Hindawi
IET Software
Volume 2024, Article ID 5550801, 16 pages
https://doi.org/10.1049/2024/5550801

Research Article

Cross-Project Defect Prediction Using Transfer Learning with Long Short-Term Memory Networks

Hongwei Tao, Lianyou Fu, Qiaoling Cao, Xiaoxu Niu, Haoran Chen, Songtao Shang, and Yang Xian

School of Computer Science and Technology, ZhengZhou University of Light Industry, ZhengZhou, China

Correspondence should be addressed to Hongwei Tao; hongweitao@zzuli.edu.cn

Received 22 August 2023; Revised 15 January 2024; Accepted 6 March 2024; Published 18 March 2024

Academic Editor: Cristina David

With the increasing number of software projects, within-project defect prediction (WPDP) has already been unable to meet the demand, and cross-project defect prediction (CPDP) is playing an increasingly significant role in the area of software engineering. The classic CPDP methods mainly concentrated on applying metric features to predict defects. However, these approaches failed to consider the rich semantic information, which usually contains the relationship between software defects and context. Since traditional methods are unable to exploit this characteristic, their performance is often unsatisfactory. In this paper, a transfer long short-term memory (TLSTM) network model is first proposed. Transfer semantic features are extracted by adding a transfer learning algorithm to the long short-term memory (LSTM) network. Then, the traditional metric features and semantic features are combined for CPDP. First, the abstract syntax trees (AST) are generated based on the source codes. Second, the AST node contents are converted into integer vectors as inputs to the TLSTM model. Then, the semantic features of the program can be extracted by TLSTM. On the other hand, transferable metric features are extracted by transfer component analysis (TCA). Finally, the semantic features and metric features are combined and input into the logical regression (LR) classifier for training. The presented TLSTM model performs better on the *f*-measure indicator than other machine and deep learning models, according to the outcomes of several open-source projects of the PROMISE repository. The TLSTM model built with a single feature achieves 0.7% and 2.1% improvement on Log4j-1.2 and Xalan-2.7, respectively. When using combined features to train the prediction model, we call this model a transfer long short-term memory for defect prediction (DPTLSTM). DPTLSTM achieves a 2.9% and 5% improvement on Synapse-1.2 and Xerces-1.4.4, respectively. Both prove the superiority of the proposed model on the CPDP task. This is because LSTM capture long-term dependencies in sequence data and extract features that contain source code structure and context information. It can be concluded that: (1) the TLSTM model has the advantage of preserving information, which can better retain the semantic features related to software defects; (2) compared with the CPDP model trained with traditional metric features, the performance of the model can validly enhance by combining semantic features and metric features.

1. Introduction

Software defect prediction (SDP) has attracted widespread attention in recent years. It can supply effective guidelines for software developers, and enable them to spend more time and energy on defective software modules (such as files, classes, functions, etc.). By reducing the time spent on non-defect software modules, the software testing cycle can be shortened [1, 2]. According to the data and feature distribution of the project, SDP can be classified as within-project defect prediction (WPDP) [3], cross-project defect prediction (CPDP) [4], and cross-company defect prediction (CCDP) [5]. Existing research has mainly absorbed in WPDP, using historical data to train machine learning models [6]. Nevertheless, WPDP does not perform well with incipient programs that have insufficient data. To fill this gap, CPDP [7, 8] has been proposed. That is, a model is built using the mature projects (called source projects) and then used to predict whether a new project (called target projects) contains defective modules. WPDP relies on single-project historical data for model training, whereas CPDP and CCDP use data from multiple projects or companies for more robust models.

To improve the performance of CPDP, various methods have been proposed. At present, the mainstream CPDP mainly introduces transfer learning approaches [9] to match the features. Nevertheless, existing models have poor prediction performance in CPDP. This is because transfer learning models assume that the source and target domains have similar distributions, which may not be the case in CPDP. The reasons for this problem can be analyzed from three aspects. Firstly, an inappropriate transfer learning [10] approach is selected for CPDP. Transfer learning includes both instance-based and feature-representation methods. Instance-based transfer learning involves directly transferring knowledge or models from one domain to another by reusing labeled data instances. This method adjusts source domain data to match the target domain distribution. It assigns weights to each source domain sample based on its similarity to the target domain. The feature-representation transfer learning refers to the process of using the knowledge learned from a source domain, such as an existing model or dataset, to extract transferable features that can be applied to a target domain. This approach is commonly used in CPDP, where the source and target domains have different feature distributions. It can fit the boundaries of diverse domains so that the features in the domain are also mapped to the same space [11]. In summary, to improve the prediction performance of transfer learning models in CPDP, appropriate transfer learning methods should be selected based on specific scenarios.

Secondly, network models have different scopes of application. Nowadays, many neural network models are used to extract semantic features [12–15]. However, the semantic features extracted by traditional neural network models usually cannot explain how these defects are generated. An example is the convolutional neural network (CNN), which is mainly used to process images; while code defects are usually textual information, CNN is not suitable for this case. Therefore, it is worth considering the network model commonly used to analyze textual information, and use it to extract semantic features that contain defective information.

