InC: A Vertical Federated Learning Framework with Multiple Noisy Labels

Ruixuan Zhang[1], Ziyi Li[1], Xiao Yan[2], Xiaokai Zhou[1], Hao Wang[1]✉, Hao Huang[1], and Jiawei Jiang[1]✉

[1] School of Computer Science, Wuhan University
`{zrx123, lzy0323, xiaokaizhou, wanghao.cs, haohuang, jiawei.jiang}@whu.edu.cn`

[2] Centre for Perceptual and Interactive Intelligence (CPII)
`yanxiaosunny@gmail.com`

Abstract. Vertical Federated Learning (VFL) enables collaborative model training in a privacy-preserving way when data features are distributed across different parties. Typical VFL systems assume only one party holds the training labels. However, in real-world scenarios, each party may annotate its local data independently, resulting in significant label noise due to incomplete feature spaces and varying annotation quality. To address this challenge, we propose Initialization Correction (InC), a VFL framework designed to handle multiple noisy labels. In the first stage, we train a consensus classifier by converting party labels into probability distributions to enhance noise tolerance. In the second stage, a temporary corrected label is inferred and iteratively refined through an EM process. The expertise matrix of each party is evaluated and used to weight the classifier predictions, which serve as the corrected label. Throughout the entire process, we leverage techniques such as Homomorphic Encryption (HE) to protect the original labels' privacy. Extensive experiments on public datasets demonstrate that InC significantly outperforms the baselines across various settings, and the results are comparable to training with clean labels.

Keywords: Vertical federated learning · Split learning · Noisy label.

1 Introduction

Vertical Federated Learning. Federated Learning (FL) is a collaborative learning paradigm that allows different data sources to jointly train models without sharing private data [20]. Vertical Federated Learning (VFL) is one category of FL scenarios, in which the federated parties hold the same set of data instances, but have different feature spaces [34, 22, 41, 39, 16]. VFL incorporates various feature information from collaborative parties or organizations, thereby improving the quality of machine learning models.

✉ Corresponding author

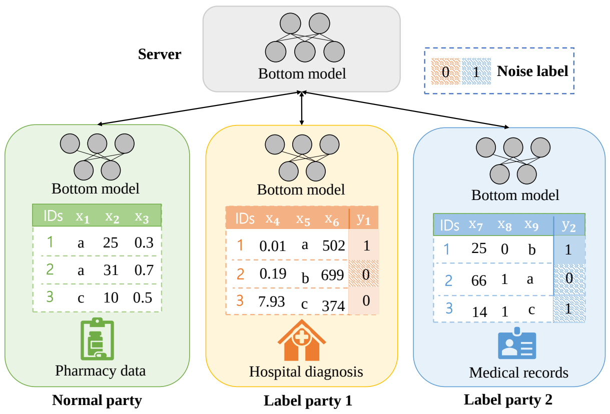

Fig. 1. The VFL-MNL example. For one instance, each federal party holds a partial set of its features. Multiple parties have instance labels which can be noisy. Under the split learning framework, each party holds a bottom model and jointly trains a top model.

VFL with Multiple Labels. In the typical VFL setting, multiple parties hold vertically partitioned features, but only one federated party has the training labels [34]. In the meantime, it assumes that these labels are correct and clean. Nevertheless, this setting is often unrealistic in the real world. For example, multiple financial institutions, e.g., banks, insurance companies, and loan platforms, may hold diverse features of the same set of customers. In this typical VFL setting, the involved parties may want to collaboratively estimate credit scores for the purpose of risk management [6]. Since each party has its own business, it can independently annotate its customers according to the local features, yielding a VFL scenario with multiple labels. Unlike the conventional VFL setting, the training system can obtain multiple labels for the same data instance.
Challenges. In the presence of multiple labels, the training of vertical federated learning faces the following challenge:

> *Multiple parties perform annotation independently according to their own local features; therefore, the labels are not entirely trustworthy and inevitably contain noises.*

Figure 1 shows an example of VFL in the medical field, where the VFL system combines the electronic health records of patients from multiple medical institutions to diagnose the type of disease [14, 25, 27]. Each party is an independent medical institution and may conduct a diagnosis based on its local data before the VFL training. However, these parties can be significantly different regarding the data in patient medical records (i.e., local feature space), such as the period of diagnosis, the equipment used to collect disease characteristics, and the types of examinations. Therefore, the diagnosis and treatment for the same patient may vary among these medical institutions. At the same time, due to the complexity of disease and subjectivity in diagnosis, the diagnostic results may inevitably have errors, causing the multiple labels in VFL inconsistent and noisy. Motivated by this VFL-MNL (Vertical Federated Learning with Multiple Noisy Labels) issue, in this work we try to address the following problem:

How to train a VFL model in the presence of multiple noisy labels, while safeguarding the privacy of local data?

