electronics

Article

Cross-Project Software Defect Prediction Using Differential Perception Combined with Inheritance Federated Learning

Aili Wang [1], Yanxiang Feng [1], Mingji Yang [1], Haibin Wu [1], Yuji Iwahori [2] and Haisong Chen [3,*]

[1] Heilongjiang Province Key Laboratory of Laser Spectroscopy Technology and Application, Harbin University of Science and Technology, Harbin 150080, China; aili925@hrbust.edu.cn (A.W.); 2220610131@stu.hrbust.edu.cn (Y.F.); yangmingji@hrbust.edu.cn (M.Y.); woo@hrbust.edu.cn (H.W.)
[2] Department of Computer Science, Chubu University, Kasugai 487-8501, Japan; iwahori@isc.chubu.ac.jp
[3] School of Undergraduate Education, Shenzhen Polytechnic University, Shenzhen 518115, China
* Correspondence: hschen@szpu.edu.cn

Abstract: Cross-project software defect prediction (CPDP) refers to the construction of defect prediction models by collecting multi-source project data, but the heterogeneity of data among projects and the modern problem of "data islands" hinder its development. In response to these challenges, we propose a CPDP algorithm based on differential perception combined with inheritance federated learning (FedDPI). Firstly, we design an efficient data preprocessing scheme, which lays a reliable data foundation for federated learning by integrating oversampling and optimal feature selection methods. Secondly, a two-stage collaborative optimization mechanism is proposed in the federated learning framework: the inheritance private model (IPM) is introduced in the local training stage, and the differential perception algorithm is used in the global aggregation stage to dynamically allocate aggregation weights, forming positive feedback for training to overcome the negative impact of data heterogeneity. In addition, we utilize the Ranger optimization algorithm to improve the convergence speed and privacy security of the model through its irreversible mixed optimization operation. The experimental results show that FedDPI significantly improves predictive performance in various defect item data combination experiments. Compared with different deep learning and federated learning algorithms, the average improvement in AUC and G-mean indicators is 0.2783 and 0.2673, respectively, verifying the practicality and effectiveness of federated learning and two-stage collaborative optimization mechanisms in the field of CPDP.

Keywords: cross-project software defect prediction; federated learning; inheritance private model; differential perception

check for updates

Citation: Wang, A.; Feng, Y.; Yang, M.; Wu, H.; Iwahori, Y.; Chen, H. Cross-Project Software Defect Prediction Using Differential Perception Combined with Inheritance Federated Learning. *Electronics* **2024**, *13*, 4893. https://doi.org/10.3390/electronics13244893

Academic Editors: Krzysztof Szczypiorski and Ping-Feng Pai

Received: 8 November 2024
Revised: 7 December 2024
Accepted: 10 December 2024
Published: 11 December 2024

Copyright: © 2024 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (https://creativecommons.org/licenses/by/4.0/).

1. Introduction

With the rapid development of intelligence and digitization in modern society, software is constantly iterating with the advancement of technology and people's needs. The scale and complexity of software are constantly expanding, which puts forward greater requirements for software quality while facilitating public life. Software quality is an important indicator to measure software reliability and ensure user experience. Poor software quality often causes huge economic losses to enterprises, and even endangers people's lives in serious cases [1]. Software defects are the main factors affecting software quality, and refer to problems, errors, or hidden functional defects in programs that disrupt normal operating capabilities [2]. In the process of writing software code, programmers do not have a fixed writing approach or format, and their writing habits vary from person to person without clear norms [3]. Therefore, subjective factors such as lack of work experience and inadequate understanding of requirements can lead to software defects.

With the development of artificial intelligence, software defect prediction (SDP) technology provides a feasible strategy for this purpose. As one of the scientific methods for quality assurance in modern software engineering, SDP significantly improves the

efficiency of defect detection in the software development process and reduces labor costs by training effective predictive models [4]. Its core value lies in its ability to accurately locate potential defect modules during software development, helping development teams allocate testing resources and optimize code in a targeted manner. By integrating machine learning, data mining, and statistical methods, SDP technology can extract code features from historical data of software projects, thereby constructing high-precision defect prediction models. Therefore, overall, SDP can achieve optimization of the allocation of testing resources, reductions in the cost of defect module detection, and improvements in software development efficiency and overall quality. In today's rapidly developing software world, SDP is even more important for modern large-scale software because there are numerous program modules and the programs are more complex and diverse. If defects cannot be quickly and accurately identified, the resource cost required for subsequent testing will increase dramatically over time [5]. SDP provides an efficient detection tool for this, transforming software quality-related work from passive defect repair to active risk prevention. With the continuous progress of research, SDP technology is gradually becoming an important solution in software quality work and a hot direction for innovative development in the field of software engineering in the future.

Early research on SDP mainly focused on within-project defect prediction (WPDP), which constructs prediction models by mining historical data of the same project as training data. However, for initial or lightweight projects, the lack of historical data has become a common problem, and there is a lack of sufficient generalization for new projects. Cross-project defect prediction (CPDP) [6] compensates for the problem of insufficient training data by mining multi-source project-related data. This is because there are similarities in software development processes, compilation rules, etc., and the existing large amount of data must have referenceable parts. However, due to differences in data distribution, measurement methods, and even programming languages among different project data, this data heterogeneity [7] still leaves room for optimization in the CPDP method. In recent years, researchers have proposed many improved algorithms for CPDP. For instance, Jing et al. proposed unified metric representation (UMR) for source and target project data [8], and based on this, introduced canonical correlation analysis (CCA) [9] into CPDP to make the data distribution of the target project close to that of the source project. However, the machine learning methods all involve the sharing of data between the source and target projects, which can easily lead to privacy leakage in practical scenarios. In the era of big data, society is gradually realizing the importance of data ownership. With the improvement of privacy- and security-related laws, the owners of software defect data only allow such data to be stored privately and cannot share them freely, thus forming "data islands" [10].

Federated learning (FL) emerged from the demands for privacy protection and data sharing. Google proposed this concept in 2016, which involves training models in the local environment of each participant and uploading encrypted model parameters to a central server for aggregation. By integrating the training knowledge of all parties, it indirectly increases the amount of training data and achieves joint modeling of distributed datasets without directly sharing original data. This method effectively solves the problems of privacy security and data islands, providing new ideas for cross-project data collaboration and knowledge sharing.

In this paper, we propose an algorithm framework based on differential perception and inheritance federated learning applied to CPDP, called FedDPI, aimed at addressing the client and model drift problems caused by data heterogeneity in federated learning. The core idea of this framework is to improve and integrate local training and global aggregation separately, forming a two-stage collaborative optimization that efficiently enhances model performance while achieving privacy protection. The contribution of this paper can be summarized as follows.

1. Our method proposes applying improved federated learning to CPDP. Federated learning allows multiple data participants to jointly participate in training and build

local models that meet their own data needs without exposing their private data. In the case of limited participant datasets, it indirectly increases the amount of training data and solves the problem of "data islands" in the era of big data.

2. By integrating oversampling and optimal feature selection methods, the problem of extreme class imbalance and uneven feature dimensions in the data is solved, providing a comprehensive data foundation for the entering of federated learning.
3. Our method innovatively proposes a two-stage collaborative optimization mechanism. Unlike most previous studies, this mechanism fully utilizes the characteristics of federated learning at different stages and provides two different improvements for each stage. Adding inherited private models (IPMs) during the local training phase solves the knowledge-forgetting problem caused by model drift. In the global aggregation stage, the difference-aware algorithm is introduced to alleviate the negative impact of client drift. The two improvements cooperate fully to form positive training feedback, effectively improving the predictive performance and convergence speed of the model.
4. A Ranger optimization algorithm is introduced as a privacy protection strategy. Its advanced optimization approach and irreversible hybrid computing method not only improve convergence speed but also effectively defends against network security attacks, ensuring the privacy and security of participants. Through the combination of multiple software projects, we constructed a diverse dataset and distributed it to multiple clients for experimentation. The results showed that our algorithm performed well in all data allocation combinations, significantly improving the performance of defect prediction and verifying the effectiveness of the two-stage collaborative optimization mechanism and privacy protection strategy.

The structure of the rest of this paper is as follows. Section 2 will provide a detailed introduction to the algorithm framework and its steps, Section 3 will present the experimental design, including experimental data, evaluation indicators, and result analysis, and finally, Section 4 will discuss the experimental setup and summarize the entire article.

2. Related Works

To the best of our knowledge, Briand et al. [11] were the first to conduct research on machine learning methods for software defect prediction, analyzing whether the CPDP model is theoretically feasible. They analyzed whether the CPDP model was theoretically feasible, but their method performed poorly in terms of results. After years of development in the SDP field, many improved machine learning algorithms have been derived to achieve more efficient defect prediction performance. In addition, due to the emergence of data silos, the application of federated learning in the SDP field has become a popular direction. Therefore, existing related research can be roughly divided into two categories: machine learning methods and federated learning methods.

The advantage of machine learning lies in its ability to automatically extract features without relying on manually designed feature engineering, which makes it perform well in handling defect data and more efficiently than manual detection. In recent years, an increasing number of improved and intersecting machine learning techniques have been used in the SDP field. For example, some studies have proposed models based on convolutional neural networks (CNNs) [12] and long short-term memory (LSTM) [13] that can effectively handle the structural information and time series features of software code. In addition, machine learning strategies such as transfer learning [14] and ensemble learning [15] have also been introduced to enhance the adaptability of models across different projects and domains. These methods not only improve the accuracy of defect prediction but also provide new perspectives and methodologies for research in the field of software engineering.

Wu et al. established an intermediate domain from the target domain to the source domain [16] to break the distribution gap and narrow the difference between the source and target domains. Jin et al. proposed the kernel twin support vector machine (KTSVM)

method, which uses domain adaptation (DA) to match the training data distribution differences between source and target items, thereby improving the effectiveness of cross-project prediction [17]. Aiming at the problem of class imbalance and incomplete labeling in software defect data, Shen et al. proposed a semi-supervised software defect prediction method based on sampling and integration, aiming to improve model performance by enhancing the representativeness of minority samples [18]. Tong et al. proposed a method that combines kernel spectrum embedding, transfer learning, and ensemble learning to improve defect prediction performance between heterogeneous projects by capturing potential common features between the source and target datasets [19]. Bal et al. proposed a data transfer and correlated measure matching method for heterogeneous software defect prediction, which effectively transmits source project data by selecting correlation measures between source and target projects, reducing the differences in cross-project data distribution [20].

Šikić et al. proposed a source code defect prediction method based on a graph convolutional neural network (GNN). By extracting seven types of node features from the source code and representing them in a graph structure, a graph neural network was used to capture complex structural and semantic information in the code, thereby improving prediction performance [21]. Pornprasit et al. proposed a deep learning model called DeepLineDP, which effectively improves the accuracy of defect prediction, especially in line-level defect prediction tasks, by converting code lines into word vectors and using a bidirectional recurrent neural network (BiRNN) to learn the context information of code lines [22]. Abdu et al. proposed a deep learning method that integrates semantic features and traditional features. CNN is used to process semantic features extracted from the abstract syntax tree (AST) of the project, while a multilayer perceptron (MLP) is used to process traditional features extracted from the dataset. The CNN-MLP fusion model can effectively handle structured and unstructured code data, thereby achieving higher prediction performance [23].

At present, there are more and more industries holding sensitive information. If large-scale data are needed for model training, there may be a risk of information leakage during data collection and transmission [24]. Due to the emergence of data privacy laws, more and more researchers are paying attention to the solutions provided by federated learning, which allows multiple participants to collaboratively train neural network models without sharing raw data, effectively protecting data privacy. Since Google proposed it, federated learning has continuously developed and improved in different fields, demonstrating enormous potential for application and academic value. Wang et al. proposed the knowledge distillation-based federated transfer learning method (FTLKD) [25], which alleviates the problem of insufficient training data in practical scenarios and improves the performance of heterogeneous defect prediction models. Chen et al. proposed a federated transfer learning (FTL) framework based on discrepancy-based weighted federated averaging (D-WFA) [26], which utilizes locally labeled source domain data and unlabeled target domain data to update multiple local models with generalization ability, in order to collaboratively train a good global model and protect data privacy. Wang et al. proposed a federated reinforcement learning method based on gradient clustering called FRLGC [27], and used Gaussian differential privacy to encrypt parameters, which avoided estimation redundancy and low-value prediction while protecting privacy. In order to reduce local learning bias caused by data heterogeneity, Guo et al. proposed a federated learning on heterogeneous data via local learning bias reduction (FedBR) model, which improves model performance by balancing local classification outputs and learning local features that are similar to global features [28]. Shu et al. adaptively adjusted the threshold using the concept of self-paced learning by capturing different data distributions between clients [29], in order to filter the training data of each client and select appropriate clients to participate in each round of global aggregation. Lee et al. combined the concepts of global alignment and local generalization to propose a framework called FedSoL, which effectively reduces conflicts between local targets of different clients and improves local model performance

by introducing proximal constraints in local training [30]. Wang et al. proposed a federated prototype learning method (FPLPA) applied to CPDP that combines federated learning and prototype learning. By constructing a prototype network to generate local prototypes for training, it alleviates the problem of data heterogeneity while effectively protecting data privacy [31]. Nandi et al. proposed a Docker-based federated learning framework and developed Docker-supported federated learning (DFL) by utilizing client independent technologies such as Docker containers to simplify the deployment of federated learning frameworks for data stream processing on heterogeneous clients [32]. Lu et al. proposed an adaptive asynchronous federated learning scheme called FedAAM, which combines an adaptive weight allocation algorithm with a new asynchronous federated learning framework to improve the convergence speed and performance of the model by dynamically adjusting the training frequency and update strategy of the client, fully utilizing the computing resources of each client without affecting the performance of the global model [33].