Thirdly, traditional machine learning models only consider the metric features and do not take full advantage of the information involved in semantic features. In the research of CPDP [16], some researchers used metric features and semantic features to construct prediction models respectively. It was found that the model trained by the latter had better prediction performance. This is because compared with metric features, semantic features contain more information related to defects. Semantic features typically refer to aspects of source code that reflect programmer intent, functionality, and logic. This may include the naming of variables, functions, classes, and methods, the presence and content of comments, code structure, etc. For example, the function in Figure 1 is named "calculate_average" and this naming itself is a semantic feature as it gives information about the purpose and functionality of the function. In addition, the comments for a function increase the readability of the code by providing more information about the function's functionality, inputs, and outputs. Therefore, semantic features are important for understanding the purpose, design intent, and functionality of the code. Metric features are numerical features obtained by measuring or statistics on the code. These measures can relate to the number of lines of code, complexity, coupling, cohesion, and other aspects. In this study, integrating semantic features and metric metafeatures can provide more comprehensive code information for the model.

```
def calculate_average (numbers):
    """
    Calculate the average of a list of numbers.

    Parameters:
    - numbers: List of numeric values.

    Returns:
    - average: The average value.
    """
    if not numbers:
        return None # Avoid division by zero if the list is empty
    total = sum(numbers)
    average = total/len(numbers)
    return average
```

Figure 1: Source code example.

To compensate for the existing shortcomings of traditional neural network models in CPDP, a transfer long short-term memory (TLSTM) network is proposed. The transfer learning approach is associated with the network model to raise the behavior of the CPDP model. The TLSTM network adds a matching layer to the infrastructure of the LSTM network, which is used to match the transferable semantic features. Since LSTM has the function of memory retention and forgetting [17], the irrelevant information can be effectively filtered. It also avoids the gradient vanishing phenomenon that exists in the recurrent neural network (RNN). Compared with CNN (suitable for processing images), LSTM (suitable for processing textual information) is better for capturing the semantic features of code. Specifically, the source file is first parsed into an abstract syntax tree (AST) and converted into the token vector. It is then transformed into an integer vector through a dictionary. Before input into the TLSTM network, the ensemble learning [18] method is adopted to preprocess the data. Then, the semantic features are obtained by the TLSTM network and combined with the metric features extracted by transfer component analysis (TCA). The combined features are input into the logistic regression (LR) [19] classifier for training, and the CPDP model is constructed.

The contributions of this article are summed up as follows.

(1) In this article, a TLSTM model for CPDP is put forward. Due to the LSTM network having the function of memory retention and forgetting, it can effectively extract the semantic features related to code defects. Therefore, compared with other network models, LSTM is more suitable for mining the intrinsic information of defects. Experimental results on six different projects indicate that the TLSTM model outperforms the other models in CPDP.
(2) For the study of CPDP, this paper maps the features of diversity projects into a reproducing kernel Hilbert

space (RKHS) and uses the maximum mean discrepancy (MMD) [20] to measure the distance between features of different projects. Transfer features with similar distances. Moreover, semantic features acquired by the TLSTM and the metric features are combined to construct the CPDP model.

(3) This paper explains why LR is selected as the classifier through comparative experiments. In the experiments, the combined features are input into LR, support vector machine (SVM), and random forest (RF) classifiers to build CPDP models respectively. The outcomes demonstrated that contrasted with SVM and RF, LR has better outcomes.

This article consists of the following parts. Section 2 describes the related work from the three steps of CPDP. Section 3 introduces the research methodology, including the overall framework and each link. The analysis of results and model introduction are reported in Section 4. Section 5 discusses the TLSTM model and the threats to validity. Finally, Section 6 summarizes the full text and points out the research direction.

2. Related Work

Section 2 introduces the latest progress in SDP and the shortcomings and limitations of these methods. It will give a brief description from the following three directions: data preprocessing, feature extraction, and classifier selection.

2.1. Data Preprocessing. Data preprocessing is an essential step in SDP that involves various techniques such as data cleaning, missing value supplement, and class imbalance processing [21]. Since the open-source dataset selected for this paper does not contain dirty data and missing values, the class imbalance problem is mainly considered. Methods for solving class imbalance problems can be summarized in the following three categories: (1) random sampling, (2) cost-sensitive learning, and (3) ensemble learning.

The operation object of random sampling is data. A sample proportion is rebalanced before the model is constructed. It can prevent the classifier from biasing the labels of the majority of class samples during training. Although using the random undersampling method [22] to preprocess the dataset can alleviate the class imbalance problem. However, random undersampling selects bug-free instances in the dataset, so that the proportion of nondefective samples and defective samples is equal. This will lead to a decline in the quantity of defective-free samples, and some important information related to code defects may be lost. Feng et al. [23] adopted random oversampling to tackle the imbalance matter. Although it could increase defective samples, making the proportion of samples with and without defects equal. However, expanding the dataset by generating repeated defective samples will produce redundant information, which makes the model overfit to the defective samples.