Federated Noisy Label Learning (FNLL). In the prior federated learning literature, several works have studied federated noisy label learning (FNLL) to solve the problem of noisy labels in the FL training process [18]. However, these methods focus on the Horizontal Federation Learning (HFL) scenario [11, 5, 33, 37, 35], and cannot be directly applied to VFL tasks due to differences in data distribution and training process. One type of these methods typically requires a completely clean dataset on the server. For example, RoFL [38] and FOCUS [5] train a noise-tolerant model by exchanging information with the benchmark dataset. Another type is based on sampling or correction. FedNoRo [35], FedNoil [33] and FedCorr [37] propose a two-stage scheme, which firstly selects noisy clients and then performs noise-robust training or noise correction process.
FNLL in VFL. In conventional VFL, most studies are based on the SplitNN architecture [4], which divides neural networks into multiple parts and lets each party train a local model (a.k.a. bottom model) over its local features. Assuming only one party has the labels, the SplitNN framework establishes a server, which aggregates the output of bottom models on the parties, performs forward computation over a top model, and sends the output to the party holding labels. The label party then calculates the loss and sends it back to the server for back-propagation. As can be seen, the training process of VFL is much more complex than the typical training paradigm of HFL, in which each party can locally perform forward computation and back-propagation. Under this training scheme, the data protection in VFL training also encounters difficulties. The previous works [4, 26, 2, 13, 12, 1] generally focus on protecting features by transferring intermediate gradients, but overlook the protection of labels. When multiple parties in VFL have labels, it is critical to carefully design approaches for FNLL under the split learning framework, while protecting the data labels.
Contributions. To address the above challenges, we propose a two-stage VFL training process called Initialization Correction (InC). In the first stage, we train an initialized consensus classifier using the soft labels generated from all available label parties. We encrypt the labels of each client by Homomorphic Encryption (HE), and the server sums the encrypted values to obtain the probability distribution as soft labels. In the second stage, we correct noises in the labels using the classifier obtained in the first step through an EM-style method. Specifically, in the E-step, we infer a temporary pseudo-label in each iteration by combining the prediction probability of the consensus classifier with the expertise of each client. The assessment of the expertise is inspired by the confusion matrix, which describes the probability distribution of noise. In the M-step, the expertise matrices will be updated based on the prediction of the consensus classifier. Overall, we continuously correct noisy labels through the EM strategy. The key contributions of this work are summarized as follows:

- We propose a two-stage training method, called Initialization Correction (InC), for vertical federated learning in the presence of multiple noisy labels.
- In the first stage, we train an initialization consensus classifier by converting client labels into probability distributions, using Homomorphic Encryption (HE) to protect data privacy.
- In the second stage, we propose an EM-style algorithm that dynamically corrects the noises in the labels. We assess the expertise of each client by the confusion matrix of labeling inconsistency and combine it with the classifier predictions to generate temporary pseudo-labels.
- We conduct extensive experiments over various workloads and datasets. The experimental results demonstrate that, compared to the baselines, our proposed InC can effectively correct noisy labels and significantly improve the model accuracy by up to 10.84%.

2 Related Work

Vertical Federated Learning. Existing studies of VFL are mainly based on SplitNN, which is a popular multi-party VFL architecture using split neural networks [4]. Split Learning is applied in various domains, such as healthcare and the Internet of Things (IoT). SplitNN [31] integrates different modalities of patient information and avoids sharing original data. Duan et al. [9] mention that SplitNN trains a global model collaboratively without directly accessing decentralized raw data, which is effective for applications that benefit from the data generated in IoT environments. PyVertical [26] introduces a dual-headed scenario and aligns data from two parties and a data scientist, who holds labels.
VFL with Multiple Label Parties. In VFL studies, most studies explore the issue of features, with a few involving labels. Multi-VFL [21] considered a scenario where the sample labels are distributed over multiple parties [29]. It uses a third-party server to aggregate bottom models from different label parties by federated averaging (FedAvg) [17]. CVFL [36] raises the issue of parties having only partial labels and explores the privacy protection of labels in VFL. HeteroVFL [40] addresses the complexity of data distribution and regional heterogeneity. Different from the above VFL label studies, which handle non-IID label owners [21], partial labels [36] and label distribution differences [40], our work focuses on the scenario of multiple noisy labels in VFL.
Federated Noisy Label Learning (FNLL). Federated Noisy Label Learning (FNLL) was proposed to solve the noisy label problem in FL training [18]. Existing studies on FNLL have mostly focused on HFL and can be roughly divided into two types: selection-free and selection-based. In selection-free method, a noise-tolerant loss function or regularization is usually designed to offset the harm of noise to the model [11, 38, 5]. RoFL [38] reduces the problem of inconsistent decision boundaries caused by noise by exchanging class centroids. FOCUS [5] uses mutual cross-entropy to quantify the credibility of the clients. These methods usually require a completely clean dataset on the server, which is unrealistic. Selection-based method selects the most likely clean clients and samples the

clients [33, 37, 35]. FedNoRo [35], FedNoil [33] and FedCorr [37] all propose a two-stage scheme, which firstly selects noisy clients by certain indicators (i.e., per-class loss indicators, confidence scores and the dimensionalities of the model prediction subspaces) and then designs noise-robust training or noise correction methods.

Comparing to Our Method. Due to the huge differences between HFL and VFL, the aforementioned methods cannot be directly applied to the VFL scenario with multiple noisy labels. Unlike the previous works that train multiple classifiers on the parties, VFL needs to let the parties collaboratively train a classifier [19, 4]. In previous works, multiple label parties without overlapping labels, therefore have no noise. Therefore, the corresponding strategy cannot directly solve VF-MNL.

