Privacy Protection Optimization for Federated Software Defect Prediction via Benchmark Analysis

Ying Liu[1,2], Yong Li[1,2], Ming Wen[2], Wenjing Zhang[1]*

[1] College of Computer Science and Technology, Xinjiang Normal University, China
[2] Xinjiang Electronics Research Institute, China
liuying.void@gmail.com, liyong@live.com, wmconet@126.com, zwenj0801@163.com

Abstract

Federated learning is a privacy-preserving machine learning technique that coordinates multi-participant co-modeling. It can alleviate the privacy issues of software defect prediction, which is an important technical way to ensure software quality. In this work, we implement Federated Software Defect Prediction (FedSDP) and optimize its privacy issues while guaranteeing performance. We first construct a new benchmark to study the performance and privacy of Federated Software defect prediction. The benchmark consists of (1) 12 NASA software defect datasets, which are all real software defect datasets from different projects in different domains, (2) Horizontal federated learning scenarios, and (3) the Federated Software Defect Prediction algorithm (FedSDP). Benchmark analysis shows that FedSDP provides additional privacy protection and security with guaranteed model performance compared to local training. It also reveals that FedSDP introduces a large amount of model parameter computation and exchange during the training process. There are model user threats and attack challenges from unreliable participants. To provide more reliable privacy protection without losing prediction performance we proposed optimization methods that use homomorphic encryption model parameters to resist honest but curious participants. Experimental results show that our approach achieves more reliable privacy protection with excellent performance on all datasets.

Keywords: Software defect prediction, Federated learning, Homomorphic encryption, Privacy protection

1 Introduction

The increasing awareness of personal data protection [1] has limited the development of software defect prediction. Software defect prediction is vital in the modern industry to improve software reliability and avoid software problems during operation [2]. However, as data collection and use continue to expand, there are growing concerns about the security and privacy of the data being collected and shared [3]. On the other hand, software vendors and organizations have realized the potential benefits of consolidating their data assets, particularly improving statistical capabilities for analytical and predictive tasks. This has led to the problem of software defect data silos, which has stalled the development of software defect prediction techniques with extreme reliance on data. Therefore, it is necessary to navigate its development under the premise of privacy protection. In the past five years of research, different privacy-preserving approaches for software defect prediction have been explored. They generally use differential privacy techniques to achieve privacy preservation, but this does not adequately guarantee the performance of software defect prediction models.

In recent years, how to further improve data security and user privacy and unleash data values while ensuring model performance without loss has become a hot research topic in academia and industry. Federated learning is a privacy-preserving machine learning technique that trains models on dispersed software defect data through edges to share model updates with servers [4]. Thus, Federated Learning can effectively mitigate the risk of potential privacy breaches. In addition to privacy protection, implementing federated software defect prediction has other advantages: the communication overhead is reduced by avoiding massive data uploads [4], global models can be applied to different projects, and participants can obtain models adapted to local data.

This work focuses on the privacy-preserving aspects of federated learning in software defect prediction while guaranteeing the performance of software defect prediction models. Privacy preservation is further optimized by constructing a new benchmark and performing benchmark analysis, using the Paillier technique to encrypt the large number of model parameters introduced in the training of the federated model, and using polynomial Taylor expansions to approximate the base model loss function to improve the computational efficiency of the algorithm. Comprehensive experimental results of the analysis demonstrate the practicality and effectiveness of the approach. We summarize the contributions of this paper as follows:

(I) We constructed a new benchmark for FedSDP and analyzed its privacy issues through the benchmark. The constructed federated software defect prediction benchmark has the following features: (1) Using the publicly available NASA Software Defect Dataset, which includes 12 software project modules and acts as a publicly available dataset specifically designed to study software defect prediction models, (2) Setting federated scenarios based on software defect data features, (3) Designing a suitable algorithm for

*Corresponding Author: Yong Li; E-mail: liyong@live.com
DOI: 10.53106/160792642023112406001

FedSDP, (4) Standardizing base models and performance evaluation metrics.

(II)We propose an approach to optimize privacy protection: homomorphic encryption. Encrypting federated training model parameters to avoid attackers launching inference attacks based on model parameters.

The rest of this paper is organized as follows. In Section 2, we introduce the related work. Section 3 describes the FedSDP benchmark. Section 4 analyzes the benchmark results and provides insights. In Section 5, we propose optimization methods to improve the privacy protection of FedSDP. Finally, Section 6 concludes the paper and presents future work.

2 Related Work

2.1 Software Defect Prediction

Currently, most of the research work on software defect prediction methods is based on machine learning techniques, with data as the key driving the performance of defect prediction algorithm models [5]. With the development of machine learning techniques, methods for building software defect prediction models have tended to be enriched, such as decision trees [6], support vector machines [7], and Naive Bayes [8] algorithms, among which the Naive Bayes algorithm has a better performance with an accuracy rate of 71% [9]. Some researchers have also proposed a cross-project software defect prediction method, which uses the labeled data from related source projects to build a defect prediction model to achieve target project defect prediction. For example, in a cross-project prediction method based on feature migration, He et al [10] obtained the optimal subset of attributes from the source and target projects to build defect prediction models as a way to alleviate the problem of insufficient data for the target projects. Mi et al [11] proposed an active learning based data selection algorithm considering that the prior knowledge of the target item can match the source item with the defect pattern of the target item. Menzies et al [12] implemented cross-item defect prediction by building local models through clustering clusters of source and target items. In software defect prediction model training, software defect data plays a crucial role, containing sensitive information such as code data and software version data, which will directly threaten the reputation and property security of the organization or enterprise if leaked or exploited during the training process. Therefore, there is a need to explore more effective privacy protection methods for software defect prediction to deal with the strict regulatory environment of user privacy and data security.

2.2 Federated Learning

2.2.1 Federated Learning Benchmarks

Caldas et al. proposed LEAF in [13], which focuses on a benchmark framework for natural language processing and image classification. Luo et al in [14] proposed a real-world dataset for object detection. Both works use the Federated Averaging (FedAvg) algorithm as a baseline implementation. In this work, we introduce a new benchmark for federated learning combined with software defect prediction and analyze the benchmark to reveal potential privacy issues.

2.2.2 Privacy Protection in Federated Learning

Unlike centralized algorithm training, federated learning algorithm training only requires model parameters to be computationally exchanged between the participants and the server, avoiding direct exposure of data and providing natural protection for data privacy. However, the model parameters and gradients, as private data for the model training task, contain information about the data features, posing the risk of leakage. It has been shown that some of the original data can be restored by gradients. Zhu et al [15] explored the problem of sensitive information leakage using model gradients and proposed a privacy attack method. Tramèr et al [16] incorporated machine learning models through prediction API and proposed a corresponding defense. Song et al [17] exploit the machine learning model over-memory to obtain user privacy information. Fredrikson et al [18] first proposed an approach to protect sensitive data using cryptography and security protocols. In subsequent work, Fredrikson et al [19] introduced a model inversion attack method based on confidential information. Shokri et al [20] proposed a neural network-based model membership inference attack that reveals the privacy leakage problem in machine learning models. Unreliable participants will exacerbate the risk of privacy breaches. It inferred other participant labels or data based on legitimately obtained intermediate parameters. Geyer et al [21] first proposed a federated learning differential privacy-preserving algorithm for the user level. Huang et al [22] proposed the alternating direction method of multipliers (ADMM) perturbation algorithm, which provides a highly usable protection scheme for nonsmooth convex objective functions. Aono et al [23] used encryption mechanisms to protect federated training in a client-server architecture. Inspired by encryption mechanisms, we use homomorphic encryption algorithms in federal software defect prediction to provide more reliable privacy protection.