Based on the above-mentioned problems of oversampling and undersampling, a cost-sensitive [24] learning method has been proposed to address this question. In SDP, the cost of classifying defective modules as defective-free modules exceeded the other case. Instead of balancing the data distribution by subsampling, cost-sensitive learning sets corresponding prices for diverse classification situations by using a cost matrix. Various learning approaches have been innovated based on cost-sensitive learning, like cost-sensitive decision trees [25], cost-sensitive neural networks [26], etc. These methods can effectively address the class imbalance problem and improve the classification performance of the models in SDP. However, the cost matrix needs to be carefully designed based on the specific problem and domain knowledge to achieve optimal results.

Finally, a popular approach today is to cope with the class imbalance issue through ensemble learning in the data preprocessing step. This type of method combines the strengths of individual learners. The class imbalance is dealt with by choosing suitable weights for learners to underline minority groups. The most widespread means in ensemble learning are bagging [27] and boosting [28]. Bagging combines multiple models, and the final output is based on their majority vote. Boosting trains models sequentially and assigns higher weights to misclassified instances. Gao et al. [29] proposed a new ensemble approach on the foundation of boosting, which combines random sampling methods to address the high dimensionality in features. Contrasting the sampling methods introduced in the previous part, the ensemble learning method is adopted in the experiments. The reason is that compared to other methods like random sampling or cost-sensitive learning, ensemble learning has less prone to overfitting and can better capture the complexity of the data. This method can validly wipe out the problem of class imbalance in the dataset. After the preprocessing operation, the obtained balanced dataset will not have a biased effect on the classification results when it is input to the classifier for training.

2.2. Feature Extraction. Feature extraction is an indispensable link in determining the performance of SDP models, and good features can validly promote the prediction ability of the models. In the field of SDP, features can be broadly classified into two types. One is the metric features of source code, such as Halstead complexity metric, McCabe cyclomatic complexity metric, and object-oriented CK metric [30]. Another type is the semantic features based on the source program. Semantic features capture the context and meaning of the software development process and can include factors such as developer experience, team dynamics, and software architecture. The features are extracted from the program and provided to the classifier to build the SDP models [31].

Methods for operating features fall into two main categories. First is the selection of features, which aims to select a subset of relevant features from the original feature set. This helps reduce the dimensionality and eliminate redundant or irrelevant features. Different weights are set for different features according to their relevance to the category, and the higher the relevance to the category, the greater the weight; then the first few features with a large percentage of weight are selected as key features. Classical feature selection approaches such as the χ^2 test [32], information gain [33], and distance

correlation [34, 35] have been used to study software defect features. The second category is feature extraction, which refers to the process of transforming the original feature set into a new feature space that is more suitable for the classification task. Numerous network models have been applied to extract features from source code. For instance, Zhou et al. [36] proposed a convolutional-graph convolutional network (CGCN) model to extract semantic information. This method combined semantic and metric features and proved that the combined features can elevate the performance.

Feature transfer learning is a popular CPDP approach, addressing domain shift due to differences in data distributions between source and target projects. Bai et al. [37] proposed a three-stage weighting framework for multisource transfer learning (3SW-MSTL) in CPDP. This method aims to improve the transferability of features and achieve better performance than other multisource CPDP methods. Jin [38] used kernel twin support vector machines (KTSVMs) to perform domain adaptation and fit the data of diverse projects with large differences in data distribution. Chen and Ding [39] proposed the transfer AdaBoost (TrAdaBoost) algorithm, which is a type of domain adaptation algorithm that transfers knowledge learned from a source project to a target project with different feature distributions. In the experiments, weights are set for various samples according to the similarity of the range, and TraAdaBoost is used for training. The paired CPDP approach was applied in their research. Nam et al. [40] extended the original TCA algorithm with custom standardization regulations and produced a feature transfer approach TCA+. Xia et al. [41] came up with a hybrid model reconstruction approach (HYDRA) that combines multiple classifiers trained on different subsets of data. The first step of this method is to use a genetic algorithm (GA) to build the classifiers, and the boosting algorithm is adopted to assign weights to the classifiers. These transfer learning methods are effective in handling the domain shift problem in CPDP.

Deep learning [42] and transfer learning have become increasingly popular in the field of CPDP due to their superior performance in feature extraction and knowledge transfer. Wang et al. [43] proposed utilizing the deep belief network (DBN) to abstract semantic information from the programs, and after that take these features to establish a CPDP model. It is proved that the features picked up by network models are superior to the traditional metric features. Qiu et al. [44] united transfer learning with the neural network model and proposed a transfer convolutional neural network (TCNN) model. They used TCNN to extract transferable features among distinct projects, but the model's ability to analyze textual information is still worth exploring. In addition, Deng et al. [45] studied the performance of the LSTM and other network models in WPDP. Their study compared the performance of different machine learning and neural network models. The experimental results showed that the LSTM model outperformed other models.

After summarizing the previous research results, this paper proposes using the TLSTM model to extract transferable semantic features. Specifically, after inputting data into the network model, it first goes through the embedding layer, which maps the input data to a low-dimensional vector space. In the matching layer, the features of different projects are mapped to the kernel space, and the distance is compared to select the correlation features across projects. Then, the iterative training of the network is used to continuously reduce the loss, and finally output the extracted transferable semantic features. The use of deep learning and transfer learning techniques can potentially improve prediction accuracy and reduce the cost of building and maintaining a CPDP model.