3 The InC Framework

3.1 Problem Formulation

Data Layout. In our setting, the training dataset D contains N samples with sample identifiers (IDs), and there are m parties. The feature space $X = [X_1 \dots X_i \dots X_m]$ is vertically partitioned among each party $P_i \in P$. P_i holds a subset of features (columns) of X, denoted by X_i. As shown in Figure 1, unlike the traditional VFL scenario where there is only one special party that owns labels, the VFL-MNL scenario has K parties holding labels y_i^k. Let c denote the number of class categories, then the individual label $y_i^k \in \{1, 2, \dots, c\}$. We call them label parties $L_k \in L$, and assume that all label parties labeled all samples, where for i-th sample has labels $y_i^k, k \in \{1, ..., K\}$. In VFL-MNL, all label parties have a certain degree of noise. Due to that, the labels of the i-th sample may be different from each other. We denote the noise rate of each label party as ε^k.

SplitNN Framework. We choose the SplitNN framework since it is a widely adopted standard for VFL [26]. We split the model into local models $\{G_i, i \in \{1, ..., m\}\}$, and a top model F. Each party has its own bottom model, and the server holds the top model. Each local model G_i is parameterized by θ_i and operates only on local data. The top model F is parameterized by θ_f, which cannot be accessed by the parties. Assuming there is only one label party, the training process works as follows:

1. Each bottom model G_i uses its own local features X_i to perform forward propagation, and sends the split layer output $\hat{X}_i$ to the server.
2. The server concatenates the outputs from all parties. After that, the top model F on the server uses the concatenated values to perform forward propagation.
3. The label party receives the prediction results $\hat{y}$ from the top model F. And it computes the loss using its local labels y^k.
4. The top model F on the server performs back propagation using the loss and distributes gradients to all parties.

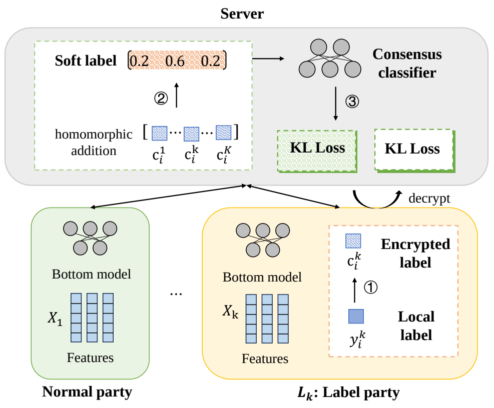

Fig. 2. Initlization Stage: Training Consensus Classifier with Soft Labels

3.2 Framework Overview

To deal VFL-MNL problem, we propose a two-stage training framework InC (InitializationCorrection). In the first Initialization stage, we obtain an initial predictive model; while in the second correction stage, we correct noise gradually.
Initialization Stage. In the first stage, we train an initialized consensus classifier by using all label information. Here we do not choose a specific client because blindly choosing one under high uncertainty will inevitably introduce bias, especially when all label parties are not experts and noisy. Instead, we collect all labels from the label parties and convert the labels into probability distributions (a.k.a., soft labels), which makes the model training more tolerant to noises [30]. Since the labels need to be transferred to the server, it is necessary to protect the labels from any curious party. To achieve this, we use Homomorphic Encryption (HE) to encrypt the local label distribution before transmission. In this manner, the original label value will not be leaked to the server or other involved parties.
Correction Stage. In the second stage, we leverage the consensus classifier in the first stage to infer temporary pseudo labels for data instances and iteratively correct the noisy labels via an EM-style algorithm. Specifically, in the E-step, we combine the prediction probability of the consensus classifier with the expertise of each client to generate the pseudo labels. Our assessment of the expertise is inspired by the *confusion matrix* that measures the inconsistenty between the consensus classifier's prediction and each label party's annotations. Intuitively, this confusion matrix can reflect the distribution of noises. In the M-step, the expertise matrix is updated according to the prediction of the classifier. The EM process is iteratively executed until reaching a stopping criterion.

3.3 Initialization Stage: Consensus Classifier with Soft Labels

Figure 2 shows the initialization stage in which a consensus classifier is trained using soft labels. In the following, we describe the major steps.

❶ **Send Local Labels.** Each label party sends the local label distribution to the server to summarize the global distribution. For example, if one party out of four annotates the sample as positive, the probability of positive is 0.25 and the distribution of labels is $[0.25, 0.75]$. In order to protect data privacy, we use homomorphic encryption (HE) [23] to encrypt the transferred labels. Specifically, each label party L_k first converts its own labels into one-hot encoding (e.g., the local label distribution of each data sample), and then uses HE to encrypt the local distribution into ciphertext $c_i^k = \mathrm{Enc}(pk, y_i^k)$, where pk is the public key. After encryption, the label party sends c_i^k to the server.
❷ **Generate Soft Labels.** Once the server receives all the encrypted label distributions, it uses homomorphic addition to sum them under ciphertext:

$$c_{\mathrm{sum}} = \sum_{k=1}^{K} c_i^k \tag{1}$$

In this way, we obtain the global probability of labels for data samples, which we regard as *soft labels.*
❸ **Train Consensus Classifier.** We run the traditional SplitNN routine over all the federated parties. After the forward propagation of the bottom model, the server concatenates the outputs of the bottom models and executes the forward propagation of the top model. In regular SplitNN, the server sends the predicted probability $\hat{y}$ to the party that holds the labels for computing the training loss. However, in our Inc framework, since the server holds the soft labels, there is no need to send the soft labels to the label party for loss calculation. Instead, the server can directly calculate the loss, using the predictions of the consensus classifier and the soft labels. Here, we choose KL divergence as the loss function:

$$\mathcal{L}_{\mathrm{KL}} = \sum_{i=1}^{N} \sum_{c=1}^{C} p_i(c) \log \frac{p_i(c)}{\hat{y}_i(c)} \tag{2}$$

where $p_i(c)$ denotes the soft label probability distribution of sample i on category c and $\hat{y}_i(c)$ denotes prediction probability of sample i on category c. Since $p_i(k)$ is in encrypted by HE, $\mathcal{L}_{\mathrm{KL}}$ is ciphertext as well. Hence, we send $\mathcal{L}_{\mathrm{KL}}$ to a label party to decrypt the loss. Afterwards, the server performs back propagation on the top model and distributes the gradientto update the bottom models.