3 FedSDP Benchmark

In this section, we introduce the FedSDP benchmark, a new benchmark for implementing federated learning for software defect prediction. It includes 12 datasets (Section 3.1), horizontal federated learning scenarios (Section 3.2), model selection (Section 3.3), the federated software defect prediction algorithm (Section 3.4), and performance metrics (Section 3.5).

3.1 Datasets

To simulate a realistic scenario of FedSDP, we selected 12 NASA software defect prediction datasets from the PROMISE database [24], which is a publicly available dataset published by NASA specifically for building software defect prediction models, with the data described in Table 1. These data differ significantly in terms of design language, defect rate, and example number. Maurice et al [25] proposed a series of metrics and measures to measure software complexity and quality. McCabe et al [26] proposed a measure of software complexity, the McCabe complexity. Radjenović et al [27] summarized various metrics for

evaluating software defect prediction. Basili et al [28] validated the effectiveness of object-oriented design metrics as quality indicators. Elish et al [29] compared the effectiveness of three package-based software defect prediction metric suites in Eclipse. Olague et al [30] empirically validated three software metrics suites to predict the propensity for errors in object-oriented classes developed using highly iterative or agile software development processes. Each software module in the dataset contains public attributes such as McCabe, Lines of Code, and Halatead. It can effectively model the participating parties in federated learning.

Table 1. The characteristics of 12 datasets for the FedSDP benchmark

Data	Language	Examples	Attributes	Defect (%)
JM1	C	7,720	22	20.9
PC2	C	5,589	37	0.41
MC1	C	1,952	39	1.8
PC5	C	1,694	39	27.0
PC3	Java	1,409	38	10.5
PC4	C++	1,270	38	13.9
KC1	C++	1,162	22	25.3
PC1	C	919	22	6.5
CM1	C	505	38	48.0
MW1	C	375	38	7.5
KC3	Java	324	40	13.0
MC2	C++	155	40	32.9

3.2 Horizontal Federated Learning

Yang et al [31] first divided federated learning into horizontal FL, vertical FL, and federated migration learning. Kairouz et al [32] then further explored different forms of division in federated learning, including device-based division, data-based division, and model-based division. In horizontal federated learning, different participants have different sample spaces but intersect in the feature space (Figure 1). Software defect data has overlapping data characteristics, so this paper is concerned with software defect prediction horizontal federated learning scenarios. A client-server architecture is used, where each dataset is defined as a separate client that communicates directly with the server for federated learning. In this case, keeping the data in the client can greatly reduce the risk of a privacy breach.

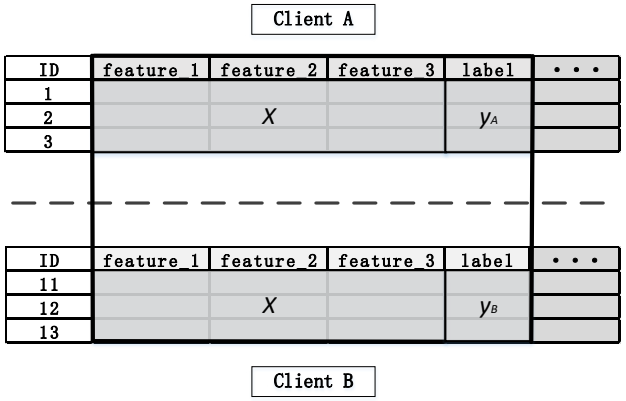

Figure 1. Horizontal federated learning scenario data segmentation

3.3 Model Selection

Software defect prediction targets to measure the code properties of software modules that contain defects in historical releases. Machine learning techniques are usually used to construct software defect prediction models. The logistic regression algorithm (LR) belongs to the generalized linear regression model, where the probability value of the predicted outcome is calculated by the attribute eigenvector coefficients and is chosen as the baseline in software defect prediction [33]. Therefore, we use logistic regression as the base model to perform federated learning.

3.4 Federated Software Defect Prediction Algorithm

The Federated Software Defect Prediction algorithm uses the standard Federated Average Algorithm (FedAvg), which involves the client training the model using a local dataset and uploading model parameter updates to the server; the server is responsible for initializing the model and aggregating model updates from the client by weighted averaging. Figure 2 shows the implementation of the FedAvg to Federated Software Defect Prediction algorithm, where the client and server train and share the global model.

Assume that the aggregation server model parameters are W^t, the data characteristics of participant A and participant B are X_A and X_B ($X_A = X_B$), the initial model parameters are W_A and W_B ($W_A = W_B = W^t$), the label of participant A is $y_A \in \{-1,1\}^{N_A}$, and the label of participant B is $y_B \in \{-1,1\}^{N_B}$, with 1 denoting a defective module and -1 denoting a non-defective module; The data features and label space of the participants are the same for (X_A, y_A) and (X_B, y_B), where N is the number of sample entries in the dataset and t is the number of model training epochs. The predicted output of the logistic regression model is $\hat{y} = Sigmoid(Z)$, where $Z = W^t X_i$, $i = A, B$ …. The training objective of the logistic regression algorithm is to find the model parameters that minimize the value of the loss function, such as the participant A, $W_A^* = \arg\min_{W_A} L(\hat{y}, y)$, where $L(\cdot,\cdot)$ is the cross-entropy loss function, $(\hat{y}, y) = \Sigma_{i=1}^{N} (y_i \log \hat{y}_i + (1 - y_i) \log(1 - \hat{y}_i))$.

In each epoch round, training is performed using the small-batch Stochastic Gradient Gradient (mini-batch SGD) algorithm, which first samples small batches of sample data $X^{(b)}$, $Y^{(b)}$, from the participants. Computational model gradients for participant P on sampled small batch sample data.

$$\nabla W_P^{(b)} = \frac{\partial L(\hat{y}^{(b)}, y^{(b)})}{\partial W_P} = \frac{1}{BS} X_P^{(b)T} \nabla Z^{(b)}, \quad \text{where } \nabla Z^{(b)} = \frac{\partial L(\hat{y}^{(b)}, y^{(b)})}{\partial Z} = \hat{y}^{(b)} - y^{(b)}. \tag{1}$$

BS is the number of sampling training samples (Batch Size). The participant P model is updated as $W_P = W_P - \eta \nabla W_P^{(b)}$, where η is the gradient descent step size. The model update of the aggregation server is

$W^{t+1} = W^t + \lambda \sum_{i=1}^{m} \left(W_i^{t+1} - W_i^t \right)$, where W_i^{t+1} denotes the model of the (i)th participant after the (t + 1)th round of local update and λ is the learning rate. We summarize the Federated Software Defect Prediction algorithm (FedSDP) in Algorithm 1.