On the other hand, when all edge computing devices build a global model together, the heterogeneous private datasets of a large number of devices will lead to the deterioration of model quality in the training process. EK et al. proposed a new aggregation algorithm called FedDist (federated distance), which can identify differences between specific neurons and modify their model structure [34]. Pang et al. proposed an intelligent central server that can recognize heterogeneity and help guide most clients to achieve better performance [35]. To address the issue of data distribution differences among local clients, Sattler et al. utilized the geometric properties of the FL loss surface to divide participants into clusters with jointly trainable data distributions [36]. Zhang et al. proposed a data-free knowledge distillation method, FedFTG, which achieves global model fine-tuning through knowledge transfer under data-free conditions. Custom label sampling and difficult sample mining are used to address data distribution differences between clients, thereby improving the generalization performance of the global model under data heterogeneity [37]. Chen et al. used hyper-knowledge distillation to improve the accuracy of the global model by extracting and aggregating the mean representation of the client and soft prediction (i.e., hyper-knowledge), achieving an effective combination of personalized and global models while avoiding dependence on public datasets [38]. Acun et al. proposed a dynamic regularization framework called FedDyn, which enhances the accuracy of global model aggregation by adding personalized regularization terms for local model training while reducing communication costs [39]. Sittijuk et al. proposed a new aggregation algorithm, Fed-RHLP, which increases the probability of high-performance clients being selected by a roulette strategy and allows low-performance clients to participate in global model aggregation proportionally according to their local performance. This method improves the convergence speed and final accuracy of the model, especially in the case of data heterogeneity [40]. At present, the mainstream research directions of federated learning mostly focus on improving local training or global aggregation schemes, and have achieved considerable leading results. Chen et al. proposed a federated meta-learning algorithm based on efficient communication, called ECFML, which combines lightweight MobileViT networks with model agnostic meta-learning (MAML) algorithms to reduce unnecessary iterations and parameter count, thereby improving the efficiency of federated communication [41].

3. Materials and Methods

Figure 1 shows the algorithm framework of FedDPI applied to CPDP, which mainly includes four parts: data preprocessing, local inheritance training, global difference aggregation, and privacy protection strategy. In the data preprocessing stage, the SMOTETomek algorithm is used for comprehensive oversampling, removing noisy samples and achieving an approximate balance of class distribution. Then, the kernel principal component analysis (KPCA) algorithm is used to extract the optimal features of the dataset, preparing it for input into convolutional neural networks (CNNs). Before the first round of federated learn-

ing training, each client calculates the difference between local and global data distribution and uploads it to the central server. The server allocates aggregation weights through a difference-aware collaborative algorithm. In the local training phase, the client sends the preprocessed data to the newly downloaded global model and introduces inherited private models (IPMs) to jointly train the training model knowledge and private data from the historical rounds. In the global aggregation stage, the server updates the global model based on the difference aggregation weights obtained in the first round. Finally, the Ranger algorithm is adopted for iterative optimization of the model, which improves convergence speed through collaborative optimization while effectively avoiding network security attacks and reducing privacy leakage risks. The collaborative improvement measures of local training and global aggregation form positive feedback for training, thereby efficiently enhancing model performance.

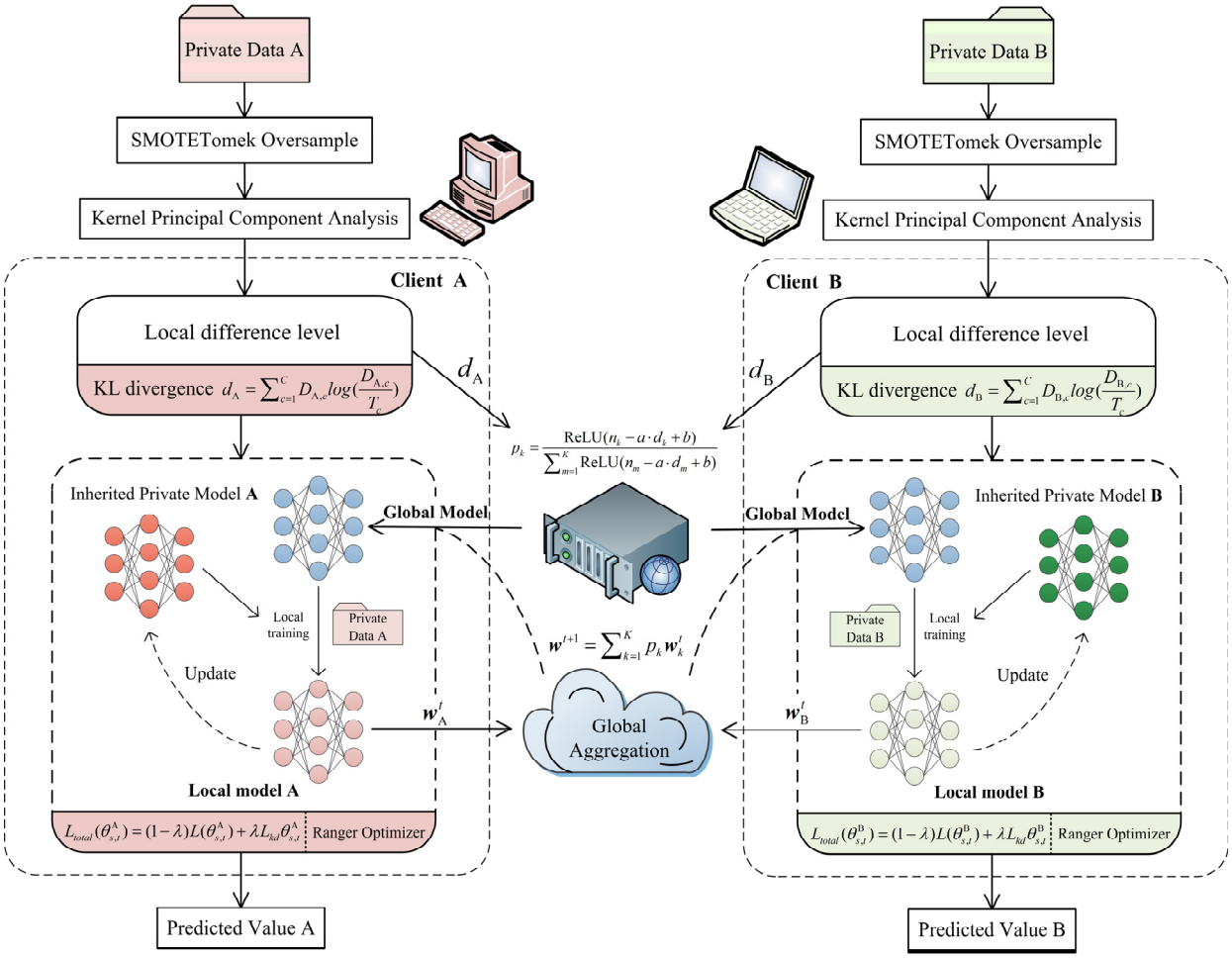


Figure 1. The proposed algorithm framework of FedDPI applied to CPDP.

3.1. Data Denoising Based on SMOTETomek Algorithm

The SMOTETomek algorithm achieves the dual goals of data denoising and class distribution balance through the collaboration of oversampling and undersampling methods. The specific process is as follows.

Firstly, SMOTE oversampling is performed on the sample data. The core idea is to use the k-nearest neighbor algorithm to find neighboring samples for all minority class samples, and then perform linear random interpolation to synthesize new samples. The interpolation positions are randomly selected, and the number of interpolation points for each sample point is equal, as shown in Figure 2. The circle represents the majority class sample, and the asterisk represents the minority class sample.

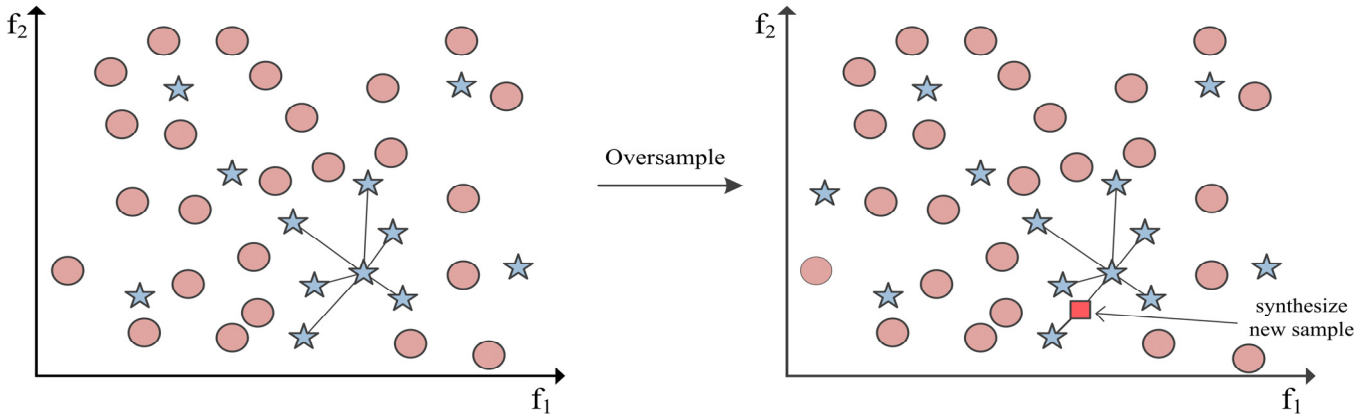


Figure 2. SMOTE oversampling algorithm.

However, SMOTE oversampling has limitations: newly synthesized samples will overlap with most of the surrounding class samples, becoming new overlapping noise samples and increasing classification difficulty. Therefore, it is necessary to clean the samples after oversampling.

To meet the requirements of data cleaning, the SMOTETomek algorithm adopts Tomek link undersampling. This method marks two adjacent and different categories of samples as a pair of Tomek links, considering that these samples either contain noisy data or are located near the category boundary. By removing Tomek link, overlapping noise samples between classes can be "cleaned" to ensure that the nearest neighbors belong to the same category, making the classification boundary clear, as shown in Figure 3. On the other hand, using Tomek link undersampling alone may result in the loss of valuable data information for small-sample data owners, affecting the effectiveness of model training. Therefore, the SMOTETomek algorithm cleverly solves the problems of SMOTE oversampling, introducing new noise and Tomek link undersampling, causing information loss. The two methods independently form a collaboration, achieving a balance between positive and negative class samples while retaining most of the data information, reducing the interference of noisy data on the classifier, and making the classification boundary clearer. This comprehensive method demonstrates significant advantages in handling imbalanced datasets, laying a solid foundation for subsequent prediction tasks.

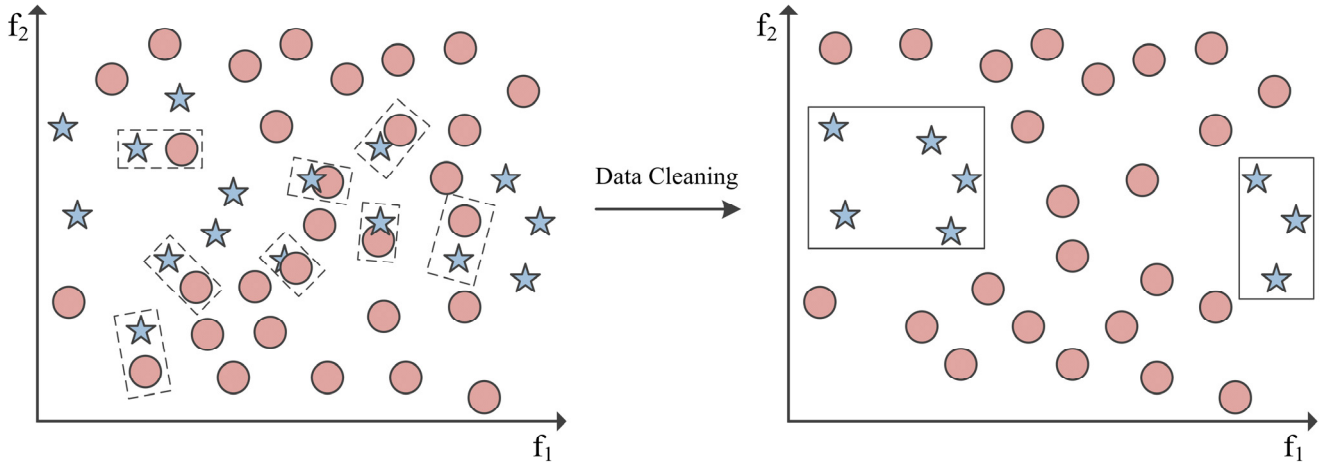


Figure 3. Tomek link undersampling algorithm.

3.2. Optimal Feature Selection Combined with KPCA

In complex software systems, there are often numerous metrics included to ensure the smooth operation of the software, which are referred to as the measurement elements of the

software project. The measurement elements used in different software projects are different. For example, a NASA project [42] uses 37 measurement elements to measure software modules, while an AEEEM software project [43] uses 61 measurement elements. Therefore, these historical software project data belong to high-dimensional data, which contain many irrelevant and redundant features that do not significantly affect the model prediction performance. This not only increases computation time but also reduces classification accuracy. Therefore, after comprehensive oversampling, the kernel principal component analysis (KPCA) algorithm is used to screen the features of software defect data and obtain the optimal feature subset.

