide sufficient training information and the agents with multiview data are of similar importance which is also reflected by our linear head weights.

TABLE X: Functionality of client summarization enabled by VIMADMM.

Client ratio	Type	MNIST	CIFAR	NUS-WIDE
100%	all	97.13	75.25	88.51
50%	important	96.58	70.28	87.29
	unimportant	78.11	62.67	75.80
20%	important	88.72	66.06	80.28
	unimportant	29.11	54.99	59.34

VII. Conclusions

We propose a VFL framework with multiple linear heads (VIM) and an ADMM-based method (VIMADMM) for efficient communication. We provide the convergence guarantee for VIMADMM. We also introduce user-level differential privacy mechanism for VIM and prove the privacy guarantee. Extensive experiments verify the superior performance of our algorithms under vanilla VFL and DP VFL and show that VIM enables client-level explainability.

ACKNOWLEDGEMENT

The authors thank Yunhui Long, Linyi Li, Yangjun Ruan, Weixin Chen, and the anonymous reviewers for their valuable feedback and suggestions.

This work is partially supported by the National Science Foundation under grant No. 1910100, No. 2046726, No. 2229876, DARPA GARD, the National Aeronautics and Space Administration (NASA) under grant No. 80NSSC20M0229, Alfred P. Sloan Fellowship, the Amazon research award, and the eBay research grant.

REFERENCES

[1] Martin Abadi, Andy Chu, Ian Goodfellow, H Brendan McMahan, Ilya Mironov, Kunal Talwar, and Li Zhang. Deep learning with differential privacy. In *Proceedings of the 2016 ACM SIGSAC conference on computer and communications security*, pages 308–318, 2016.

[2] Naman Agarwal, Ananda Theertha Suresh, Felix Yu, Sanjiv Kumar, and H Brendan McMahan. cpsgd: communication-efficient and differentially-private distributed sgd. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, pages 7575–7586, 2018.

[3] Borja Balle, Gilles Barthe, Marco Gaboardi, Justin Hsu, and Tetsuya Sato. Hypothesis testing interpretations and renyi differential privacy. In *International Conference on Artificial Intelligence and Statistics*, pages 2496–2506. PMLR, 2020.

[4] Michael Ben-Or, Shafi Goldwasser, and Avi Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation. In *Proceedings of the twentieth annual ACM symposium on Theory of computing*, pages 1–10, 1988.

[5] Abhishek Bhowmick, John Duchi, Julien Freudiger, Gaurav Kapoor, and Ryan Rogers. Protection against reconstruction and its applications in private federated learning. *arXiv preprint arXiv:1812.00984*, 2018.

[6] Keith Bonawitz, Vladimir Ivanov, Ben Kreuter, Antonio Marcedone, H Brendan McMahan, Sarvar Patel, Daniel Ramage, Aaron Segal, and Karn Seth. Practical secure aggregation for privacy-preserving machine learning. In *proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 1175–1191, 2017.

[7] Stephen Boyd, Neal Parikh, and Eric Chu. *Distributed optimization and statistical learning via the alternating direction method of multipliers*. Now Publishers Inc, 2011.

[8] Theodora S Brisimi, Ruidi Chen, Theofanie Mela, Alex Olshevsky, Ioannis Ch Paschalidis, and Wei Shi. Federated learning of predictive models from federated electronic health records. *International journal of medical informatics*, 112:59–67, 2018.

[9] Adam Byerly, Tatiana Kalganova, and Ian Dear. No routing needed between capsules. *Neurocomputing*, 463:545–553, 2021.

[10] Timothy Castiglia, Shiqiang Wang, and Stacy Patterson. Flexible vertical federated learning with heterogeneous parties. *IEEE Transactions on Neural Networks and Learning Systems*, 2023.

[11] Timothy J Castiglia, Anirban Das, Shiqiang Wang, and Stacy Patterson. Compressed-vfl: Communication-efficient learning with vertically partitioned data. In *International Conference on Machine Learning*, pages 2738–2766. PMLR, 2022.

[12] Tianyi Chen, Xiao Jin, Yuejiao Sun, and Wotao Yin. Vafl: a method of vertical asynchronous federated learning. *arXiv preprint arXiv:2007.06081*, 2020.

[13] Kewei Cheng, Tao Fan, Yilun Jin, Yang Liu, Tianjian Chen, Dimitrios Papadopoulos, and Qiang Yang. Secureboost: A lossless federated learning framework. *IEEE Intelligent Systems*, 36(6):87–98, 2021.

[14] Tat-Seng Chua, Jinhui Tang, Richang Hong, Haojie Li, Zhiping Luo, and Yantao Zheng. Nus-wide: a real-world web image database from national university of singapore. In *Proceedings of the ACM international conference on image and video retrieval*, pages 1–9, 2009.

[15] Vincent Cohen-Addad, Praneeth Kacham, Vahab Mirrokni, and Peilin Zhong. Differentially private vertical federated learning primitives.

[16] Jiahao Ding, Xinyue Zhang, Mingsong Chen, Kaiping Xue, Chi Zhang, and Miao Pan. Differentially private robust admm for distributed machine learning. In *2019 IEEE International Conference on Big Data (Big Data)*, pages 1302–1311. IEEE, 2019.

[17] Jinshuo Dong, Aaron Roth, and Weijie J Su. Gaussian differential privacy. *arXiv preprint arXiv:1905.02383*, 2019.

[18] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. In *International Conference on Learning Representations*, 2021.

[19] Cynthia Dwork. A firm foundation for private data analysis. *Communications of the ACM*, 54(1):86–95, 2011.

[20] Cynthia Dwork, Krishnaram Kenthapadi, Frank McSherry, Ilya Mironov, and Moni Naor. Our data, ourselves: Privacy via distributed noise generation. In *Advances in Cryptology – EUROCRYPT*, 2006.

[21] Cynthia Dwork and Aaron Roth. The algorithmic foundations of differential privacy. *Foundations and Trends in Theoretical Computer Science*, 9(3-4):211–407, 2014.

[22] Anis Elgabli, Jihong Park, Sabbir Ahmed, and Mehdi Bennis. L-fgadmm: Layer-wise federated group admm for communication efficient decentralized deep learning. In *2020 IEEE Wireless Communications and Networking Conference (WCNC)*, pages 1–6. IEEE, 2020.

[23] Anis Elgabli, Jihong Park, Amrit S Bedi, Mehdi Bennis, and Vaneet Aggarwal. Gadmm: Fast and communication efficient framework for distributed machine learning. *J.*

Mach. Learn. Res., 21(76):1–39, 2020.

[24] Siwei Feng and Han Yu. Multi-participant multi-class vertical federated learning. *arXiv preprint arXiv:2001.11154*, 2020.

[25] Chong Fu, Xuhong Zhang, Shouling Ji, Jinyin Chen, Jingzheng Wu, Shanqing Guo, Jun Zhou, Alex X Liu, and Ting Wang. Label inference attacks against vertical federated learning. In *31st USENIX Security Symposium (USENIX Security 22)*, Boston, MA, August 2022. USENIX Association.

[26] Fangcheng Fu, Xupeng Miao, Jiawei Jiang, Huanran Xue, and Bin Cui. Towards communication-efficient vertical federated learning training via cache-enabled local updates. *Proc. VLDB Endow.*, 15(10):2111–2120, jun 2022.

[27] Robin C Geyer, Tassilo Klein, and Moin Nabi. Differentially private federated learning: A client level perspective. *arXiv preprint arXiv:1712.07557*, 2017.

[28] Ran Gilad-Bachrach, Nathan Dowlin, Kim Laine, Kristin Lauter, Michael Naehrig, and John Wernsing. Cryptonets: Applying neural networks to encrypted data with high throughput and accuracy. In *International Conference on Machine Learning*, pages 201–210. PMLR, 2016.

[29] Prashant Gohel, Priyanka Singh, and Manoranjan Mohanty. Explainable ai: current status and future directions. *arXiv preprint arXiv:2107.07045*, 2021.

[30] Bin Gu, Zhiyuan Dang, Xiang Li, and Heng Huang. Federated doubly stochastic kernel learning for vertically partitioned data. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 2483–2493, 2020.

[31] Andrew Hard, Kanishka Rao, Rajiv Mathews, Françoise Beaufays, Sean Augenstein, Hubert Eichner, Chloé Kiddon, and Daniel Ramage. Federated learning for mobile keyboard prediction. *arXiv preprint arXiv:1811.03604*, 2018.

[32] Stephen Hardy, Wilko Henecka, Hamish Ivey-Law, Richard Nock, Giorgio Patrini, Guillaume Smith, and Brian Thorne. Private federated learning on vertically partitioned data via entity resolution and additively homomorphic encryption. *arXiv preprint arXiv:1711.10677*, 2017.

[33] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[34] Mingyi Hong, Zhi-Quan Luo, and Meisam Razaviyayn. Convergence analysis of alternating direction method of multipliers for a family of nonconvex problems. *SIAM Journal on Optimization*, 26(1):337–364, 2016.

[35] Yaochen Hu, Peng Liu, Linglong Kong, and Di Niu. Learning privately over distributed features: An admm sharing approach. *arXiv preprint arXiv:1907.07735*, 2019.

[36] Yaochen Hu, Di Niu, Jianming Yang, and Shengping Zhou. Fdml: A collaborative machine learning framework for distributed features. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 2232–2240, 2019.

[37] Yuzheng Hu, Fan Wu, Qinbin Li, Yunhui Long, Gonzalo Munilla Garrido, Chang Ge, Bolin Ding, David Forsyth, Bo Li, and Dawn Song. Sok: Privacy-preserving data synthesis. *S&P*, 2024.

[38] Zonghao Huang, Rui Hu, Yuanxiong Guo, Eric Chan-Tin, and Yanmin Gong. Dp-admm: Admm-based distributed learning with differential privacy. *IEEE Transactions on Information Forensics and Security*, 15:1002–1012, 2019.

[39] Xiao Jin, Pin-Yu Chen, Chia-Yi Hsu, Chia-Mu Yu, and Tianyi Chen. Catastrophic data leakage in vertical federated learning. *Advances in Neural Information Processing Systems*, 34, 2021.

[40] Yan Kang, Yang Liu, and Tianjian Chen. Fedmvt: Semi-supervised vertical federated learning with multiview training. *arXiv preprint arXiv:2008.10838*, 2020.

[41] Prannay Khosla, Piotr Teterwak, Chen Wang, Aaron Sarna, Yonglong Tian, Phillip Isola, Aaron Maschinot, Ce Liu, and Dilip Krishnan. Supervised contrastive learning. *Advances in Neural Information Processing Systems*, 33:18661–18673, 2020.

[42] Alex Krizhevsky. Learning multiple layers of features from tiny images. Technical report, 2009.

[43] Yann LeCun and Corinna Cortes. MNIST handwritten digit database. 2010.

[44] Tian Li, Shengyuan Hu, Ahmad Beirami, and Virginia Smith. Ditto: Fair and robust federated learning through personalization. In *International Conference on Machine Learning*, pages 6357–6368. PMLR, 2021.

[45] Tian Li, Maziar Sanjabi, Ahmad Beirami, and Virginia Smith. Fair resource allocation in federated learning. In *International Conference on Learning Representations*, 2020.

[46] Yang Liu, Yan Kang, Xinwei Zhang, Liping Li, Yong Cheng, Tianjian Chen, Mingyi Hong, and Qiang Yang. A communication efficient collaborative learning framework for distributed features. *arXiv preprint arXiv:1912.11187*, 2019.

[47] Yang Liu, Zhihao Yi, and Tianjian Chen. Backdoor attacks and defenses in feature-partitioned collaborative learning. *arXiv preprint arXiv:2007.03608*, 2020.

[48] Yang Liu, Xinwei Zhang, Yan Kang, Liping Li, Tianjian Chen, Mingyi Hong, and Qiang Yang. Fedbcd: A communication-efficient collaborative learning framework for distributed features. *IEEE Transactions on Signal Processing*, 70:4277–4290, 2022.

