RAP: Random Projection is What You Need for Vertical Federated Learning

Qinbo Zhang[1]*, Xiao Yan[2]*, Yukai Ding[1], Fangcheng Fu[3], Chuang Hu[1], Quanqing Xu[4]✉, Xu Chen[1], and Jiawei Jiang[1]✉

[1] School of Computer Science, Wuhan University
{qinbo_zhang, yukai.ding, handc, xuchen, jiawei.jiang}@whu.edu.cn
[2] Centre for Perceptual and Interactive Intelligence (CPII)
yanxiaosunny@gmail.com
[3] School of Computer Science, Peking University
ccchengff@pku.edu.cn
[4] OceanBase, Ant Group
xuquanqing.xqq@oceanbase.com

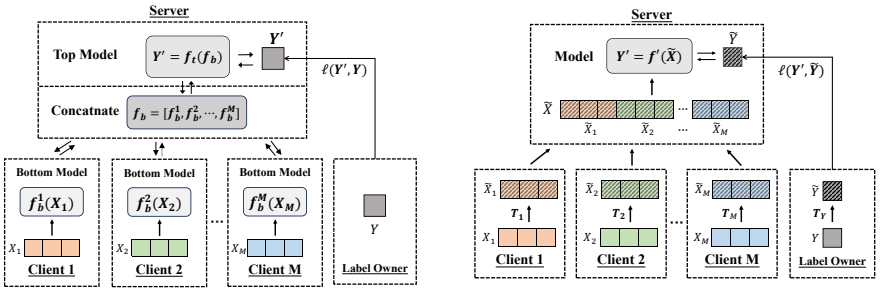
Abstract. Vertical federated learning (VFL) considers model training when the features of data samples are partitioned over a set of clients. As the standard practice of VFL, SplitNN decomposes a model into a bottom part on the clients and a top part on a server, and requires the clients and server to exchange activations/gradients in every mini-batch. We observe that SplitNN is inefficient due to frequent client-server communication and propose *random projection* (i.e., RAP) to improve efficiency. RAP is radically simple, i.e., the clients transform their local features and transfer the transformed data to the server, and the server trains on the transformed data without communicating with the clients. As only one round of client-server communication is required, RAP is much more efficient than SplitNN. RAP uses Gaussian projection matrix as the data transformation. Data privacy is preserved because the projection matrix of each client is private and resembles the bottom model of SplitNN. Model accuracy is not affected because the Gaussian projection matrix is invertible and preserves the geometry of the original space. To evaluate RAP, we experimented with both regression and classification tasks on six datasets. The results show that RAP matches SplitNN in model accuracy and accelerates training by over 630×.

Keywords: vertical federated learning · random projection · communication efficiency.

1 Introduction

Federated learning coordinates multiple participants cooperate to train a model while safeguarding their data [23, 25], and we focus on vertical federated learning (VFL) in this paper. Specifically, VFL involves M participants (also called clients)

* Equal contribution ✉ Corresponding author



(a) SplitNN. (b) RAP.

Fig. 1. The SplitNN framework and our RAP for vertical federated learning (VFL).

Table 1. Performance on YP for regression and SU for classification. Smaller is better for RMSE, and larger is better for accuracy. *Centralized* (Central) is the ideal case. With *homomorphic encryption* (HE), clients encrypt data and server decrypts data for training with a trusted execution environment. *Local differential privacy* (LDP) adds noise to client data before sending to server.

Method		Central	SplitNN	HE	LDP	RAP
YP	RMSE ↓	8.83	8.91	8.90	9.71	8.87
	Time (s)	75	5.7e4	3.6e3	92	90
SU	Acc (%) ↑	80.31	80.30	80.30	75.22	80.32
	Time (s)	140	8.0e4	4.6e3	165	164

and a central server. The dataset $\mathcal{D}$ has N data samples, and $\mathbf{X} \in \mathbb{R}^{N \times F}$ contains all features with F being the feature dimension. Each participant holds a subset of the features for all samples, i.e., $\mathbf{X}_m \in \mathbb{R}^{N \times F_m}$ with F_m being the local feature dimension on client m, and we have $F = \sum_{m=1}^{M} F_m$. One special client holds the labels for all data samples (denoted as $\mathbf{Y}$) and is called the label owner.

To conduct VFL, a framework is required to coordinate the participants, and SplitNN [33] is the most widely used [37, 38, 18] and illustrated in Figure 1(a). In particular, SplitNN decomposes the model into a *bottom part* (e.g., first few layers of a neural network) and a *top part* (e.g., last layers of a neural network). The clients run the bottom model on their local features and transfer activations to the server. The server concatenates the activations from all clients and feeds them to the top model. During backward propagation, the server transfers gradients to update the client bottom models.

We observe that SplitNN suffers from a long training time because client-server communication is required in every mini-batch. As reported in Table 1, SplitNN's training time can be over 500 times of centralized training, which keeps all features on the server and represents an ideal case. As alternatives for SplitNN, homomorphic encryption (HE) requires a trusted execution environment (TEE)

to protect data privacy and involves encryption and decryption overheads [29]. Local differential privacy (LDP) degrades model accuracy because it adds noise to client data to protect privacy [2]. The limitations of existing solutions motivate us to ask the following research question:

> *Is it possible to design a VFL framework that protects data privacy as SplitNN and achieves efficiency and accuracy as ideal centralized training?*

To address the research question, we propose *RAP* as a novel VFL framework, which is depicted in Figure 1(b). In particular, clients independently transform their local features and send the transformed data to the server. The server aggregates the transformed data and trains the model without client interaction. Thus, RAP achieves high efficiency by using a single round of client-server communication. As the transformations are private to the clients, like SplitNN's bottom models, data privacy is protected since the server cannot decode client features from transformed data.

Although RAP's concept is simple, the main challenge is designing transformations so that the server can train on the transformed data without compromising model accuracy. We use Gaussian projection matrix, which transforms a feature vector $\mathbf{x} \in \mathbb{R}^d$ as $\tilde{\mathbf{x}} = \mathbf{A}\mathbf{x}$. Denote the model produced by centralized training as $g(x)$, model accuracy will not be affected if training can learn to reverse the transformation *implicitly*, i.e., producing $f(\tilde{\mathbf{x}}) = g \odot \mathbf{A}^{-1}$ ($\odot$ denotes function composition). Gaussian matrix is inevitable with a probability of almost 1. Moreover, Gaussian projection preserves the geometry of the original space, i.e., $\|\mathbf{x}_1 - \mathbf{x}_2\| \approx \|\mathbf{A}\mathbf{x}_1 - \mathbf{A}\mathbf{x}_2\|$ with high probability. This suggests that for models relying on spatial geometry, the original and transformed features are similar.