The KPCA algorithm is a nonlinear extension of the traditional PCA algorithm, specifically designed for mining complex nonlinear data that cannot be fully captured by linear methods. PCA relies on the feature decomposition of the covariance matrix to extract the main features [44], as shown in Formula (1):

$$\mathbf{x}\mathbf{x}^{\mathrm{T}}\mathbf{w} = \lambda\mathbf{w} \Rightarrow \left(\sum_{i=1}^{m} x_i x_i^{\mathrm{T}}\right)\mathbf{w} = \lambda\mathbf{w} \tag{1}$$

where x_i is the eigenvector of the i-th sample, λ is the corresponding eigenvalue, and w is the eigenvector direction to be solved. The premise of applying PCA is to assume that there is a linear hyperplane, but when the data have a nonlinear structure, the effect of PCA is limited. The idea of the KPCA algorithm is to first map the original data from n-dimensional to linearly separable high-dimensional $N > n$, and then use PCA to reduce the N-dimensional to a low-latitude $n'(n' < n < N)$, thereby completing the dimensionality reduction process. Based on Formula (1), the high-dimensional mapping and feature decomposition of KPCA are shown in Formula (2):

$$ZZ^{\mathrm{T}}\mathbf{W} = \phi(X)\phi(X)^{\mathrm{T}}\mathbf{W} \tag{2}$$

where $Z = \phi(X)$ is the image of the whole sample X mapped to the high-dimensional space. Since the mapping function $\phi(\cdot)$ of the high-dimensional feature space is difficult to solve, by introducing the kernel function $K = \phi(X)\phi(X)^{\mathrm{T}}$, the direct calculation can be avoided and the computational complexity can be greatly reduced.

$$K\mathbf{W} = \lambda\mathbf{W} \tag{3}$$

The radial basis function kernel is one of the most common and effective kernel functions. Its mathematical expression is written as follows:

$$K(x_i, x_j) = \exp\left(-\frac{\|x_i - x_j\|^2}{2\sigma^2}\right) \tag{4}$$

where σ controls the shape of kernel function. The kernel function itself does not explicitly define the high-dimensional feature space, but realizes the mapping of data from low-dimensional to high-dimensional feature space through kernel techniques. Therefore, the complexity of the model can further adjust the parameters to better adapt to different data scenarios and requirements.

KPCA preserves nonlinear feature information in high-dimensional feature space, effectively capturing complex patterns in data and greatly increasing the scope of feature selection. The software defect dataset comprises typical nonlinear separable data. Through KPCA dimensionality reduction, the model can find key features in the complex nonlinear feature space, thereby improving the model's generalization ability and prediction accuracy. In addition, a significant advantage of using the KPCA algorithm is that it can make the feature dimensions of different datasets consistent, ensuring that the local model structure of each participant is the same, providing a more stable foundation for subsequent federated training.

3.3. Differential Perception Combined with Inheritance Federated Learning Algorithm

Personal federated learning (PFL) divides each round of global communication into two stages, local training and global aggregation, and continues multiple iterations until the model converges. Due to the heterogeneity of multi-source client data, global aggregation often leads to significant performance degradation of the model, resulting in poor performance of the global model on local datasets. However, the global model will directly overwrite the existing personalized model on the client, resulting in the loss of learned training knowledge and further causing performance degradation [45]. This negative feedback after iteration may lead to catastrophic consequences. This phenomenon stems from two main issues: client drift in local training and model drift in global aggregation [46]. To address these issues, FedDPI innovatively proposes a two-stage collaborative optimization mechanism. This mechanism integrates improvement and collaboration in two stages: local training and global aggregation. In the local stage, an inherited private model is introduced to fully preserve and utilize historical training knowledge, reducing the impact of client drift. In the global aggregation stage, the differential aggregation-aware algorithm is adopted to enable the server side to adapt to the heterogeneity of data from different clients during model aggregation, alleviating the problem of model drift. Effective training positive feedback is formed between the two stages, enabling global and local models to collaborate and optimize, complementing each other and efficiently improving model convergence speed and prediction accuracy. This mechanism solves the performance degradation problem that can only be avoided in traditional federated learning by fully combining the advantages of global and local models, and improves the learning effect across client datasets.

3.3.1. Local Training Stage of FedDPI

During the local training phase, client drift can lead to knowledge-forgetting issues. Figure 4 shows this phenomenon in the federal average algorithm (FedAvg) [47]. For visualization purposes, the local loss curves of the two clients are plotted separately in the figure to avoid overlap. After t rounds of local training, the local loss of the client significantly decreased. However, before the start of the t+1 round of training, the newly downloaded global model caused an increase in local loss compared to the previous round. This indicates that the performance of the global model obtained by the federal average is poor, but it directly covers the local model on the client that retains the effective training knowledge from the previous round, forcing the client to start training the global model from scratch in a new round. The problem of forgetting historical model knowledge seriously hinders the training effectiveness and convergence speed of the model.

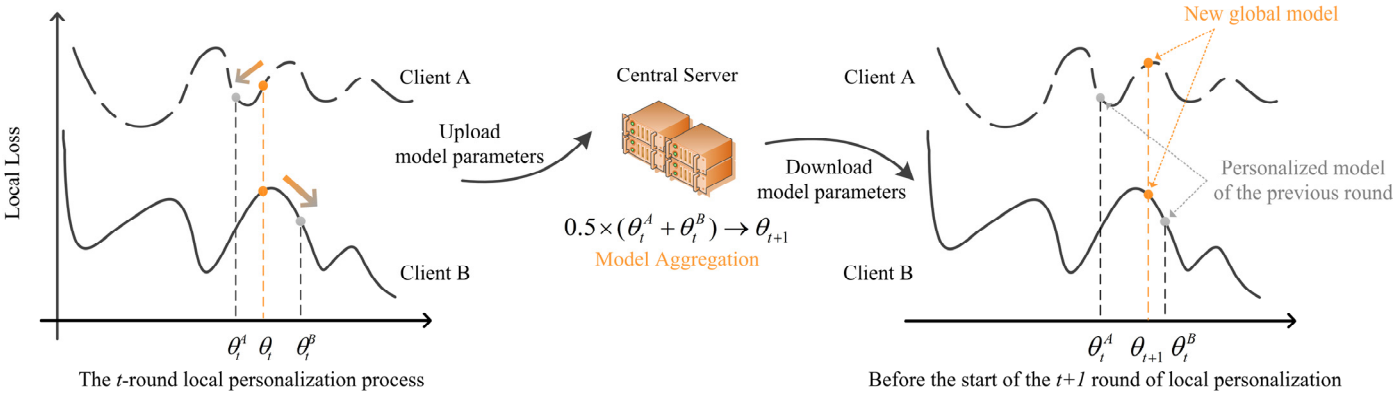


Figure 4. The process visualization of the knowledge-forgetting problem in the FedAvg algorithm.

To address this issue, this chapter proposes a strategy of introducing inherited private models (IPMs) during the local training phase. IPMs can dynamically average the local models in the second round of historical training and save them in the client, then use these

to "supervise" the local "personalization" process of the global models downloaded in the new round. IPMs are a new type of private model in federated learning that can inherit the historical training knowledge of the local model, preserve and transfer valuable results, fully utilize the advantages of time integration, and alleviate the knowledge-forgetting problem in PFL, thereby improving training effectiveness and enhancing model stability. An IPM exists on each client, independent of the local model, participates in training together with private data, and updates personalized models based on the training results.

More specifically, assuming there are K distributed clients, each with a different data distribution, the optimization objective of the k-th client can be rewritten from the general PFL objective function as follows:

$$\min_{\theta^k} F^k\left(\theta^k\right) \triangleq E_{(x^k,y^k)\sim D^k}\left[\ell\left(f\left(x^k,\theta^k\right),y^k\right)\right] \tag{5}$$

where $f\left(\cdot;\theta^k\right)$ is a prediction function based on the model parameter θ^k, $f(x^k;\cdot)$ returns the "logits" before the softmax operation of the neural network, and $\ell(\cdot;\cdot)$ is a loss function, such as a cross-entropy loss function. In this paper, all the model parameters are divided into the global shared parameter θ_s and the private parameter $\left\{\theta_p^k\right\}_{k=1}^K$ of the local client. At the beginning of the t-th round of training, the client downloads the global model: $\theta_{s,t}^k \leftarrow \theta_{s,t}$. According to the above, the IPM aggregates historical model knowledge, and its performance is better than the newly downloaded global model. Therefore, during local training, the IPM transferred the knowledge contained in the output layer through knowledge distillation (KD) [48]. KD can guide the output of the global model to approach the output of the IPM in the form of Equation (6) to achieve the purpose of knowledge migration:

$$L_{kd}(\theta_{s,t}^k) = \tau^2 \frac{1}{B}\sum_{i=1}^{B} D_{KL}\left(\frac{\sigma\left(g_{p,i}\right)}{\tau}\right) \parallel \frac{\sigma\left(g_{s,i}\right)}{\tau}\right) \tag{6}$$

where $L_{kd}(\cdot)$ is the distillation loss function, D_{KL} is the KL divergence, $\sigma(\cdot)$ represents the softmax operation, τ is the temperature, B is the batch data size, and $g_{p,i}, g_{s,i}$ are the "logit" outputs of the samples through the IPM and the global model, respectively. In this paper, asymmetric knowledge distillation is adopted. An IPM is used as the teacher model. The distillation loss function constitutes the total loss function through Formula (7) and private data training loss, and realizes collaborative training to generate personalized model $\hat{\theta}_{s,t}^k$, which replaces the strategy of the global model directly covering the previous round of personalized model in standard federated learning.

$$L_{total}\left(\theta_{s,t}^k\right) = (1-\lambda)L\left(\theta_{s,t}^k\right) + \lambda L_{kd}\left(\theta_{s,t}^k\right) \tag{7}$$

Here, λ is used to adjust the weight of knowledge transfer. After the local training is completed, the IPM is updated by Equation (8), and the t-th round model knowledge is aggregated to supervise the t + 1-th round training process. When the initial convergence of the model is detected, the IPM stops updating, retains the existing historical model knowledge, and no longer aggregates new personalized models to reduce the computational overhead of this method.

$$\theta_{p,t+1}^k \leftarrow \left(1-\mu_t^k\right)\hat{\theta}_{s,t}^k + \mu_t^k \theta_{p,t}^k \tag{8}$$

Equation (8) retains the personalized model knowledge of the t-th round through dynamic averaging, where $\theta_{p,t}^k$ denotes the IPM parameter of the t-th round, $\mu_t^k \in [0,1]$ is the momentum coefficient of the k-th client, defined as $\mu_t^k = \mu \times (t-1)/T$, T is the global maximum number of training rounds, μ is the control variable, and the default value is

0.9. When $\mu_t^k = 0$, the IPM abandons the model knowledge in the historical rounds and only retains the personalized model $\hat{\theta}_{s,t}^k$ of the current round. When $\mu_t^k = 1$, the IPM is no longer updated, retains the existing historical model knowledge, and degenerates into an independent private model. Figure 5 shows the local training process of FedDPI.

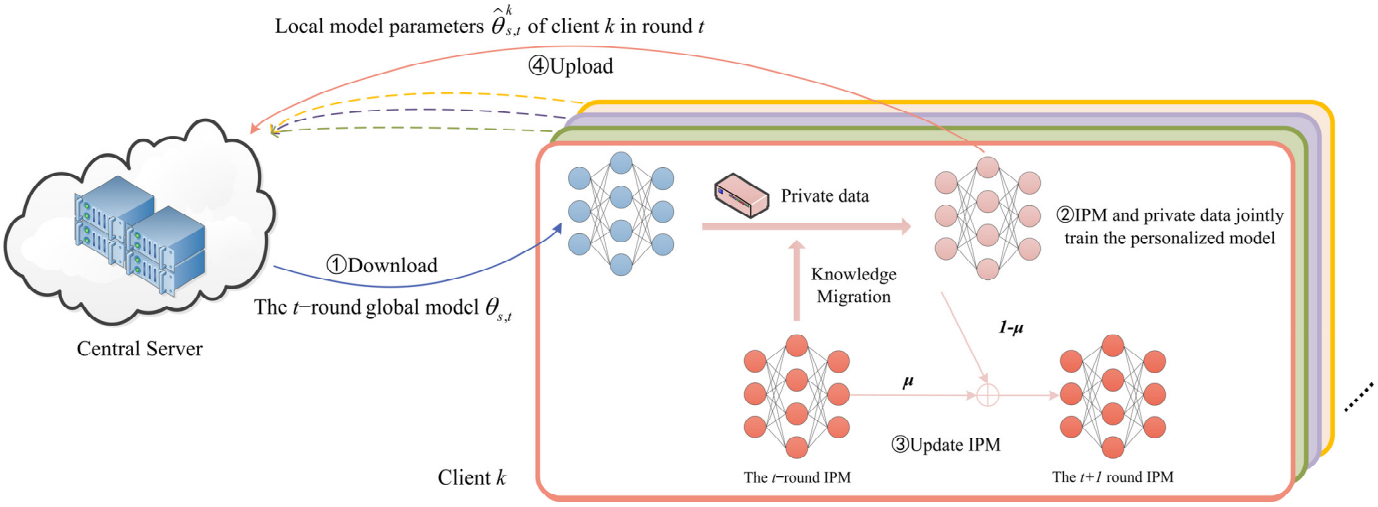


Figure 5. The local training process of FedDPI.

3.3.2. Global Aggregation Stage of FedDPI

Introducing the inheritance private model algorithm during the local training phase can effectively alleviate the problem of knowledge forgetting and improve the performance of the local model. However, the global aggregation stage of data heterogeneity remains a key challenge. Due to the use of private data with different distributions by different participants, model optimization is carried out for different local objectives, resulting in inconsistent optimization directions. Therefore, it is difficult to obtain a high-performance global model through simple federated averaging aggregation. To substantially alleviate the negative impact of data heterogeneity, a new global aggregation scheme should be designed. Therefore, this chapter introduces the differential perception collaboration (DC) algorithm in the global aggregation stage, aiming to dynamically allocate aggregation weights to each client based on their data distribution differences to minimize the decline in global model performance. At the same time, the algorithm forms positive feedback with the local training process, forming a two-stage collaborative optimization, thereby further improving the overall model performance and convergence speed.

The traditional aggregation weight allocation method mainly measures based on the size of the dataset, but this standard cannot reflect the differences in the distribution of client data, thus lacking sufficient information to allocate weights reasonably. This method not only considers the size of the dataset but also incorporates the level of difference between local and global data distributions into the aggregation weight calculation. The lower the difference value, the more similar the distribution between the private data of the local client and the assumed global data, indicating that the client should make a greater contribution to global aggregation. Therefore, assigning higher aggregation weights to clients with larger datasets and smaller differences is the best aggregation solution.