[49] Scott M Lundberg and Su-In Lee. A unified approach to interpreting model predictions. *Advances in neural information processing systems*, 30, 2017.

[50] Aravindh Mahendran and Andrea Vedaldi. Understanding deep image representations by inverting them. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 5188–5196, 2015.

[51] Mani Malek Esmaeili, Ilya Mironov, Karthik Prasad, Igor Shilov, and Florian Tramer. Antipodes of label differential

privacy: Pate and alibi. *Advances in Neural Information Processing Systems*, 34:6934–6945, 2021.

[52] Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Aguera y Arcas. Communication-Efficient Learning of Deep Networks from Decentralized Data. In *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics*, volume 54 of *Proceedings of Machine Learning Research*, pages 1273–1282. PMLR, 20–22 Apr 2017.

[53] H. Brendan McMahan, Daniel Ramage, Kunal Talwar, and Li Zhang. Learning differentially private recurrent language models. In *International Conference on Learning Representations*, 2018.

[54] H Brendan McMahan, Daniel Ramage, Kunal Talwar, and Li Zhang. Learning differentially private recurrent language models. In *International Conference on Learning Representations*, 2018.

[55] Frank D McSherry. Privacy integrated queries: an extensible platform for privacy-preserving data analysis. In *Proceedings of the 2009 ACM SIGMOD International Conference on Management of data*, pages 19–30, 2009.

[56] Ilya Mironov. Rényi differential privacy. In *2017 IEEE 30th computer security foundations symposium (CSF)*, pages 263–275. IEEE, 2017.

[57] Nicolas Papernot, Patrick McDaniel, Arunesh Sinha, and Michael P Wellman. Sok: Security and privacy in machine learning. In *2018 IEEE European Symposium on Security and Privacy (EuroS&P)*, pages 399–414. IEEE, 2018.

[58] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.

[59] Natalia Ponomareva, Hussein Hazimeh, Alex Kurakin, Zheng Xu, Carson Denison, H Brendan McMahan, Sergei Vassilvitskii, Steve Chien, and Abhradeep Guha Thakurta. How to dp-fy ml: A practical guide to machine learning with differential privacy. *Journal of Artificial Intelligence Research*, 77:1113–1201, 2023.

[60] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, et al. Learning transferable visual models from natural language supervision. In *International Conference on Machine Learning*, pages 8748–8763. PMLR, 2021.

[61] Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv preprint arXiv:1511.06434*, 2015.

[62] Thilina Ranbaduge and Ming Ding. Differentially private vertical federated learning. *arXiv preprint arXiv:2211.06782*, 2022.

[63] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. " why should i trust you?" explaining the predictions of any classifier. In *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*, pages 1135–1144, 2016.

[64] Bita Darvish Rouhani, M Sadegh Riazi, and Farinaz Koushanfar. DeepSecure: Scalable provably-secure deep learning. In *Proceedings of the 55th Annual Design Automation Conference*, pages 1–6, 2018.

[65] Reza Shokri, Marco Stronati, Congzheng Song, and Vitaly Shmatikov. Membership inference attacks against machine learning models. In *2017 IEEE symposium on security and privacy (SP)*, pages 3–18. IEEE, 2017.

[66] Jong-Chyi Su, Matheus Gadelha, Rui Wang, and Subhransu Maji. A deeper look at 3d shape classifiers. In *Second Workshop on 3D Reconstruction Meets Semantics, ECCV*, 2018.

[67] Davoud Ataee Tarzanagh, Bojian Hou, Boning Tong, Qi Long, and Li Shen. Fairness-aware class imbalanced learning on multiple subgroups. In *Uncertainty in Artificial Intelligence*, pages 2123–2133. PMLR, 2023.

[68] Linh Tran, Timothy Castiglia, Stacy Patterson, and Ana Milanova. Privacy tradeoffs in vertical federated learning. In *Federated Learning Systems (FLSys) Workshop @ MLSys 2023*, 2023.

[69] Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of Machine Learning Research*, 9(86):2579–2605, 2008.

[70] Praneeth Vepakomma, Otkrist Gupta, Tristan Swedish, and Ramesh Raskar. Split learning for health: Distributed deep learning without sharing raw patient data. *arXiv preprint arXiv:1812.00564*, 2018.

[71] Rahul Vigneswaran, Marc T Law, Vineeth N Balasubramanian, and Makarand Tapaswi. Feature generation for long-tail classification. In *Proceedings of the twelfth Indian conference on computer vision, graphics and image processing*, pages 1–9, 2021.

[72] Boxin Wang, Fan Wu, Yunhui Long, Luka Rimanic, Ce Zhang, and Bo Li. Datalens: Scalable privacy preserving training via gradient compression and aggregation. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, pages 2146–2168, 2021.

[73] Wenju Wang, Yu Cai, and Tao Wang. Multi-view dual attention network for 3d object recognition. *Neural Computing and Applications*, 34(4):3201–3212, 2022.

[74] Yuncheng Wu, Shaofeng Cai, Xiaokui Xiao, Gang Chen, and Beng Chin Ooi. Privacy preserving vertical federated learning for tree-based models. *Proceedings of the VLDB Endowment*, 13(12):2090–2103, 2020.

[75] Chulin Xie, Yunhui Long, Pin-Yu Chen, Qinbin Li, Sanmi Koyejo, and Bo Li. Unraveling the connections between privacy and certified robustness in federated learning against poisoning attacks. In *Proceedings of*

the 2023 ACM SIGSAC Conference on Computer and Communications Security, pages 1511–1525, 2023.

[76] Han Xu, Xiaorui Liu, Yaxin Li, Anil Jain, and Jiliang Tang. To be robust or to be fair: Towards fairness in adversarial training. In *International conference on machine learning*, pages 11492–11501. PMLR, 2021.

[77] Qiang Yang, Yang Liu, Tianjian Chen, and Yongxin Tong. Federated machine learning: Concept and applications. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 10(2):12, 2019.

[78] Shengwen Yang, Bing Ren, Xuhui Zhou, and Liping Liu. Parallel distributed logistic regression for vertical federated learning without third-party coordinator. *arXiv preprint arXiv:1911.09824*, 2019.

[79] Timothy Yang, Galen Andrew, Hubert Eichner, Haicheng Sun, Wei Li, Nicholas Kong, Daniel Ramage, and Françoise Beaufays. Applied federated learning: Improving google keyboard query suggestions. *arXiv preprint arXiv:1812.02903*, 2018.

[80] Wensi Yang, Yuhang Zhang, Kejiang Ye, Li Li, and Cheng-Zhong Xu. Ffd: a federated learning based method for credit card fraud detection. In *International Conference on Big Data*, pages 18–32. Springer, 2019.

[81] Sheng Yue, Ju Ren, Jiang Xin, Sen Lin, and Junshan Zhang. Inexact-admm based federated meta-learning for fast and continual edge learning. In *Proceedings of the Twenty-second International Symposium on Theory, Algorithmic Foundations, and Protocol Design for Mobile Networks and Mobile Computing*, pages 91–100, 2021.

[82] Muhammad Bilal Zafar, Isabel Valera, Manuel Gomez Rodriguez, and Krishna P Gummadi. Fairness beyond disparate treatment & disparate impact: Learning classification without disparate mistreatment. In *Proceedings of the 26th international conference on world wide web*, pages 1171–1180, 2017.

[83] Jie Zhang, Song Guo, Zhihao Qu, Deze Zeng, Haozhao Wang, Qifeng Liu, and Albert Y Zomaya. Adaptive vertical federated learning on unbalanced features. *IEEE Transactions on Parallel and Distributed Systems*, 33(12):4006–4018, 2022.

[84] Qingsong Zhang, Bin Gu, Cheng Deng, and Heng Huang. Secure bilevel asynchronous vertical federated learning with backward updating. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 10896–10904, 2021.

[85] Ciyou Zhu, Richard H Byrd, Peihuang Lu, and Jorge Nocedal. Algorithm 778: L-bfgs-b: Fortran subroutines for large-scale bound-constrained optimization. *ACM Transactions on mathematical software (TOMS)*, 23(4):550–560, 1997.

APPENDIX

The Appendix is organized as follows:

- Appendix A provides algorithm details for `Split Learning` [70](Algorithm 2) and `VIMADMM-J` (Algorithm 3);
- Appendix B provides the proofs for convergence guarantees in Theorem 1;
- Appendix C provides the proofs for privacy guarantee in Theorem 2;
- Appendix D provides more details on experimental setups and the additional experimental results;
- Appendix E provides additional discussion on ADMM and VFL.

A. Algorithm Details

1) `Split Learning` [70]: At each communication round t, the server samples a set of data indices, $B(t)$, with batch size $|B(t)| = b$. Then we describe the key steps `Split Learning` (Algorithm 2) as follows:

(1) *Communication from client to server.* Each client k sends a batch of embeddings $\{h_j^{k(t)}\}_{j\in B(t)}$ to the server, where $h_j^{k(t)} = f(x_j^k;\theta_k^{(t)}), \forall j \in B(t)$.

(2) *Sever updates server model* θ_0. According to VFL objective in Eq. 1, the server model is updated as:

$$\theta_0^{(t+1)} \leftarrow \theta_0^{(t)} - \eta \nabla_{\theta_0^{(t)}} \mathcal{L}_{\mathrm{VFL}}(\theta_0^{(t)}), \forall k \in [M] \tag{14}$$

where η is the server learning rate, and

$$\nabla_{\theta_0^{(t)}} \mathcal{L}_{\mathrm{VFL}}(\theta_0^{(t)}) = \nabla_{\theta_0^{(t)}} \left(\frac{1}{N} \sum_{j=1}^{N} \ell([h_j^{1(t)}, \ldots, h_j^{M(t)}], y_j; \theta_0^{(t)}) + \beta \mathcal{R}(\theta_0^{(t)}) \right). \tag{15}$$

Here $[h_j^{1(t)}, \ldots, h_j^{M(t)}]$ denotes the concatenated local embeddings.

(3) *Communication from server to client.* Server computes gradients w.r.t each local embedding $\nabla_{h_j^{k(t)}} \mathcal{L}_{\mathrm{VFL}}(\theta_0^{(t+1)})$ by the VFL objective in Eq. 1, where

$$\nabla_{h_j^{k(t)}} \mathcal{L}_{\mathrm{VFL}}(\theta_0^{(t+1)}) = \nabla_{h_j^{k(t)}} \ell([h_j^{1(t)}, \ldots, h_j^{M(t)}], y_j; \theta_0^{(t+1)}), \forall j \in B(t), k \in [M] \tag{16}$$

Server sends gradients $\{\nabla_{h_j^{k(t)}} \mathcal{L}_{\mathrm{VFL}}(\theta_0^{(t+1)})\}_{j\in B(t)}$ to each client $k, \forall k \in [M]$.

(4) *Client updates local model parameters* θ_k. Finally, every client k locally updates the model parameters θ_k according to the VFL objective in Eq. 1 as follows:

$$\theta_k^{(t+1)} \leftarrow \theta_k^{(t)} - \eta^k \nabla_{\theta_k^{(t)}} \mathcal{L}_{\mathrm{VFL}}(\theta_0^{(t+1)}), \forall k \in [M] \tag{17}$$

where η^k is the local learning rate for client k, and

$$\nabla_{\theta_k^{(t)}} \mathcal{L}_{\mathrm{VFL}}(\theta_0^{(t+1)}) = \frac{1}{N} \sum_{j=1}^{N} \nabla_{\theta_k^{(t)}} h_j^{k(t)} \nabla_{h_j^{k(t)}} \mathcal{L}_{\mathrm{VFL}}(\theta_0^{(t+1)}) + \beta \nabla_{\theta_k^{(t)}} \mathcal{R}(\theta_k^{(t)}) \tag{18}$$

These four steps of `Split Learning` are summarized in Algorithm 2.