To make RAP work, special cases that affect data privacy or model accuracy need to be addressed. For instance, when a client has only one feature, Gaussian projection becomes linear scaling. If the server knows the average of the feature, it might solve the scaling factor and decode the original features. To tackle these cases, we generalize the idea of random projection as transformations that cannot be inverted explicitly for data reconstruction but can be inverted implicitly for model training, and design specialized transformations. We also analyze for the privacy protection of RAP theoretically.

We conduct extensive experiments to evaluate RAP and compare it with SplitNN for both classification and regression tasks. The results show that RAP matches SplitNN in model accuracy but requires significantly shorter training time. As shown in Table 1, RAP runs only slightly longer than ideal centralized training and can speed up SplitNN by 600×. RAP is robust against different numbers of clients, feature distributions, and feature types. To summarize, we make the following contributions:

- We observe that the popular SplitNN framework for vertical federated learning (VFL) suffers from a long training time due to high communication costs.
- We propose the RAP framework for VFL, which achieves efficiency with only one round of client-server communication and preserves privacy with local transformations on the clients.

Algorithm 1: The procedure of the RAP framework.

1. **Data transformation** //performed on the clients and label owner
   Client: $\tilde{\mathbf{X}}_m = T_m(\mathbf{X}_m)$; LabelOwner: $\tilde{\mathbf{Y}} = T_Y(\mathbf{Y})$
2. **Data transfer** //send transformed data and labels to the server
   Client: $send(\tilde{\mathbf{X}}_m)$; LabelOwner: $send(\tilde{\mathbf{Y}})$
3. **Model training** //conducted entirely on the server
   Concatenate the transformed features $\tilde{\mathbf{X}} = [\tilde{\mathbf{X}}_1, \cdots, \tilde{\mathbf{X}}_m, \cdots, \tilde{\mathbf{X}}_M]$
   Compute $\mathcal{L}(\tilde{\mathcal{D}}, \tilde{\theta}) := \sum_{\tilde{\mathbf{x}} \in \tilde{\mathbf{X}}, \tilde{\mathbf{y}} \in \tilde{\mathbf{Y}}} \ell(f'(\tilde{\mathbf{x}}, \tilde{\theta}), \tilde{\mathbf{y}})$, update parameter $\tilde{\theta}$
4. **Model serving** //$\mathbf{x}_t$ is the target data sample for serving
   Client: $\tilde{\mathbf{x}}_{tm} = T_m(\mathbf{x}_{tm})$, $send(\tilde{\mathbf{x}}_{tm})$
   Server: $\tilde{\mathbf{x}}_t = [\tilde{\mathbf{x}}_{t1}, \tilde{\mathbf{x}}_{t2}, \cdots, \tilde{\mathbf{x}}_{tM}]$, $\mathbf{y}'_t = f'(\tilde{\mathbf{x}}_t, \tilde{\theta})$, $send(\mathbf{y}'_t)$
   LabelOwner: $y_t = T_Y^{-1}(\mathbf{y}'_t)$ // T_Y^{-1} is the inverse of T_Y

– We adopt Gaussian matrix projection to transform data for good model accuracy and tailor the transformations for special cases that harm model accuracy or data privacy.

2 The RAP Framework

In this part, we first introduce the overall procedure of RAP framework, then analyze the rationale of using Gaussian projection matrix, and finally discuss how to handle special cases of the feature and label.

2.1 Workflow Overview

Figure 1(b) illustrates RAP, and Algorithm 1 summarizes its workflow with the following steps:

Data transformation. Each client m transforms its features $\mathbf{X}_m$ independently with a private transformation and the label owner transforms the label $\mathbf{Y}$. The transformations are expressed as $\tilde{\mathbf{X}}_m = T_m(\mathbf{X}_m)$ and $\tilde{\mathbf{Y}} = T_Y(\mathbf{Y})$ for the features and labels, respectively. $T_m(\cdot)$ is usually a Gaussian projection, i.e., $\tilde{\mathbf{X}}_m = \mathbf{X}_m \times \mathbf{A}_m$, with $\mathbf{A}_m \in \mathbb{R}^{F_m \times F_m}$ being an Gaussian matrix generated by client m. The label transformation $T_Y(\cdot)$ also uses random projection but is carefully designed to prevent label leakage. More design rationale and details are in the following section.

Data transfer. Each client m sends its transformed feature $\tilde{\mathbf{X}}_m$ to the server, and the label owner also sends the transformed label $\tilde{\mathbf{Y}}$. The server concatenates the transformed features from all clients as $\tilde{\mathbf{X}} = [\tilde{\mathbf{X}}_1, \tilde{\mathbf{X}}_2, \cdots, \tilde{\mathbf{X}}_M]$. $\tilde{\mathbf{X}}$ and $\tilde{\mathbf{Y}}$ are used as features and labels for model training on the server.

Model training. The server conducts training on $\tilde{\mathbf{X}}$ and $\tilde{\mathbf{Y}}$ without communicating with the clients:

$$\mathcal{L}(\tilde{\mathcal{D}}, \tilde{\theta}) := \sum_{\tilde{\mathbf{x}} \in \tilde{\mathbf{X}}, \tilde{\mathbf{y}} \in \tilde{\mathbf{Y}}} \ell(f'(\tilde{\mathbf{x}}, \tilde{\theta}), \tilde{\mathbf{y}}). \tag{1}$$

The server model $f'(\tilde{\mathbf{x}}, \tilde{\theta})$ takes transformed features $\tilde{\mathbf{x}}$ as input and predicts transformed label $\tilde{\mathbf{y}}$. As $f'(\tilde{\mathbf{x}}, \tilde{\theta})$ is different from the target model $f(\mathbf{x}, \theta)$ that works on the original features, special considerations are required for model serving (i.e., computing the predicted label for a sample).