The process of calculating the local difference level for each client is shown in Figure 6, assuming there are K participants and the dataset contains C classes with different inter-class probability distributions. In the first round of communication, each client calculates the difference between its local data distribution and the global data distribution. Please note that this article assumes that the global data distribution is uniformly distributed, as uniform distribution can naturally promote fairness among all categories and the generalization ability of the global model. In addition, the difference calculation is completely

completed locally without the need for additional data sharing, effectively avoiding the risk of privacy leakage.

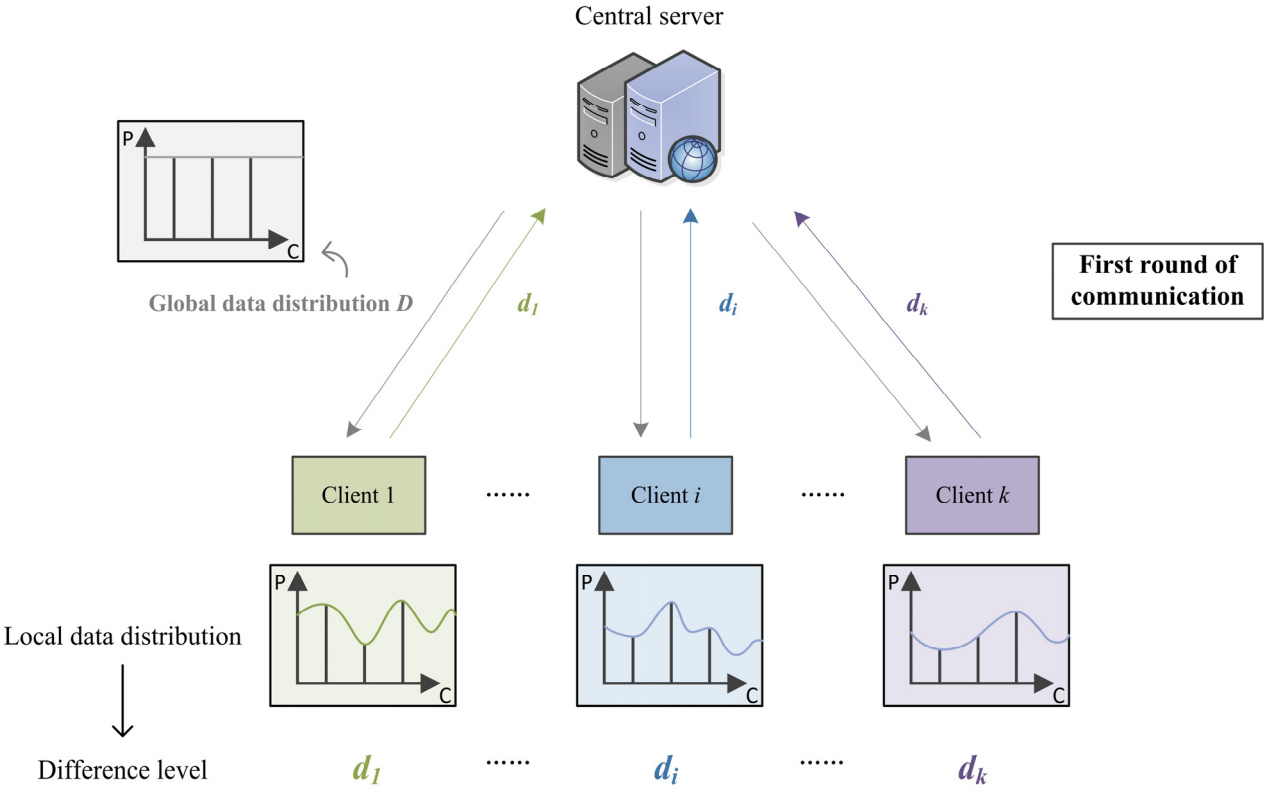


Figure 6. Calculation of the difference level in the first communication round.

More specifically, D_k and T are used to represent the local and global data class distributions, respectively. According to the hypothesis, set $T_c = 1/C$ to treat all categories C fairly. Each client k evaluates the difference between the two distributions through $d_k \leftarrow e(D_k, T)$ to calculate its local difference value d_k, where $e(\cdot)$ is a predefined metric function, which can be L-norm, KL divergence, etc. In this paper, KL divergence is used to measure the difference.

$$d_k = \sum_{c=1}^{C} D_{k,c} \log\left(\frac{D_{k,c}}{T_c}\right) \tag{9}$$

Finally, the central server collects the difference value d_k uploaded by the client, and based on this, assigns different weights to the parameters of different clients in the global aggregation stage. As mentioned above, the aggregation weight p_k of client k should comprehensively consider the relative size n_k of the data set and the local difference level d_k. Therefore, the calculation formula of the difference-aware aggregation weight is defined as follows:

$$p_k = \frac{\mathrm{ReLU}(n_k - a \cdot d_k + b)}{\sum_{m=1}^{K} \mathrm{ReLU}(n_m - a \cdot d_m + b)} \tag{10}$$

where $\mathrm{ReLU}(\cdot)$ is a nonlinear activation function considering negative values. A is a hyperparameter that balances n_k and d_k, and b is a hyperparameter that adjusts the weight. In this way, the DC algorithm dynamically adjusts the aggregation weights based on the data distribution characteristics of different clients, allowing clients with larger data volumes and smaller distribution differences to play a more significant role in model training.

The global aggregation process of the t-round communication is shown in Figure 7, which omits the local personalization stage and the parameter uploading process. The

central server will perform model aggregation based on the difference-aware aggregation weights, as shown in Formula (11), and w_k^t represents the k th client in the t-round.

$$\boldsymbol{w}^{t+1} = \sum_{k=1}^{K} p_k \boldsymbol{w}_k^t \tag{11}$$

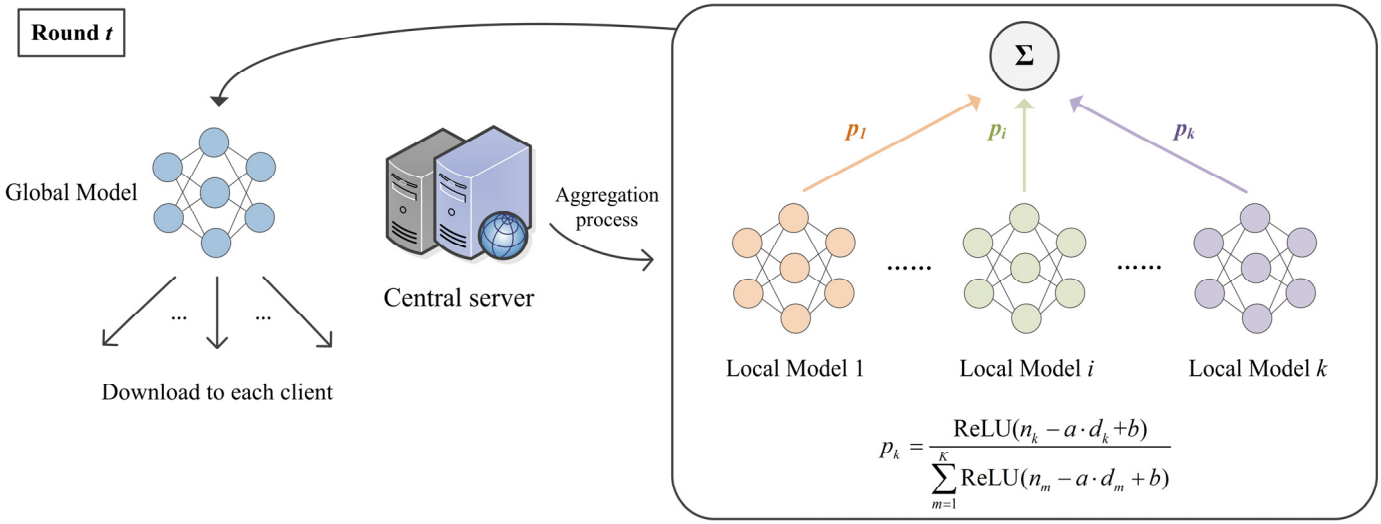


Figure 7. The global aggregation process of FedDPI.

The DC algorithm has the following advantages.

Privacy protection. Since the difference value is calculated locally on the client side, only the difference level measurement value p_k is uploaded to the central server, so it cannot obtain any distribution or specific information of private data during transmission, and there is no privacy leakage problem.

Low communication and computational overhead. As shown in Figure 6, the difference value communication between the client and server only occurs in the first round and only transmits a single value, so the communication overhead can be ignored. The calculation of the difference level only involves simple operations between two vectors, and is effectively distributed to each client. Compared with the overall computational cost of model training, the computational overhead is also minimal. On the other hand, unlike previous differential algorithms that required fine-tuning of each local model on the server, the DC algorithm significantly reduces the computational pressure on the central server while maintaining efficient aggregation of the global model.

Overall, the introduction of the DC algorithm in the aggregation stage has negligible computational and communication overhead and does not pose a threat to privacy and security. This algorithm combines data size and distribution differences to assign differentiated aggregation weights to different clients, ensuring that clients with larger data volumes and smaller distribution differences receive higher weights, thereby alleviating the negative impact of data heterogeneity in the global aggregation stage from the root. At the same time, the optimization of the global model can directly promote local training. For clients with similar data distributions, the subsequent optimization direction will be more consistent and the local model and IPM will be better, thereby promoting further improvement of the global model and forming positive feedback for training. This two-stage collaborative optimization mechanism provides an efficient solution to the problem of data heterogeneity.

3.3.3. Design of Convolutional Neural Network

Existing convolutional neural network models, such as VGG, ResNet, etc., are mostly constructed for processing two-dimensional data such as images. In order to better apply

heterogeneous software defect data and extract data features more conveniently, we conducted data preprocessing as described earlier and designed a CNN model that adapts to the dataset used as the initial network model for the server and various clients as shown in Figure 8. The convolutional neural network we designed is shown in Figure 4, where the convolutional kernel size of the convolutional layer is set to 2 × 2. We set the maximum pooling core size of the maximum pooling layer to 2 × 2. The steps are all 1.

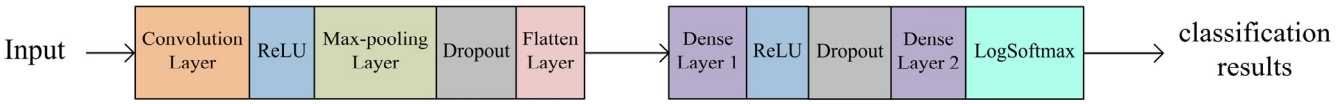


Figure 8. The convolutional neural network of FedDPI.

3.3.4. Optimization and Privacy Protection Strategy

In the communication phase of federated learning, each participant sends model parameters to the central server for aggregation, instead of directly sharing private data, to achieve the goal of collaborative construction of shared models without data leaving the local area. However, research has shown that obtaining model parameters for multiple rounds of communication for differential analysis also carries the risk of exposing private data. Furthermore, if the client only transmits optimized gradient values, a malicious party may expose the model parameters through gradient inversion attacks. Therefore, data security during transmission is difficult to fully guarantee, and privacy and security must be protected through encrypted transmission or the use of irreversible parameter transmission.

This method chooses the latter as the privacy protection scheme and adopts the Ranger optimizer in parameter optimization, which combines the advanced RAdam optimizer and lookahead algorithm, and is currently one of the most advantageous optimization strategies. The lookahead algorithm is based on the latest research results of neural network loss hyperplanes, and proposes an innovative method for stabilizing deep learning training and accelerating convergence. Unlike most optimizers based on stochastic gradient descent (SGD) algorithms, lookahead adopts a completely new optimization approach. Specifically, the lookahead algorithm maintains two sets of weights and updates them through interpolation: one set of weights updates quickly and explores with a "look ahead" approach; the other group of weights updates slowly and is used to provide long-term update stability. After each k-step update, the slow weight is replaced with the current fast weight. This mechanism reduces the variance during the training process, significantly reduces sensitivity to suboptimal hyperparameters, and reduces the dependence on tuning a large number of hyperparameters, thereby achieving faster convergence in various deep learning tasks.

The RAdam optimizer is based on an adaptive momentum optimizer and introduces a learning rate warm-up mechanism. In the early stages of training, due to insufficient data from the optimizer to make accurate adaptive momentum decisions, the variance of model weights is large. Warm-up reduces the fluctuation of model learning in the early stages of training by controlling the learning rate, but requires manual adjustment of the warm-up value. On this basis, RAdam adaptively determines the warm-up value through a rectification function, which adjusts based on the variance of actual data and dynamically starts and stops adaptive momentum until the model training stabilizes.