Algorithm 2 `Split Learning` [70]

Input: number of communication rounds T, number of clients M, number of training samples N, batch size b , input features $\{\{x_j^1\}_{j=1}^N, \{x_j^2\}_{j=1}^N, \ldots, \{x_j^M\}_{j=1}^N\}$, the labels $\{y_j\}_{j=1}^N$, local model $\{\theta_k\}_{k=1}^M$; linear heads $\{W_k\}_{k=1}^M$; server learning rate η; client learning rate $\{\eta^k\}_{k=1}^M$;
for communication round $t \in [T]$ **do**
  Server samples a set of data indices $B(t)$ with $|B(t)| = b$
  for client $k \in [M]$ **do**
    generates a local training batch $\{x_j^k\}_{j\in B(t)}$
    computes local embeddings $h_j^{k(t)} \leftarrow f(x_j^k;\theta_k), \forall j \in B(t)$
    sends local embeddings $\{h_j^{k(t)}\}_{j\in B(t)}$ to the server
  Server **updates** server model $\theta_0^{(t+1)}$ by Eq. 14
  Server **computes** gradients w.r.t embeddings $\nabla_{h_j^{k(t)}} \mathcal{L}_{\mathrm{VFL}}(\theta_0^{(t+1)})$ by Eq. 16 , $\forall j \in B(t)$
  Server **sends** gradients $\{\nabla_{h_j^{k(t)}} \mathcal{L}_{\mathrm{VFL}}(\theta_0^{(t+1)})\}_{j\in B(t)}$ to each client $k, \forall k \in [M]$
  for client $k \in [M]$ **do**
    updates local model $\theta_k^{(t+1)}$ by Eq. 17

2) `VIMADMM-J`: At each communication round t, the server samples a set of data indices, $B(t)$, with batch size $|B(t)| = b$. Then we describe the key steps of `VIMADMM-J` (Algorithm 3) as follows:

(1) *Communication from client to server.* Each client k sends a batch of local logits $\{o_j^{k(t)}\}_{j\in B(t)}$ to the server, where $o_j^{k(t)} = f(x_j^k;\theta_k^{(t)})W_k^{(t)}, \forall j \in B(t)$

Algorithm 3 `VIMADMM-J` (with differentially privacy)

Input: number of communication rounds T, number of clients M, number of training samples N, batch size b , input features $\{\{x_j^1\}_{j=1}^N, \{x_j^2\}_{j=1}^N, \ldots, \{x_j^M\}_{j=1}^N\}$, the labels $\{y_j\}_{j=1}^N$, local model $\{\theta_k\}_{k=1}^M$; linear heads $\{W_k\}_{k=1}^M$; auxiliary variables $\{z_j\}_{j=1}^N$; dual variables $\{\lambda_j\}_{j=1}^N$; noise parameter σ, clipping constant C
for communication round $t \in [T]$ **do**
  Server samples a set of data indices $B(t)$ with $|B(t)| = b_s$
  for client $k \in [M]$ **do**
    generates a local training batch $\{x_j^k\}_{j \in B(t)}$
    computes local logits $o_j^{k(t)} = f(x_j^k; \theta_k^{(t)}) W_k^{(t)}, \forall j \in B(t)$
    clips and perturbs local logit matrix $\{o_j^{k(t)}\}_{j \in B(t)} \leftarrow \texttt{Clip}\left(\{o_j^{k(t)}\}_{j \in B(t)}, C\right) + \mathcal{N}\left(0, \sigma^2 C^2\right)$
    sends local logits $\{o_j^{k(t)}\}_{j \in B(t)}$ to the server
  Server **updates** auxiliary variables $z_j^{(t)}$ by Eq. 19, $\forall j \in B(t)$
  Server **updates** dual variables $\lambda_j^{(t)}$ by Eq. 20 , $\forall j \in B(t)$
  Server **computes** residual variables $s_j^{k(t)}$ by Eq. 21 , $\forall j \in B(t), k \in [M]$
  Server **sends** $\{\lambda_j^{(t)}\}_{j \in B(t)}$, $\{s_j^{k(t)}\}_{j \in B(t)}$ to each client $k, \forall k \in [M]$
  for client $k \in [M]$ **do**
    for local step $e \in [\tau]$ **do**
      updates local linear head $W_k^{(t+1)}$ by Eq. 22 with SGD
      updates local model $\theta_k^{(t+1)}$ by Eq. 23 with SGD

(2) *Sever updates auxiliary variables* $\{z_j\}$. After receiving the local logits from all clients, the server updates the auxiliary variable for each sample j as:

$$z_j^{(t)} = \underset{z_j}{\operatorname{argmin}} \quad \ell(z_j, y_j) - \lambda_j^{(t-1)\top} z_j + \frac{\rho}{2} \left\| \sum_{k=1}^{M} o_j^{k(t)} - z_j \right\|^2, \forall j \in B(t) \tag{19}$$

Since the optimization problem in Eq. 19 is convex and differentiable with respect to z_j, we use the L-BFGS-B algorithm [85] to solve the minimization problem.

(3) *Sever updates dual variables* $\{\lambda_j\}$. After the updates in Eq. 19, the server updates the dual variable for each sample j as:

$$\lambda_j^{(t)} = \lambda_j^{(t-1)} + \rho \left(\sum_{k=1}^{M} o_j^{k(t)} - z_j^{(t)} \right), \forall j \in B(t) \tag{20}$$

(4) *Communication from server to client.* After the updates in Eq. 20, we define a residual variable $s_j^{k(t+1)}$ for each sample j of k-th client, which provides supervision for updating local model:

$$s_j^{k(t)} \triangleq z_j^{(t)} - \sum_{i \in [M], i \neq k} o_j^{i(t)} \tag{21}$$

The server sends the dual variables $\{\lambda_j^{(t)}\}_{j \in B(t)}$ and the residual variables $\{s_j^{k(t)}\}_{j \in B(t)}$ of all samples to each client k.

(5) *Client updates linear head* W_k *and local model* θ_k *alternatively.* The linear head of each client is locally updated as:

$$W_k^{(t+1)} = \underset{W_k}{\operatorname{argmin}} \quad \beta \mathcal{R}(W_k) + \frac{1}{b} \sum_{j \in B(t)} \lambda_j^{(t)\top} f(x_{j_k}; \theta_k^{(t)}) W_k + \sum_{j \in B(t)} \frac{\rho}{2b} \left\| s_j^{k(t)} - f(x_{j_k}; \theta_k^{(t)}) W_k \right\|_F^2, \forall k \in [M] \tag{22}$$

Each client updates the local model parameters θ_k as follows:

$$\theta_k^{(t+1)} = \underset{\theta_k}{\operatorname{argmin}} \quad \beta \mathcal{R}(\theta_k) + \frac{1}{b} \sum_{j \in B(t)} \lambda_j^{(t)\top} f(x_{j_k}; \theta_k) W_k^{(t+1)} + \sum_{j \in B(t)} \frac{\rho}{2b} \left\| s_j^{k(t)} - f(x_{j_k}; \theta_k) W_k^{(t+1)} \right\|_F^2. \tag{23}$$

Due to the nonconvexity of the loss function of DNN, we use τ local steps of SGD to update W_k and θ_k alternatively at each round with the objective of Eq. 22 and Eq. 23. Specifically, at each local step, we first update W_k and then update θ_k.

These five steps of `VIMADMM-J` are summarized in Algorithm 3.

B. Convergence Guarantees

1) Additional Notations and Supporting Lemmas: To help theoretical analysis, we denote the objective functions in Eq. (5), Eq. (7) , Eq. (9) as

$$
\begin{aligned}
h(z_j) &= \ell(z_j) - {\lambda_j^{(t)}}^\top z_j + \frac{\rho}{2}\left\|\sum_{k=1}^{M} f(x_j^k;\theta_k^{(t+1)})W_k^{(t+1)} - z_j\right\|_F^2 \\
g_k(W_k) &= \beta_k \mathcal{R}_k(W_k) + \frac{1}{N}\sum_{j\in[N]} {\lambda_j^{(t)}}^\top f(x_j^k;\theta_k^{(t)})W_k \\
&\quad + \frac{\rho}{2N}\sum_{j\in[N]}\left\|\sum_{\substack{i\in[M],\\ i\neq k}} f(x_j^i;\theta_i^{(t)}){W_i}^{(t)} + f(x_j^k;\theta_k^{(t)})W_k - {z_j}^{(t)}\right\|_F^2 \\
q_k(\theta_k) &= \beta_k \mathcal{R}_k(\theta_k) + \frac{1}{N}\sum_{j\in[N]} {\lambda_j^{(t)}}^\top f(x_j^k;\theta_k)W_k^{(t+1)} \\
&\quad + \frac{\rho}{2N}\sum_{j\in[N]}\left\|\sum_{\substack{i\in[M],\\ i\neq k}} f(x_j^i;\theta_i^{(t)}){W_i}^{(t+1)} + f(x_j^k;\theta_k)W_k^{(t+1)} - {z_j}^{(t)}\right\|_F^2
\end{aligned}
\tag{24}
$$

Before delving into the main proofs, we introduce the bellow supporting lemmas.

Lemma 1.

$$\nabla \ell(z_j^{(t)}) = \lambda_j^{(t)} \tag{25}$$

Proof. According to the optimality of $z_j^{(t)}$ Eq. (5)

$$\nabla \ell(z_j^{(t)}) - \lambda_j^{(t-1)} - \rho\left(\sum_{k=1}^{M} f(x_j^k;\theta_k^{(t)})W_k^{(t)} - z_j^{(t)}\right) = 0, \forall j \in B(t) \tag{26}$$

then invoke Eq. (6) $\lambda_j^{(t)} = \lambda_j^{(t-1)} + \rho\left(\sum_{k=1}^{M} f(x_j^k;\theta_k^{(t)})W_k^{(t)} - z_j^{(t)}\right)$, so we have $\nabla \ell(z_j^{(t)}) = \lambda_j^{(t)}$. □

Lemma 2.

$$\|\lambda_j^{(t)} - \lambda_j^{(t-1)}\| \leq L\|z_j^{(t)} - z_j^{(t-1)}\| \tag{27}$$

Proof. According to Assumption 1 and Lemma 1, we have

$$\|\lambda_j^{(t)} - \lambda_j^{(t-1)}\| = \|\nabla \ell(z_j^{(t)}) - \nabla \ell(z_j^{(t-1)})\| \leq L\|z_j^{(t)} - z_j^{(t-1)}\| \tag{28}$$

□

Lemma 3. *[35, Lemma 3]*

$$\left(\left\|\sum_{m=1}^{M} x_m^{t+1} - z\right\|^2 - \left\|\sum_{m=1}^{M} x_m^{t} - z\right\|^2\right) \tag{29}$$

$$\leq \sum_{m=1}^{M}\left(\left\|\sum_{\substack{k=1\\ k\neq m}}^{M} x_k^t + x_m^{t+1} - z\right\|^2 - \left\|\sum_{m=1}^{M} x_m^t - z\right\|^2\right) + \sum_{m=1}^{M}\left\|x_m^{t+1} - x_m^t\right\|^2 \tag{30}$$

2) Proofs for Theorem 1: We restate our assumptions here in Theorem 1:

Assumption 1. $\ell(z;\cdot)$ is L-Lipschitz smooth w.r.t z.

Assumption 2. $\mathcal{L}_{\text{ADMM}}$ is strongly convex w.r.t z, W, θ with constant μ_z, μ_W, μ_θ respectively.

Assumption 3. The norm of W_k is bounded $\|W_k\| \leq \sigma_W$. The local model $f(\cdot;\theta)$ has bounded gradient $\|\nabla f(\cdot;\theta)\| \leq L_\theta$ and bounded output norm $\|f(\cdot;\theta)\| \leq \sigma_\theta$.