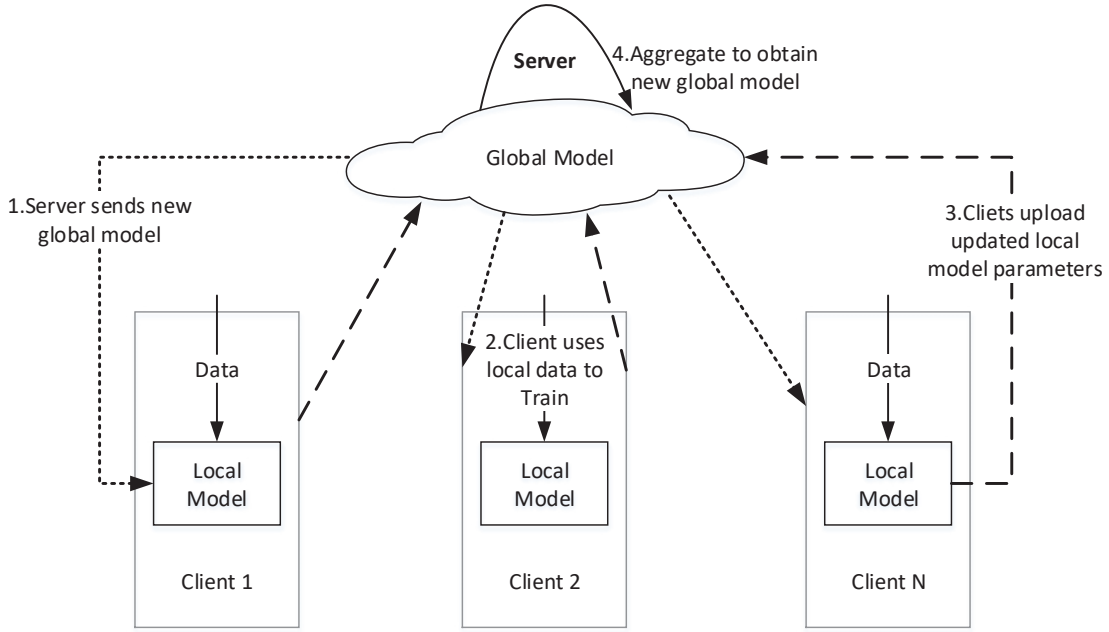

Figure 2. FedSDP algorithm process using FedAvg

(Each epoch consists of the following steps: (1) The server sends the global model to the client. (2) The client trains the model using local data. (3) The client uploads the model parameters to the server. (4) The server aggregates the model updates from the client by weighted average to obtain a new global model.)

Algorithm 1. Federated Software Defect Prediction (FedSDP) with FedAvg

Input: E, B, K, η, T, N, n_k, n

Output: w^T, w_k^T

Server:

Initialize w^0;

for each round t = 0 to T − 1 **do**

$C_t \leftarrow$ (randomly select K out of N clients);

for *each client* $k \in C_t$ *concurrently* **do**

$w_k^{t+1} \leftarrow$ **Client** (w^t, k, t);

end

// n: total size of dataset; n_k: size of client k's dataset;

$w^{t+1} \leftarrow \Sigma_{k \in C_t} \frac{n_k}{n} w_k^{t+1}$;

end

return w^T;

Client(w, k, t):

$\beta \leftarrow$ (divide local data into batches of size B);

for each local epoch e = 0 to E-1 **do**

for $b \in \beta$ **do**

// parameter w update;

$w \leftarrow w - \eta \nabla \mathcal{L}(w; b)$;

end

end

return w ;

return

3.5 Performance Metrics

Software defect prediction is a binary classification problem where the goal is to study the classification of software modules into defective and non-defective modules. A few defective modules are defined as positive cases and most non-defective modules are defined as negative cases in the classification learning task. Table 2 shows the confusion matrix for software defect prediction.

Table 2. Confusion matrix description

Defective module status	Predicted results	
	Defective module	Non-defective module
Defective module	True Positive (TP)	False Negative (FN)
Non-defective module	False Positive (FP)	True Negative (TN)

The task of software defect prediction is to find as many defective modules as possible, and its prediction accuracy (Precision, P) can be calculated based on the confusion matrix, $P = TP/(TP + FP)$, which represents the proportion of defective modules correctly predicted; Recall (R) represents the "ability" of the algorithm model to correctly predict software modules "metric, $R = TP/(TP + FN)$; on this basis, F1 (F1-score) value is defined: $F1 = 2PR/(P + R)$, which represents the weighted summed average of accuracy and recall, which can visually reflect the stability of software

defect prediction model. Therefore, we used Precision (P), F1-score (F1), and Recall metrics for FedSDP model evaluation.

4 Benchmark Analysis

Using the benchmarks in Section 3, we conducted extensive experiments on different federated settings, analyzed the results of the algorithmic model experiments, and investigated the effects of batch size B and the number of local epochs E, as well as the performance comparison with local training, and gained meaningful insights by analyzing the experimental results. We constructed 12 clients for the following experiments, each using one of the 12 datasets for training. In each round of communication, we select all clients for aggregation.

4.1 Impact of Batch Size

Batch size is an important hyperparameter in FedSDP that affects the client's computations. A smaller batch size will lead to increased computation for the client in each training round, assuming a fixed number of local epochs and dataset size. Fixing local epoch number $E = 1$ global training is 300 rounds. We compare the performance of different batch sizes $B = 32$, $B = 64$, and $B = 128$ in Figure 3. The performance of most clients improved slightly when we reduced the batch size to increase the amount of computation. Therefore, we use $B = 32$ as the default batch size setting.

4.2 Communication Cost

In FedSDP, the trade-off between communication cost and performance depends on the number of local epochs. Setting the default batch size $B = 32$ and fixing the global training for 300 rounds, Figure 4 shows the performance for different local epochs $E = 1$, $E = 5$, and $E = 10$. Although $E = 1$ and $E = 10$ perform better than $E = 5$ for a few clients, $E = 5$ substantially outperforms $E = 1$ and $E = 10$ for all clients. Even though lowering the local epoch E generally improves performance, it adds additional communication costs.