Based on the working principles of RAdam and lookahead mentioned above, the parameter optimization scheme for the Ranger algorithm is as follows:

$$g_t = g_t - \text{mean}(g_t) \tag{12}$$

$$m_t \leftarrow {\beta_1}^2 m_{t-2} + \left(1 - {\beta_1}^2\right) g_t \tag{13}$$

$$\hat{m}_t \leftarrow \frac{((1+\beta_0)m_t - \beta_0 m_{t-1})}{(1-\beta_1^t)} \tag{14}$$

$$v_t \leftarrow \beta_2 v_{t-1} + (1-\beta_2) g_t^2 \tag{15}$$

$$\hat{v}_t \leftarrow \frac{v_{max}}{(1-\beta_2^t)} \tag{16}$$

$$u_t \leftarrow \frac{\hat{m}_t}{\left(\sqrt{(1+\beta_0)^2 + \beta_0^2}\,(\sqrt{\hat{v}_t} + \epsilon)\right)}) \tag{17}$$

$$\eta_t = \min(1, \max(\frac{1-\beta_2}{2} \cdot t, \frac{t}{t_{warmup}}), \frac{t_{max} - t}{t_{warmup}})\eta \tag{18}$$

$$d_t = \frac{\eta_t}{\sqrt{mean(\hat{v}_t)}}\lambda(1 - \frac{1}{\|\theta_{t-1}\|})\theta_{t-1} \tag{19}$$

$$\theta_t = \theta_{t-1} - \eta_t u_t - \eta_t d_t \tag{20}$$

where θ_t represents the current parameter to be optimized, g_t is the gradient of the current parameter, m_t and v_t are the first-order and second-order momentum of the historical gradient, respectively, and $\hat{m}_t$ and $\hat{v}_t$ are the momentum after bias correction. The hyperparameters β_0, β_1 and β_2 are responsible for controlling the momentum, u_t is the parameter update vector, which is determined by the first-order and second-order momentum, η is the learning rate, and Formula (18) adjusts the learning rate adaptively. t_{max} is the number of iterations, and t_{warmup} is the number of iterations for preheating the learning rate. d_t is the weight attenuation term; ϵ is used for numerical stability and is generally set to e^{-8}. Formula (20) is used to update the parameters in each batch. When the iteration of k batches is completed, lookahead fuses the slow weights it saves with the latest fast weights of RAdam, and updates the final model parameters through Equations (21) and (22):

$$l_{t/k} \leftarrow \beta_{lookahead} l_{t/k-1} + (1 - \beta_{lookahead})\theta_t \tag{21}$$

$$\theta_t \leftarrow l_{t/k} \tag{22}$$

where $\beta_{lookahead}$ is the slow-weight step size, usually equal to 0.5. As it can be seen, due to the introduction of various new irreversible mixed operations such as second-order momentum, bias correction, weight decay, weight fusion, etc. by the Ranger optimizer, combined with multiple hyperparameter settings, it is impossible to obtain the original gradient information of transmission parameters through reverse operations in the process of federated communication. At the same time, the lookahead algorithm adopts a new strategy independent of SGD in optimal parameter search. Therefore, compared with the method of gradient descent as the kernel, privacy and security attacks based on gradient difference analysis are also ineffective against the lookahead algorithm.

In addition, RAdam and lookahead have made breakthroughs in different aspects of deep learning optimization, and the collaborative effect of their combination has enabled the Ranger optimizer to achieve two optimal improvements. The Ranger optimizer records first-order and second-order moment information, reflecting historical learning inertia and environmental perception ability, thereby dynamically adjusting the learning rate and avoiding the dilemma of local optima or inability to converge. Incorporating the warm-up mechanism, the Ranger optimizer not only improves the robustness of the model under multi-source heterogeneous data but also accelerates the optimization convergence of the objective function.

The overall pseudocode of the proposed FedDPI in this chapter is shown as Algorithm 1.

Algorithm 1 FedDPI

Initialization: Initialize the global model θ_s for the server
Input: number of clients k, number of communication rounds t, number of local update rounds E, learning rate η, local data set D_k

Server processing:
Initialize the global model θ_s and send it to each client to form the local model θ_s^k
if t = 1
    $d_k \leftarrow$ local difference measure (D_k, T)
    The aggregation weight p_k is calculated by DC algorithm
for each communication round t = 1, 2, . . ., T **do**
    The server sends the model to the local $\theta_{s,t}^k \leftarrow \theta_{s,t}$
    for each client k **in parallel do**
        $\hat{\theta}_{s,t}^k \leftarrow$ local update $\left(k, \theta_{s,t}^k\right)$
    end for
    Update the global model $\theta_{s+1} \leftarrow \sum_{k=1}^{K} p_k \hat{\theta}_{s,t}^k$ according to p_k.
end for
return global model θ_s and personalized model $\hat{\theta}_{s,t}^k$

Local difference measure (D_k, T):
for each client k **in parallel do**
    Difference measure $d_k \leftarrow e(D_k, T)$ with global data distribution
    Upload the difference value d_k to the server
end for

Local update (k, $\theta_{s,t}^k$):
for local epoch e = 1, 2, . . ., E **do**
    for every mini-batch **do**
        Calculate the local loss $L(\theta_{s,t}^k)$
        Private data and IPM jointly obtain the total loss $L_{total} \leftarrow (1-\lambda)L + \lambda L_{kd}$
        Calculate the gradient based on the loss and update the local model $\hat{\theta}_{s,t}^k \leftarrow \text{Ranger}(L_{total})$
    end for
if convergence detection = **false**
    Update IPM $\theta_{p,t+1}^k \leftarrow \theta_{p,t}^k$
return personalized model parameter $\hat{\theta}_{s,t}^k$

4. Results

4.1. Experimental Dataset Description

In order to promote research in the field of SDP, multiple public datasets have emerged. The NASA, AEEEM, and Relink datasets, widely used in the SDP field, were used in this experiment [49]. Table 1 shows the various attributes of project data in the dataset.

1. The NASA dataset is sourced from multiple software projects of the National Aeronautics and Space Administration (NASA) in the United States, initially collected by the Jet Propulsion Laboratory. This dataset contains defect history data for 14 projects involving different software systems and programming languages, including JAVA, C++, etc., [50]. Each project represents a software system or subsystem of NASA, and the data consist of defect information and static code features. A total of 37 software quality-related features were used in the NASA dataset. The diversity and representativeness of the data are mainly composed of metrics such as in McCabe and Halstead [51], making it one of the benchmarks for defect prediction research [52–54]. This article uses five projects from the NASA dataset for experiments, namely, CM1, MW1, PC1, PC3, and PC4.

2. The AEEEM dataset was compiled by D'Ambros's research team in 2012, covering five open-source Java projects [55]: Equinox (EQ), Eclipse JDT Core (JDT), Apache, Lucene (LC), Mylyn (ML), and Eclipse PDE UI (PDE) [56]. Each project's data contain 61 features, including 17 source code features, 5 previous defect features, 5 entry of change features, 17 entry of source code features, and 17 church of source code features. This dataset has also been widely used in the field of SDP research [57–59].
3. The Relink dataset was collected by Wu et al. [60] and manually validated to ensure the accuracy of sample defect information. It contains data from three projects, namely, Apache HTTP Server (Apache), Open Intents Safe (Safe), and ZXing. The Relink dataset contains 26 features extracted by the Understanding tool focusing on the complexity metrics of the code [60], which has been adopted in multiple SDP academic studies [61–63].

Table 1. Brief attributes of the datasets used in the experiment.

Dataset	Project	Number of Metrics	Number of Total Instances	Number of Defective Instances	% of Defective Instances
NASA	PC1	37	705	61	8.65
	PC3	37	1077	134	12.44
	PC4	37	1458	178	12.21
	MW1	37	253	27	10.67
	CM1	37	327	42	12.84
AEEEM	EQ	61	324	129	39.81
	JDT	61	997	206	20.66
	LC	61	691	64	9.26
	ML	61	1862	245	13.16
	PDE	61	1497	209	13.96
Relink	Apache	26	194	98	50.52
	Safe	26	56	22	39.29
	ZXing	26	399	118	29.57

4.2. Experimental Settings

Our experiment sets the maximum number of global training rounds to 500, with a total of five clients participating in federated training and model aggregation. Each local epoch is set to 2, and the batch size of the model input is 128. The default settings are used in the Ranger optimizer. In the knowledge transfer of HPM, the weight of the loss function obtained from knowledge distillation is set to 0.05. In the model aggregation method, KL divergence is selected as the parameter between the client and server. In order to fully integrate the influence of the relative size and difference value of the dataset on the aggregation weight, the control parameters a and b are set to 0.5.

4.3. Evaluation Performance Metrics

In our experiment, we used two evaluation metrics to measure the performance of the model, namely, AUC and G-mean. AUC is a commonly used metric to evaluate the performance of machine learning models and is typically used to assess the overall predictive accuracy of classification models. AUC is the area under the receiver operating characteristic (ROC) curve, which is a curve plotted on the model's true positive (TP) as the vertical axis and false positive (FP) as the horizontal axis. In addition, the G-mean [64] indicator is selected to evaluate the model performance for extreme class imbalance problems. The G-mean index is a geometric mean that considers the classification performance of software defect prediction models for positive and negative classes in class imbalance scenarios (in SDP, defective instances are referred to as positive classes, and defect free

instances are referred to as negative classes). G-mean can be calculated using a confusion matrix as shown in Table 2.

$$G - \text{mean} = \sqrt{\frac{TP}{FN + TP} \times \frac{TN}{TN + FP}} \tag{23}$$

Table 2. Confusion matrix.

		Ground Truth	
		Positive	**Negative**
Prediction label	Positive	True Positive (TP)	False Positive (FP)
	Negative	False Negative (FN)	True Negative (TN)

4.4. Experimental Environment

Our experiment used Python 3.6.5 as the programming language, PyCharm 2022.1.22 as the compiler, and builds a federated learning framework and neural network model based on the PyTorch environment. The computer processor used was an AMD Ryzen 7 4800H, the operating system Windows 10, and the display adapter an Nvidia GeForce RTX 2060.

4.5. Comparative Experiment and Analysis

To verify the advantages of inheriting federated learning in HSDP compared to other methods, we used CCA+ [9], KCCA+ [65], FedAvg [47], FTLKD [25], FRLGC [27], and FPLPA [31] as comparative methods. CCA+ and KCCA+ are deep learning methods, while FedAvg, FTLKD, FRLGC, and FPLPA are all methods of federated learning in HSDP using AUC and G-mean as evaluation indexes for model performance to measure the prediction accuracy of the model under data heterogeneity and extreme class imbalance problems.

Table 3 shows the experimental results of using software projects from NASA and AEEEM datasets as private data for five clients, namely, PC3, PC4, MW1, JDT, and LC. It can be seen that the AUC index of each client in this method ranges from 0.7782 to 0.9375, with an average of 0.8617. The G-mean index ranges from 0.7287 to 0.9137, with a mean of 0.7993. Under two indicators, the results of our method on all clients are superior to different deep learning methods and federated learning methods. Comparing by means, our method is higher than CCA+, KCCA+, FedAvg, FTLKD, FRLGC, and FPLPA (0.3631, 0.2978), (0.3533, 0.3001), (0.3135, 0.2843), (0.3036, 0.2901), (0.2866, 0.2927), and (0.1046, 0.2157), respectively. Among the comparison methods, the deep learning method, as well as the FedAvg, FTLKD, and FRLGC methods, achieved an AUC index of around 0.5, indicating that the model approximates random classification in classification and has significant room for improvement. FPLPA shows significant improvement in AUC, but there is still room for improvement in G-mean, indicating that the model's ability to recognize negative class samples can continue to be optimized. In this experiment, this method achieved significant improvements in both indicators, demonstrating the advantage of forming positive feedback through collaborative improvement in both stages of federated learning. It can effectively overcome the problems of data heterogeneity and class imbalance and provide a good local model for all parties while effectively protecting their privacy.

Table 4 shows the experimental results of using software projects from NASA and Relink datasets as private data for five clients, namely, PC3, PC4, MW1, Apache, and Safe. The AUC index of each client in this method ranges from 0.7356 to 0.9402, with a G-mean higher than CCA+, KCCA+, FedAvg, FTLKD, FRLGC, and FPLPA by (0.3214, 0.2849), (0.3055, 0.2773), (0.3293, 0.2862), (0.2940, 0.2813), (0.2796, 0.2811), and (0.1100, 0.2142), respectively.

Table 3. Comparison of NASA and AEEEM as datasets with other methods.

Project	Index	CCA+	KCCA+	FedAvg	FTLKD	FRLGC	FPLPA	FedDPI
PC3	AUC	0.4703	0.6029	0.5361	0.5417	0.5753	0.7673	0.7951
	G-mean	0.4830	0.5232	0.5117	0.5092	0.5071	0.5889	0.7287
PC4	AUC	0.4618	0.4861	0.5222	0.5530	0.6148	0.8098	0.9202
	G-mean	0.4859	0.4859	0.5071	0.5014	0.5238	0.6030	0.8427
MW1	AUC	0.5322	0.5317	0.5734	0.7091	0.6083	0.6360	0.9375
	G-mean	0.5048	0.5180	0.5242	0.5685	0.5208	0.5384	0.9137
JDT	AUC	0.4920	0.4933	0.5336	0.5119	0.5174	0.7964	0.8775
	G-mean	0.4972	0.4917	0.5079	0.5013	0.4808	0.5978	0.8012
LC	AUC	0.5367	0.4279	0.5755	0.4750	0.5597	0.7761	0.7782
	G-mean	0.5367	0.4774	0.5243	0.4655	0.5007	0.5901	0.7103
Average	AUC	0.4986	0.5084	0.5482	0.5581	0.5751	0.7571	0.8617
	G-mean	0.5015	0.4992	0.5150	0.5092	0.5066	0.5836	0.7993

Table 4. Comparison of NASA and Relink as datasets with other methods.

Project	Index	CCA+	KCCA+	FedAvg	FTLKD	FRLGC	FPLPA	FedDPI
PC3	AUC	0.5156	0.6294	0.5179	0.5190	0.4988	0.7578	0.8074
	G-mean	0.5040	0.5293	0.5059	0.5024	0.4711	0.5856	0.7362
PC4	AUC	0.6000	0.4806	0.5377	0.4788	0.5561	0.7952	0.9327
	G-mean	0.5177	0.4940	0.5116	0.467	0.4979	0.5981	0.8632
MW1	AUC	0.4579	0.5155	0.5051	0.7463	0.5000	0.6434	0.9402
	G-mean	0.4873	0.5068	0.5000	0.5827	0.5000	0.5404	0.8788
JDT	AUC	0.5373	0.5275	0.5109	0.5602	0.5718	0.6782	0.7356
	G-mean	0.5151	0.5127	0.5032	0.5102	0.5172	0.5594	0.6931
LC	AUC	0.5537	0.5940	0.5535	0.5000	0.7500	0.8500	0.8586
	G-mean	0.5205	0.5397	0.5171	0.5000	0.5774	0.6146	0.7977
Average	AUC	0.5329	0.5494	0.5250	0.5609	0.5753	0.7449	0.8549
	G-mean	0.5089	0.5165	0.5076	0.5125	0.5127	0.5796	0.7938

In this experiment, the AUC and G-mean metrics of each client model were higher than those of other methods. From the experimental results, it can be seen that due to the limited data of the two software projects in the Relink dataset, the AUC is lower compared to other clients, but the G-mean metric is comparable. It has been proven that this method can fully solve the problem of insufficient training data brought by deep learning and achieve a high level of recognition in minority classes.