Assumption 4. The original objective function $\mathcal{L}_{\text{VIM}}$ is bounded from below over Θ and $\mathcal{W}$, that is $\underline{e} := \min_{\{\theta_k\}\in\Theta,\{W_k\}\in\mathcal{W}} \mathcal{L}_{\text{VIM}}(\{W_k\}_{k=1}^M, \{\theta_k\}_{k=1}^M) > -\infty$.

3) Proofs for Theorem 1 Part (A): We decompose Theorem 1 part (A) into the below two lemmas and prove them one-by-one.

Lemma 4. *Let Assumption 1 to Assumption 3 hold, and there exists a penalty parameter ρ satisfying*

$$\max\{L, \frac{2L^2}{\mu_z}\} < \rho < \min\{\frac{\mu_\theta}{L_\theta^2\sigma_W^2}, \frac{\mu_W}{\sigma_\theta^2}\} \tag{31}$$

then $\mathcal{L}_{\text{ADMM}}$ is monotonically decreasing:

$$\mathcal{L}_{\text{ADMM}}(\{W_k^{(t+1)}\}, \{\theta_k^{(t+1)}\}, \{z_j^{(t+1)}\}, \{\lambda_j^{(t+1)}\}) - \mathcal{L}_{\text{ADMM}}(\{W_k^{(t)}\}, \{\theta_k^{(t)}\}, \{z_j^{(t)}\}, \{\lambda_j^{(t)}\}) < 0. \tag{32}$$

Lemma 5. *Let Assumption 1 to Assumption 4 hold, then the following limit exists and $\mathcal{L}_{\mathrm{ADMM}}$ is lower bounded by $\underline{e}$ defined in Assumption 4:*

$$\lim_{t\to\infty} \mathcal{L}_{\mathrm{ADMM}}(\{W_k^{(t)}\}, \{\theta_k^{(t)}\}, \{z_j^{(t)}\}, \{\lambda_j^{(t)}\}) \geq \underline{e}. \tag{33}$$

We first present the proof for the monotonically decreasing property of $\mathcal{L}_{\mathrm{ADMM}}$ in Lemma 4.

Proof for Lemma 4.

$$\begin{aligned}
&\mathcal{L}_{\mathrm{ADMM}}(\{W_k^{(t+1)}\}, \{\theta_k^{(t+1)}\}, \{z_j^{(t+1)}\}, \{\lambda_j^{(t+1)}\}) - \mathcal{L}_{\mathrm{ADMM}}(\{W_k^{(t)}\}, \{\theta_k^{(t)}\}, \{z_j^{(t)}\}, \{\lambda_j^{(t)}\}) \\
=&\underbrace{\mathcal{L}_{\mathrm{ADMM}}(\{W_k^{(t+1)}\}, \{\theta_k^{(t+1)}\}, \{z_j^{(t+1)}\}, \{\lambda_j^{(t+1)}\}) - \mathcal{L}_{\mathrm{ADMM}}(\{W_k^{(t+1)}\}, \{\theta_k^{(t+1)}\}, \{z_j^{(t+1)}\}, \{\lambda_j^{(t)}\})}_{T_1} \\
&+\underbrace{\mathcal{L}_{\mathrm{ADMM}}(\{W_k^{(t+1)}\}, \{\theta_k^{(t+1)}\}, \{z_j^{(t+1)}\}, \{\lambda_j^{(t)}\}) - \mathcal{L}_{\mathrm{ADMM}}(\{W_k^{(t+1)}\}, \{\theta_k^{(t+1)}\}, \{z_j^{(t)}\}, \{\lambda_j^{(t)}\})}_{T_2} \\
&+\underbrace{\mathcal{L}_{\mathrm{ADMM}}(\{W_k^{(t+1)}\}, \{\theta_k^{(t+1)}\}, \{z_j^{(t)}\}, \{\lambda_j^{(t)}\} - \mathcal{L}_{\mathrm{ADMM}}(\{W_k^{(t+1)}\}, \{\theta_k^{(t)}\}, \{z_j^{(t)}\}, \{\lambda_j^{(t)}\})}_{T_3} \\
&+\underbrace{\mathcal{L}_{\mathrm{ADMM}}(\{W_k^{(t+1)}\}, \{\theta_k^{(t)}\}, \{z_j^{(t)}\}, \{\lambda_j^{(t)}\} - \mathcal{L}_{\mathrm{ADMM}}(\{W_k^{(t)}\}, \{\theta_k^{(t)}\}, \{z_j^{(t)}\}, \{\lambda_j^{(t)}\})}_{T_4}
\end{aligned} \tag{34}$$

Recall ADMM objective function

$$\begin{aligned}
\mathcal{L}_{\mathrm{ADMM}} =& \frac{1}{N}\sum_{j=1}^{N} \ell(z_j, y_j) + \sum_{k=1}^{M} \beta_k \mathcal{R}_k(\theta_k) + \sum_{k=1}^{M} \beta_k \mathcal{R}_k(W_k) \\
&+ \frac{1}{N}\sum_{j=1}^{N} \lambda_j^\top \left(\sum_{k=1}^{M} f(x_j^k;\theta_k)W_k - z_j\right) + \frac{\rho}{2N}\sum_{j=1}^{N}\left\|\sum_{k=1}^{M} f(x_j^k;\theta_k)W_k - z_j\right\|_F^2
\end{aligned}$$

Then we have

$$\begin{aligned}
T_1 &= \mathcal{L}_{\mathrm{ADMM}}(\{W_k^{(t+1)}\}, \{\theta_k^{(t+1)}\}, \{z_j^{(t+1)}\}, \{\lambda_j^{(t+1)}\}) - \mathcal{L}_{\mathrm{ADMM}}(\{W_k^{(t+1)}\}, \{\theta_k^{(t+1)}\}, \{z_j^{(t+1)}\}, \{\lambda_j^{(t)}\}) \\
&= \frac{1}{N}\sum_{j=1}^{N} (\lambda_j^{(t+1)} - \lambda_j^{(t)})^\top \left(\sum_{k=1}^{M} f(x_j^k;\theta_k^{(t+1)})W_k^{(t+1)} - z_j^{(t+1)}\right) \\
&\overset{(a)}{=} \frac{1}{N}\sum_{j=1}^{N} \frac{1}{\rho}\|\lambda_j^{(t+1)} - \lambda_j^{(t)}\|^2 \\
&\overset{(b)}{\leq} \sum_{j=1}^{N} \frac{L^2}{\rho N}\|z_j^{(t+1)} - z_j^{(t)}\|^2
\end{aligned}$$

where (a) we use the Eq. (6) that $\frac{1}{\rho}(\lambda_j^{(t)} - \lambda_j^{(t-1)}) = \left(\sum_{k=1}^{M} f(x_j^k;\theta_k^{(t)})W_k^{(t)} - z_j^{(t)}\right)$; (b) we use Lemma 2.

$$\begin{aligned}
T_2 &= \mathcal{L}_{\mathrm{ADMM}}(\{W_k^{(t+1)}\}, \{\theta_k^{(t+1)}\}, \{z_j^{(t+1)}\}, \{\lambda_j^{(t)}\}) - \mathcal{L}_{\mathrm{ADMM}}(\{W_k^{(t+1)}\}, \{\theta_k^{(t+1)}\}, \{z_j^{(t)}\}, \{\lambda_j^{(t)}\}) \\
&= \frac{1}{N}\sum_{j=1}^{N}\left(\ell(z_j^{(t+1)}) - \ell(z_j^{(t)})\right) - \frac{1}{N}\sum_{j=1}^{N} \lambda_j^{(t)\top}\left(z_j^{(t+1)} - z_j^{(t)}\right) \\
&+\frac{\rho}{2N}\sum_{j=1}^{N}\left(\left\|\sum_{k=1}^{M} f(x_j^k;\theta_k^{(t+1)})W_k^{(t+1)} - z_j^{(t+1)}\right\|_F^2 - \left\|\sum_{k=1}^{M} f(x_j^k;\theta_k^{(t+1)})W_k^{(t+1)} - z_j^{(t)}\right\|_F^2\right) \\
&= \frac{1}{N}\sum_{j=1}^{N}\left(h(z_j^{(t+1)}) - h(z_j^{(t)})\right) \\
&\overset{(a)}{\leq} \frac{1}{N}\sum_{j=1}^{N}\left(\left\langle \nabla h(z_j^{(t+1)}), z_j^{(t+1)} - z_j^{(t)}\right\rangle - \frac{\mu_z}{2}\|z_j^{(t+1)} - z_j^{(t)}\|^2\right) \\
&\overset{(b)}{=} -\frac{\mu_z}{2N}\sum_{j=1}^{N}\|z_j^{(t+1)} - z_j^{(t)}\|^2
\end{aligned}$$

where (a) strong convex of h Assumption 2 , (b) optimality of z update at Eq. (5).

$$
\begin{aligned}
T_3 =& \mathcal{L}_{\text{ADMM}}(\{W_k^{(t+1)}\}, \{\theta_k^{(t+1)}\}, \{z_j^{(t)}\}, \{\lambda_j^{(t)}\} - \mathcal{L}_{\text{ADMM}}(\{W_k^{(t+1)}\}, \{\theta_k^{(t)}\}, \{z_j^{(t)}\}, \{\lambda_j^{(t)}\}) \\
=& \sum_{k=1}^{M} \beta_k \left(\mathcal{R}_k(\theta_k^{(t+1)}) - \mathcal{R}_k(\theta_k^{(t)})\right) + \frac{1}{N}\sum_{j=1}^{N} \lambda_j^{(t)\top} \left(\sum_{k=1}^{M}\left(f(x_j^k;\theta_k^{(t+1)})W_k^{(t+1)} - f(x_j^k;\theta_k^{(t)})W_k^{(t+1)}\right)\right) \\
&+ \frac{\rho}{2N}\sum_{j=1}^{N}\left(\left\|\sum_{k=1}^{M} f(x_j^k;\theta_k^{(t+1)})W_k^{(t+1)} - z_j^{(t)}\right\|_F^2 - \left\|\sum_{k=1}^{M} f(x_j^k;\theta_k^{(t)})W_k^{(t+1)} - z_j^{(t)}\right\|_F^2\right) \\
\overset{(a)}{\leq}& \sum_{k=1}^{M} \beta_k \left(\mathcal{R}_k(\theta_k^{(t+1)}) - \mathcal{R}_k(\theta_k^{(t)})\right) + \frac{1}{N}\sum_{j=1}^{N} \lambda_j^{(t)\top} \left(\sum_{k=1}^{M}\left(f(x_j^k;\theta_k^{(t+1)})W_k^{(t+1)} - f(x_j^k;\theta_k^{(t)})W_k^{(t+1)}\right)\right) \\
&+ \frac{\rho}{2N}\sum_{j=1}^{N}\sum_{k=1}^{M}\left(\left\|\sum_{\substack{i\in[M],\\ i\neq k}} f(x_j^i;\theta_i^{(t)})W_i^{(t+1)} + f(x_j^k;\theta_k^{(t+1)})W_k^{(t+1)} - z_j^{(t)}\right\|_F^2 - \left\|\sum_{k=1}^{M} f(x_j^k;\theta_k^{(t)})W_k^{(t+1)} - z_j^{(t)}\right\|_F^2\right) \\
&+ \frac{\rho}{2N}\sum_{j=1}^{N}\sum_{k=1}^{M}\left\|f(x_j^k;\theta_k^{(t+1)})W_k^{(t+1)} - f(x_j^k;\theta_k^{(t)})W_k^{(t+1)}\right\|_F^2 \\
=& \sum_{k=1}^{M}\left(q_k(\theta_k^{(t+1)}) - q_k(\theta_k^{(t)})\right) + \frac{\rho}{2N}\sum_{j=1}^{N}\sum_{k=1}^{M}\left\|f(x_j^k;\theta_k^{(t+1)})W_k^{(t+1)} - f(x_j^k;\theta_k^{(t)})W_k^{(t+1)}\right\|_F^2 \\
\overset{(b)}{\leq}& \sum_{k=1}^{M}\left(\left\langle \nabla q_k(\theta_k^{(t+1)}), \theta_k^{(t+1)} - \theta_k^{(t)}\right\rangle - \frac{\mu_\theta}{2}\|\theta_k^{(t+1)} - \theta_k^{(t)}\|^2\right) \\
&+ \frac{\rho}{2N}\sum_{j=1}^{N}\sum_{k=1}^{M}\left\|f(x_j^k;\theta_k^{(t+1)})W_k^{(t+1)} - f(x_j^k;\theta_k^{(t)})W_k^{(t+1)}\right\|_F^2 \\
\overset{(c)}{=}& \sum_{k=1}^{M} -\frac{\mu_\theta}{2}\|\theta_k^{(t+1)} - \theta_k^{(t)}\|^2 + \frac{\rho}{2N}\sum_{j=1}^{N}\sum_{k=1}^{M}\left\|f(x_j^k;\theta_k^{(t+1)}) - f(x_j^k;\theta_k^{(t)})\right\|^2 \|W_k^{(t+1)}\|^2 \\
\overset{(d)}{\leq}& \sum_{k=1}^{M} -\frac{\mu_\theta}{2}\left\|\theta_k^{(t+1)} - \theta_k^{(t)}\right\|^2 + \frac{\rho L_\theta^2}{2}\sum_{k=1}^{M}\left\|\theta_k^{(t+1)} - \theta_k^{(t)}\right\|^2 \|W_k^{(t+1)}\|^2 \\
\overset{(e)}{\leq}& \sum_{k=1}^{M} \frac{-\mu_\theta + \rho L_\theta^2 \sigma_W^2}{2}\left\|\theta_k^{(t+1)} - \theta_k^{(t)}\right\|^2
\end{aligned}
$$