3.4 Correction Stage: EM-Based Label Correction

Figure 3 shows the overview of EM-based label correction. To improve the model performance, our idea is to jointly use the classifier output and multiple noisy labels for noisy label correction. We describe the major steps as below:
❶ **Initialize Expertise Matrix.** We initialize an annotation expertise matrix $T^k \in [0, 1]^{C \times C}$ in each label party [7] to model its expertise. The annotation expertise matrix T^k represents the flipping probabilities between all possible classes, using the consensus classifier's predictions as the true labels. Each row of T^k, denoted by T_j^k, represents the probabilities of class j on client k

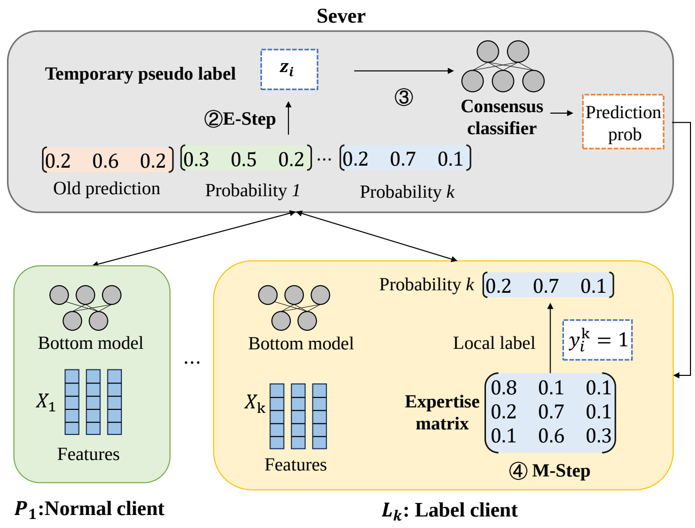

Fig. 3. Correction Stage: EM-Based Label Correction.

being misclassified into other classes. Specifically, T_j^k is a vector of multivariate parameters: $T_j^k = (T_{j,1}^k, \ldots, T_{j,c}^k)$, where each item is defined as:

$$T_{j,l}^k := \Pr[y^k = l \mid Y = j] \tag{3}$$

where y^k is the label from label party k, and Y is the predicted label produced by the consensus classifier. In other words, $T_{j,l}^k$ denotes the probability that the label party k provides a label l to the data instance whose predicted label is j. This means that when the instance is labeled as class l by the label party k, its true class is characterized by the probability distribution of $T_{j,l}^k$. Note that, $\sum_{l=1}^{c} T_{j,l}^k = 1$. In the beginning, we initialize probabilities on the diagonal of the matrix to 1, which is regarded as completely trustworthy. To optimize the expertise matrix in the VFL scenario, we design a maximum likelihood estimation method, including an Expectation (E)-step and a Maximization (M)-step to correct noisy labels and update the expertise matrix.

❷ E-step: Correct Pseudo-Labels and Retrain Consensus Classifier. After the phase of soft label-based training, the consensus classifier has learned some knowledge from the features of the underlying data. Though the labels are noisy, the learned model still has a certain level of classification ability. We use the classifier trained in the first step as the initial state to start the EM algorithm. Motivated by prior EM strategies [7], for a specific data sample i, we combine the prediction $\hat{y}_i$ of the consensus classifier F with the expertise matrix T^k in each label party to correct the original label y^k. Specifically, we calculate the corrected label $\tilde{y}_i$ of data sample i as:

$$\tilde{y}_i = \frac{\hat{y}_i + \sum_{k=1}^{K} T_{y_i^k,\cdot}}{\sum_{c=1}^{C}\left(\hat{y}_i + \sum_{k=1}^{K} T_{y_i^k,c}\right)} \tag{4}$$

where $\hat{y}_i$ is the predicted probability generated by the global model classifier F using the old parameter θ_f in the last training epoch. $T_{y_i^k,\cdot}$ is the expertise matrix probabilities in each label party. For example, if a data is labeled as class j by the label party k, the original label is transformed into $T_{j,\cdot}$. We sum the probability values $T_{y_i^k,\cdot}$ and $\hat{y}_i$, then normalize them to obtain the correction value $\tilde{y}_i$. During the above process, we also leverage homomorphic encryption to protect data privacy. Each label party encrypts the expertise matrix using HE and sends it to the server. The server aggregates the expertise matrices and the predicted probability as the corrected labels, which are ciphertext. Afterward, the server sends the corrected labels to one label party for decryption. In this way, the server can only know the aggregate results.