2.3. Classifier Selection. After the two steps of data preprocessing and feature extraction, the next step is to input the obtained features into the classifier for model construction. Numerous algorithms have already been applied to SDP, like LR [46], naive Bayes (NB) [47], SVM [48], decision tree (DT) [49], RF [50], etc.

Zain et al. [51] conducted experiments to compare various classifiers in machine learning. The study found that in SDP, the RF has better results on specific datasets. Probably because the RF is obtained by integrating multiple DTs, with more balanced results. In addition, Hall et al. [52] found that models that perform well in SDP are often based on simple modeling techniques, such as NB, LR, and other simple classifiers. Summarizing the previous studies, it is found that different classifiers have their application scenario. It is challenging to find a single classification model that works well on all datasets. Therefore, it is necessary to select the appropriate classifier to build the model according to the specific dataset. If the training set is insufficient, NB is preferable to k-nearest neighbors (KNN), which the latter leads to overfitting [53]. If the training set is large, the opposite is true.

Similarly, the appropriate machine learning algorithm can be selected as a classifier according to the following situations. For example, the NB algorithm assumes that the features are conditionally independent of each other given the class variable. If the condition of the dataset is independent, it can be selected as the classifier. The LR classifier needs to focus on the correlation between features, which cannot be achieved in DTs or support vector machines. It happens that the CPDP should focus on the interrelated features between different projects. Therefore, this experiment chooses logistic regression to construct the CPDP model.

3. Methodology

For the shortcomings in other studies, this paper makes the following improvements. In data preprocessing, an ensemble learning approach is applied to handle the imbalance issue. In feature extraction, an improved model based on the LSTM network, called TLSTM, is proposed to extract the transferable semantic features. It is joined with the metric features extracted by TCA to construct transferable combined features. Finally, in the choice of classifier, LR is selected for experiments and the results of LR and other classifiers on CPDP are compared.

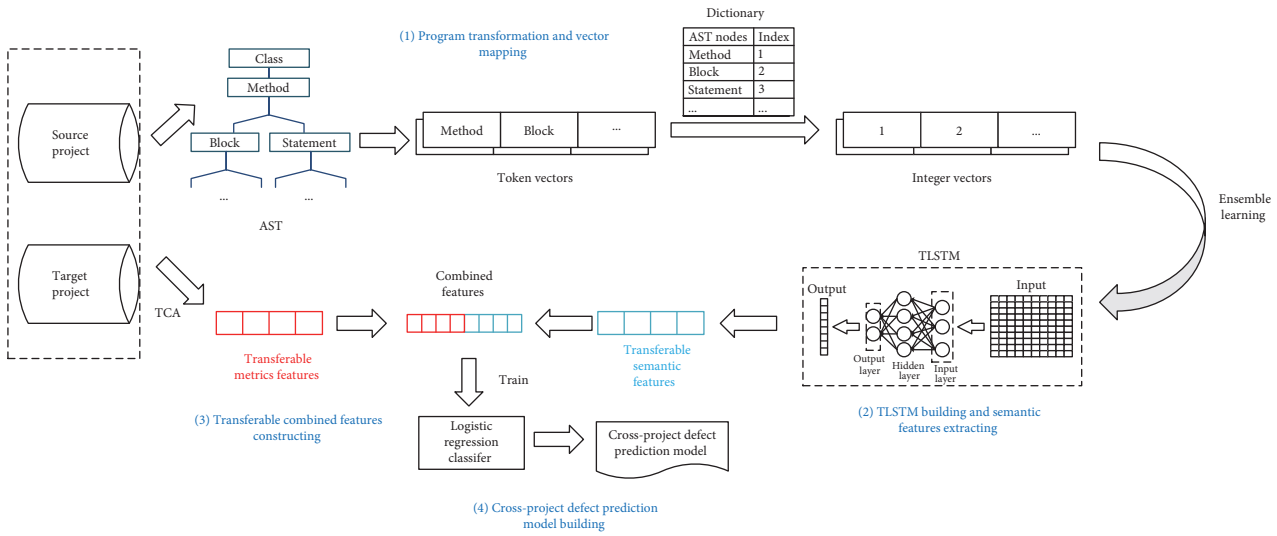

FIGURE 2: Overall framework of CPDP.

3.1. Overall Framework. The overall framework of the experiments is displayed in Figure 2. We will introduce the experimental process in three steps.

Step 1-Source code parsing and vector mapping: in this step, the collected open-source PROMISE dataset is parsed. The source and target codes are converted into AST form respectively, and then the AST node is transformed into token vectors according to the tree structure. Since the network model requires that the inputs be numeric vectors, a dictionary is constructed to map token vectors to integer vectors.

Step 2-TLSTM building and feature extracting: in this step, the TLSTM is used to extract transferable semantic features. Before input into the TLSTM network, the ensemble learning approach is applied to preprocess the imbalance question. When passing the TLSTM model, the first stage is the embedding layer, which is responsible for mapping the integer vectors to a continuous vector space. Then, the input information is filtered in the LSTM cell layer. Finally, the transferable semantic features between different projects are extracted by the matching layer.