Model serving. Given a data sample $\mathbf{x}_t$, RAP takes three steps to obtain its predicted label. First, each client transforms its local feature of $\mathbf{x}_t$ reusing its feature transformations for model training (i.e., $\tilde{\mathbf{x}}_{tm} = T_m(\mathbf{x}_m)$). Second, the server collects and concatenates the transformed features from all clients $\tilde{\mathbf{x}}_t = [\tilde{\mathbf{x}}_{t1}, \tilde{\mathbf{x}}_{t2}, \cdots, \tilde{\mathbf{x}}_{tM}]$, and then feeds the feature $\tilde{\mathbf{x}}_t$ to $\mathbf{y}'_t = f'(\tilde{\mathbf{x}}, \tilde{\theta})$. Finally, the model predicted label $\mathbf{y}'_t$ is sent to the label owner to decode $\mathbf{y}_t$ as $\mathbf{y}_t = T_Y^{-1}(\mathbf{y}'_t)$, where $T_Y^{-1}(\cdot)$ is the inverse function of the label transformation $T_Y(\cdot)$.

2.2 Design Rationale

RAP ensures data privacy because the feature transformation $T_m(\cdot)$ and label transformation $T_Y(\cdot)$ are private like SplitNN's bottom models. As such, the server cannot reconstruct the original client data. Using the Gaussian projection will not harm model accuracy for the following reasons.

Corollary 1. *Random Gaussian matrix $\mathbf{A}_m \in \mathbb{R}^{F_m \times F_m}$ has full-rank (invertible) with probability 1.*

This is proved by [15] and suggests that $\mathbf{A}_m$ has an inverse matrix $\mathbf{A}_m^{-1}$. For client m with $\tilde{\mathbf{X}}_m = \mathbf{X}_m \times \mathbf{A}_m$, applying $\mathbf{A}_m^{-1}$ reverses the transformation, i.e., $\mathbf{X}_m = \tilde{\mathbf{X}}_m \times \mathbf{A}_m^{-1}$. Let the model trained on the original features be $f(\mathbf{x}, \theta)$ and the model trained on the transformed features in RAP be $f'(\tilde{\mathbf{x}}, \tilde{\theta})$, $f'(\tilde{\mathbf{x}}, \tilde{\theta})$ and $f(\mathbf{x}, \theta)$ produce the same output if

$$f'(\tilde{\mathbf{x}}, \tilde{\theta}) = \mathbf{A}_m^{-1} \odot f(\mathbf{x}, \theta) = f(\tilde{\mathbf{x}} \times \mathbf{A}_m^{-1}, \theta), \tag{2}$$

where $\odot$ denotes the composition of two functions. That is, if model training can learn to reverse the transformation implicitly, the two models produce the same output. For instance, if the model is a multi-layer perceptron (MLP) and the first layer of $f(\mathbf{x}, \theta)$ is $\mathbf{x} \times \mathbf{W}$, and the first layer of $f'(\tilde{\mathbf{x}}, \tilde{\theta})$ becomes $\tilde{\mathbf{x}} \times \mathbf{A}_m^{-1} \times \mathbf{W}$, which changes the model parameters. In fact, if the original feature space yields good accuracy, gradient-based training is likely to learn that $f'(\tilde{\mathbf{x}}, \tilde{\theta}) = \mathbf{A}_m^{-1} \odot f(\mathbf{x}, \theta)$.

Corollary 2. *For two feature vectors $\mathbf{x}_{m1}, \mathbf{x}_{m2} \in \mathbb{R}^{F_m}$ and a Gaussian matrix $\mathbf{A}_m \in \mathbb{R}^{F_m \times F_m}$, given an error parameter $0 < \epsilon < 1$, the following holds with probability $p \geq 1 - 2e^{-F_m(3\epsilon^2 - 2\epsilon^3)/12}$*

$$(1-\epsilon)||\mathbf{x}_{m1} - \mathbf{x}_{m2}||^2 \leq \frac{1}{F_m}||\mathbf{A}_m\mathbf{x}_{m1} - \mathbf{A}_m\mathbf{x}_{m1}||^2 \leq (1+\epsilon)||\mathbf{x}_{m1} - \mathbf{x}_{m2}||^2.$$

The corollary suggests that Gaussian projection preserves the distance between two feature vectors with high probability and is indicated by the famous Johnson-Lindenstrauss Lemma [12]. This means models that depend on geometric properties, such as SVM and logistic regression, can achieve similar accuracy on the original and transformed features. It also suggests that the singular values of $\mathbf{A}_m$ are likely to be close to 1, making its inverse $\mathbf{A}_m^{-1}$ both stable and easy to learn.

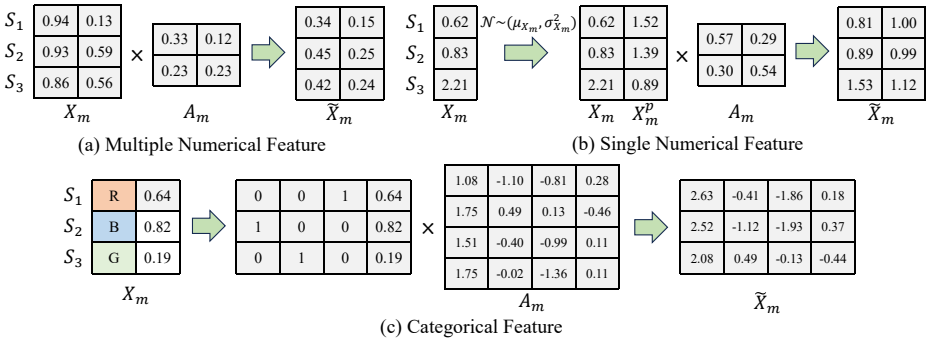


Fig. 2. Illustrations of the transformation for features.

2.3 Transformations for Special Cases

The common case for RAP is when a client m holds multiple numerical features ($\mathbf{X}_m \in \mathbb{R}^{N \times F_m}, F_m > 1$), applying random projection as shown in Figure 2(a). The original feature $\mathbf{X}_m$ cannot be recovered from $\tilde{\mathbf{X}}_m$ without the projection matrix $\mathbf{A}_m$. However, some special cases, such as single numerical features or categorical data, require tailored transformations to ensure privacy and accuracy. While these transformations vary, they follow the Gaussian projection principle—making the transformation irreversible to the server while allowing the model to implicitly recover information for training.