The server next retrains the consensus classifier using the corrected labels. Here, we choose the class with the maximal probability, instead of the soft labels in the initialization stage. The reason we choose deterministic categories instead of probabilities for training is that the correction stage needs a classifier with stronger judgment capabilities. Compared to the initialization stage, where the entire set of labels are noisy, the corrected labels are relatively closer to the clean labels. Therefore, introducing additional probabilities is unnecessary. Thus, the cross-entropy loss of retraining is formulated as:

$$\ell = -\frac{1}{N}\sum_{i=1}^{N}\sum_{c=1}^{C} \overline{y}_i \log(\hat{y}_{i,c}), \quad \overline{y}_i = \arg\max \tilde{y}_i \tag{5}$$

where $\hat{y}_{i,c}$ denotes the prediction probability of category c by the current classifier F and $\overline{y}_i$ denotes the category with the maximal probability in the corrected label $\tilde{y}_i$. After obtaining the loss, the top model performs back propagation, divides model gradients for back propagation of bottom models.

❸ **M-step: Update Expertise Matrix.** The server sends $\hat{y}$ in the current epoch to the label parties, in order to update the confusion matrix T^k. The following equation describes the M-step:

$$T_{j,l}^{k} = \frac{\sum_{i=1}^{N} q(\hat{y}_i = j) \cdot I(y_i^k = l)}{\sum_{i=1}^{N} q(\hat{y}_i = j)} \tag{6}$$

Among them, $I(y_i^k = l)$ means that, when the original label of sample i from label party k is equal to l, the value is 1, $q(\hat{y}_i = j)$ refers to the probability of j in the model prediction probability $\hat{y}_i$ for sample i. In summary, we sum up the predicted probabilities of all data with the original label l to update the values of the expertise matrix $T_{j,l}^{k}$ in each label party k. Following the above scheme, the probability of the matrix dynamically changes.

Table 1. Evaluated datasets.

Dataset	# instances	# features	# class
Letter	20,000	16	26
Sensorless	58,483	48	11
SensIT Vehicle	98,582	100	3
Covtype	396,112	54	2

3.5 Privacy Analysis

Threat Model. In line with existing VFL systems [31], we assume that the aggregation server and parties are semi-honest. That is, they all follow the designed protocol but try to speculate the benign parties from the received data. Besides, some parties may collude with each other and try to infer sensitive information about the benign parties.
Feature Security denotes that the raw features of parties should be maintained locally. We adopt the SplitNN architecture, in which each party only transfer the bottom model output rather than the raw local features to the server, and the parties do not communicate directly with each other. Feature security is ensured against any curious party. Even if some parties collude, the security of features remains intact, as they cannot infer the features of other honest clients.
Label Security denotes the label of parties can not be shared with any external party. In our framework, we utilize homomorphic encryption to secure all intermediate data related to labels, such as y_i^k and $T_{y_i^k,\cdot}$. At the same time, the server can only obtain the encrypted value after computation (i.e., $\mathcal{L}_{\text{KL}}$ and $\mathbf{z}_i$), and cannot directly infer the original label value after decryption. Additionally, each party only receives the gradients of the last layer to update its local model. As a result, label security is safeguarded against any curious party or collusion. However, if the server colludes with any party, this protection is obviously compromised since the server has the private key.

4 Experiment

4.1 Experimental Setting

Datasets We validate the proposed method by several widely used public datasets — Letter [15], Sensorless [32], SensITVehicle [10] and covtype [3]. In Table 1, we summarize the statistics of these datasets. Before training, features are equally divided into four parts and sent to four parties randomly, and they all have multiple noisy labels. For all datasets, we randomly sample 80% of data for training and the rest for testing.
Setting of Noisy Labels. In the experiments, we introduce symmetric noise by randomly flipping clean labels into all other possible classes with probability ε, the reason is that previous studies [24] have shown that symmetric label noise is more challenging than asymmetric label noise. In setup stage, we set four parties,

and each party have different levels of label noises. Specially, we set up four scenarios for the noise span $\varepsilon \in [\varepsilon_{\min}, \varepsilon_{\max}]$: 1) low ratio & small range: the noise ratios are randomly chosen from $\varepsilon \in [0.1, 0.2]$; 2) median ratio & small range: the span is $\varepsilon \in [0.3, 0.4]$; 3) median ratio & large range: the span is $\varepsilon \in [0.2, 0.5]$; 4) high ratio & large range: the span is $\varepsilon \in [0.3, 0.6]$. To guarantee the robustness of results, we train three times and average the results. In this way, we consider different noise levels and diverse annotation expertise.

Baselines. We compared our method with four baselines designed for VFL with multiple noisy labels.

- **Clean**: Only one label party, and the labels are totally clean without any noise, which is an unrealistic setting in the real world, we choose it to assess whether our method can achieve comparable performance to the ideal case.
- **Random**: train a model under SplitNN and randomly select one label party for each batch of training.
- **MJ** [28]: train a model with the label of the highest probability of occurrence among all label parties. If multiple labels appear the same number of times, we randomly select one.
- **EM** [7]: train a model using EM algorithm to infer latent ground truth. The inference process is executed on the server and the original values are encrypted using HE techniques.

The reason why we chose these baselines is that they infer a potentially correct label before the model training begins; therefore, preventing potential privacy leakage from intermediate information during training.

Implementations and Protocols. We choose Python 3.8.18 and Pytorch 2.0.1 to implement InC. Besides, each party including the server is deployed on one NVIDIA RTX A6000 GPU instance. We choose multiple linear regression (MLR) and multi-layer perceptron (MLP) as machine learning tasks. We apply Adam [8] as the optimization algorithm with a learning rate of $1e-3$. Besides, we set the batch size to 1% of the total amount of the dataset, and the number of iterations to 100 in baseline. In our Inc framework, we also set the iterations to 100 – 50 for stage 1 and 50 for stage 2. For EM [7], the maximum tolerance is set to $1e-5$, and the maximum number of iterations is set to 100.