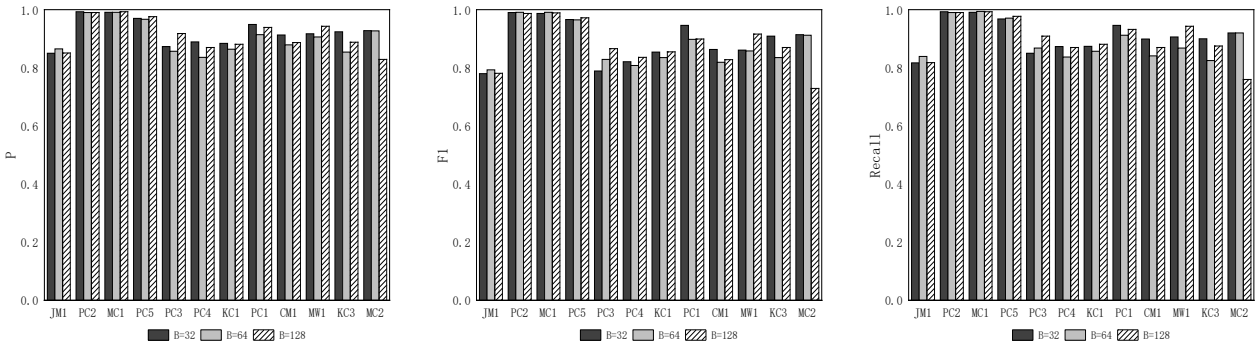

Figure 3. Performance comparison of different batch sizes. Batch size $B = 32$ performs better in most clients

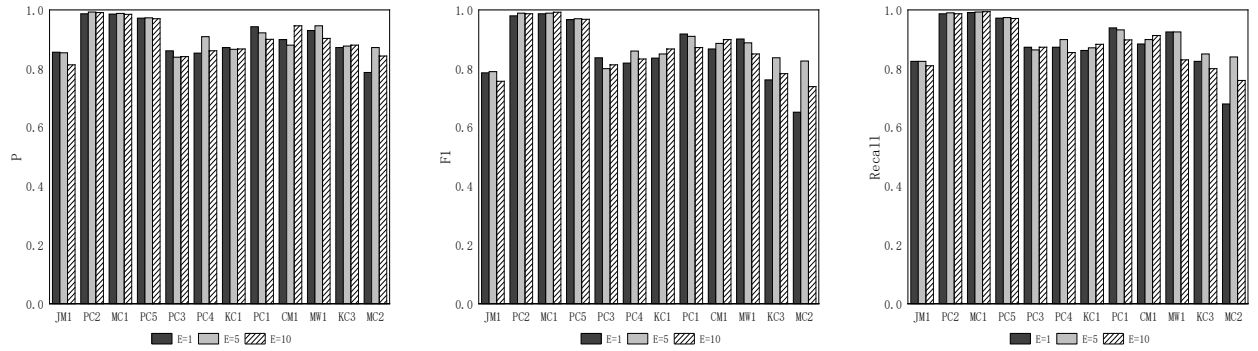

Figure 4. Performance comparison for the different number of local epochs

(The default batch size $B = 32$ and the total number of training rounds $T = 300$. The number of local epochs $E = 5$ performs better in most clients.)

4.3 Upper Bound of FedSDP

We compare the model performance obtained from FedSDP and local training. Following the previous conclusion, we use the local epoch E = 5 and batch size B = 32 as the default hyperparameter settings for the FedSDP algorithm. Table 3 and Table 4 show the experimental results of local training and FedSDP and the F1-scores are compared in Figure 5. Although the federated model performed slightly worse than local training on the larger sample size datasets PC3 and PC2 (Figure 5(a)), it outperforms local training on most of the smaller datasets (Figure 5(b)). These results suggest that clients with smaller datasets can more effectively acquire knowledge from other clients. The dataset size has a significant impact on the performance and generalization ability of the federated model. Since clients with larger datasets dominate the federated aggregation, less knowledge is obtained from other clients; the models trained by clients with smaller datasets have poor generalization ability, so more knowledge is needed from other clients to improve their ability. Therefore, FedSDP can facilitate knowledge sharing and collaboration among different clients, thus improving the performance and efficiency of the whole model, especially in the case of uneven distribution of data sets.

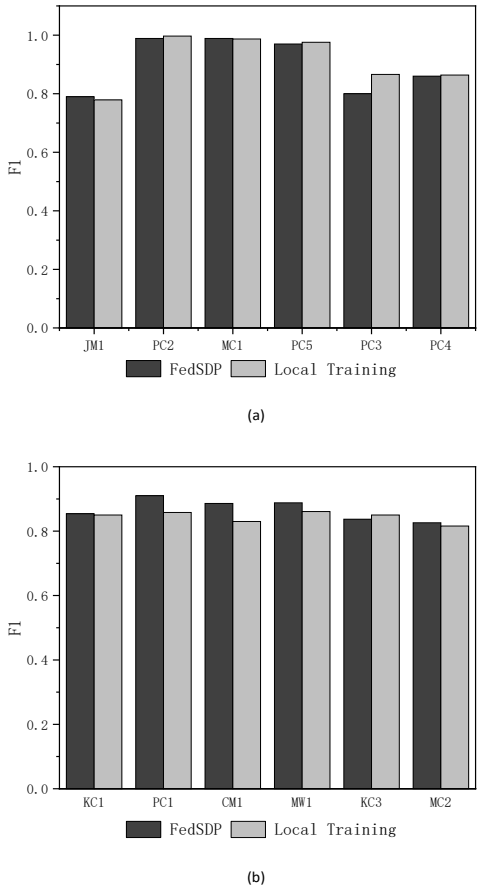

Figure 5. Performance (F1-score) of FedSDP compared with local training

(Although in (a) the federated models perform slightly worse than local training on larger datasets, they outperform local training on smaller datasets in (b).)

Table 3. Performance of models trained on each dataset (Local Training)

Datasets	P	F1	Recall
JM1	0.854	0.779	0.823
PC2	0.998	0.997	0.997
MC1	0.991	0.987	0.991
PC5	0.978	0.976	0.978
PC3	0.917	0.866	0.909
PC4	0.915	0.864	0.906
KC1	0.870	0.850	0.874
PC1	0.893	0.858	0.878
CM1	0.899	0.830	0.884
MW1	0.917	0.861	0.906
KC3	0.883	0.850	0.850
MC2	0.869	0.816	0.840

Table 4. Performance of models trained on each dataset (FedSDP)

Datasets	P	F1	Recall
JM1	0.854	0.790	0.825
PC2	0.993	0.989	0.990
MC1	0.988	0.989	0.993
PC5	0.973	0.970	0.974
PC3	0.839	0.800	0.864
PC4	0.909	0.860	0.899
KC1	0.866	0.850	0.871
PC1	0.922	0.910	0.932
CM1	0.880	0.886	0.899
MW1	0.946	0.888	0.925
KC3	0.877	0.837	0.850
MC2	0.872	0.826	0.840

4.4 Privacy of FedSDP

FedSDP has a higher privacy guarantee than traditional centralized software defect prediction model training. However, FedSDP is based on client-server architecture, including the parameter upload and parameter distribution phases. As a result, it may not offer comprehensive and adequate privacy protection for parameters and still faces the risk of information leakage. For example, Lyv et al [34] and Wang et al [35] elaborated on the possible attacks on federated learning. Liu et al [36] analyzed parameter privacy risk in federated learning, where malicious attackers can launch reconstruction and inference attacks based on model parameters.