A single numerical feature. A client may hold only a single numerical feature, i.e., $\mathbf{X}_m \in \mathbb{R}^{N \times F_m}$ with $F_m = 1$ (e.g., the salary of all customers for a bank). If we still generate the Gaussian projection matrix $\mathbf{A}_m \in \mathbb{R}^{1 \times 1}$, the transformation becomes a linear scaling, i.e., $\tilde{x} = ax$ for each sample $x \in \mathbf{X}_m$. If the server has the mean of the feature μ_x (e.g., by collecting a small sample of users), it can estimate a as $\tilde{a} = \sum_{\tilde{x} \in \tilde{\mathbf{X}}_m} \tilde{x}/(N\mu_x)$ and reverse the transformation, which leads to the leakage of client data. To avoid such data leakage, we generate a pseudo numerical feature $\mathbf{X}_m^p$ to accompany the real feature $\mathbf{X}_m$. In particular, the client randomly generates each value in $\mathbf{X}_m^p$ using normal distribution $\mathcal{N}(\mu_{\mathbf{X}_m}, \sigma^2_{\mathbf{X}_m})$, where $\mu_{\mathbf{X}_m}$ and $\sigma^2_{\mathbf{X}_m}$ are the empirical mean and variance of $\mathbf{X}_m$. The client then concatenates the real and pseudo feature, and projects them with a Gaussian matrix $\mathbf{A}_m \in \mathbb{R}^{2 \times 2}$, i.e., $\tilde{\mathbf{X}}_m = [\mathbf{X}_m, \mathbf{X}_m^p] \times \mathbf{A}_m$. Figure 2(b) provides an example for this case. The idea is that the pseudo feature $\mathbf{X}_m^p$ works as 'interference' that prevents the server from decoding the original feature. This does not harm model accuracy if training can learn to reverse the projection $\mathbf{A}_m$ implicitly and assign a weight of 0 for the pseudo feature in the model parameter.

Categorical feature. Categorical features are common in practice, e.g., for object colors, product categories, and geographic regions. Categorical features are usually encoded using letters from an alphabet (e.g., R, G, B for colors) and thus cannot be processed with numerical projection. Thus, we map each categorical feature to 1-hot representation before applying Gaussian projection. In particular,

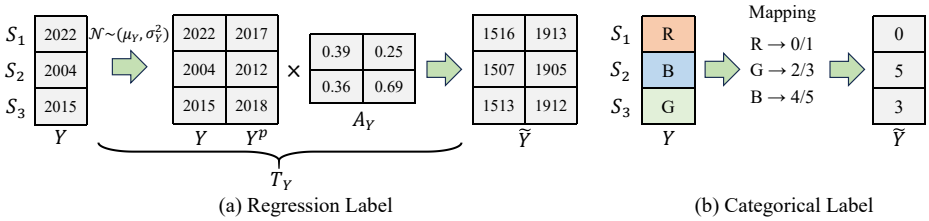


Fig. 3. Illustrations for the transformation for labels.

a feature with K categories is mapped to a numerical vector with length K, and category k sets the k^{th} entry of the vector as 1. Figure 2(c) provides an example that involves one categorical feature with 3 categories and one numerical feature. After mapping the categorical features, $\mathbf{X}_m$ has 4 columns, and thus the projection matrix is $\mathbf{A}_m \in \mathbb{R}^{4\times4}$. The transformed feature $\tilde{\mathbf{X}}_m$ has more columns than the original feature.

Numerical label. In regression tasks, the label owner holds a single numerical value of each data sample for the model to predict, which we call numerical label. Like the case of a single numerical feature, the label owner cannot directly transform the numerical label $\mathbf{Y}$ by Gaussian projection as this may leak data. Similarly, we generate a pseudo label $\mathbf{Y}^p$ using normal distribution $\mathcal{N}(\mu_Y, \sigma_Y^2)$, where μ_Y and σ_Y^2 are the empirical mean and variance of the real numerical label $\mathbf{Y}$. Then, $\mathbf{Y}^p$ is concatenated with $\mathbf{Y}$ and projected by a Gaussian matrix $\mathbf{A}_Y \in \mathbb{R}^{2\times2}$ as $\tilde{\mathbf{Y}} = [\mathbf{Y}, \mathbf{Y}^p] \times \mathbf{A}_Y$, which is sent to the server for model training. Figure 3 (a) provides an example of such a case. Due to the transformation, model training and serving are different from SplitNN. First, instead of training a single model, RAP trains two models with identical configurations, one using the first column of $\tilde{\mathbf{Y}}$ as numerical label, and the other using the second column of $\tilde{\mathbf{Y}}$ as numerical label. This increases the cost of model training but we will show that this cost is marginal compared to the client-server communication cost of SplitNN. Second, for model serving, the server uses the two models to predict $\tilde{\mathbf{y}} \in \mathbb{R}^{1\times2}$ for each sample. With $\tilde{\mathbf{y}}$, the label owner obtains the numerical label as the first column of $\tilde{\mathbf{y}} \times \mathbf{A}_Y^{-1}$, where $\mathbf{A}_Y^{-1}$ is the inverse matrix of the label transformation matrix $\mathbf{A}_Y$. The idea is that if the server-side models can predict $\tilde{\mathbf{y}}$ accurately, then the first column of $\tilde{\mathbf{y}} \times \mathbf{A}_Y^{-1}$ will approximate the true numerical label according to our transformation.

Categorical label. For classification tasks, the label is the category of the data samples. We do not adopt the 1-hot representation technique for categorical features to process categorical labels. This is because it yields numerical labels and requires the server to train regression models. To prevent data leakage, we randomly map the C categories of the label $\mathbf{Y}$ to $2C$ categories before sending them to the server. In particular, denote the original category set as $\mathcal{C} = \{c_1, c_2, \cdots, c_K\}$, the new category set becomes $\mathcal{C}' = \{c'_1, c'_2, \cdots, c'_{2K}\}$, each category $c_k \in \mathcal{C}$ is mapped to two categories $c'_i, c'_j \in \mathcal{C}'$, and the mapped categories

of any pair $c_{k_1}, c_{k_2} \in \mathcal{C}$ do not overlap. The label owner records a mapping from $c_k \in \mathcal{C}$ to $c'_i, c'_j \in \mathcal{C}'$; and to transform each entry $\mathbf{y} \in \mathbf{Y}$, the label owner first looks up the category of $\mathbf{y}$ in $\mathcal{C}$ as c_k, and then maps y to $c'_i, c'_j \in \mathcal{C}'$ with equal probability. Figure 3(b) shows an example of such case. R is mapped to 0 and 1 of the new classes, G is mapped to 2 and 3, and B is mapped to 4 and 5. For model training on the server, the model classifies the data samples into $2K$ classes instead of K classes. This requires a wider output layer for the model and more computation for model training but we note that such cost is significantly smaller than the client-server communication cost of SplitNN. For model serving, the server obtains the mapped class $\tilde{\mathbf{y}}$ of the sample, and the label owner determines the original class $\mathbf{y}$ for $\tilde{\mathbf{y}}$ according to its local category mapping. The rationale is that if the server-side model can predict the mapped category accurately, the label owner can construct the original category.