Metrics We measure the accuracy of the model and the label correction, respectively. For the model accuracy, we use prediction accuracy on the test dataset. For the label correction accuracy, we calculate the ratio of correct corrections and the total number of corrections. Besides, we also compare the efficiency of the baselines by end-to-end runtime.

4.2 Main Experiments

Evaluation of Model Accuracy. We evaluated the accuracy of the vertical federated model and the results are shown in table 2. In comparison with the baselines, the accuracy of the VFL model can be improved by up to 10.84%. And in the vast majority of cases, it outperforms the baseline method in both

Table 2. Evaluation of model accuracy. The best results are highlighted in bold.

Dataset	**Method**		**LR**				**MLP**			
			Noise Setting							
		ratio:	**low**	**median**	**median**	**high**	**low**	**median**	**median**	**high**
		range:	**small**	**small**	**large**	**large**	**small**	**small**	**large**	**large**
Letter	**Clean**		**0.7720**				**0.9205**			
	Random		0.7297	0.7055	0.7170	0.6967	0.8842	0.8660	0.8612	0.8472
	MJ		0.7630	0.7437	0.7490	0.7227	0.9085	0.8690	0.7897	0.7902
	EM		0.7622	0.7492	0.7525	0.7410	0.9197	0.8995	0.9005	0.8775
	InC		**0.7750**	**0.7637**	**0.7662**	**0.7667**	**0.9245**	**0.9060**	**0.9132**	**0.9055**
Sensorless	**Clean**		**0.9157**				**0.9872**			
	Random		0.8107	0.6939	0.7014	0.6443	0.9847	0.9755	0.9569	0.9526
	MJ		0.8728	0.8105	0.8078	0.7416	0.9850	0.8735	0.8650	0.8703
	EM		0.8744	0.8498	0.8654	0.7987	0.9869	0.9703	0.9657	0.9626
	InC		**0.9181**	**0.9109**	**0.9111**	**0.9071**	**0.9935**	**0.9925**	**0.9927**	**0.9771**
SensIT Vehicle	**Clean**		**0.8022**				**0.8428**			
	Random		0.7917	0.7849	0.7853	0.7789	0.8328	0.7252	0.8189	0.8038
	MJ		0.7968	0.7797	0.7774	0.7509	0.8239	0.8076	0.8096	0.7132
	EM		0.7900	0.7854	0.7878	0.7507	0.8321	0.8071	0.8275	0.8134
	InC		**0.8057**	**0.8039**	**0.8053**	**0.7935**	**0.8386**	**0.8353**	**0.8303**	**0.8225**
Covtype	**Clean**		**0.7723**				**0.8261**			
	Random		0.7728	0.7681	0.7670	0.7388	0.8161	0.7935	0.4283	0.4283
	MJ		0.7659	0.6020	0.6241	0.4322	0.8184	0.4283	0.4283	0.4238
	EM		0.7732	0.7724	0.7717	**0.7728**	**0.8254**	0.8066	0.8072	**0.8064**
	InC		**0.7750**	**0.7753**	**0.7726**	0.7724	0.8212	**0.8068**	**0.8080**	0.7932

Table 3. Evaluation of end-to-end training time.

Method \ **Dataset**	**Letter**	**Sensorless**	**SensIT Vehicle**	**Covtype**
	LR			
Clean	134.03	391.86	651.55	370.04
Random	125.93	385.61	628.36	400.05
MJ	137.01	384.15	648.61	442.30
EM	312.68	477.84	805.01	2262.29
InC	185.71	548.65	1069.35	637.42
	MLP			
Clean	154.21	491.36	886.26	521.44
Random	177.37	466.58	871.31	751.36
MJ	176.84	451.39	890.13	725.16
EM	339.97	612.61	1036.66	2482.71
InC	262.02	602.60	903.95	1562.15

LR and MLP models, with an average improvement of 3% to 4%. Especially in the high ratio & large range setting, InC can still maintain an accuracy close to clean label training, while improving by an average of 2% to 3% compared to the runner-up method. In order to clarify the gap of improvement, we also tested the results of training with only one active party holding complete correct labels. From the result, it can be concluded that our method is almost close to the accuracy of clean label training, and even exceed.

Evaluation of Label Correction Accuracy. In our training process, the accuracy of predicted labels gradually increases with dynamic iterations. The data in Table 4 shows the label correction accuracy in LR, indicating that the

Table 4. Evaluation of label correction accuracy on the logistic regression model. The best results are highlighted in bold.