Step 3-Construct defect prediction model: at this stage, the CPDP model is constructed. On the one hand, TCA is used to extract the transferable metric features. On the other hand, the transferable semantic features and transferable metric features are combined. Finally, the combined features are transmitted in the LR classifier for training, and the CPDP model is constructed.

We will depict the three steps concretely in the following parts.

3.2. Dataset Selection. The selection of a dataset is the first stage of SDP, and selecting an appropriate dataset is helpful in constructing an accurate prediction model. Currently, the datasets widely used in the research mainly come from the following repositories, including PROMISE [54], AEEEM [55], NASA [56], Softlab, Relink [57], and the JiT data repository. Since the PROMISE data repository is free of missing values and embodies a great number of samples with features and category labels, which is available for training and testing. Therefore, this experiment select multiple projects from this repository as datasets, which is also an extensively manipulated data repository in the field of SDP. The selected projects include Camel, Forrest, Log4j, Synapse, Xalan, and Xerces. These projects contain defective and nondefective modules, providing training data for building prediction models. Table 1 describes the relevant information on the selected six projects, and the defect rate of different projects varies greatly, with the highest even reaching 98.8%. Therefore, it is necessary to preprocess the dataset. The selected projects in this experiment contain traditional metric features and the source code, and the metric attributes used in the experiments are listed in Table 2. By selecting an appropriate dataset and preprocessing the data, the researchers can improve the accuracy and performance of the prediction model and obtain reliable results.

The following is a detailed description of the 20 metric properties:

(1) WMC (weighted methods per class): this indicates the number of methods in a class. This is usually obtained by calculating the complexity of each method and summing it up. This reflects the complexity of the class.
(2) DIT (depth of inheritance tree): this indicates the depth of inheritance of a class. The length of the longest path a class can take from the root of the class hierarchy to the class. This can be used to assess the complexity of the inheritance hierarchy of a class.
(3) NOC (number of children): this indicates the number of direct children in a class. This gives information about the complexity and maintainability of the class.
(4) CBO (coupling between objects): this indicates the degree of coupling between classes, that is, the number of connections between a class and other classes.

Table 1: Information for the projects in PROMISE.

Projects in PROMISE	Number of instances	Number of defects	Defect ratio
Camel-1.6	965	188	19.5%
Forrest-0.8	32	2	6.3%
Log4j-1.2	205	189	92.2%
Synapse-1.2	256	86	33.6%
Xalan-2.7	909	898	98.9%
Xerces-1.4.4	588	437	74.3%

Table 2: List of the metric attributes.

ID	Metrics	Meaning	Type
1	wmc	Method weights in a class	Integer
2	dit	Inheritance tree depth	Integer
3	noc	The number of direct subclasses of a class	Integer
4	cbo	Coupling between objects	Integer
5	rfc	The size of the response set of the class	Integer
6	lcom	Lack of methodological cohesion	Integer
7	ca	Depends on the number of classes in the current class	Integer
8	ce	The number of classes that the current class depends on	Integer
9	npm	The number of public methods in the class	Integer
10	lcom3	Cohesion metrics	Float
11	loc	The number of lines in the Java binary of the class	Integer
12	dam	The ratio of all private attributes in the class	Float
13	moa	The number of fields in a class that a user defines as a class	Integer
14	mfa	The percentage of methods in a class that are inherited	Float
15	cam	The proportion of arguments in a method that are of the type of the argument	Float
16	ic	The number of inheritance coupling classes of a class	Integer
17	cbm	Number of new methods coupled to all inherited methods	Integer
18	amc	Average size of the methods in the class	Float
19	max_cc	The maximum cyclomatic complexity of all methods in a class	Integer
20	avg_cc	The arithmetic average of the cyclomatic complexity of all methods in the class	Float

Highly coupled code can be more difficult to understand and maintain.

(5) RFC (response for a class): this indicates the number of responses a class has to an external request, including method calls and property accesses. This reflects the complexity and responsibility of the class.

(6) LCOM (lack of cohesion in methods): this measures the extent to which there is a lack of cohesion between methods in a class. A high LCOM value may indicate that the methods in the class are not relevant enough and there may be a design problem.

(7) CA (afferent connections): this represents the number of times a class is referenced by other classes, i.e., the dependance of external classes on the class. This can be used to assess the complexity and importance of the class.

(8) CE (efferent connections): it means a kind of the number of references to other classes, namely this kind of dependance on foreign values. This also concerns the complexity of the class and its relationship to other classes.

(9) NPM (number of public methods): this indicates the number of public methods in a class. This provides information about the class interface.

(10) LCOM3: it is an improved version of LCOM that measures the lack of cohesion between methods in a class.

(11) LOC (lines of code): represents the number of physical lines in the source code. This is a basic metric, but often not the best indicator of code quality.

(12) DAM (data access metric): this metric measures access to external data structures. It can be used to evaluate the complexity of a class and its interaction with data.