3 Discussions

In this part, we analyze RAP's properties, including its guarantee for data privacy, comparison with SplitNN, alternative applications, and possible limitations.

Privacy protection. We consider a threat model where the server and all clients are honest-but-curious, which is widely adopted in federated learning [31, 7, 27]. In RAP, the clients have no direct communication with each other, and they all communicate with the server. Thus, there is no risk of leaking data from one client to the other clients but the server may infer the data of the clients. Therefore, we analyze server's ability to decode client features. We consider the following server decoding model. The client feature is $\mathbf{X}_m \in \mathbb{R}^{N \times F_m}$; the server has the transformed feature $\tilde{\mathbf{X}}_m = \mathbf{X}_m \times \mathbf{A}_m$, and attempts to estimate $\mathbf{X}_m$ via $\bar{\mathbf{X}}_m = \tilde{\mathbf{X}}_m \times \mathbf{R}$, where $\mathbf{R}$ is a reconstruction matrix used by the server. In particular, when $\mathbf{R} = \mathbf{A}_m^{-1}$, the server can decode client feature without error. We assume that the server may have some information θ about $\mathbf{X}_m$ and chooses the reconstruction matrix by minimizing the difference between θ and the information computed from the reconstructed feature $\bar{\mathbf{X}}_m$. That is,

$$\arg \min_{\mathbf{R} \in \mathbb{R}^{F_m \times F_m}} \mathcal{U}(\theta, g(\tilde{\mathbf{X}}_m \times \mathbf{R})), \tag{3}$$

where $g(\cdot)$ is the function to compute information θ from the reconstructed feature, and function $\mathcal{U}(\theta, \theta')$ measures how much θ and θ' differ. We state the results of different cases below:

- **Case 1**: The server lacks information about the client feature. It can only randomly guess a projection matrix, and thus cannot reconstruct the features.
- **Case 2**: The server knows the distribution of the client feature. In particular, if the server knows the mean of the features and obtains $\mathbf{R}$ by mean squared error (MSE, for function $\mathcal{U}$), it cannot reconstruct the client feature as there are an infinite number of solutions for $\mathbf{R}$.

– **Case 3**: The server knows the features of n data samples and obtains $\mathbf{R}$ by MSE. (a) If $n < F_m$, there are an infinite number of solutions for $\mathbf{R}$. (b) If $n \geq F_m$ and no client colludes with the server, it cannot solve $\mathbf{R}$. However, a unique solution for $\mathbf{R}$ can be found if the server colludes with a client to acquire the ID of the exposed samples.

Thus, RAP only leaks data privacy when the server knows the features of $n \geq F_m$ data samples, and a client colludes with the server. We note that such a case is considered challenging for federated learning [25], and SplitNN will also leak data. In particular, SplitNN uses function f_b to transform client features, and the server receives activations $\tilde{\mathbf{X}}_m = f_b(\mathbf{X}_m)$. With collusion in case 3(b), the server can collect samples of $(\tilde{\mathbf{X}}_m, \mathbf{X}_m)$ and train a model to predict $\mathbf{X}_m$ using $\tilde{\mathbf{X}}_m$. To address this issue, the clients can jointly run the secure multi-party shuffling protocol [9, 28], so that no clients can tell the exact ordering after the shuffling. Thus, the ID information is hidden from the server even though it may collude with some clients. Although the analysis focuses on a single client, it extends to multiple clients since they all apply the same transformation procedure, which similarly applies to labels due to their feature transformation similarity.

Communication cost. SplitNN's communication volume is $O(E \times F \times N)$ and RAP's is $O(F \times N)$, where E is the number of training epochs, F is the feature dimension, and N is the data sample count. RAP has a much smaller communication cost than SplitNN because training can run many epochs (i.e., large E) and we observe $100\times$ communication volume reduction in the experiments.

Other applications. Corollary 2 in § 2.2 shows that Gaussian random projection preserves the distance between data samples with high probability. Given this property, RAP can be extended to applications that rely on the distance between data samples while preserving data privacy. In order to identify anomaly samples, existing approaches search for samples within a distance threshold from the target sample and classify the target sample as an anomaly if the number of similar samples is small [8, 1, 5]. With RAP, this procedure can be applied directly to the projected samples on the server. Distance-based clustering such as K-means is widely used to organize similar samples into a group [30, 17, 20]. RAP allows to conduct K-means on the projected samples. Recently, similarity-based vector search is becoming popular for embeddings generated by machine learning models [11, 26, 13]. We can compute vector similarity such as Euclidean distance, inner product, and angular similarity on the projected samples.

Limitations. As discussed earlier, RAP increases server computation by requiring two models for regression and doubling the categories for classification. When server resource is limited, RAP can transfer the transformed features to the server and exchange predicted labels/gradients with the label owner like SplitNN. Besides, RAP is designed for multivariate data, which is common in vertical federated learning. Complex data types like images, with multiple channels and similar

Table 2. Statistics of the datasets, first 3 for classification, last 3 for regression, and CO has categorical features.

Dataset	SU	HI	CO	YP	HP	WO
# samples	1.0M	1.0M	0.6M	0.5M	0.5M	0.05M
# features	18	28	54	90	11	20
# classes	2	2	7	/	/	/

adjacent pixels, require special operations like convolution. Flattening images for random projection may affect their semantics and convolution operations.

Relation to existing works. Many researchers aim to improve VFL efficiency. Yang cuts communication cost using the quasi-Newton method [36], while VAFL enhances training efficiency with asynchronous stochastic gradient descent [10]. However, these methods are model-specific. SplitNN, a general VFL framework, splits the model between clients and the server to protect data privacy [18, 33], but suffers from frequent client-server communication. RAP addresses this by reducing communication to a single shot while preserving accuracy and privacy. Other optimizations include Fu's local updates and cached statistics [16], Jiang's complementary client selection [21], and Feng's VFLFS for feature selection [14]. Li's FedSDG-FS applies a Gaussian stochastic dual-gate for feature selection [24], while V-coreset reduces training samples via coreset selection [19]. RAP is orthogonal to these techniques, as it can be applied afterward.