where (a) is due to Lemma 3, (b) is due to the strong convex of q_k Assumption 2, (c) is due to the optimality of θ_k in Eq. (9), (d) is due to the Lipschitz continuity of local model $f(\theta)$ Assumption 3 (bounded gradient implies Lipschitz continuity), and (e) is due to the upper bound of the linear weights in Assumption 3.

$$
\begin{aligned}
T_4 =& \mathcal{L}_{\text{ADMM}}(\{W_k^{(t+1)}\}, \{\theta_k^{(t)}\}, \{z_j^{(t)}\}, \{\lambda_j^{(t)}\} - \mathcal{L}_{\text{ADMM}}(\{W_k^{(t)}\}, \{\theta_k^{(t)}\}, \{z_j^{(t)}\}, \{\lambda_j^{(t)}\}) \\
=& \sum_{k=1}^{M} \beta_k \left(\mathcal{R}_k(W_k^{(t+1)}) - \mathcal{R}_k(W_k^{(t)}) \right) + \frac{1}{N} \sum_{j=1}^{N} \lambda_j^{(t)\top} \left(\sum_{k=1}^{M} f(x_j^k; \theta_k^{(t)}) \left(W_k^{(t+1)} - W_k^{(t)} \right) \right) \\
&+ \frac{\rho}{2N} \sum_{j=1}^{N} \left(\left\| \sum_{k=1}^{M} f(x_j^k; \theta_k^{(t)}) W_k^{(t+1)} - z_j^{(t)} \right\|_F^2 - \left\| \sum_{k=1}^{M} f(x_j^k; \theta_k^{(t)}) W_k^{(t)} - z_j^{(t)} \right\|_F^2 \right) \\
\overset{(a)}{\leq} & \sum_{k=1}^{M} \beta_k \left(\mathcal{R}_k(W_k^{(t+1)}) - \mathcal{R}_k(W_k^{(t)}) \right) + \frac{1}{N} \sum_{j=1}^{N} \lambda_j^{(t)\top} \left(\sum_{k=1}^{M} f(x_j^k; \theta_k^{(t)}) \left(W_k^{(t+1)} - W_k^{(t)} \right) \right) \\
&+ \frac{\rho}{2N} \sum_{j=1}^{N} \sum_{k=1}^{M} \left(\left\| \sum_{\substack{i \in [M], \\ i \neq k}} f(x_j^i; \theta_i^{(t)}) W_i^{(t)} + f(x_j^k; \theta_k^{(t)}) W_k^{(t+1)} - z_j^{(t)} \right\|_F^2 - \left\| \sum_{k=1}^{M} f(x_j^k; \theta_k^{(t)}) W_k^{(t)} - z_j^{(t)} \right\|_F^2 \right) \\
&+ \frac{\rho}{2N} \sum_{j=1}^{N} \sum_{k=1}^{M} \left\| f(x_j^k; \theta_k^{(t)}) W_k^{(t+1)} - f(x_j^k; \theta_k^{(t)}) W_k^{(t)} \right\|_F^2 \\
=& \sum_{k=1}^{M} \left(g_k(W_k^{(t+1)}) - g_k(W_k^{(t)}) \right) + \frac{\rho}{2N} \sum_{j=1}^{N} \sum_{k=1}^{M} \left\| f(x_j^k; \theta_k^{(t)}) W_k^{(t+1)} - f(x_j^k; \theta_k^{(t)}) W_k^{(t)} \right\|_F^2 \\
\overset{(b)}{\leq} & \sum_{k=1}^{M} \left(\left\langle \nabla g_k(W_k^{(t+1)}), W_k^{(t+1)} - W_k^{(t)} \right\rangle - \frac{\mu_W}{2} \| W_k^{(t+1)} - W_k^{(t)} \|^2 \right) \\
&+ \frac{\rho}{2N} \sum_{j=1}^{N} \sum_{k=1}^{M} \left\| f(x_j^k; \theta_k^{(t)}) W_k^{(t+1)} - f(x_j^k; \theta_k^{(t)}) W_k^{(t)} \right\|_F^2 \\
\overset{(c)}{\leq} & \sum_{k=1}^{M} -\frac{\mu_W}{2} \left\| W_k^{(t+1)} - W_k^{(t)} \right\|^2 + \frac{\rho N}{2} \sum_{k=1}^{M} \left\| W_k^{(t+1)} - W_k^{(t)} \right\|^2 \| f(x_j^k; \theta_k^{(t)}) \|^2 \\
\overset{(d)}{\leq} & \sum_{k=1}^{M} \frac{-\mu_W + \rho N \sigma_\theta^2}{2} \left\| W_k^{(t+1)} - W_k^{(t)} \right\|^2
\end{aligned}
$$

where (a) is due to Lemma 3, (b) is due to strong convex of g_k Assumption 2, (c) is because of the optimality of W_k in Eq. (7) and (d) is due to upper bound of the model outputs in Assumption 3.

Combining the above bounds for T_1, T_2, T_3, T_4 together and recall the condition for ρ in Eq. (31), we have

$$
\begin{aligned}
& \mathcal{L}_{\text{ADMM}}(\{W_k^{(t+1)}\}, \{\theta_k^{(t+1)}\}, \{z_j^{(t+1)}\}, \{\lambda_j^{(t+1)}\}) - \mathcal{L}_{\text{ADMM}}(\{W_k^{(t)}\}, \{\theta_k^{(t)}\}, \{z_j^{(t)}\}, \{\lambda_j^{(t)}\}) \\
=& T_1 + T_2 + T_3 + T_4 \\
\leq & \frac{1}{N} \sum_{j=1}^{N} \left(-\frac{\mu_z}{2} + \frac{L^2}{\rho} \right) \| z_j^{(t+1)} - z_j^{(t)} \|^2 + \sum_{k=1}^{M} \frac{-\mu_\theta + \rho L_\theta^2 \sigma_W^2}{2} \left\| \theta_k^{(t+1)} - \theta_k^{(t)} \right\|^2 + \sum_{k=1}^{M} \frac{-\mu_W + \rho \sigma_\theta^2}{2} \left\| W_k^{(t+1)} - W_k^{(t)} \right\|^2 \\
<& 0
\end{aligned}
$$

Thus, proved. □

Then we provide the proof for the lower-bounded property of $\mathcal{L}_{\text{ADMM}}$ for Lemma 5.

Proof for Lemma 5.

$$
\begin{aligned}
&\mathcal{L}_{\mathrm{ADMM}}(\{W_k^{(t)}\},\{\theta_k^{(t)}\},\{z_j^{(t)}\},\{\lambda_j^{(t)}\})\\
=&\frac{1}{N}\sum_{j=1}^{N}\ell(z_j^{(t)},y_j)+\sum_{k=1}^{M}\beta_k\mathcal{R}_k(\theta_k^{(t)})+\sum_{k=1}^{M}\beta_k\mathcal{R}_k(W_k^{(t)})\\
&+\frac{1}{N}\sum_{j=1}^{N}\lambda_j^{(t)\top}\left(\sum_{k=1}^{M}f(x_j^k;\theta_k^{(t)})W_k^{(t)}-z_j^{(t)}\right)+\frac{\rho}{2N}\sum_{j=1}^{N}\left\|\sum_{k=1}^{M}f(x_j^k;\theta_k^{(t)})W_k^{(t)}-z_j^{(t)}\right\|_F^2\\
\overset{(a)}{=}&\frac{1}{N}\sum_{j=1}^{N}\ell(z_j^{(t)},y_j)+\sum_{k=1}^{M}\beta_k\mathcal{R}_k(\theta_k^{(t)})+\sum_{k=1}^{M}\beta_k\mathcal{R}_k(W_k^{(t)})\\
&+\frac{1}{N}\sum_{j=1}^{N}\nabla\ell(z_j^{(t)})^{\top}\left(\sum_{k=1}^{M}f(x_j^k;\theta_k^{(t)})W_k^{(t)}-z_j^{(t)}\right)+\frac{\rho}{2N}\sum_{j=1}^{N}\left\|\sum_{k=1}^{M}f(x_j^k;\theta_k^{(t)})W_k^{(t)}-z_j^{(t)}\right\|_F^2\\
\overset{(b)}{\geq}&\frac{1}{N}\sum_{j=1}^{N}\ell\left(\sum_{k=1}^{M}f(x_j^k;\theta_k^{(t)})W_k^{(t)},y_j\right)+\sum_{k=1}^{M}\beta_k\mathcal{R}_k(\theta_k^{(t)})+\sum_{k=1}^{M}\beta_k\mathcal{R}_k(W_k^{(t)})\\
&+\frac{\rho-L}{2N}\sum_{j=1}^{N}\left\|\sum_{k=1}^{M}f(x_j^k;\theta_k^{(t)})W_k^{(t)}-z_j^{(t)}\right\|_F^2\\
\overset{(c)}{\geq}&\frac{1}{N}\sum_{j=1}^{N}\ell\left(\sum_{k=1}^{M}f(x_j^k;\theta_k^{(t)})W_k^{(t)},y_j\right)+\sum_{k=1}^{M}\beta_k\mathcal{R}_k(\theta_k^{(t)})+\sum_{k=1}^{M}\beta_k\mathcal{R}_k(W_k^{(t)})\\
=&\mathcal{L}_{\mathtt{VIM}}(\{W_k^{(t)}\},\{\theta_k^{(t)}\})\\
\geq&\underline{e}
\end{aligned}
$$

where (a) is due to Lemma 1; (b) is due to Lipschitz continuity of gradient of ℓ in Assumption 1 that

$$
\ell\left(\sum_{k=1}^{M}f(x_j^k;\theta_k^{(t)})W_k^{(t)}\right)-\ell\left(z_j^{(t)}\right)-\frac{L}{2}\left\|\sum_{k=1}^{M}f(x_j^k;\theta_k^{(t)})W_k^{(t)}-z_j^{(t)}\right\|\leq\left\langle\nabla\ell(z_j^{(t)}),\left(\sum_{k=1}^{M}f(x_j^k;\theta_k^{(t)})W_k^{(t)}-z_j^{(t)}\right)\right\rangle
$$

and (c) is due to $\rho\geq L$ from the condition Eq. (31).