Dataset	Method	Noise			
		low ratio small range	median ratio small range	median ratio large range	high ratio large range
Letter	**Random**	0.8515	0.6330	0.6643	0.5587
	MJ	0.9314	0.8772	0.9010	0.8056
	EM	0.9589	0.8824	0.9133	0.8212
	InC	**0.9626**	**0.89.65**	**0.9599**	**0.8552**
Seneorless	**Random**	0.8274	0.6436	0.6513	0.5531
	MJ	0.9294	0.8792	0.8837	0.7823
	EM	0.8992	0.8834	0.9039	0.8111
	InC	**0.9839**	**0.9835**	**0.9843**	**0.9663**
SensIT Vehicle	**Random**	0.8526	0.6502	0.6535	0.5550
	MJ	0.9109	0.7758	0.7794	0.6464
	EM	0.8980	0.8223	0.8504	0.7354
	InC	**0.9485**	**0.8487**	**0.8584**	**0.8090**
Covtype	**Random**	0.8500	0.6507	0.6520	0.5525
	MJ	0.9049	0.6749	0.6791	0.5649
	EM	0.8998	0.7005	0.8006	0.6886
	InC	**0.9172**	**0.8138**	**0.8194**	**0.6338**

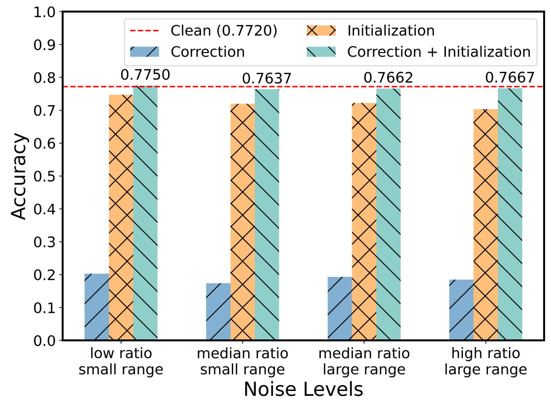

Fig. 4. Ablation experiment on the Letter dataset and LR.

final predicted labels in LR have exceeded the original accuracy of all parties (i.e., in the low ratio & small range setting, the label correction accuracy is lower than $\varepsilon_{\min} \in [0.1, 0.2]$) and are almost entirely correct. This indicates that our strategy improved the quality of the original dataset to a certain content.

Evaluation of Efficiency. In terms of time consumption, compared to the baseline, InC does not introduce significant additional time overhead. On the contrary, as the size of the dataset expands, the time consumption of EM increases significantly. For example, on Covtype dataset, InC compared to one clean label party, InC increased by 267 seconds, but EM consumed an additional 1892 seconds. On Letter dataset and model LR, InC is almost as time-consuming as random, with a difference of only 51 seconds.

4.3 Ablation Experiment

In the ablation experiment, we tested the accuracy of the model in four symmetric noise scenarios on the Letter dataset.

The Effect of Initialization. As can be seen from Fig. 4, first, the results of Correct after removing Initialize are tested. It's obvious that only Correct cannot make any effect. Compared with the correction process only, the complete process with initialization has increased model accuracy by 70%.

The Effect of Correction. We show only Initialize training results in Fig. 4. After adding the correction process, the model accuracy improved by an average of 4% to 5% points compared to initialization. The results in the table 4 also indicate that our complete processes effectively brings improvements.

5 Conclusions

In this work, we introduced a two-stage process to ensure the accuracy and privacy of VFL with multiple noisy labels. In the first initialization stage, we train a consensus classifier using the soft labels without labels disclosure. In the second correction stage, we use this classifier to dynamically correct noise by an EM-style algorithm. Extensive experiments on various datasets and tasks demonstrate the effectiveness of our proposed Inc framework.

6 Acknowledgments

This work was sponsored by Key R&D Program of Hubei Province (2023BAB077), National Natural Science Foundation of China (62472327), Science and Technology Development Fund (SKL-IOTSC(UM)-2021-2023), and the State Key Laboratory of Internet of Things for Smart City (University of Macau) (Ref. No.: SKL-IoTSC(UM)-2024-2026/ORP/GAxx/2023)

References

1. Abedi, A., Khan, S.S.: Fedsl: federated split learning on distributed sequential data in recurrent neural networks. Multimedia Tools and Applications **83**(10), 28891–28911 (2024)
2. Abuadbba, S., Kim, K., Kim, M., Thapa, C., Camtepe, S.A., Gao, Y., Kim, H., Nepal, S.: Can we use split learning on 1d cnn models for privacy preserving training? In: Proceedings of the ACM Asia Conference on Computer and Communications Security. pp. 305–318 (2020)
3. Blackard, J.: Covertype (1998), uCI Machine Learning Repository
4. Ceballos, I., Sharma, V., Mugica, E., Singh, A., Roman, A., Vepakomma, P., Raskar, R.: Splitnn-driven vertical partitioning. arXiv preprint arXiv:2008.04137 (2020)
5. Chen, Y., Yang, X., Qin, X., Yu, H., Chan, P., Shen, Z.: Dealing with label quality disparity in federated learning. Federated Learning: Privacy and Incentive pp. 108–121 (2020)