5 Privacy Protection Optimization

Based on the conclusions of the benchmark analysis, we investigate further and optimize the privacy protection of FedSDP. In this section, we apply partially homomorphic encryption (Paillier) to the FedSDP algorithm to improve its privacy-preserving capability. As described in Section 4, the FedSDP algorithm neglects the protection of the model parameters, and there is a risk of private information leakage.

5.1 Paillier Partially Homomorphic Encryption

Homomorphic Encryption (HE) [37] is a cryptographic method that allows operations on the ciphertext space and the result of the decrypted ciphertext operations is the same

as the result of the plaintext operations. Homomorphic encryption can be classified as Fully HE (FHE), Somewhat HE (SHE), and Partially HE (PHE) based on the supported operations. We use the partially homomorphic encryption algorithm Paillier [38], which has the advantage of ciphertext computation of fully homomorphic encryption and greatly improves the computational efficiency with half the effort. The Paillier encryption algorithm generates the key pair <*pk*, *sk*> in the initialization phase. The public key <*pk*> is used for encryption and can be disclosed to each participating party, and the private key <*sk*> is used for decryption and is kept only in the trusted party without being disclosed. For a given integer *x*. *y*, the Paillier encryption algorithm can perform the following operations.

- Encryption: $\underline{Enc}\ (x, pk) \rightarrow [\![x]\!]$;
- Decryption: $\underline{Dec}\ ([\![x]\!], sk); \rightarrow x$
- Homomorphic addition: $\underline{HAdd}\ ([\![x]\!], [\![y]\!]) \rightarrow [\![z]\!]$, where $[\![z]\!]$ satisfies $\underline{Dec}\ ([\![z]\!], sk) = x + y$;
- Scalar addition: $\underline{SAdd}\ ([\![x]\!], y) \rightarrow [\![z]\!]$, where $[\![z]\!]$ satisfies $\underline{Dec}\ ([\![z]\!], sk) = x + y$;
- Scalar multiplication: $\underline{SMul}\ (x, [\![y]\!]) \rightarrow [\![z]\!]$, where $[\![z]\!]$ satisfies $\underline{Dec}\ ([\![z]\!], sk) = x \times y$.

Similarly, the above operations are also applicable to homomorphic operations between vectors or matrices. It should be noted that in the above operations in cipher space, such as the "$[\![z]\!] = x[\![y]\!]$" expresses the meaning of "$\underline{Dec}\ (\underline{SMul}\ (x, [\![y]\!], sk) = x \times y$", that is, "$z = x \times y$". The result of the decrypted ciphertext space operation is exactly the same as the corresponding plaintext space operation.

5.2 FedSDP with Paillier

From (1), it can be seen that the LR algorithm model gradient calculation involves a large number of complex logarithmic and exponential operations, so to apply the Paillier encryption algorithm, the method of Taylor loss approximation of the original target loss function was used inspired by Hardy et al [39], and the Taylor expansion of the logistic regression original logarithmic loss function $L = \frac{1}{N}\Sigma_{i=1}^{N} \log(1 + e^{-y_i \theta^T x_i})$ was performed to finally obtain the encryption gradient calculation.

$$\left[\left[\frac{\partial L}{\partial \theta}\right]\right] = \frac{1}{n}\sum_{i=1}^{n}\left(\frac{1}{4}\left[\left[\theta^T\right]\right]x_i + \frac{1}{2}\left[[-1]\right]y_i\right)x_i. \tag{2}$$

The gradient operation after Taylor's second-order expansion involves only addition and number multiplication operations, which can be directly applied to the Paillier encryption algorithm for parameter privacy protection. We summarize the secure logistic regression algorithm in Algorithm 2. Before the algorithm model is trained, the server generates a key pair for the selected cryptosystem and shares the public key with the participants, and then the server sends the encrypted ciphertext $[\![m]\!]$ to the participants.

Algorithm 3 summarizes the training process with Paillier: (1) At the beginning of training, we disclose the public key *pk* and make the encrypted initialized model $[\![w^0]\!]$ available to all clients. (2) Each client computes the encryption gradient using the local dataset and performs encryption training. (3) The clients upload the updated model parameters $[\![w^k]\!]$ to the server. (4) The server securely aggregates the model updates from the client to obtain the new global model. Figure 6 shows the algorithm process of FedSDP with Paillier.

Figure 7 compares the performance of FedSDP and FedSDP with Paillier on all participants. The FedSDP with Paillier approach not only encrypts the FedSDP algorithm model training parameters and reduces the risk of privacy leakage, but also shows better performance across all participants, such as the datasets KC3 and CM1 (Figure 7(b)) which both have improved performance. Table 5 shows the F1-score changes of FedSDP and FedSDP with Paillier compared to local training. Experimental results show that although FedSDP with Paillier uses an approximate polynomial Taylor loss function for security gradient calculation, the model performance is approximately lossless. It also added extra security and privacy protection, which is undoubtedly more valuable.

Algorithm 2. Secure logistic regression

Input: m, η, s'

Output: θ
create an homomorphic encryption key pair
send the public key to Client
$\underline{Enc}\ (m, pk) \rightarrow [\![m]\!]$, send $[\![m]\!]$ to Client
$\theta \leftarrow 0$, $\ell_H \leftarrow \infty$
repeat
  for every mini-batch S' **do**
    secure gradient $\nabla \ell_{S'}(\theta)$
    $\theta \leftarrow \theta - \eta\ (\nabla \ell_{S'}(\theta) + \Gamma\theta)$;
  secure logistic loss $\ell_H(\theta)$
  if $\ell_H(\theta)$ stable **then break**
until max epoch
return θ

Algorithm 3. FedSDP with Paillier

Input: E, B, K, η, T, N, n_k, n

Output: $[\![w^T]\!]$, $[\![w_k^T]\!]$

Server:
Initialize w^0, <*pk*, *sk*>;
$[\![w^0]\!] \leftarrow \underline{Enc}\ (w^0, pk)$
for each round t = 0 to T-1 **do**
  $C_t \leftarrow$ (randomly select K out of N clients);
  for each client $k \in C_t$ concurrently **do**
    $[\![w_k^{t+1}]\!] \leftarrow$ **Client**($[\![w^t]\!]$, k, t, pk);

End

// n: total size of dataset; n_k: size of client k's dataset;

$[\![w^{t+1}]\!] \leftarrow \Sigma_{k \in C_t} \frac{n_k}{n} [\![w_k^{t+1}]\!]$;

End

return $[\![w^T]\!]$;

Client($[\![w]\!]$, k, t, pk):