Table 5 shows the experimental results of using software projects from the AEEEM and Relink datasets as private data for five clients, namely, EQ, JDT, LC, Apache, and Safe. The AUC of all clients is 0.7448–0.9293, and the G-mean is 0.6898–0.8409, with average values of 0.8649 and 0.7786, respectively. Compared with CCA+, KCCA+, FedAvg, FTLKD, FRLGC, and FPLPA, the average values are higher: (0.3348, 0.2682), (0.3289, 0.2643), (0.3366, 0.2696), (0.2651, 0.2528), (0.2281, 0.2380), and (0.1518, 0.2122), respectively.

In this experiment, the AUC and G-mean metrics of each client model were higher than those of other methods. It can be seen that the test results obtained from Relink and Safe project data significantly improved compared to the previous experiment, indicating that the project data distribution of the AEEEM dataset replaced in this experiment is similar to that of the Safe project and can provide more training information under the federated learning framework. Similarly, the client testing results of the JDT and LC projects also showed corresponding improvements compared to the first set of experiments. The experimental results show that federated learning can play an important role in cross-project software defect prediction and achieve greater model performance improvement

when the data distribution is somewhat similar, reflecting the advantages of federated learning and this method on HSDP.

Table 5. Comparison of AEEEM and Relink as datasets with other methods.

Project	Index	CCA+	KCCA+	FedAvg	FTLKD	FRLGC	FPLPA	FedDPI
PC3	AUC	0.5171	0.5715	0.5120	0.6500	0.6389	0.7099	0.9097
	G-mean	0.5060	0.5314	0.5038	0.5477	0.5307	0.5671	0.7998
PC4	AUC	0.5010	0.5009	0.5199	0.5119	0.6857	0.7844	0.9002
	G-mean	0.5026	0.4980	0.5066	0.4907	0.5367	0.5934	0.8215
MW1	AUC	0.5961	0.5147	0.5336	0.7000	0.5718	0.7314	0.8404
	G-mean	0.5323	0.5009	0.5106	0.5578	0.5543	0.5731	0.7412
JDT	AUC	0.5281	0.5331	0.5104	0.6429	0.7045	0.6971	0.7448
	G-mean	0.5094	0.5154	0.5035	0.5429	0.5172	0.5640	0.6898
LC	AUC	0.5084	0.5596	0.5655	0.4944	0.5833	0.6429	0.9293
	G-mean	0.5018	0.5256	0.5205	0.4898	0.5641	0.5345	0.8409
Average	AUC	0.5301	0.5360	0.5283	0.5998	0.6368	0.7131	0.8649
	G-mean	0.5104	0.5143	0.5090	0.5258	0.5406	0.5664	0.7786

In the comparative experiments conducted in this chapter, we used project data from three different datasets, conducted three sets of experiments through various combinations, and selected six representative deep learning and federated learning methods for comparison. The experimental results showed that FedDPI achieved the best model performance in all data combination experiments, with average improvements of 0.2783 and 0.2673 in AUC and G-mean metrics, respectively, verifying the effectiveness of our method for data heterogeneity and extreme class imbalance problems.

5. Discussion

5.1. The Influence of Feature Dimensions on Model Accuracy

In order to more effectively capture the complex correlations between features in high-dimensional data, this method designs an adaptive two-dimensional convolutional neural network that requires two-dimensional reconstruction of the data to meet the input format requirements. Based on the above two points, when using KPCA for optimal feature selection, this method fixes the optimal subset range at F = 25, F = 16, and F = 9 through high-dimensional mapping and feature decomposition. The performance of the three dimensions of data is evaluated through experiments to determine the optimal feature dimension selection strategy. The experimental results are shown in the following figures. Figures 9–11 gave the experimental results of different feature dimensions on different datasets.

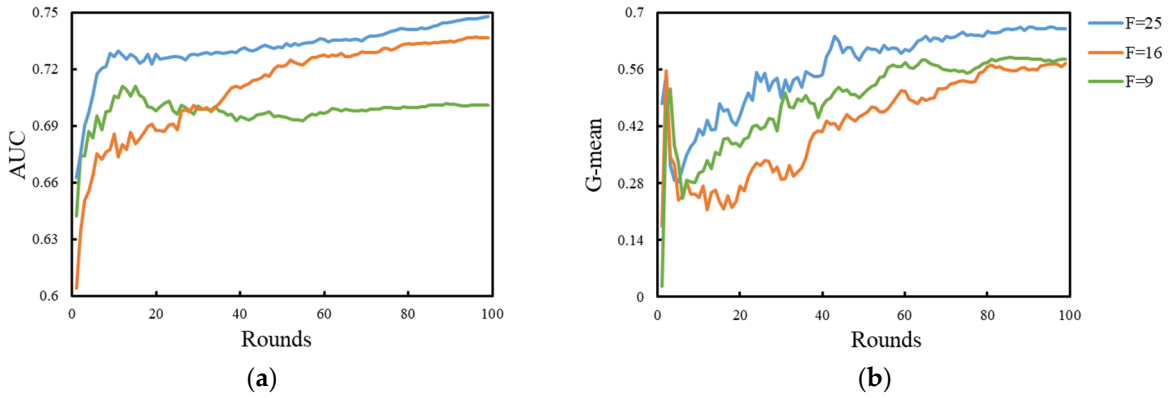
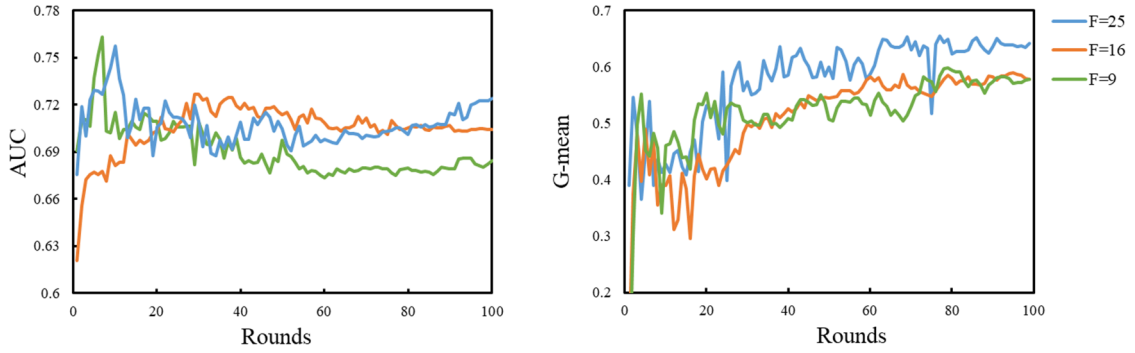
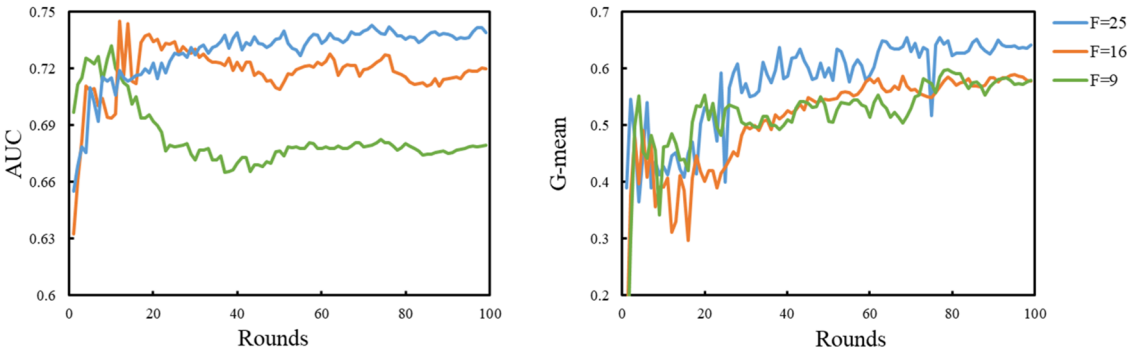


Figure 9. Experimental results of different feature dimensions (using NASA and AEEEM as datasets). (**a**) Learning curve of AUC; (**b**) learning curve of G-mean.



(**a**) (**b**)

Figure 10. Experimental results of different feature dimensions (using NASA and Relink as datasets). (**a**) Learning curve of AUC; (**b**) learning curve of G-mean.



(**a**) (**b**)

Figure 11. Experimental results of different feature dimensions (using AEEEM and Relink as datasets). (**a**) Learning curve of AUC; (**b**) learning curve of G-mean.

Experiments were conducted using pairwise combinations of three datasets, and the results of each group showed that, under similar convergence times, the model exhibited better learning curves and higher accuracy in AUC and G-mean metrics when feature selection reached 25 dimensions. This indicates that within the optional dimension range, the 25-dimension feature subset can maximize the preservation of effective information in heterogeneous data, extracting the more discriminative features, and effectively improving the generalization ability of the model. In contrast, when the dimensions are reduced to 16 or 9, too few features result in the loss of local defect data information, limiting the performance of the prediction model. Therefore, 25 dimensional features were selected for subsequent experiments in this paper.

5.2. The Impact of Different Numbers of Participants on Model Performance

In federated learning, the number of clients also affects the training performance of the model, so it is necessary to determine the optimal number of clients. Considering the possible situations that may occur in practical applications, experiments were conducted to compare and analyze the number of clients, set to three to seven. Figure 12 shows the model learning curves for different numbers of clients, while Table 6 lists the corresponding training times. The experimental results show that the learning curves of the five participants gradually increase during the training process and eventually converge to the optimal AUC level, which is better than the setting of the number of other participants.

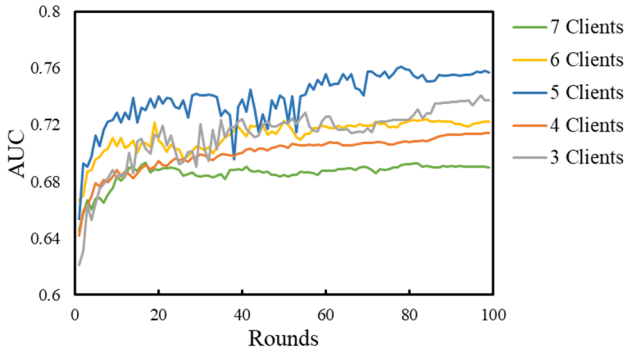


Figure 12. Comparison of experimental results from different clients.

Table 6. Comparison of experiment time for different clients.

Comparison Items	**Result**				
Number of Clients	7	6	5	4	3
Training Time	15	24	23	62	37

In addition, selecting five participants is the optimal configuration in that it achieves a balance between performance and efficiency. As shown in Table 6, although the training time with seven participants is slightly lower than that with five clients, its improvement on the AUC index of the model is limited and its predictive performance is poor. In contrast, the setting of five participants significantly reduces training time while ensuring good prediction performance and convergence speed. Therefore, considering both the predictive performance and training efficiency of the model, this experiment adopted a configuration of five participants as the optimal solution.

5.3. Analysis of Model Convergence for Different Clients

In order to investigate the trend of predictive performance of each client with increasing training epochs, a combination of NASA and AEEEM, NASA and Relink, and AEEEM and Relink datasets were used as local data for the client, and the changes in AUC and G-mean evaluation indicators with increasing training epochs were observed. The average prediction results of each client model are shown in Figure 13. Overall, as the number of training iterations increases, our proposed method demonstrates a stable convergence trend under cross-project data training, demonstrating the advantages of the two-stage collaborative optimization mechanism in addressing data heterogeneity issues in federated learning. In addition, the significant improvement in the G-mean index further validates the effectiveness of our method in handling extreme class-imbalanced data.

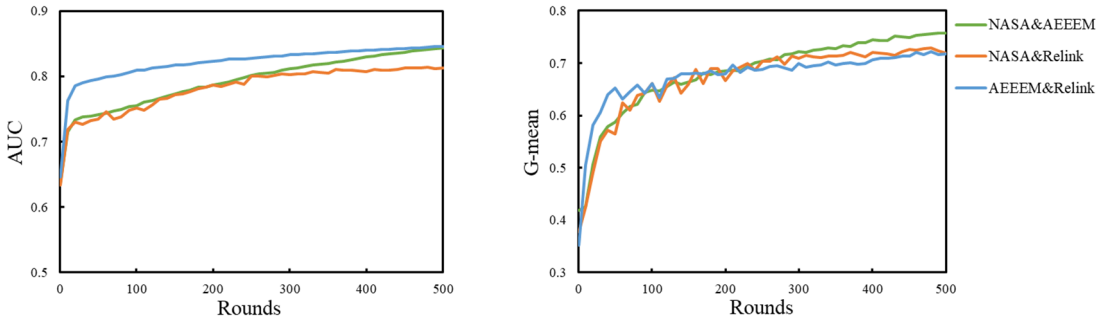
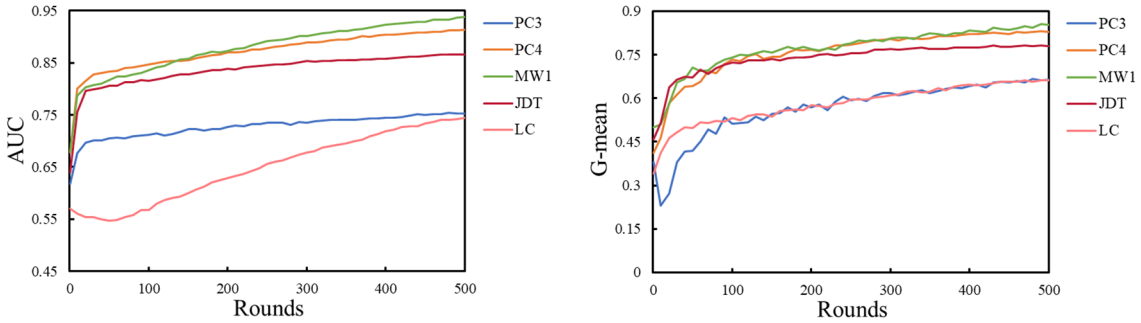


Figure 13. Experimental results of three combinations of datasets. (**a**) Learning curve of AUC; (**b**) learning curve of G-mean.