The result show that $\mathcal{L}_{\mathrm{ADMM}}(\{W_k^{(t)}\},\{\theta_k^{(t)}\},\{z_j^{(t)}\},\{\lambda_j^{(t)}\})$ is lower bounded. Thus, proved. □

Proof for Theorem 1 (A). Combining Lemma 4 and Lemma 5, we show that $\mathcal{L}_{\mathrm{ADMM}}(\{W_k^{(t)}\},\{\theta_k^{(t)}\},\{z_j^{(t)}\},\{\lambda_j^{(t)}\})$ is monotonically decreasing and is convergent. This completes the proof. □

4) Proofs for Theorem 1 Part (B):

Proofs for Theorem 1 (B). Lemma 4 implies that

$$
\begin{aligned}
&\mathcal{L}_{\mathrm{ADMM}}(\{W_k^{(t+1)}\},\{\theta_k^{(t+1)}\},\{z_j^{(t+1)}\},\{\lambda_j^{(t+1)}\})-\mathcal{L}_{\mathrm{ADMM}}(\{W_k^{(t)}\},\{\theta_k^{(t)}\},\{z_j^{(t)}\},\{\lambda_j^{(t)}\})\\
\leq&\frac{1}{N}\sum_{j=1}^{N}\left(-\frac{\mu_z}{2}+\frac{L^2}{\rho}\right)\|z_j^{(t+1)}-z_j^{(t)}\|^2+\sum_{k=1}^{M}\frac{-\mu_\theta+\rho L_\theta^2\sigma_W^2}{2}\left\|\theta_k^{(t+1)}-\theta_k^{(t)}\right\|^2\\
&+\sum_{k=1}^{M}\frac{-\mu_W+\rho\sigma_\theta^2}{2}\left\|W_k^{(t+1)}-W_k^{(t)}\right\|^2
\end{aligned}
$$

Using the fact that $\mathcal{L}_{\mathrm{ADMM}}$ is monotonically decreasing and lower-bounded (in Lemma 5) as well as the bounds for ρ in Eq. (31), we have $\forall j\in[N],k\in[M]$,

$$
\lim_{t\to\infty}\left\|z_j^{(t+1)}-z_j^{(t)}\right\|^2\to0,\lim_{t\to\infty}\left\|\theta_k^{(t+1)}-\theta_k^{(t)}\right\|^2\to0,\lim_{t\to\infty}\left\|W_k^{(t+1)}-W_k^{(t)}\right\|^2\to0. \tag{35}
$$

By Lemma 2 that $\|\lambda_j^{(t+1)}-\lambda_j^{(t)}\|\leq L\|z_j^{(t+1)}-z_j^{(t)}\|$, we further obtain

$$
\lim_{t\to\infty}\left\|\lambda_j^{(t+1)}-\lambda_j^{(t)}\right\|^2\to0,\forall j\in[N] \tag{36}
$$

In light of the dual update step of Algorithm 1, Eq. (36) implies that

$$
\lim_{t\to\infty}\left\|\sum_{k=1}^{M}f(x_j^k;\theta_k^{(t+1)})W_k^{(t+1)}-z_j^{(t+1)}\right\|^2\to0,\forall j\in[N] \tag{37}
$$

Using the limit points, we have $W_k^{(t+1)}\to W_k^*,\theta_k^{(t+1)}\to\theta_k^*,z_j^{(t+1)}\to z_j^*,\lambda_j^{(t+1)}\to\lambda_j^*$.

Based on Eq. (37), we have

$$\sum_{k=1}^{M} f(x_j^k;\theta_k^*)W_k^* = z_j^* \tag{38}$$

Then, we examine the optimality condition for the $\{W_k^{(t+1)}\}$ subproblems at iteration $t+1$:

$$\begin{aligned} 0 =& \beta_k \nabla \mathcal{R}_k(W_k^{(t+1)}) + \frac{1}{N}\sum_{j=1}^{N} {\lambda_j^{(t)}}^\top f(x_j^k;\theta_k^{(t)}) \\ &+ \sum_{j=1}^{N}\frac{\rho}{N}\left(\sum_{i\in[M],i\neq k} f(x_j^i;\theta_i^{(t)}){W_i}^{(t)} + f(x_j^k;\theta_k^{(t)})W_k^{(t+1)} - {z_j}^{(t)}\right) f(x_j^k;\theta_k^{(t)}) \end{aligned} \tag{39}$$

According to Eq. (35) and Eq. (37), we have

$$0 = \beta_k \nabla \mathcal{R}_k(W_k^*) + \frac{1}{N}\sum_{j=1}^{N} {\lambda_j^*}^\top f(x_j^k;\theta_k^*) \tag{40}$$

Similarly, the optimality condition for the $\{\theta_k^{(t+1)}\}$ subproblems at iteration $t+1$ indicates that:

$$\begin{aligned} 0 =& \beta_k \nabla \mathcal{R}_k(\theta_k^{(t+1)}) + \frac{1}{N}\sum_{j=1}^{N} {\lambda_j^{(t)}}^\top \nabla f(x_j^k;\theta_k^{(t+1)})W_k^{(t+1)} \\ &+ \sum_{j=1}^{N}\frac{\rho}{N}\left(\sum_{i\in[M],i\neq k} f(x_j^i;\theta_i^{(t)}){W_i}^{(t+1)} + f(x_j^k;\theta_k^{(t)})W_k^{(t+1)} - {z_j}^{(t)}\right) \nabla f(x_j^k;\theta_k^{(t+1)})W_k^{(t+1)} \end{aligned} \tag{41}$$

According to Eq. (35) and Eq. (37), we have

$$0 = \beta_k \nabla \mathcal{R}_k(\theta_k^*) + \frac{1}{N}\sum_{j=1}^{N} {\lambda_j^*}^\top \nabla f(x_j^k;\theta_k^*)W_k^* \tag{42}$$

Based on the optimality condition for the $\{z_j^{(t+1)}\}$ subproblems at iteration $t+1$, we have

$$0 = \nabla \ell(z_j^{(t)}) - \lambda_j^{(t-1)} - \rho\left(\sum_{k=1}^{M} f(x_j^k;\theta_k^{(t)})W_k^{(t)} - z_j^{(t)}\right), \forall j \in B(t) \tag{43}$$

Based on the strongly convexity w.r.t z_j in Assumption 2, there exists a subgradient $\eta^{(t)} \in \partial\ell\left(z_j^{(t)}\right)$ such that

$$\left\langle z - z_j^{(t)}, \eta^{(t)} - \left(\lambda_j^{(t-1)} + \rho\left(\sum_{k=1}^{M} f(x_j^k;\theta_k^{(t)})W_k^{(t)} - z_j^{(t)}\right)\right)\right\rangle \geq 0, \quad \forall z \tag{44}$$

It implies that

$$\ell\left(z;y_j\right) - \ell\left(z_j^{(t)};y_j\right) + \left\langle z - z_j^{(t)}, -\left(\lambda_j^{(t-1)} + \rho\left(\sum_{k=1}^{M} f(x_j^k;\theta_k^{(t)})W_k^{(t)} - z_j^{(t)}\right)\right)\right\rangle \geq 0, \forall z \tag{45}$$

Taking the limits for Eq. (45) and using the results in Eq. (35) Eq. (36) Eq. (37), we have

$$\ell\left(z;y_j\right) - \ell\left(z_j^*;y_j\right) + \left\langle z - z_j^*, -\lambda_j^*\right\rangle \geq 0, \forall z \tag{46}$$

That is:

$$\ell\left(z;y_j\right) + {\lambda_j^*}^\top\left(\sum_{k=1}^{M} f(x_j^k;\theta_k^*)W_k^* - z\right) \geq \ell\left(z_j^*;y_j\right) + {\lambda_j^*}^\top\left(\sum_{k=1}^{M} f(x_j^k;\theta_k^*)W_k^* - z_j^*\right)$$

It implies that

$$z_j^* \in \arg\min_{z_j} \ell\left(z_j;y_j\right) + {\lambda_j^*}^\top\left(\sum_{k=1}^{M} f(x_j^k;\theta_k^*)W_k^* - z_j\right)$$

This completes the proof.

□

C. Privacy Guarantees

1) Preliminaries: We utilize Rényi Differential Privacy (RDP) to perform the privacy analysis since it supports a tighter composition of privacy budget [56] than the moments accounting technique [1] for Differential Privacy (DP).

We start by introducing the definition of RDP as a generalization of DP, which leverages the α-Rényi divergence between the output distributions of two neighboring datasets. The definition of the neighboring dataset in this work follows the *client-level* differentially private FL framework [54]. *The neighboring datasets would differ in all samples associated with a single client, that is, one user is present or absent in the VFL global dataset.* (Definition 2)

Definition 3. (Rényi Differential Privacy [56]) We say that a mechanism $\mathcal{M}$ is (α, ϵ)-RDP with order $\alpha \in (1, \infty)$ if for all neighboring datasets D, D'

$$D_\alpha\left(\mathcal{M}(D) \| \mathcal{M}\left(D'\right)\right) := \frac{1}{\alpha-1} \log \mathbb{E}_{\theta \sim \mathcal{M}(D')}\left[\left(\frac{\mathcal{M}(D)(\theta)}{\mathcal{M}\left(D'\right)(\theta)}\right)^\alpha\right] \leq \epsilon \tag{47}$$

RDP guarantee can be converted to DP guarantee as follows:

Theorem 3. *(RDP to (ϵ, δ)-DP Conversion [3])* [6] *If f is an (α, ϵ)-RDP mechanism, it also satisfies $(\epsilon + \log \frac{\alpha-1}{\alpha} - \frac{\log \delta + \log \alpha}{\alpha-1}, \delta)$-differential privacy for any $0 < \delta < 1$.*

Here, we highlight three key properties that are relevant to our analyses.

Theorem 4. *(RDP Composition [56]) Let $f : \mathcal{D} \mapsto \mathcal{R}_1$ be (α, ϵ_1)-RDP and $g : \mathcal{R}_1 \times \mathcal{D} \mapsto \mathcal{R}_2$ be (α, ϵ_2)-RDP, then the mechanism defined as (X, Y), where $X \sim f(D)$ and $Y \sim g(X, D)$, satisfies $(\alpha, \epsilon_1 + \epsilon_2)$-RDP.*

Theorem 5. *(RDP Guarantee for Gaussian Mechanism [56]) If f is a real-valued function, the Gaussian Mechanism for approximating f is defined as $\mathbf{G}_\sigma f(D) = f(D) + \mathcal{N}\left(0, \sigma^2\right)$. If f has ℓ_2 sensitivity 1, then the Gaussian Mechanism $\mathbf{G}_\sigma f$ satisfies $(\alpha, \alpha/(2\sigma^2))$-RDP.*

2) Proof of Theorem 2: We aim to protect the local training data of each client under client-level (ϵ, δ)-DP guarantee (Definition 2) during VFL training. Let X be the VFL global dataset, i.e., the union of local feature sets $X_1, \ldots, X_M$ from all M clients. We denote the output of client k as a matrix $\mathcal{A}_k$, where each row is the embedding or logit of one local training sample. With a loss of generality, we consider the embedding matrix $\mathcal{A}_k = [h_1^k, \ldots, h_N^k]^\top$ as local output, and our analysis directly applies to the logit matrix $\mathcal{A}_k = [o_1^k, \ldots, o_N^k]^\top$. The local outputs from all clients can be concatenated as a global embedding matrix $\mathcal{A}$:

$$\mathcal{A} = [\mathcal{A}_1, \mathcal{A}_2, \ldots, \mathcal{A}_M] \tag{48}$$

For our algorithms (Algorithm 1, Algorithm 3) that sample a mini-batch of data with data indices $B(t)$ at each round t for clients to compute their embeddings, we view the corresponding $\mathcal{A}_k$ for each client k as:

$$\mathcal{A}_k^{(t)}[j] = h_j^{k(t)} \quad \text{if} \quad j \in B(t), \tag{49}$$

$$\mathcal{A}_k^{(t)}[j] = 0 \qquad \text{if} \quad j \notin B(t). \tag{50}$$

Here we can fill in the rows of the output matrix for non-sampled indices (i.e., $j \notin B(t)$) as all zeros for privacy analysis.