(13) MOA (number of methods added): this indicates the number of methods added to the child class relative to the parent class. This provides information on the evolution of the class.

(14) MFA (method flow analysis): this is used to measure the complexity of methods and call relationships in a class.

(15) CAM (cohesion among methods of class): measures the cohesion between methods of a class, that is, whether they share the same data. High CAM values may indicate that the methods of the class are more relevant.

(16) IC (inheritance cohesion): represents the cohesion of inheritance relationships within a class. High IC values may indicate more consistent inheritance relationships.

(17) CBM (coupling between methods): this measures the degree of coupling between methods in a class. High CBM values may indicate a strong dependency between methods.

(18) AMC (average method complexity): represents the average complexity of the methods in a class. This can be used to evaluate the overall complexity of the class.

(19) Max_CC (maximum cyclomatic complexity): this represents the cyclomatic complexity of the most complex method in the class. Cyclomatic complexity is a measure of the complexity of a method.

(20) Avg_CC (average cyclomatic complexity): this represents the average cyclomatic complexity of the methods in the class. Cyclomatic complexity is a measure of code complexity.

3.3. Source Code Parsing and Vector Mapping. After obtaining the dataset, the next step is to consider how to transform the source code into integer vectors. Since the code in the project is written in JAVA language, the source code can be converted into AST through java.lang method. Then it is converted into token vectors according to the structure of AST. Because the vectors are composed of tree node types instead of numeric vectors, they cannot be directly used as inputs to the deep learning model. In this case, we need a dictionary to map. The dictionary is shown in Table 3. Through the key-value pairs in the dictionary, we can map the node types in the vector to unique corresponding numbers. After the above procedure, the code is transformed into the integer vector, which can be used as the input of the TLSTM. The details of the LSTM are represented in Section 3.4. In summary, the experiments use the source code to construct the AST and transform it into integer vectors. It should be noted that the class imbalance issue needs to be dealt with before input into the network, which is solved by the ensemble learning method.

3.4. TLSTM Building and Feature Extracting. This section introduces the improved TLSTM model based on the LSTM network used in the experiments. Before that, a brief understanding of the LSTM is given. LSTM is a variant of RNN and is designed to handle long sequence input problems. It is similar to the baseline RNN, yet the way used to compute the hidden state is diverse. The hidden state (h) can abstract information from the sequence of data and change them into outputs. The "memory" of the LSTM, also known as the cell, and its input is the hidden state h_{t-1} and the input x_t. It can decide which information to keep in long-term memory and which irrelevant information to forget. With this distinguishing characteristic, the information related to defects can be discovered. Specifically, the LSTM module contains four interacting layers, i.e., three sigmoid and one tanh layer. The structure is shown in Figure 3.

Table 3: Dictionary mapping table.

ID	Token vector	ID	Token vector
1	PackageDeclaration	17	IfStatement
2	ClassDeclaration	18	CatchClause
3	ConstructorDeclaration	19	CatchClauseParameter
4	StatementExpression	20	ReturnStatement
5	FormalParameter	21	ForStatement
6	ReferenceType	22	ForControl
7	MemberReference	23	ThrowStatement
8	MethodInvocation	24	SuperMemberReference
9	MethodDeclaration	25	SynchronizedStatement
10	SuperMethodInvocation	26	BreakStatement
11	VariableDeclarator	27	InterfaceDeclaration
12	TryStatement	28	SwitchStatement
13	ClassCreator	29	SwitchStatementCase
14	BasicType	30	EnhancedForControl
15	WhileStatement	31	ContinueStatement
16	BlockStatement		

In addition, an important concept in LSTM, "gate", is a method to allow selective passage of information. It is used to raise or reduce the ability of information to reach the cell state. The forget gate, input gate, and output gate are contained in the cell. The functions of these gates and related theoretical equations are introduced in the following.

3.4.1. Forget Gate. The main task of the LSTM cell is to forget irrelevant information from the cell state and retain useful information. This step is achieved by a structure called the "forget gate", which reads the last hidden layer output h_{t-1} and the current input x_t, does the sigmoid nonlinear mapping, and outputs a vector f_t, which is multiplied by the cell state C_{t-1}. The calculation process is shown in Equation (1).

$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] + b_f\right). \tag{1}$$

3.4.2. Input Gate. The next step is to determine which new information is stored in the cell state. There are two parts here, the first part is the sigmoid layer, also known as the input layer, which determines what information will be updated. The second part is the tanh layer, which creates a new vector of candidate values $\widetilde{C}_t$ and adds them to the cell state. The calculation process is shown in Equation (2).

$$\begin{cases} i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i) \\ \widetilde{C}_t = \tanh(W_c \cdot [h_{t-1}, x_t] + b_c) \end{cases}. \tag{2}$$