Below is an analysis of all client prediction results in the combination of three datasets. Figure 14 shows the experimental results using a combination of NASA and AEEEM datasets. Overall, compared to the first training round, the AUC and G-mean evaluation metrics after model convergence increased by an average of 0.2061 and 0.4182, respectively. In Figure 14a, the AUC learning curve shows that PC3, PC4, MW1, and JDT rapidly increase and gradually converge during the period of 0 < rounds < 100, while LC first decreases and then increases during the 100 rounds, and then steadily increases and shows a convergence trend after rounds > 100. This phenomenon is related to the data distribution characteristics in the LC dataset and the small number of negative class samples. The initial data lack sufficient information, making it difficult to adjust the LC model to the correct optimization direction. In the G-mean learning curve shown in Figure 14b, each client exhibits a similar growth trend, gradually converging after rounds >100.



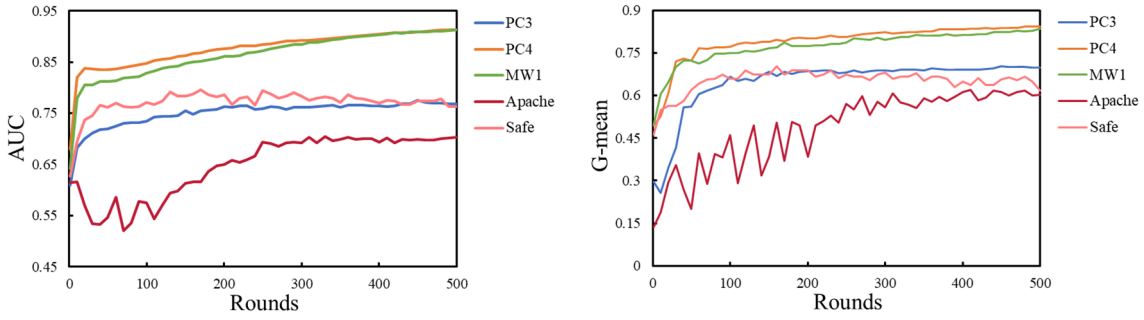
(**a**) (**b**)

Figure 14. Prediction results using NASA and AEEEM as datasets. (**a**) Learning curve of AUC; (**b**) learning curve of G-mean.

Figure 15 shows the experimental results using a combination of NASA and Relink datasets. In Figure 15a,b, the AUC and G-mean evaluation metrics after model convergence have increased by an average of 0.2061 and 0.4182, respectively, compared to the first training round. According to the results of the two graphs, during the period of 0 < rounds < 100, PC3, PC4, MW1, and Safe showed a stable growth trend and tended to converge. Apache's AUC index fluctuated, first decreasing and then increasing within 100 rounds, and eventually returned to growth and tended to converge. However, the G-mean index showed relatively large fluctuations within 200 rounds, but overall increased. This is due to the low similarity between Apache's sample distribution and other data, as well as significant differences in optimization directions, making it difficult to extract effective training knowledge for its model prediction in the early stages. This fluctuation gradually decreases to the normal range as training progresses, demonstrating the difficulty of optimizing the model for dealing with Apache datasets and the generalization performance of our method.

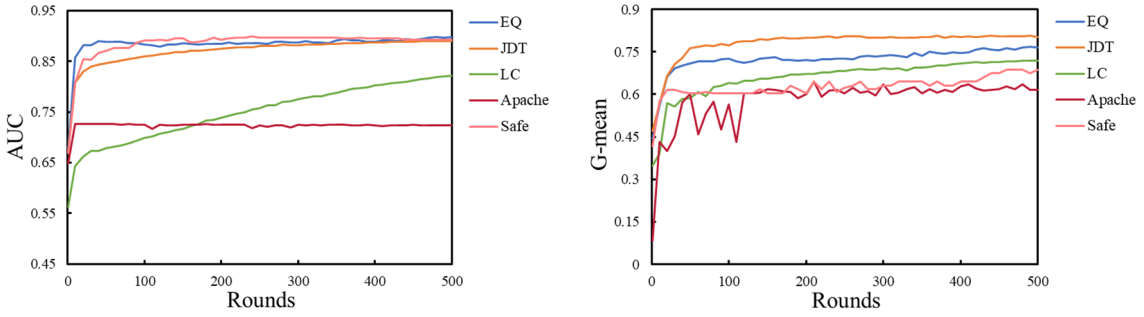
Figure 16 shows the experimental results using a combination of AEEEM and Relink datasets. In Figure 16a,b, the AUC and G-mean evaluation metrics after model convergence improved by an average of 0.1881 and 0.3663, respectively, compared to the first training round. Overall, EQ and JDT clients show significant performance improvement and tend to converge when 0 < rounds < 100, demonstrating good stability; The learning curve of LC also gradually increases, but the convergence speed is lower than other clients, and the overall performance is better than the first group of experiments. This may be due to the fact that the feature distribution between LC and other datasets in this group is more similar, which alleviates the negative impact of LC's own data problems on model learning. Apache's performance lags behind other clients, but the fluctuation has decreased compared to the previous set of experiments, and the convergence value has slightly improved. This indirectly indicates that the data distribution similarity for this set of experiments is higher,

which can obtain more effective federated training knowledge to improve the optimization direction of the model.



(**a**) (**b**)

Figure 15. Prediction results using NASA and Relink as datasets. (**a**) Learning curve of AUC; (**b**) learning curve of G-mean.



(**a**) (**b**)

Figure 16. Prediction results using AEEEM and Relink as datasets. (**a**) Learning curve of AUC; (**b**) learning curve of G-mean.

We first conducted a systematic experiment on the selection of the number of clients. The results showed that when there were five clients, the model performance was optimal and the training time was moderate, resulting in the most reasonable number setting. Secondly, we conducted convergence analysis on the model training of FedDPI on different project data. The experiments showed that FedDPI can achieve varying degrees of convergence on all project data, and combined with the heterogeneity in data distribution, we explored the potential reasons for changes in the learning curve.

6. Conclusions

This paper proposes a cross-project software defect prediction algorithm called FedDPI based on differential sensing and inheritance federated learning to address the problem of data heterogeneity in the field of cross-project software prediction. This algorithm innovatively introduces a two-stage collaborative optimization mechanism, aiming to simultaneously alleviate the negative impact of data heterogeneity in both local and global stages, form positive feedback for training, and efficiently improve the performance and convergence speed of defect prediction models. Firstly, we used a data preprocessing scheme that combines SMOTETomek and KPCA to obtain the optimal feature subset while removing data noise, resulting in high-quality local data. The inheritance private model algorithm is introduced in the local training stage, effectively alleviating the knowledge-forgetting problem caused by model drift. In the global aggregation stage, the difference-aware collaboration algorithm was used to dynamically adjust the aggregation weights based on different data distributions, thereby overcoming the problem of client drift,

improving the performance of the global model, and bringing benefits to the next round of local training. In addition, we combined the algorithm advantages of Ranger optimizer and its irreversible parameter calculation process to enhance the robustness of the model and the privacy protection ability of the algorithm. We conducted experiments using project data combinations from three different datasets, and obtained AUC indicators ranging from 0.7356 to 0.9402. The G-mean index ranged from 0.6898 to 0.9137. Compared with different deep learning and federated learning algorithms, the average improvement in AUC and G-mean metrics reached 0.2783 and 0.2673. The experimental results show that FedDPI can achieve the best predictive performance in various data combination experiments, demonstrating good robustness in the face of data heterogeneity and extreme class imbalance challenges, proving the feasibility and reliability of our method in the SDP field, providing different improvement directions for the development of federated learning in the SDP field, and also providing low-cost and high-efficiency defect detection solutions for software engineering in practice.

In the future, we will further investigate the challenges faced by federated learning in cross-project software defect prediction, particularly in innovative improvements to communication efficiency. In addition to improving the federated learning framework, we also plan to incorporate more practical challenges such as communication protocols and network attacks into our research scope. By deeply studying the performance characteristics of different communication protocols and systematically simulating network security attacks, we aim to enhance the security and reliability of our methods in practical scenarios and open up a more secure and efficient innovative path for the widespread application of deep learning.

Author Contributions: Conceptualization, A.W., Y.F., M.Y., H.W., Y.I. and H.C.; methodology, A.W., Y.F., M.Y., H.W., Y.I. and H.C.; software Y.F. and M.Y.; validation A.W., Y.F., M.Y., H.W., Y.I. and H.C.; writing—review and editing A.W., Y.F., M.Y., H.W., Y.I. and H.C. All authors have read and agreed to the published version of the manuscript.

Funding: This work was funded by Shenzhen Polytechnic University-Julongji Satellite Earth Communication Big Data Application Research and Development Center (No. 602431014PQ), the Key Research and Development Plan Project of Heilongjiang (JD2023SJ19), the Natural Science Foundation of Heilongjiang Province (LH2023F034), and the Science and Technology Project of Heilongjiang Provincial Department of Transportation (HJK2024B002). Yuji Iwahori's research was supported by a Japan Society for the Promotion of Science (JSPS) Grant-in-Aid for Scientific Research(C) (24K15019) and a Chubu University grant.

Data Availability Statement: NASA, AEEEM, Relink: https://github.com/bharlow058/AEEEM-and-other-SDP-datasets, accessed on 28 June 2015.

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Lin, W.T. Research on Cross-Project Software Defect Prediction Method Based on Machine Learning. Master's Thesis, Beijing University of Posts and Telecommunications, Beijing, China, 2023.
2. Hu, J.H. Research on Software Defect Prediction Method Based on Random Forest and SVM. Master's Thesis, Zhejiang University of Technology, Hangzhou, China, 2020.
3. Ma, Y. Research on Software Defect Prediction Techniques Based on Machine Learning. Master's Thesis, University of Electronic Science and Technology of China, Chengdu, China, 2012.
4. Shen, Z.D.; Chen, S. A Survey of Automatic Software Vulnerability Detection, Program Repair, and Defect Prediction Techniques. *Secur. Commun. Netw.* **2020**, *2020*, 8858010. [CrossRef]
5. Cambridge Judge Business School. Available online: https://www.jbs.cam.ac.uk/2013/research-by-cambridge-mbas-for-tech-firm-undo-finds-software-bugs-cost-the-industry-316-billion-a-year (accessed on 3 December 2024).
6. Herbold, S.; Trautsch, A.; Grabowski, J. A comparative study to benchmark cross-project defect prediction approaches. *IEEE Trans. Softw. Eng.* **2017**, *44*, 811–833. [CrossRef]
7. He, Z.; Shu, F.; Yang, Y.; Li, M.; Wang, Q. An investigation on the feasibility of cross-project defect prediction. *Autom. Softw. Eng.* **2012**, *19*, 167–199. [CrossRef]