4 Experimental Evaluation

In this part, we evaluate our proposed RAP framework for both regression and classification tasks and on datasets with diverse properties. We try to answer the following questions:

- Q1: Can RAP maintain *good model accuracy* and achieve *short training time* for VFL compared with baselines?
- Q2: How robust is RAP to diffident VFL configurations, e.g., the number of clients, ML model structures, feature types and distributions?

4.1 Experiment Settings

Datasets and models. Table 2 reports the statistics of the datasets used in our experiments. Among them, SU [35] and HI [34] are for binary classification, and we randomly select 1 million samples from their original datasets; CO [6] has seven distinct categories, and 44 out of its 54 features are categorical; YP [3], HP [32], and WO [4] are for regression. For each dataset, we group the samples into a training set (70%), a validation set (10%), and a test set (20%). By default, we first permute the feature dimensions and then assign them to the clients via

Table 3. Accuracy comparison for regression and classification tasks with MLP model. (RMSE and Accuracy). We highlight the top-2 results for each dataset.

	RMSE ↓			Acc(%) ↑		
	YP	HP	WO	SU	HI	CO
Centralized	**8.83**±0.23	**0.08**±0.02	**98.27**±0.15	**80.36**±0.25	**74.12**±0.13	**87.67**±0.11
Single	9.61±0.23	75.24±0.27	208.93±0.19	78.54±0.05	65.81±0.29	70.59±0.21
SplitNN	8.91±0.24	0.11±0.03	98.95±0.21	80.30±0.19	73.96±0.23	87.62±0.24
RAP	**8.87**±0.18	**0.10**±0.2	**98.42**±0.12	**80.33**±0.08	**74.10**±0.15	**87.65**±0.19

Table 4. Training time comparison with MLP model. The numbers are in seconds.

	YP	HP	WO	SU	HI	CO
Centralized	75±3	94±2	11±3	107±2	188±1	111±1
Single	65±1	82±1	9±2	87±1	146±1	97±3
SplitNN(10^4)	5.6±0.6	5.0±0.6	0.7±0.1	8.0±0.6	8.5±0.8	5.5±0.5
RAP	90±2	107±6	11±1	124±1	255±1	128±3

round-robin to ensure that different clients have a similar number of features. We mainly use a multi-layer perceptron (MLP) as the model to train in our experiments. The MLP has a single hidden layer with a default dimension of 128. We also experiment with logistic regression (LR) to assess the adaptability of RAP to different machine learning models.

Baselines. We compare our RAP with the three alternatives listed below.

- **Centralized** assumes that all data features are on the server, and the server trains model on its local data. It serves as a ideal case with good model accuracy and short training time and a reference.
- **Single** assumes that each client conducts independent model training using its local features. We report the time and accuracy for the client that exhibits the highest accuracy on the test set. It can validate the benefit of coordinating the clients to learn a good model jointly via VFL.
- **SplitNN** segments the model into a bottom part on clients and a top part on server; client-server communication is required for activation/gradient in every mini-batch as discussed earlier.

We exclude homomorphic encryption (HE) and local differential privacy (LDP) because Table 1 shows that they have either longer running time or poorer model accuracy compared with RAP. Besides, HE requires a trusted execution environment (TEE) on server, which incurs additional cost.

Evaluation protocol. By default, we use four clients and one label owner. We adopt the Adam [22] optimizer for both regression and classification tasks and

conduct a grid search for the optimal learning rate within $\{3, 2, 1, 0.1, 0.01, 0.001\}$ in each experiment (i.e., dataset, model, and VFL method). We also tune the batch size from 0.1% to 1% of the training samples for all datasets to optimize model accuracy. We continue model training until the relative change in the loss is below a threshold (set as $1e-4$) for five epochs. For regression tasks, we use the Root-Mean-Square-Error (RMSE) to evaluate model accuracy and note that *smaller value* indicates better accuracy. For classification tasks, we use classification accuracy, and *higher value* is better.

Implementation. We run VFL on a cluster, where each machine has a Intel-i9 CPU and 24GB memory, and the machines are connected via 10GBps Ethernet. The machines use gRPC with the proto3 library for communication, and PyTorch for model training. Each machine serves as a client, and one machine serves as the server. Note that this implementation may underestimate communication costs (and thus favors SplitNN) because client-server communication may go through public network in practice, which is slower than our inter-cluster connection.

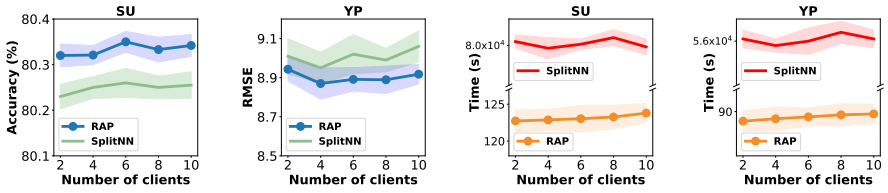
4.2 Experiment Results

Table 3 compares the model accuracy of RAP with the baselines, and Table 4 reports the model training time.

Model accuracy. Table 3 shows that the model accuracy of RAP matches the ideal centralized training. For regression tasks, the RMSE scores of RAP and Centralized differ by less than 0.15; for classification tasks, the accuracy of RAP and Centralized differ by at most 0.03%. These differences are nearly negligible, indicating that RAP achieves good model accuracy. SplitNN's accuracy is similar to RAP and Centralized for both classification and regression. In contrast, Single has much larger RMSE for regression tasks and lower accuracy for classification tasks. This is because it conducts learning independently on each client and suggests that VFL is necessary to learn high-quality models.