$\beta \leftarrow$(divide local data into batches of size B);

for each local epoch e = 0 to E-1 **do**

for $b \in \beta$ **do**

// encryption parameter $[\![w]\!]$ update;

$[\![w]\!] \leftarrow [\![w]\!] - \eta \nabla \mathscr{L}([\![w]\!], b, pk)$;

end

end

return $[\![w]\!]$;

return

Table 5. F1-score of Local Training, FedSDP, and FedSDP with Paillier

Datasets	Local Training	FedSDP	FedSDP with Paillier
JM1	0.779	**+0.011**	-0.007
PC2	0.997	-0.008	-0.017
MC1	0.987	**+0.002**	**+0.003**
PC5	0.976	-0.006	-0.005
PC3	0.866	-0.066	-0.024
PC4	0.864	-0.004	-0.045
KC1	0.850	**+0.004**	-0.033
PC1	0.858	**+0.052**	**+0.021**
CM1	0.830	**+0.056**	**+0.073**
MW1	0.861	**+0.027**	**+0.018**
KC3	0.850	-0.013	**+0.017**
MC2	0.816	**+0.010**	-0.049

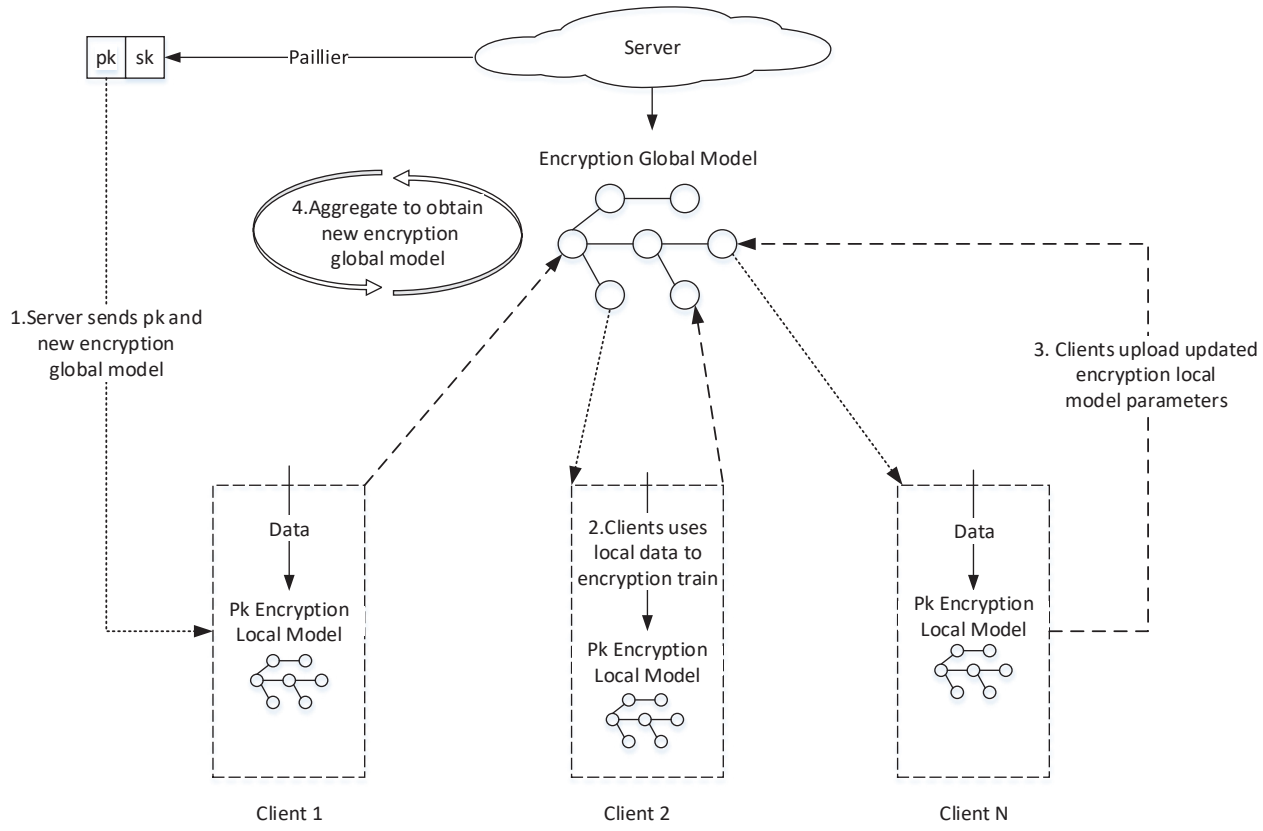

Figure 6. FedSDP with Paillier process

(Each epoch consists of the following steps: (1) The server sends the public key and encrypted global model to the client. (2) The client encrypts the training model using local data. (3) The client uploads the encrypted model parameters to the server. (4) The server aggregates the model updates from the client by weighted average to obtain a new encrypted global model.)