6. Cheng, Y., Liu, Y., Chen, T., Yang, Q.: Federated learning for privacy-preserving ai. Communications of the ACM **63**(12), 33–36 (2020)
7. Dawid, A.P., Skene, A.M.: Maximum likelihood estimation of observer error-rates using the em algorithm. Journal of the Royal Statistical Society: Series C (Applied Statistics) **28**(1), 20–28 (1979)
8. Diederik, P.K.: Adam: a method for stochastic optimization (2014)
9. Duan, Q., Hu, S., Deng, R., Lu, Z.: Combined federated and split learning in edge computing for ubiquitous intelligence in internet of things: state-of-the-art and future directions. Sensors **22**(16), 5983 (2022)
10. Duarte, M.F., Hu, Y.H.: Vehicle classification in distributed sensor networks. Journal of Parallel and Distributed Computing **64**(7), 826–838 (2004)
11. Fang, X., Ye, M.: Robust federated learning with noisy and heterogeneous clients. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 10072–10081 (2022)
12. 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)
13. Fu, F., Xue, H., Cheng, Y., Tao, Y., Cui, B.: Blindfl: vertical federated machine learning without peeking into your data. In: Proceedings of the International Conference on Management of Data. pp. 1316–1330 (2022)
14. Ghassemi, M., Naumann, T., Schulam, P., Beam, A.L., Chen, I.Y., Ranganath, R.: a review of challenges and opportunities in machine learning for health. AMIA Summits on Translational Science Proceedings **2020**, 191 (2020)
15. Hsu, C.W., Lin, C.J.: A comparison of methods for multiclass support vector machines. IEEE Transactions on Neural Networks **13**(2), 415–425 (2002)
16. 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? Advances in Neural Information Processing Systems **35**, 2088–2101 (2022)
17. Li, X., Huang, K., Yang, W., Wang, S., Zhang, Z.: On the convergence of fedavg on non-iid data. arXiv preprint arXiv:1907.02189 (2019)
18. Liang, S., Huang, J., Hong, J., Zeng, D., Zhou, J., Xu, Z.: Fednoisy: federated noisy label learning benchmark. arXiv preprint arXiv:2306.11650 (2023)
19. Mammen, P.M.: Federated learning: Opportunities and challenges. arXiv preprint arXiv:2101.05428 (2021)
20. McMahan, B., Moore, E., Ramage, D., Hampson, S., y Arcas, B.A.: Communication-efficient learning of deep networks from decentralized data. In: Artificial Intelligence and Statistics. pp. 1273–1282 (2017)
21. Mugunthan, V., Goyal, P., Kagal, L.: Multi-vfl: a vertical federated learning system for multiple data and label owners. arXiv preprint arXiv:2106.05468 (2021)
22. Nock, R., Hardy, S., Henecka, W., Ivey-Law, H., Patrini, G., Smith, G., Thorne, B.: Entity resolution and federated learning get a federated resolution. arXiv preprint arXiv:1803.04035 (2018)
23. Paillier, P.: Public-key cryptosystems based on composite degree residuosity classes. In: International Conference on the Theory and Applications of Cryptographic Techniques. pp. 223–238 (1999)
24. Patrini, G., Rozza, A., Krishna Menon, A., Nock, R., Qu, L.: Making deep neural networks robust to label noise: a loss correction approach. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 1944–1952 (2017)

25. Raykar, V.C., Yu, S., Zhao, L.H., Valadez, G.H., Florin, C., Bogoni, L., Moy, L.: Learning from crowds. Journal of machine learning research **11**(4) (2010)
26. Romanini, D., Hall, A.J., Papadopoulos, P., Titcombe, T., Ismail, A., Cebere, T., Sandmann, R., Roehm, R., Hoeh, M.A.: Pyvertical: a vertical federated learning framework for multi-headed splitnn. arXiv preprint arXiv:2104.00489 (2021)
27. Rooijakkers, T.: Convinced—enabling privacy-preserving survival analyses using multi-party computation (2020)
28. Sheng, V.S., Provost, F., Ipeirotis, P.G.: Get another label? improving data quality and data mining using multiple, noisy labelers. In: Proceedings of the ACM SIGKDD international conference on Knowledge discovery and data mining. pp. 614–622 (2008)
29. Thapa, C., Arachchige, P.C.M., Camtepe, S., Sun, L.: Splitfed: when federated learning meets split learning. In: Proceedings of the AAAI Conference on Artificial Intelligence. vol. 36, pp. 8485–8493 (2022)
30. Thiel, C.: Classification on soft labels is robust against label noise. In: International Conference on Knowledge-Based and Intelligent Information and Engineering Systems. pp. 65–73 (2008)
31. 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)
32. Wang, C.C., Tan, K.L., Chen, C.T., Lin, Y.H., Keerthi, S.S., Mahajan, D., Sundararajan, S., Lin, C.J.: Distributed newton methods for deep neural networks. Neural Computation **30**(6), 1673–1724 (2018)
33. Wang, Z., Zhou, T., Long, G., Han, B., Jiang, J.: Fednoil: a simple two-level sampling method for federated learning with noisy labels. arXiv preprint arXiv:2205.10110 (2022)
34. Wei, K., Li, J., Ma, C., Ding, M., Wei, S., Wu, F., Chen, G., Ranbaduge, T.: Vertical federated learning: challenges, methodologies and experiments. arXiv preprint arXiv:2202.04309 (2022)
35. Wu, N., Yu, L., Jiang, X., Cheng, K.T., Yan, Z.: Fednoro: towards noise-robust federated learning by addressing class imbalance and label noise heterogeneity. arXiv preprint arXiv:2305.05230 (2023)
36. Xia, W., Li, Y., Zhang, L., Wu, Z., Yuan, X.: Cascade vertical federated learning towards straggler mitigation and label privacy over distributed labels. IEEE Transactions on Big Data (2023)
37. Xu, J., Chen, Z., Quek, T.Q., Chong, K.F.E.: Fedcorr: multi-stage federated learning for label noise correction. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 10184–10193 (2022)
38. Yang, S., Park, H., Byun, J., Kim, C.: Robust federated learning with noisy labels. IEEE Intelligent Systems **37**(2), 35–43 (2022)
39. 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)
40. Zhang, R., Li, H., Tian, L., Hao, M., Zhang, Y.: Vertical federated learning across heterogeneous regions for industry 4.0. IEEE Transactions on Industrial Informatics (2024)
41. 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)