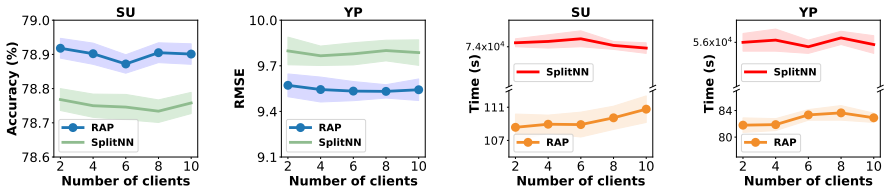
Training time. Table 4 shows that the training time of RAP is only slightly longer than ideal centralized training, and the blowup is usually around 10-20%. This is mainly due to the costs of transforming the data by Gaussian projection and sending the transformed data to the server. Single has the shortest training time among the methods because it works with fewer features on each individual client. Compared with SplitNN, RAP has significantly shorter model training time. The speedup of RAP over SplitNN is usually two orders of magnitude and reaches 630× for regression task on the WO dataset. This is because RAP conducts only one round of client-server communication while SplitNN involves many rounds of client-server communication.

We choose two representative datasets, i.e., YP for regression task and SU for classification task, to conduct further investigations. We check how RAP performs when changing the number of clients and the clients that have only one



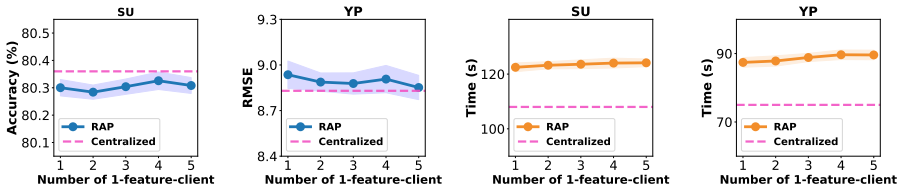
(a) SU accuracy. (b) YP RMSE. (c) SU time. (d) YP time.

Fig. 4. Scalability of RAP and SplitNN on MLP model.



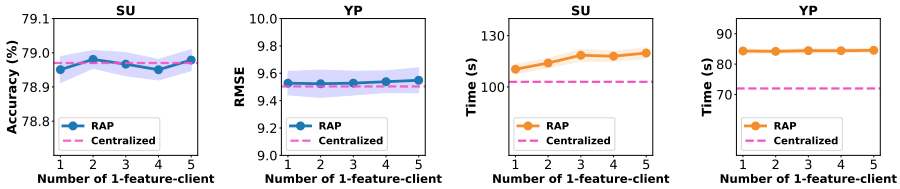
(a) SU accuracy. (b) YP RMSE. (c) SU time. (d) YP time.

Fig. 5. Scalability of RAP and SplitNN on LR model.



(a) SU accuracy. (b) YP RMSE. (c) SU time. (d) YP time.

Fig. 6. Sensitivity study on MLP model by varying the number of one-feature-client.



(a) SU accuracy. (b) YP RMSE. (c) SU time. (d) YP time.

Fig. 7. Sensitivity study on LR model by varying the number of one-feature-client.

feature. We also experiment with another model, i.e., logistic regression (LR), to assess the adaptability of RAP to different machine learning models.

Number of clients. As depicted in Figures 4 and 5, we vary the number of clients from 2 to 10 and compare RAP with SplitNN. RAP has stable model

accuracy and running time when changing the number of clients that participate in VFL. The accuracy is quite consistent, i.e., the deviations on the SU dataset are around 0.02% and the variations on the YP dataset are smaller than 0.2. Similar to the case of model accuracy, the training time of RAP also does not change much with the number of clients. Moreover, we also observe that RAP consistently runs much faster then SplitNN for different number of clients. The good scalability of RAP makes it suitable for cases with many clients for VFL.

Number of one-feature-clients. In Section § 2, we discuss how RAP can handle clients that hold only one feature and call such client *one-feature-client* here. In particular, for a one-feature-client, we generate a pseudo-feature and integrate it with the original features to confuse the server. As shown in Figure 6 and Figure 7, we adjust the number of one-feature-clients in our experiments, ranging from 1 to 5, while the total number of clients remains at 10. For model accuracy, we observe that, when increasing the number of one-feature-clients, RAP consistently maintains similar performance compared to ideal centralized training. Regarding the training time, using more one-feature-clients only introduces a modest increase in the training time, primarily due to an increase in the number of model parameters caused by the increased feature dimension. These results suggest that RAP is robust to the number of one-feature-clients.

Model compatibility. To evaluate RAP's robustness and compatibility across machine learning models, we extended our experiments beyond MLP to include logistic regression (LR) with varying client counts and one-feature clients. Figure 5 and 7 show RAP's consistent performance on LR, similar to MLP. With different client numbers, RAP maintains stable accuracy and RMSE on SU and YP datasets, along with consistent training times. For one-feature clients, RAP remains robust, with SU's accuracy fluctuating only by 0.1% and YP's RMSE changes under 0.1. Notably, in both scenarios, there are marginal and stable increases regarding training time. According to the above experiments, RAP can perform well across various machine learning models.

5 Conclusion

In this paper, we propose a privacy-preserving and communication-efficient vertical federated learning (VFL) framework, termed RAP. In particular, RAP uses Gaussian random matrix to project the features on each client and sends the transformed data to the server to conduct model training without communicating with the clients. RAP requires only one round of client-server communication and thus is much more efficient than the popular SplitNN framework for VFL, which conducts client-server communication for each mini-batch of training. To make RAP general, we carefully design data transformations for special cases that may arise in practice. Moreover, we also analyze the privacy guarantee of RAP and discuss its extensions and applications. Extensive experiments show that RAP can speed up SplitNN by up to 630× while maintaining model accuracy and being robust to different model and data configurations.

6 Acknowledgments

This work was sponsored by National Natural Science Foundation of China (62472327) and Key R&D Program of Hubei Province (2023BAB077). This work was supported by Ant Group through CCF-Ant Research Fund (CCF-AFSG RF20240104).