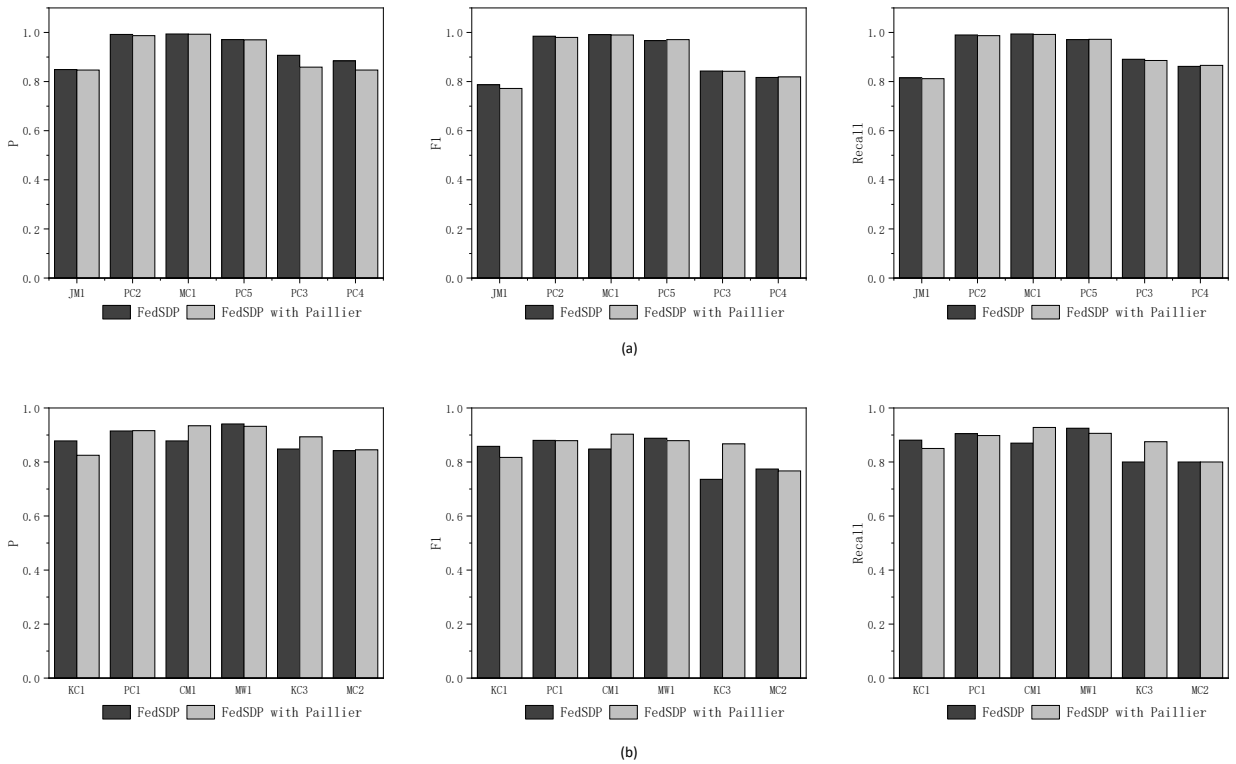

Figure 7. Performance comparison of FedSDP and FedSDP with Paillier with local epochs $E = 5$ and batch size $B = 32$

6 Conclusion

In this paper, we investigate the privacy-preserving challenges of implementing federated learning in software defect prediction by constructing a new benchmark and performing a benchmark analysis. This benchmark defines a horizontal federated software defect prediction scheme. The benchmark analysis presents privacy issues and useful insights into FedSDP. FedSDP can avoid centralized data sharing and protect data privacy, while allowing multiple organizations to cooperate in software defect prediction research, improving the efficiency of data utilization and sample diversity. Integrating software defect data from different organizations can improve the generalization ability and prediction accuracy of the model, while also avoiding the problem of poor model prediction due to the lack of data from some organizations. We propose optimization methods to solve the privacy issues in FedSDP. To tackle the challenges of honest and curious participants and parameter privacy, we employ partially homomorphic encryption (Paillier) to mask the large number of model parameters introduced in the FedSDP algorithm model training. To improve the computational efficiency of the algorithm, a polynomial Taylor expansion is used to approximate the base model loss function so that the target parameters involve only additive and multiplicative operations. Numerical results show that our method effectively protects the privacy of software defect data and solves the problem of data islands under the premise of ensuring performance. This paper focuses only on the privacy preservation of software defect prediction data in the real world. For future work, the challenge of data privacy protection for software defect prediction systems will be considered.

Acknowledgment

This work is sponsored by Natural Science Foundation of Xinjiang Uygur Autonomous Region (2022D01A225), the Xinjiang Key Research and Development Program (2022B01007-1) and the National Natural Science Foundation of China (62241209).

References

[1] B. Custers, A. M. Sears, F. Dechesne, I. Georgieva, T. Tani, S. van der Hof, *EU personal data protection in policy and practice*, TMC Asser Press, 2019.

[2] M. Pandit, D. Gupta, Performance of Genetic Programming-based Software Defect Prediction Models, *International Journal of Performability Engineering*, Vol. 17, No. 9, pp. 787-795, September, 2021.

[3] P. Esperanca, L. Aslett, C. Holmes, Encrypted accelerated least squares regression, *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics*, Fort Lauderdale, Florida, USA, 2017, pp. 334-343.

[4] B. McMahan, E. Moore, D. Ramage, S. Hampson, B. A. y. Arcas, Communication-efficient learning of deep networks from decentralized data, *Proceedings of the 20th International Conference on Artificial Intelligence*

and Statistics, Fort Lauderdale, Florida, USA, 2017, pp. 1273-1282.

[5] K. E. Rao, G. A. Rao, P. S. Rao, A Weighted Ada-Boosting Approach for Software Defect Prediction using Characterized Code Features Associated with Software Quality, *International Journal of Performability Engineering*, Vol. 18, No. 11, pp. 798-807, November, 2022.

[6] J. Wang, B.-J. Shen, Y.-T. Chen, Compressed C4.5 models for software defect prediction, *2012 12th International Conference on Quality Software*, Xi'an, China, 2012, pp. 13-16.

[7] H.-M. Zhu, Y.-P. Wu, A pso algorithm with high speed convergence, *Control and Decision*, Vol. 25, No. 1, pp. 20-24, January, 2010.

[8] P. Subramanian, N. Ramkumar, T. T. Narendran, K. Ganesh, A technical note on 'Analysis of closed loop supply chain using genetic algorithm and particle swarm optimisation', *International Journal of Production Research*, Vol. 50, No. 2, pp. 593-602, 2012.

[9] T. Menzies, J. Greenwald, A. Frank, Data mining static code attributes to learn defect predictors, *IEEE transactions on software engineering*, Vol. 33, No. 1, pp. 2-13, January, 2007.

[10] P. He, B. Li, X. Liu, J. Chen, Y.-T. Ma, An empirical study on software defect prediction with a simplified metric set, *Information and Software Technology*, Vol. 59, pp. 170-190, March, 2015.

[11] W.-B. Mi, Y. Li, M. Wen, Y.-R. Chen, Using active learning selection approach for cross-project software defect prediction, *Connection Science*, Vol. 34, No. 1, pp. 1482-1499, June, 2022.

[12] T. Menzies, A. Butcher, D. Cok, A. Marcus, L. Layman, F. Shull, B. Turhan, T. Zimmermann, Local versus global lessons for defect prediction and effort estimation, *IEEE Transactions on software engineering*, Vol. 39, No. 6, pp. 822-834, June, 2013.

[13] S. Caldas, P. Wu, T. Li, J. Konečný, H. B. McMahan, V. Smith, A. Talwalkar, Leaf: A benchmark for federated settings, December, 2018, https://arxiv.org/abs/1812.01097v1.

[14] J.-H. Luo, X.-Y. Wu, Y. Luo, A.-B. Huang, Y.-F. Huang, Y. Liu, Q. Yang, Real-world image datasets for federated learning, October, 2019, https://arxiv.org/abs/1910.11089v1.

[15] H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, R. Garnett, *Advances in Neural Information Processing Systems 32 (NeurIPS 2019)*, Curran Associates, Inc., 2019.

[16] F. Tramèr, F. Zhang, A. Juels, M. K. Reiter, T. Ristenpart, Stealing Machine Learning Models via Prediction APIs, *USENIX security symposium*, Austin, Texas, USA, 2016, pp. 601-618.

[17] C.-Z. Song, T. Ristenpart, V. Shmatikov, Machine learning models that remember too much, *Proceedings of the 2017 ACM SIGSAC Conference on computer and communications security*, Dallas, Texas, USA, 2017, pp. 587-601.

[18] M. Fredrikson, E. Lantz, S. Jha, S. Lin, D. Page, T. Ristenpart, Privacy in pharmacogenetics: An end-to-end case study of personalized warfarin dosing, *Proceedings of the 23rd USENIX Security Symposium*, San Diego, California, USA, 2014, pp. 17-32.