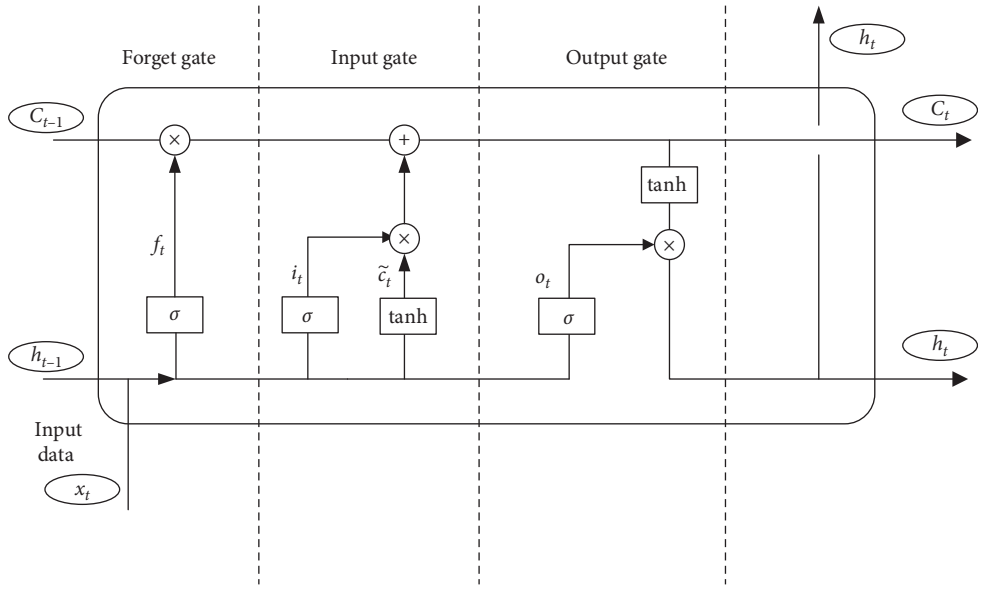

FIGURE 3: LSTM cell structure.

3.4.3. Cell State. The next step is to update the state of the old cell, i.e., C_{t-1} is updated to C_t. By multiplying the old state with f_t, the information not related to the software defect is forgotten, followed by adding $i_t * \widetilde{C}_t$, the new candidate value C_t is obtained. The process is shown in Equation (3).

$$C_t = f_t * C_{t-1} + i_t * \widetilde{C}_t. \tag{3}$$

3.4.4. Output Gate. The output gate is used to determine which value to output based on the cell state. First, which part of the cell state is output by the sigmoid layer is determined; then the cell state is processed through the tanh activation function (to obtain a value between −1 and 1) and it is multiplied with the output of the sigmoid gate, which is calculated as shown in Equation (4).

$$\begin{cases} o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o) \\ h_t = o_t * \tanh(C_t) \end{cases}. \tag{4}$$

The TLSTM model incorporates a matching process to gauge feature discrepancy across projects, adding it during network training loss calculation. The divergence is measured by MMD. By mapping features to RKHS, the MMD is used to measure the distance between features on the distribution curve and estimate the similarity. A smaller distance proves that the features are similar, and then the transfer can be performed. In general, the distribution of features varies widely between projects, and then neural networks can be used for iterative training to minimize the loss function. Particularly, we want to minimize the classification loss, denoted as Loss(c), and the distribution diversity between different features Loss(d). That is, minimizing the sum of these two losses. By minimizing both losses, the TLSTM model aims to improve the accuracy and generalization of the defect prediction model across different projects. The final function is expressed as Equation (5).

$$\min \text{Loss}(c) + \lambda \text{Loss}(d). \tag{5}$$

Here λ represents the regularization parameter.

The basic framework of the TLSTM is displayed in Figure 4, which is composed of an embedding layer, an LSTM cell layer, a linear layer, and a matching layer. The embedding layer converts the integer vectors obtained from the source code into continuous vectors. The LSTM cell layer processes the sequence of input vectors and updates the hidden state of the network accordingly. The linear layer takes the output of the LSTM layer and maps it to a lower-dimensional feature space. Finally, the matching layer measures the feature discrepancy between different projects using the MMD and adds it to the training loss of the network. The final output layer has the function of classifying. The network model is optimized by Adam optimizer and stochastic gradient descent (SGD) algorithm during training, with Adam used for the initial phase of training and SGD used for fine-tuning. This approach can help to improve the stability and generalization performance of the model. Through the TLSTM model, the transferable features are extracted for training the classifier.

3.5. Construct Defect Prediction Model. After going through the previous series of steps, the final step is to construct a CPDP model. The transferable metric features extracted from TCA and the transferable semantic features extracted from TLSTM are combined. Then the combined features are input into the logical regression classifier. After training, the CPDP model is constructed. In the original SDP, it is common to split the samples. The division methods include K-fold cross-validation, random sampling method, etc. However, the data distribution is not similar in CPDP.