References

1. Ahmed, M., Mahmood, A.N., Hu, J.: A survey of network anomaly detection techniques. Journal of Network and Computer Applications **60**, 19–31 (2016)
2. Arachchige, P.C.M., Bertok, P., Khalil, I., Liu, D., Camtepe, S., Atiquzzaman, M.: Local differential privacy for deep learning. IEEE Internet Things J **7**(7), 5827–5842 (2019)
3. Bertin-Mahieux, T.: YearPredictionMSD. UCI Machine Learning Repository (2011), DOI: https://doi.org/10.24432/C50K61
4. Bhat, V.K.: woc dataset. Kaggle Big Data Competition Platform (2023), https://www.kaggle.com/datasets/vishakkbhat/woc-data-set/data
5. Bhuyan, M.H., Bhattacharyya, D.K., Kalita, J.K.: Network anomaly detection: methods, systems and tools. IEEE Commun Surv Tut **16**(1), 303–336 (2013)
6. Blackard, J.: Covertype. UCI Machine Learning Repository (1998), DOI: https://doi.org/10.24432/C50K5N
7. Bonawitz, K., Ivanov, V., Kreuter, B., Marcedone, A., McMahan, H.B., Patel, S., Ramage, D., Segal, A., Seth, K.: Practical secure aggregation for privacy-preserving machine learning. In: SIGSAC. pp. 1175–1191 (2017)
8. Chandola, V., Banerjee, A., Kumar, V.: Anomaly detection: A survey. ACM Computing Surveys **41**(3), 1–58 (2009)
9. Chase, M., Ghosh, E., Poburinnaya, O.: Secret-shared shuffle. In: ASIACRYPT. pp. 342–372. Springer (2020)
10. Chen, T., Jin, X., Sun, Y., Yin, W.: Vafl: a method of vertical asynchronous federated learning. arXiv preprint arXiv:2007.06081 (2020)
11. Cunningham, P., Delany, S.J.: k-nearest neighbour classifiers-a tutorial. ACM Computing Surveys **54**(6), 1–25 (2021)
12. Dasgupta, S., Gupta, A.: An elementary proof of a theorem of johnson and lindenstrauss. Random Structures & Algorithms **22**(1), 60–65 (2003)
13. Deerwester, S., Dumais, S.T., Furnas, G.W., Landauer, T.K., Harshman, R.: Indexing by latent semantic analysis. JASIST **41**(6), 391–407 (1990)
14. Feng, S.: Vertical federated learning-based feature selection with non-overlapping sample utilization. Expert Systems with Applications **208**, 118097 (2022)
15. Feng, X., Zhang, Z.: The rank of a random matrix. Applied Mathematics and Computation **185**(1), 689–694 (2007)
16. Fu, F., Miao, X., Jiang, J., Xue, H., Cui, B.: Towards communication-efficient vertical federated learning training via cache-enabled local updates. arXiv preprint arXiv:2207.14628 (2022)
17. Gan, G., Ma, C., Wu, J.: Data clustering: theory, algorithms, and applications. SIAM (2020)
18. Gupta, O., Raskar, R.: Distributed learning of deep neural network over multiple agents. Journal of Network and Computer Applications **116**, 1–8 (2018)

19. Huang, L., Li, Z., Sun, J., Zhao, H.: Coresets for vertical federated learning: Regularized linear regression and k-means clustering. NeurIPS (2022)
20. Jain, A.K.: Data clustering: 50 years beyond k-means. Pattern Recognition Letters **31**(8), 651–666 (2010)
21. Jiang, J., Burkhalter, L., Fu, F., Ding, B., Du, B., Hithnawi, A., Li, B., Zhang, C.: Vf-ps: How to select important participants in vertical federated learning, efficiently and securely? NeurIPS (2022)
22. Kingma, D.P., Ba, J.: Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980 (2014)
23. Konečnỳ, J., McMahan, H.B., Yu, F.X., Richtárik, P., Suresh, A.T., Bacon, D.: Federated learning: Strategies for improving communication efficiency. arXiv preprint arXiv:1610.05492 (2016)
24. Li, A., Peng, H., Zhang, L., Huang, J., Guo, Q., Yu, H., Liu, Y.: Fedsdg-fs: Efficient and secure feature selection for vertical federated learning. arXiv preprint arXiv:2302.10417 (2023)
25. Li, T., Sahu, A.K., Talwalkar, A., Smith, V.: Federated learning: Challenges, methods, and future directions. IEEE Signal Processing Magazine pp. 50–60 (2020)
26. Li, W., Zhang, Y., Sun, Y., Wang, W., Li, M., Zhang, W., Lin, X.: Approximate nearest neighbor search on high dimensional data—experiments, analyses, and improvement. IEEE TKDE **32**(8), 1475–1488 (2019)
27. Ma, C., Li, J., Ding, M., Yang, H.H., Shu, F., Quek, T.Q., Poor, H.V.: On safeguarding privacy and security in the framework of federated learning. IEEE Network **34**(4), 242–248 (2020)
28. Movahedi, M., Saia, J., Zamani, M.: Shuffle to baffle: Towards scalable protocols for secure multi-party shuffling. In: 2015 IEEE 35th International Conference on Distributed Computing Systems. pp. 800–801. IEEE (2015)
29. Sabt, M., Achemlal, M., Bouabdallah, A.: Trusted execution environment: What it is, and what it is not. In: IEEE Trustcom/BigDataSE/Ispa. vol. 1, pp. 57–64 (2015)
30. Shirkhorshidi, A.S., Aghabozorgi, S., Wah, T.Y., Herawan, T.: Big data clustering: a review. In: ICCSA. pp. 707–720. Springer (2014)
31. Shokri, R., Shmatikov, V.: Privacy-preserving deep learning. In: Proceedings of the 22nd ACM CCS. pp. 1310–1321 (2015)
32. Sleem, A.: HousePricing. Kaggle Big Data Competition Platform (2018), https://www.kaggle.com/datasets/greenwing1985/housepricing/data
33. Vepakomma, P., Gupta, O., Swedish, T., Raskar, R.: Split learning for health: Distributed deep learning without sharing raw patient data. arXiv preprint arXiv:1812.00564 (2018)
34. Whiteson, D.: HIGGS. UCI Machine Learning Repository (2014), DOI: https://doi.org/10.24432/C5V312
35. Whiteson, D.: SUSY. UCI Machine Learning Repository (2014), DOI: https://doi.org/10.24432/C54606
36. Yang, K., Fan, T., Chen, T., Shi, Y., Yang, Q.: A quasi-newton method based vertical federated learning framework for logistic regression. arXiv preprint arXiv:1912.00513 (2019)
37. Zhang, Q., Yan, X., Ding, Y., Xu, Q., Hu, C., Zhou, X., Jiang, J.: Treecss: An efficient framework for vertical federated learning. In: International Conference on Database Systems for Advanced Applications. pp. 425–441. Springer (2024)
38. Zhou, X., Yan, X., Li, X., Huang, H., Xu, Q., Zhang, Q., Jerome, Y., Cai, Z., Jiang, J.: vfdv-im: An efficient and securely vertical federated data valuation. In: International Conference on Database Systems for Advanced Applications. pp. 409–424. Springer (2024)