We will first analyze the privacy cost for one communication round (omitting the superscript t here) and then accumulate the privacy costs over T rounds via the DP composition theorem.

We define a function $\mathcal{H}$ that outputs a global embedding matrix consisting of clipped local embedding matrices for FL global dataset X as:

$$\mathcal{H}(X) = [\hat{\mathcal{A}}_1, \ldots, \hat{\mathcal{A}_M}], \text{where} \quad \hat{\mathcal{A}}_k = \texttt{Clip}(\mathcal{A}_k, C), \forall k \in [M]. \tag{51}$$

Lemma 6. *For any neighboring datasets X, X differing by all samples associated by one single client, the ℓ_2 sensitivity for $\mathcal{H}$ is C.*

Proof. WLOG, the neighboring dataset X' differs the first client from X, i.e., $X' = \{X_1{}', X_2, \ldots, X_M\}$. Therefore the global embedding matrix $\mathcal{H}(X)$ and $\mathcal{H}(X')$ only differ by the clipped local embedding matrix from the first client ($\hat{\mathcal{A}}_1$ and $\hat{\mathcal{A}}_1{}'$). Then, the ℓ_2 sensitivity for $\mathcal{H}$ is bounded as follows:

$$\max_{X, X'} \|\mathcal{H}(X) - \mathcal{H}(X')\|_2 = \sqrt{\|\hat{\mathcal{A}}_1 - \hat{\mathcal{A}}_1{}'\|_F^2} \leq C. \tag{52}$$

□

Then, we define our Gaussian mechanism $\mathbf{G}_{\sigma C}\mathcal{H}$, which outputs a global matrix consisting of noise-perturbed local embedding matrices for VFL global dataset X:

$$\mathbf{G}_{\sigma C}\mathcal{H}(X) = [\widetilde{\mathcal{A}_1}, \ldots, \widetilde{\mathcal{A}_M}], \text{where} \quad \widetilde{\mathcal{A}_k} = \texttt{Clip}(\mathcal{A}_k, C) + \mathcal{N}\left(0, \sigma^2 C^2\right), \forall k \in [M]. \tag{53}$$

Lemma 7. *Given the function $\mathcal{H}$ with ℓ_2 sensitivity C, Gaussian standard deviation $\sigma^2 C^2$, the Gaussian mechanism with $\mathbf{G}_{\sigma C}\mathcal{H}$ satisfies client-level $(\alpha, \alpha/(2\sigma^2))$-RDP.*

Proof. The ℓ_2 sensitivity for the function $\mathcal{H}$ is C by Lemma 6. The Gaussian standard deviation for the noise-perturbed embedding is σC, which is proportional to the clipping constant C. Combining it with Theorem 5 yields the conclusion that $\mathbf{G}_{\sigma C}\mathcal{H}$ guarantees client-level $(\alpha, \alpha/(2\sigma^2))$-RDP. □

We note that the training process in the server does not access the raw data X_k, thus it does not increase the privacy budget and the whole algorithm in one round satisfies RDP by the post-processing property of RDP. For algorithms with T

[6] This theorem is tighter than the original RDP paper [56], and it is adopted in the official implementation of the PyTorch Opacus library.

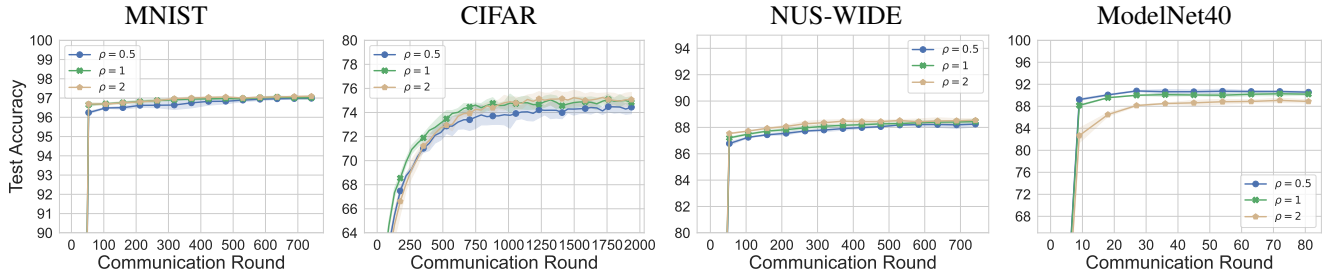

Fig. 5: Performance of VIMADMM with different penalty factor ρ on four datasets. VIMADMM is not sensitive to ρ from 0.5 to 2.

communication rounds, we use the RDP Composition theorem (Theorem 4) to accumulate the privacy budge over T rounds, and convert the RDP guarantee into DP guarantee (Theorem 3).

Finally, we recall Theorem 2 and provide the formal proof.

Theorem 2. *Given a total of M clients, T communication rounds, clipping threshold C and noise level σ, DP versions of Algorithm 1, 3 satisfy client-level $(\frac{T\alpha}{2\sigma^2} + \log\frac{\alpha-1}{\alpha} - \frac{\log\delta+\log\alpha}{\alpha-1}, \delta)$-DP for any $\alpha > 1$ and $0 < \delta < 1$.*

Proof. At each communication round, according to Lemma 7, $\mathbf{G}_{\sigma C}\mathcal{H}$ satisfies client-level $(\alpha, \frac{\alpha}{2\sigma^2})$-RDP. Due to the post-processing property of RDP, after server training, our DP algorithms (i.e., DP versions of Algorithm 1, 3) with one round still satisfy client-level $(\alpha, \epsilon'(\alpha))$-RDP. Based on RDP Composition theorem (Theorem 4), our DP algorithms with T communication rounds satisfy client-level $(\alpha, \frac{T\alpha}{2\sigma^2})$-RDP. Based on the connection between RDP and DP in Theorem 3, our DP algorithms with T communication rounds also satisfy client-level $(\frac{T\alpha}{2\sigma^2} + \log\frac{\alpha-1}{\alpha} - \frac{\log\delta+\log\alpha}{\alpha-1}, \delta)$-DP.

□

D. Experimental Details and Additional Results

1) Platform: We simulate the vertical federated learning setup (1 server and N clients) on a Linux machine with AMD Ryzen Threadripper 3990X 64-Core CPUs and 4 NVIDIA GeForce RTX 3090 GPUs. The algorithms are implemented by PyTorch [58]. Please see the submitted code for full details. We run each experiment 3 times with different random seeds.

2) Hyperparameters: We detail our hyperparameter tuning protocol and the hyperparameter values here. For all VFL training experiments, we use the SGD optimizer with learning rate η for the server's model, and the SGD optimizer with momentum 0.9 and learning rate η for client k's local model. The regularization weight β is set to 0.005. The embedding dimension d_f is set to 60, and batch size b is set to 1024 for all datasets.

a) Vanilla VFL training: For Vanilla VFL training experiments, we tune learning rates by performing a grid search separately for all methods over $\{0.05, 0.1, 0.3, 0.5, 0.8\}$ on MNIST, $\{0.003, 0.005, 0.008, 0.01, 0.05, 0.1\}$ on CIFAR, $\{0.05, 0.1, 0.5\}$ on NUS-WIDE, $\{0.0005, 0.005, 0.01, 0.05, 0.1\}$ on ModelNet40. For ADMM-based methods, we tune penalty factor ρ with a search grid $\{0.5, 1, 2\}$ on all datasets.

b) Differentially private VFL training: We leverage the PyTorch Differential Privacy library Opacus [7] to calculate the privacy budgets ϵ. In all experiments, $\delta = 1e-5$. For each privacy budget ϵ, we perform a grid search for the combination of hyperparameters (including noise scale σ, clipping threshold C, and learning rate η) for all methods for a fair comparison. The noise scale is tuned from σ $\{2, 3, 5, 8, 10, 30, 50, 70\}$ on all datasets. C is tuned from $\{0.0005, 0.001, 0.005, 0.01, 0.1, 1\}$ and η is tuned from $\{0.05, 0.3, 0.5, 1\}$ for MNIST; C is tuned from $\{0.01, 0.05, 0.1, 0.5, 1\}$ and η is tuned from $\{0.005, 0.01, 0.05, 0.1, 0.5, 1\}$ for CIFAR; C is tuned from $\{0.001, 0.005, 0.01, 0.05, 0.1\}$ and η is tuned from $\{0.05, 0.1, 0.3, 0.5, 1\}$ for NUS-WIDE; C is tuned from $\{0.01, 0.05, 0.1, 0.5, 1\}$ and η is tuned from $\{0.05, 0.1, 0.5\}$ for ModelNet40. We use the best hyper-parameters to start 3 runs with different random seeds and report the average results for each method.

c) Client-level explainability: In the experiments of *client importance validation via noisy test client*, for each time, we perturb the features of all test samples at one client by adding Gaussian noise sampled from $\mathcal{N}\left(0, \bar{\sigma}^2\right)$ to its features. In order to observe the difference in test accuracy between important clients and unimportant clients, we set $\bar{\sigma}$ to 10 for MNIST, 1 for CIFAR and NUS-WIDE, and 3 for ModelNet40.

In the experiments of *client denoising*, we construct one noisy client (i.e., client 7, 5, 2, 3 for MNIST, CIFAR, NUS-WIDE, ModelNet40 respectively) by adding Gaussian noise sampled from $\mathcal{N}\left(0, \widetilde{\sigma}^2\right)$ to all its training samples and test samples. We set $\widetilde{\sigma}$ to 1 for MNIST, NUS-WIDE and ModelNet40, and 3 for CIFAR.

3) Additional Evaluation Results:

a) Effect of ρ: Here we report the test accuracy of VIMADMM with different penalty factor ρ in Figure 5, which show that VIMADMM is not sensitive to ρ on four datasets.

[7]https://github.com/pytorch/opacus

b) Results for a large number of clients.: We evaluate baselines and our methods under 100 clients on MNIST by allowing the agents to obtain overlapped features, and the results show that our methods still outperform baselines. Specifically, we divide the features into 100 overlapped subsets for 100 clients so that each client has 14 pixels. The results in Table XI show that VIM methods (i.e., `VIMADMM`, `VIMADMM-J`) have higher accuracy than baselines in both w/ and w/o model splitting settings.

TABLE XI: Performance of Vanilla VFL when $M = 100$ on MNIST

W/ model splitting			W/o model splitting	
`VAFL`	`Split Learning`	`VIMADMM`	`FDML`	`VIMADMM-J`
95.38	95.45	**95.77**	95.85	**95.96**

c) Results for client subsampling under client-level DP: We extended our study under client-level DP by incorporating a subsampling mechanism into the `VIMADMM` framework to save the privacy budget. Specifically, during each communication round, the server receives the local embeddings from $p\%$ clients, corresponding to a participation rate of $p\%$. To address missing local embeddings, the server leverages historical local embeddings from other clients to complete their absent local embeddings.

We calculate the privacy budget ϵ_i for client following our Theorem 2, where T denotes the number of communication rounds that client i uploads local embeddings, instead of the total number of communication rounds as in our original algorithm. Due to the non-overlapping nature of local data among clients, the concatenated output matrix from all clients satisfies the $\max_i \epsilon_i$ client-level DP guarantee according to DP parallel composition.

As shown in Table XII, there is a utility loss when $p\% = 25\%, 50\%$ compared to $p\% = 100\%$ under DP and non-DP settings on MNIST and CIFAR. This discrepancy demonstrates the necessity of aggregating local outputs from all clients during training to achieve optimal utility in vertical federated learning.

TABLE XII: The utility of `VIMADMM` with different client subsampling ratio under client-level DP.