[19] M. Fredrikson, S. Jha, T. Ristenpart, Model inversion attacks that exploit confidence information and basic countermeasures, *Proceedings of the 22nd ACM Conference on Computer and Communications Security*, Denver, Colorado, USA, 2015, pp. 1322-1333.

[20] R. Shokri, M. Stronati, C.-Z. Song, V. Shmatikov, Membership inference attacks against machine learning models, *2017 IEEE symposium on security and privacy (SP)*, San Jose, California, USA, 2017, pp. 3-18.

[21] R. C. Geyer, T. Klein, M. Nabi, Differentially private federated learning: A client level perspective, December, 2017, https://arxiv.org/abs/1712.07557v1.

[22] Z.-H. Huang, R. Hu, Y.-X. Guo, E. Chan-Tin, Y.-M. Gong, DP-ADMM: ADMM-based distributed learning with differential privacy, *IEEE Transactions on Information Forensics and Security*, Vol. 15, pp. 1002-1012, July, 2019.

[23] L. T. Phong, Y. Aono, T. Hayashi, L.-H. Wang, S. Moriai, Privacy-preserving deep learning via additively homomorphic encryption, *IEEE Transactions on Information Forensics and Security*, Vol. 13, No. 5, pp. 1333-1345, May, 2018.

[24] Y. Li, Software defects prediction based on under-sampling and ensemble algorithm, *Journal of Computer Applications*, Vol. 34, No. 8, pp. 2291-2294, August, 2014.

[25] M. H. Halstead, *Elements of Software Science (Operating and programming systems series)*, Elsevier Science Inc, 1977.

[26] T. J. McCabe, A complexity measure, *IEEE Transactions on software Engineering*, Vol. SE-2, No. 4, pp. 308-320, December, 1976.

[27] D. Radjenović, M. Heričko, R. Torkar, A. Živkovič, Software fault prediction metrics: A systematic literature review, *Information and software technology*, Vol. 55, No. 8, pp. 1397-1418, August, 2013.

[28] V. R. Basili, L. C. Briand, W. L. Melo, A validation of object-oriented design metrics as quality indicators, *IEEE Transactions on software engineering*, Vol. 22, No. 10, pp. 751-761, October, 1996.

[29] M. O. Elish, A. H. Al-Yafei, M. Al-Mulhem, Empirical comparison of three metrics suites for fault prediction in packages of object-oriented systems: A case study of Eclipse, *Advances in Engineering Software*, Vol. 42, No. 10, pp. 852-859, October, 2011.

[30] H. M. Olague, L. H. Etzkorn, S. Gholston, S. Quattlebaum, Empirical validation of three software metrics suites to predict fault-proneness of object-oriented classes developed using highly iterative or agile software development processes, *IEEE Transactions on software Engineering*, Vol. 33, No. 6, pp. 402-419, June, 2007.

[31] Q. Yang, Y. Liu, T.-J. Chen, Y.-X. Tong, Federated machine learning: Concept and applications, *ACM Transactions on Intelligent Systems and Technology (TIST)*, Vol. 10, No. 2, pp. 1-19, March, 2019.

[32] P. Kairouz, H. B. McMahan, B. Avent, A. Bellet,

M. Bennis, A. N. Bhagoji, K. Bonawitz, Z. Charles, G. Cormode, R. Cummings, R. G. L. D'Oliveira, H. Eichner, S. E. Rouayheb, D. Evans, J. Gardner, Z. Garrett, A. Gascón, B. Ghazi, P. B. Gibbons, M. Gruteser, Z. Harchaoui, C.-Y. He, L. He, Z.-Y. Huo, B. Hutchinson, J. Hsu, M. Jaggi, T. Javidi, G. Joshi, M. Khodak, J. Konecný, A. Korolova, F. Koushanfar, S. Koyejo, T. Lepoint, Y. Liu, P. Mittal, M. Mohri, R. Nock, A. Özgür, R. Pagh, H. Qi, D. Ramage, R. Raskar, M. Raykova, D. Song, W.-K. Song, S. U. Stich, Z.-T. Sun, A. T. Suresh, F. Tramèr, P. Vepakomma, J.-Y. Wang, L. Xiong, Z. Xu, Q. Yang, F. X. Yu, H. Yu, S. Zhao, Advances and open problems in federated learning, *Foundations and Trends® in Machine Learning*, Vol. 14, No. 1-2, pp. 1-210, June, 2021.

[33] F. Rahman, P. Devanbu, How, and why, process metrics are better, *2013 35th International Conference on Software Engineering (ICSE)*, San Francisco, California, USA, 2013, pp. 432-441.

[34] L. Lyu, H. Yu, Q. Yang, Threats to federated learning: A survey, March, 2020, https://arxiv.org/abs/2003.02133v1.

[35] J.-Z. Wang, L.-W. Kong, Z.-C. Huang, L.-J. Chen, Y. Liu, C.-X. Lu, J. Xiao, Research advances on privacy protection of federated learning, *Journal of Big Data Research*, Vol. 7, No. 3, pp. 130-149, May, 2021.

[36] Y.-X. Liu, H. Chen, Y.-H. Liu, C.-P. Li, Privacy-preserving Techniques in Federated Learning, *Journal of Software*, Vol. 33, No. 3, pp. 1057-1092, March, 2022.

[37] R. L. Rivest, L. Adleman, M. L. Dertouzos, On data banks and privacy homomorphisms, in: R. A. DeMillo, D. P. Dobkin, A. K. Jones, R. J. Lipton (Eds.), *Foundations of secure computation*, Academic Press, 1978, pp. 169-179.

[38] P. Paillier, Public-key cryptosystems based on composite degree residuosity classes, *International Conference on the Theory and Applications of Cryptographic Techniques*, Prague, Czech Republic, 1999, pp. 223-238.

[39] S. Hardy, W. Henecka, H. Ivey-Law, R. Nock, G. Patrini, G. Smith, B. Thorne, Private federated learning on vertically partitioned data via entity resolution and additively homomorphic encryption, November, 2017, https://arxiv.org/abs/1711.10677v1.

Biographies

Ying Liu received the B.S. degree in Network Engineering from Xinjiang Normal University. He is currently a graduate student at Xinjiang Normal University. His research interests include Machine Learning and Software Reliability Engineering.

Yong Li received the Ph.D. degree in Computer Science from Nanjing University of Aeronautics and Astronautics in 2018. He is currently an Associate Professor of Xinjiang Normal University. His research interests include Machine Learning and Intelligent Software Engineering.

Ming Wen graduated from Xi'an University of Technology in 1988 with a major in automatic control. Now he is the director and researcher of software development and testing center of Xinjiang Electronic Research Institute. His research interests include Software Engineering and Artificial Intelligence.

Wenjing Zhang received the B.S. degree in Software Engineering from Xinjiang Normal University. She is currently a graduate student at Xinjiang Normal University. Her research interests include Software Data Science and Incremental Learning.