8. Jing, X.; Wu, F.; Dong, X.; Qi, F.; Xu, B. Heterogeneous cross-company defect prediction by unified metric representation and CCA-based transfer learning. In Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering, Bergamo, Italy, 30 August–4 September 2015; pp. 496–507.
9. Hardoon, D.R.; Shawe-Taylor, J.; Friman, O. Canonical correlation analysis: An overview with application to learning methods. *Neural Comput.* **2004**, *16*, 2639–2664. [CrossRef] [PubMed]
10. Yang, Q.; Liu, Y.; Chen, Y.; Kang, Y.; Chen, T.; Yu, H. *Federated Learning*; Publishing House of Electronics Industry: Beijing, China, 2020; p. 15.
11. Briand, L.C.; Melo, W.L.; Wüst, J. Assessing the Applicability of Fault-Proneness Models Across Object-Oriented Software Projects. *IEEE Trans. Softw. Eng.* **2002**, *28*, 706–720. [CrossRef]
12. Li, J.; He, P.; Zhu, J.; Lyu, M.R. Software Defect Prediction via Convolutional Neural Network. In Proceedings of the 2017 IEEE International Conference on Software Quality, Reliability and Security (QRS), Prague, Czech Republic, 25–29 July 2017; pp. 318–328.
13. Dam, H.K.; Pham, T.; Ng, S.W.; Tran, T.; Grundy, J.; Ghose, A.; Kim, T.; Kim, C.J. Lessons Learned from Using a Deep Tree-Based Model for Software Defect Prediction in Practice. In Proceedings of the 2019 IEEE/ACM 16th International Conference on Mining Software Repositories (MSR), Montreal, QC, Canada, 25–31 May 2019; pp. 46–57.
14. Liu, C.; Yang, D.; Xia, X.; Yan, M.; Zhang, X.H. A Two-Phase Transfer Learning Model for Cross-Project Defect Prediction. *Inf. Softw. Technol.* **2019**, *107*, 125–136. [CrossRef]
15. Tang, Y.; Dai, Q.; Yang, M.Y.; Du, T.; Chen, L. Software Defect Prediction Ensemble Learning Algorithm Based on Adaptive Variable Sparrow Search Algorithm. *Int. J. Mach. Learn. Cybern.* **2023**, *14*, 1967–1987. [CrossRef]
16. Wu, J.; Wu, Y.; Zhou, M.; Jiang, X. SLA+: Narrowing the Difference between Datasets in Heterogeneous Cross-Project Defect Prediction. In Proceedings of the 2020 IEEE 44th Annual Computers, Software, and Applications Conference (COMPSAC), Madrid, Spain, 13–17 July 2020; pp. 1229–1234.
17. Jin, C. Cross-project software defect prediction based on domain adaptation learning and optimization. *Expert Syst. Appl.* **2021**, *171*, 114637. [CrossRef]
18. Shen, P.; Ding, X.; Mu, X.; Xu, J. A software defect prediction method based on sampling and integration. In Proceedings of the 2nd International Seminar on Computer Science and Engineering Technology (SCSET), Shanghai, China, 30–31 October 2020; pp. 12034–12054.
19. Tong, H.; Liu, B.; Wang, S. Kernel spectral embedding transfer ensemble for heterogeneous defect prediction. *IEEE Trans. Softw. Eng.* **2021**, *47*, 1886–1906. [CrossRef]
20. Bal, P.R.; Kumar, S. A Data Transfer and Relevant Metrics Matching Based Approach for Heterogeneous Defect Prediction. *IEEE Trans. Softw. Eng.* **2023**, *49*, 1232–1245. [CrossRef]
21. Šikić, L.; Kurdija, A.S.; Vladimir, K.; Šilić, M. Graph Neural Network for Source Code Defect Prediction. *IEEE Access* **2022**, *10*, 10402–10415. [CrossRef]
22. Pornprasit, C.; Tantithamthavorn, C.K. DeepLineDP: Towards a Deep Learning Approach for Line-Level Defect Prediction. *IEEE Trans. Softw. Eng.* **2023**, *49*, 84–98. [CrossRef]
23. Abdu, A.; Zhai, Z.J.; Abdo, H.A.; Algabri, R.; Al-masni, M.A.; Muhammad, M.S.; Gu, Y.H. Semantic and Traditional Feature Fusion for Software Defect Prediction Using Hybrid Deep Learning Model. *Sci. Rep.* **2024**, *14*, 14771. [CrossRef] [PubMed]
24. Song, M.K.; Wang, Z.B.; Zhang, Z.F.; Song, Y.; Wang, Q.; Ren, J.; Qi, H.R. Analyzing User-Level Privacy Attack Against Federated Learning. *IEEE J. Sel. Areas Commun.* **2020**, *38*, 2430–2444. [CrossRef]
25. Wang, A.L.; Zhang, Y.T.; Yan, Y.X. Heterogeneous Defect Prediction Based on Federated Transfer Learning via Knowledge Distillation. *IEEE Access* **2021**, *9*, 29530–29540. [CrossRef]
26. Chen, J.; Li, J.; Huang, R.; Yue, K.; Chen, Z.; Li, W. Federated Transfer Learning for Bearing Fault Diagnosis with Discrepancy-Based Weighted Federated Averaging. *IEEE Trans. Instrum. Meas.* **2022**, *71*, 3514911. [CrossRef]
27. Wang, A.L.; Zhao, Y.H.; Yang, L.L.; Wu, H.B.; Iwahori, Y. Heterogeneous Defect Prediction Algorithm Combined with Federated Sparse Compression. *IEEE Access* **2023**, *11*, 23739–23753. [CrossRef]
28. Guo, Y.; Tang, X.; Lin, T. FedBR: Improving Federated Learning on Heterogeneous Data via Local Learning Bias Reduction. In Proceedings of the 40th International Conference on Machine Learning, Proceedings of Machine Learning Research, Honolulu, HI, USA, 23–29 July 2023; pp. 12034–12054.
29. Shu, J.; Zhang, W.; Zhou, Y.; Cheng, Z.; Yang, L.T. FLAS: Computation and Communication Efficient Federated Learning via Adaptive Sampling. *IEEE Trans. Netw. Sci. Eng.* **2022**, *9*, 2003–2014. [CrossRef]
30. Lee, G.; Jeong, M.; Kim, S.; Oh, J.; Yun, S.-Y. FedSoL: Bridging Global Alignment and Local Generality in Federated Learning. In Proceedings of the International Workshop on Federated Learning in the Age of Foundation Models in Conjunction with NeurIPS 2023, New Orleans, LA, USA, 16 December 2023.
31. Wang, A.; Yang, L.; Wu, H.; Iwahori, Y. Heterogeneous Defect Prediction Based on Federated Prototype Learning. *IEEE Access* **2023**, *11*, 98618–98632. [CrossRef]
32. Nandi, A.; Xhafa, F.; Kumar, R. A Docker-Based Federated Learning Framework Design and Deployment for Multi-Modal Data Stream Classification. *Computing* **2023**, *105*, 2195–2229. [CrossRef]
33. Liu, J.; Xu, H.; Wang, L.; Xu, Y.; Qian, C.; Huang, J.; Huang, H. Adaptive Asynchronous Federated Learning in Resource-Constrained Edge Computing. *IEEE Trans. Mob. Comput.* **2023**, *22*, 674–690. [CrossRef]

34. Ek, S.; Portet, F.; Lalanda, P.; Vega, G. Artifact: A Federated Learning Aggregation Algorithm for Pervasive Computing: Evaluation and Comparison. In Proceedings of the 2021 IEEE International Conference on Pervasive Computing and Communications Workshops and Other Affiliated Events (PerCom Workshops), Kassel, Germany, 22–26 March 2021; pp. 448–449.
35. Pang, J.; Huang, Y.; Xie, Z.; Han, Q.; Cai, Z. Realizing the Heterogeneity: A Self-Organized Federated Learning Framework for IoT. *IEEE Internet Things J.* **2021**, *8*, 3088–3098. [CrossRef]
36. Sattler, F.; Müller, K.-R.; Samek, W. Clustered Federated Learning: Model-Agnostic Distributed Multitask Optimization Under Privacy Constraints. IEEE Trans. *Neural Netw. Learn. Syst.* **2021**, *32*, 3710–3722. [CrossRef]
37. Zhang, L.; Shen, L.; Ding, L.; Tao, D.; Duan, L.-Y. Fine-Tuning Global Model via Data-Free Knowledge Distillation for Non-IID Federated Learning. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, New Orleans, LA, USA, 18–24 June 2022; pp. 10174–10183.
38. Chen, H.; Vikalo, H. The Best of Both Worlds: Accurate Global and Personalized Models Through Federated Learning with Data-Free Hyper-Knowledge Distillation. *arXiv* **2023**, arXiv:2301.08968.
39. Acar, D.A.E.; Zhao, Y.; Navarro, R.M.; Mattina, M.; Whatmough, P.N.; Saligrama, V. Federated Learning Based on Dynamic Regularization. *arXiv* **2021**, arXiv:2111.04263.
40. Sittijuk, P.; Tamee, K. Fed-RHLP: Enhancing Federated Learning with Random High-Local Performance Client Selection for Improved Convergence and Accuracy. *Symmetry* **2024**, *16*, 1181. [CrossRef]
41. Chen, H.; Yang, L.; Wang, A. Efficient Cross-Project Software Defect Prediction Based on Federated Meta-Learning. *Electronics* **2024**, *13*, 1105. [CrossRef]
42. Shepperd, M.; Song, Q.; Sun, Z.; Mair, C. Data Quality: Some Comments on the NASA Software Defect Datasets. *IEEE Trans. Softw. Eng.* **2013**, *39*, 1208–1215. [CrossRef]
43. D'Ambros, M.; Lanza, M.; Robbes, R. An Extensive Comparison of Bug Prediction Approaches. In Proceedings of the 7th IEEE Working Conference on Mining Software Repositories (MSR 2010), Cape Town, South Africa, 2–3 May 2010; pp. 31–41.
44. Xu, Z.; Liu, J.; Luo, X.; Yang, Z.; Zhang, Y.; Yuan, P.; Tang, Y.; Zhang, T. Software Defect Prediction Based on Kernel PCA and Weighted Extreme Learning Machine. *Inf. Softw. Technol.* **2019**, *106*, 182–200. [CrossRef]
45. Yu, T.; Bagdasaryan, E.; Shmatikov, V. Salvaging Federated Learning by Local Adaptation. *arXiv* **2020**, arXiv:2010.09671.
46. Karimireddy, S.P.; Kale, S.; Mohri, M.; Reddi, S.J.; Stich, S.U.; Suresh, A.T. SCAFFOLD: Stochastic Controlled Averaging for Federated Learning. In Proceedings of the 37th International Conference on Machine Learning, Proceedings of Machine Learning Research, Virtual, 13–18 July 2020; pp. 5132–5143.
47. McMahan, H.B.; Moore, E.; Ramage, D.; Hampson, S.; Agüera y Arcas, B. Communication-Efficient Learning of Deep Networks from Decentralized Data. In Proceedings of the 20th International Conference on Artificial Intelligence and Statistics (AISTATS), Proceedings of Machine Learning Research, Fort Lauderdale, FL, USA, 20–22 April 2017; pp. 1273–1282.
48. Imrana, Y.; Xiang, Y.; Ali, L.; Abdul-Rauf, Z. A Bidirectional LSTM Deep Learning Approach for Intrusion Detection. *Expert Syst. Appl.* **2021**, *185*, 115524. [CrossRef]
49. Matloob, F.; Ghazal, T.M.; Taleb, N.; Aftab, S.; Ahmad, M.; Khan, M.A.; Abbas, S.; Soomro, T.R. Software Defect Prediction Using Ensemble Learning: A Systematic Literature Review. *IEEE Access* **2021**, *9*, 98754–98771. [CrossRef]
50. Turhan, B.; Menzies, T.; Bener, A.B.; Di Stefano, J. On the Relative Value of Cross-Company and Within-Company Data for Defect Prediction. *Empir. Softw. Eng.* **2009**, *14*, 540–578. [CrossRef]
51. Uddin, M.N.; Li, B.; Mondol, M.N.; Rahman, M.M.; Mia, M.S.; Mondol, E.L. SDP-ML: An Automated Approach of Software Defect Prediction Employing Machine Learning Techniques. In Proceedings of the 2021 International Conference on Electronics, Communications and Information Technology (ICECIT), Khulna, Bangladesh, 14–16 September 2021; pp. 1–4.
52. Gong, L.; Jiang, S.; Wang, R.; Jiang, L. Empirical Evaluation of the Impact of Class Overlap on Software Defect Prediction. In Proceedings of the 34th IEEE/ACM International Conference on Automated Software Engineering (ASE), San Diego, CA, USA, 11–15 November 2019; pp. 698–709.
53. Ali, M.; Mazhar, T.; Al-Rasheed, A.; Shahzad, T.; Ghadi, Y.Y.; Khan, M.A. Enhancing Software Defect Prediction: A Framework with Improved Feature Selection and Ensemble Machine Learning. *PeerJ Comput. Sci.* **2024**, *10*, e1860. [CrossRef]
54. Zhou, Y.; Xu, B.; Leung, H.K.N.; Chen, L. An In-Depth Study of the Potentially Confounding Effect of Class Size in Fault Prediction. *ACM Trans. Softw. Eng. Methodol.* **2014**, *23*, 1–51. [CrossRef]
55. Xiao, T.; Jiyou, C.; Chi, Z.; Jingfeng, R.; Ziyu, W.; Guanghua, Z.; He, W.; Gaofei, W.; Jinglu, H.; Yuqing, Z. Survey of Open-Source Software Defect Prediction Method. *J. Comput. Res. Dev.* **2023**, *60*, 1467–1488.
56. Vashisht, R.; Rizvi, S.A.M. Feature Engineering to Heterogeneous Cross Software Projects Defect Prediction: A Novel Framework. *Arab. J. Sci. Eng.* **2023**, *48*, 2539–2560. [CrossRef]
57. Qu, Y.; Zheng, Q.H.; Chi, J.L.; Jin, Y.X.; He, A.C.; Cui, D.; Zhang, H.S.; Liu, T. Using k-Core Decomposition on Class Dependency Networks to Improve Bug Prediction Model's Practical Performance. *IEEE Trans. Softw. Eng.* **2021**, *47*, 348–366. [CrossRef]
58. Song, Q.B.; Guo, Y.C.; Shepperd, M. A Comprehensive Investigation of the Role of Imbalanced Learning for Software Defect Prediction. *IEEE Trans. Softw. Eng.* **2019**, *45*, 1253–1269. [CrossRef]
59. Li, K.; Xiang, Z.L.; Chen, T.; Wang, S.; Tan, K.C. Understanding the Automated Parameter Optimization on Transfer Learning for Cross-Project Defect Prediction: An Empirical Study. In Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering (ICSE '20), Seoul, Republic of Korea, 27 June–19 July 2020; pp. 566–577.

60. Wu, R.X.; Zhang, H.Y.; Kim, S.H.; Cheung, S.C. ReLink: Recovering Links Between Bugs and Changes. In Proceedings of the 19th ACM SIGSOFT Symposium and the 13th European Conference on Foundations of Software Engineering (ESEC/FSE'11), Szeged, Hungary, 5–9 September 2011; pp. 15–25.
61. Nam, J.; Kim, S. CLAMI: Defect Prediction on Unlabeled Datasets (T). In Proceedings of the 30th IEEE/ACM International Conference on Automated Software Engineering (ASE), Lincoln, NE, USA, 9–13 November 2015; pp. 452–463.
62. Tantithamthavorn, C.; Hassan, A.E.; Matsumoto, K. The Impact of Class Rebalancing Techniques on the Performance and Interpretation of Defect Prediction Models. *IEEE Trans. Softw. Eng.* **2020**, *46*, 1200–1219. [CrossRef]
63. Li, Z.Q.; Jing, X.Y.; Zhu, X.K.; Zhang, H.Y.; Xu, B.; Ying, S. On the Multiple Sources and Privacy Preservation Issues for Heterogeneous Defect Prediction. *IEEE Trans. Softw. Eng.* **2019**, *45*, 391–411. [CrossRef]
64. Wardhani, N.W.S.; Rochayani, M.Y.; Iriany, A.; Sulistyono, A.D.; Lestantyo, P. Cross-Validation Metrics for Evaluating Classification Performance on Imbalanced Data. In Proceedings of the 2019 International Conference on Computer, Control, Informatics and its Applications (IC3INA), Tangerang, Indonesia, 28–29 August 2019; pp. 14–18.
65. Ma, Y.; Zhu, S.; Chen, Y.; Li, J. Kernel CCA Based Transfer Learning for Software Defect Prediction. *IEICE Trans. Inf. Syst.* **2017**, *100*, 1903–1906. [CrossRef]

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.