	MNIST			CIFAR		
	25%	50%	100%	25%	50%	100%
$\epsilon = \infty$	94.20	94.52	97.13	65.78	66.10	75.25
$\epsilon = 8$	85.98	87.22	92.35	62.01	62.86	73.83
$\epsilon = 1$	85.51	86.62	91.09	46.53	46.69	61.65

d) Additional results on client denoising: Table XIII presents the test accuracy of `VAFL`, `Split Learning`, and `VIMADMM` at different epochs (communication rounds) on different datasets under one noisy client. Note that each epoch consists of N/b communication rounds. Table XIII shows that under the noisy training scenario, `VIMADMM` consistently outperform `Split Learning` and `VAFL` with faster convergence and higher test accuracy, which indicates the effectiveness of `VIM`'s multiple linear heads in client denoising.

TABLE XIII: Test accuracy under one noisy client whose training local features and test local features are perturbed by Gaussian noise.

Method	Test accuracy @ epoch (communication round)											
	MNIST			CIFAR			NUS-WIDE			ModelNet40		
	2 (106)	5 (265)	10 (530)	2 (88)	5 (220)	10 (440)	2 (106)	5 (265)	10 (530)	2 (18)	5 (45)	10 (90)
`VAFL`	91.07 ± 0.17	94.36 ± 0.16	95.59 ± 0.11	28.83 ± 1.04	38.77 ± 0.39	46.98 ± 0.70	51.88 ± 0.72	77.68 ± 0.74	85.31 ± 0.15	43.23 ± 3.07	80.13 ± 1.10	89.56 ± 0.41
`Split Learning`	95.04 ± 0.14	96.01 ± 0.03	96.43 ± 0.08	42.75 ± 0.13	50.06 ± 0.18	55.53 ± 0.37	85.35 ± 0.24	86.42 ± 0.24	87.14 ± 0.29	77.94 ± 1.00	88.74 ± 0.07	89.69 ± 0.42
`VIMADMM`	96.22 ± 0.07	96.60 ± 0.04	96.82 ± 0.07	67.08 ± 0.43	70.70 ± 0.34	71.76 ± 0.14	86.38 ± 0.20	87.00 ± 0.27	87.18 ± 0.14	90.05 ± 0.38	90.71 ± 0.31	90.59 ± 0.05

e) Reference accuracy for SOTA model and reference model in the centralized setting: we included the comparisons in Table XIV, which shows the reference accuracy for SoTA models and a simple reference model in a centralized setting on four datasets. Specifically,

- *SoTA models*: These models may employ different model architectures and training methods compared to our approach. They serve as a virtual upper bound as the highest achievable accuracy on each dataset.
- *Reference model in the centralized setting*. This reference model has the same model size as one local model coupled with a server model.

For instance, on the MNIST dataset, the latest SoTA method in a centralized setting achieves an accuracy of 99.87%, while a basic reference model (comprising one local model followed by a server model) reaches 98.19%. In comparison, our VFL model (consisting of M local models followed by a server model) demonstrates a comparable accuracy of 97.13%.

Furthermore, on datasets such as NUS-WIDE and ModelNet, our VFL model even surpasses the accuracy of the reference model in a centralized setting. This is attributable to the significantly higher number of model parameters in VFL. For example, in ModelNet, we utilized four ResNet-18 feature extractors as local models for four different clients, allowing for a more nuanced understanding and representation of the data.

TABLE XIV: Accuracy for SOTA model and reference model in the centralized setting.

	MNIST	CIFAR	NUS-WIDE (5 classes)	ModelNet40 (2D multi-views)
SOTA method in centralized setting (virtual upper bound)	99.87 [9]	99.50 [18]	88.7 [47]	96.6 [73]
Reference model in centralized setting (e.g., one local model followed by server model)	98.19	77.61	87.71	88.96
VIMADMM Model (e.g., M local models followed by server model)	97.13	75.25	88.51	91.32

f) VIMADMM on larger models: VIMADMM can scale well to large model such as ResNet-18 as shown in the experiments on ModelNet40. Leveraging the larger models as feature extractors for clients, VIMADMM can produce higher-quality local embeddings, which are also crucial for learning accurate linear heads on the server side. Here we also report the results of VIMADMM on CIFAR with CNN, ResNet-18, and ResNet-34, which are 75.25%, 81.35%, and 82.58%, respectively. It shows that a larger model can lead to higher accuracy for VIMADMM, validating its scalability and efficiency.

g) VIMADMM on non-image tasks: VIMADMM can be adapted to non-image tasks, such as datasets with both text and image modalities. For example, in the NUS-WIDE dataset, which encompasses both text and image features as local datasets, VIMADMM achieves state-of-the-art results as shown in Figure 1. This adaptability is due to VIMADMM's flexible design, which can handle heterogeneous input data types via different feature extractors (e.g., local models) in the clients, and then aggregate heterogeneous local embeddings via multiple linear heads in the server. We believe these results underscore VIMADMM's potential in a broader range of applications beyond image tasks.

h) VIMADMM under long-tail datasets: Long-tail datasets are characterized by a significant imbalance, where minority classes have far fewer samples than majority ones. *This horizontal imbalance is distinct from the challenges addressed by vertical federated learning, where the same sample (whether it belongs to a majority or minority class) is vertically split across multiple clients.* This means that *in vertical federated learning, minority class samples are still be evenly distributed among clients*. We conduct additional experiments on long-tail data.

We create long-tail training datasets following [71] with an imbalance factor of 10 (i.e., the ratio of samples in the head to tail class). Specifically, for MNIST, this resulted in class sample sizes of [6000, 4645, 3596, 2784, 2156, 1669, 1292, 1000, 774, 600] for 10 classes. For CIFAR, the class sample sizes are [5000, 3871, 2997, 2320, 1796, 1391, 1077, 834, 645, 500] across the 10 classes.

We compared the VIMADMM model, which consists M local models followed by a server model, with a reference model in a centralized setting. This reference model has the same model size as one local model coupled with a server model.

We show the results in Table V, demonstrating that our VIMADMM is still effective on challenging long-tail training datasets, yielding results comparable to those of the reference model in a centralized setting. Moreover, the long-tail version of MNIST dataset does not significantly impact the accuracy of VIMADMM compared to the original MNIST dataset.

E. Discussion

a) DP generative model in VFL: An alternative way to achieve DP in VFL is to locally train a DP generative model for each client, and then send the DP generative models to the server, which takes only one communication round. However, we identified several key challenges that make it less suitable for our VFL context:

- *Mismatch of Synthetic Partial Features Across Clients.* In VFL, there are M clients holding different subsets of features for the same training samples (denoted as $x_j^1, x_j^2, \ldots, x_j^M$ for sample j). A generative model, due to its stochastic nature, would generate synthetic partial features without correspondence to a specific training sample (e.g., the parietal features generated from a local generative model would adhere to the local data distribution, but do not correspond to a particular original sample j). This lack of correspondence means that the server cannot effectively concatenate the synthetic partial features into a cohesive "global" dataset for training. This is a limitation of the generative model in VFL.
- *Quality Concerns Due to Partial Features.* Given that each client only has partial features (see Figure 3 row 1 for the visualization of raw local features), a generative model trained locally (without FL) might yield lower-quality data. This is particularly problematic in cases where partial features are not informative (e.g., clients having only background pixels in image datasets like MNIST/CIFAR). The state-of-the-art accuracy for synthetic data in centralized learning with sample-level DP on MNIST is around 97.6% under $\epsilon = 1$ and 98.2% under $\epsilon = 10$ [37]. Since the partial features in VFL are less informative than the full features in centralized learning, the DP genetive model in VFL would lead to lower accuracy. On the other hand, our VIMADMM DP learning algorithm already achieves a promising accuracy that is close to the state-of-the-art: 91.35% under $\epsilon = 1$ and 92.35% under $\epsilon = 8$ in VFL under client-level DP, which is a stricter privacy notion than sample-level DP.

- *Scalability Issues of DP Generative Model with High-Dimensional Data.* DP generative models often struggle with high-dimensional datasets [37, 72]. For instance, their performance on datasets like CIFAR10 is limited [72], posing a challenge for more complex datasets like ModelNet40. Additionally, the generation of multi-modal data (e.g., text and image features in NUS-WIDE) remains an unresolved challenge. In our method, as we are not training generative models, the high dimensionality of data will not pose a significant challenge to `VIMADMM`.
- *Communication Overhead with Generative Model.* The model size of a standard DCGAN [61] implemented in PyTorch [8] is 13.65 MB. As it only takes one round for communication, the communication costs for each client would be 13.65 MB. In comparison, `VIMADMM` demonstrates similar communication costs on certain datasets. For example, on the ModelNet40 dataset, `VIMADMM` achieves an accuracy of 89% with a total communication cost of 11.32 MB (See Figure 1 and Table IV for details). This efficiency stems from the transmission of local embeddings and ADMM-related variables, which collectively have a smaller size than the number of parameters within a deep neural network like a DCGAN generator. This trend becomes even more evident if we use a larger generative model than DCGAN, which is a common direction in current generative AI advancements. Moreover, the high-quality local embeddings (e.g., from a pretrained ResNet-18 as local model) and multiple local updates at each communication round (enabled by ADMM) significantly aid convergence. Consequently, a relatively small number of communication rounds (approximately 10) is required to reach an accuracy of 89% on ModelNet40.

  We remain open to future research exploring the feasibility and optimization of the local training of DP generative models in VFL settings.

b) Linear head and client importance: In Section VI-C, we utilize the norm of weights in the linear head, learned by the server from local embeddings, to determine the importance of the corresponding client (and its local features). Our approach is in line with existing methods such as LIME [63], SHAP [49], and others [29] that utilize model weights to determine feature importance. In our model, we follow existing work by assuming feature independence [63, 49, 29] to simplify the interpretation of weights in terms of feature importance.

c) Challenges of ADMM algorithm design in VFL: There are several key challenges of designing ADMM algorithm in VFL for distributed optimization:

- how to ensure the consensus among clients and form it as a constrained optimization problem (e.g., from Eq. 2 to Eq. 3).
- how to decompose the optimization problem into small sub-problems that can be solved in parallel by ADMM.

For the first challenge, although ADMM is flexible to introduce auxiliary variables and thus formulate a constrained optimization problem in HFL, it raises new challenges in VFL. For example, the ADMM-based methods in HFL [23, 22, 38, 81] usually use the global model as the auxiliary variable and enforce the consistency between the global model and each local model. However, VFL communicates embeddings, and it is not feasible to enforce local embeddings from different clients to be the same as they provide unique information from different aspects. Therefore, in this paper, we introduce the auxiliary variable z_j for each sample j and construct the constraint between z_j and server's output $\sum_{k=1}^{M} h_j^k W_k$ (i.e., the logits), which enables the optimization for each W_k by ADMM.

For the second challenge, we propose the bi-level optimization for server's model and clients' models to train DNNs for VFL with model splitting, while the existing ADMM-based method in VFL [35] only considers logistic regression with linear models in client-side, which does not apply to DNNs. The initial attempt we made is to decompose the optimization for server's linear heads by ADMM while still using chain rule of SGD to update local models, which does not exhibit much superiority over pure SGD-based methods. Later, we decompose the optimization for both server's linear heads and local models by ADMM, leading to our current algorithm `VIMADMM` that enables multiple local updates for clients at each communication round and achieves significantly better performance, as we show in Sec. VI-A.

d) Limitations: Directly deploying VFL algorithms without stopping criteria or regularization techniques may lead to the over-fitting phenomenon, as in many other algorithms. Based on our experiments, we find that over-fitting is a common problem of VFL algorithms due to a large number of model parameters from all clients in the whole VFL system. Compared to centralized learning or horizontal FL, the prediction for one data sample in VFL involves M times model parameters, which corresponds to M partitions of input features. To prevent over-fitting, we use regularizers to constrain the complexity of models and adopt standard stopping criteria, i.e., stop training when the model converges or the validation accuracy starts to drop more than 2%.

[8] https://pytorch.org/tutorials/beginner/dcgan_faces_tutorial.html