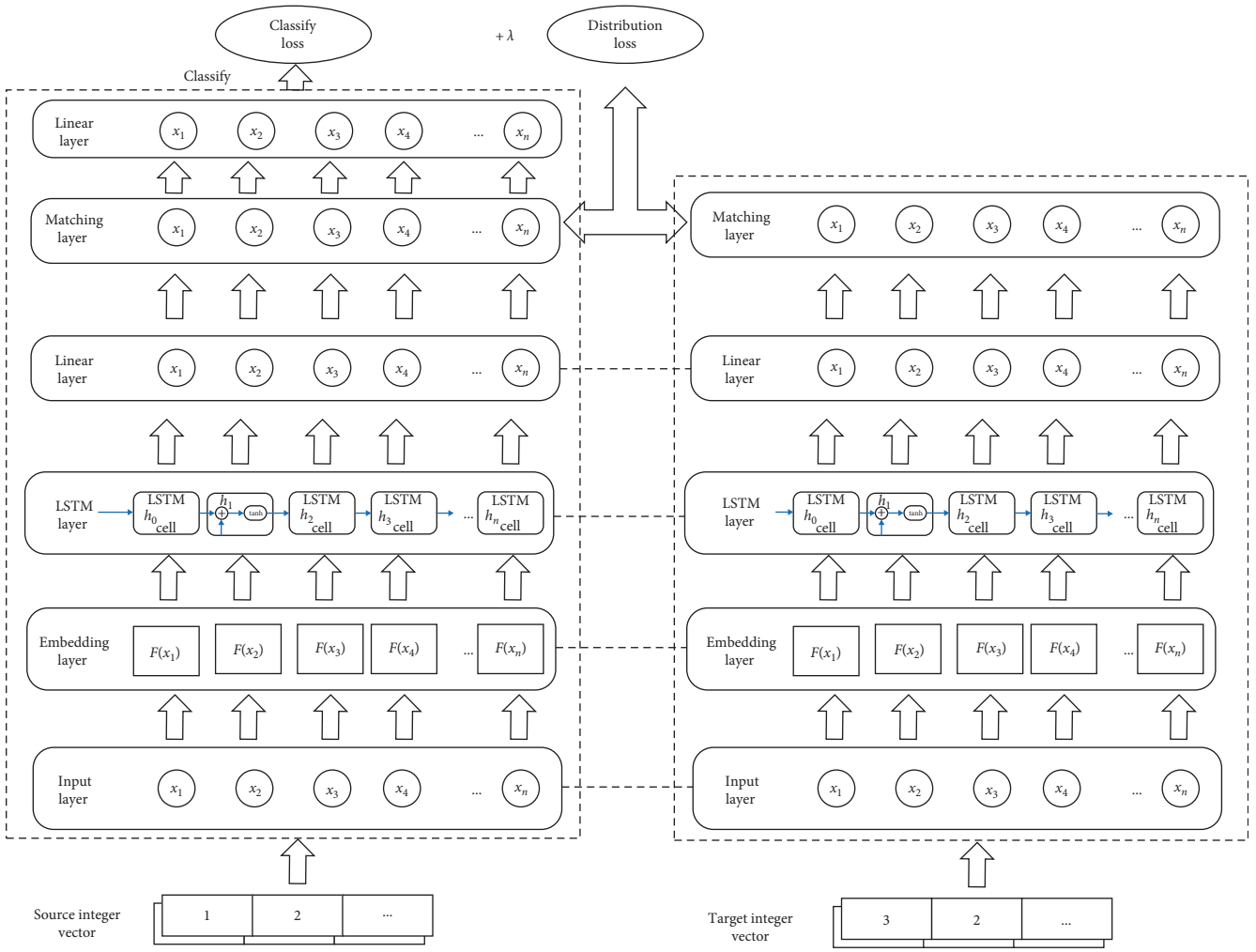

Figure 4: TLSTM network model architecture.

Therefore, the whole source project is taken for training, and the whole target project is taken for testing in the experiments. The model constructed in this method can be used to perform CPDP.

4. Experiments and Results

This section describes the experiments, including model evaluation indicators, various machine learning and neural network models used in the experiments, experimental results, and research questions. Furthermore, we evaluate the effectiveness of the TLSTM model.

4.1. Model Evaluation Indicators. SDP is a classification matter, which is to judge whether a software module is defective or not. Four values can represent various cases of prediction results, they are true positive (TP), false positive (FP), false negative (FN), and true negative (TN). With these four values, we can calculate the following indicators to estimate the CPDP model, which are accuracy, recall, precision, and *f*-measure.

(1) Accuracy (acc) measures how many samples are correctly identified in the two categories, but it does not indicate whether one category can be better recognized by the other. The accuracy calculation process is shown in Equation (6).

$$\text{acc} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}}. \tag{6}$$

(2) Recall symbolizes the probability of predicting positive samples out of those that are actually positive, in other words, how many samples are correctly identified in the whole sample set. Low recall means that many samples are not identified. The equation for calculating the recall rate is shown in Equation (7).

$$\text{recall} = \frac{\text{TP}}{\text{TP} + \text{FN}}. \tag{7}$$

(3) Precision refers to the prediction result, which means the probability that is accurately judged as a true sample. High precision means that many samples identified as true are correctly identified. The expression for the precision is shown in Equation (8).

Table 4: *f*-measure values for each model in CPDP.

Target project	LR	NNFilter	TCA	DBN	DPDBN	DPCNN	DPTCNN	TLSTM	DPTLSTM
Camel-1.6	0.328	0.325	0.324	0.31	0.336	**0.341**	0.331	0.309	0.322
Forrest-0.8	**0.192**	0.165	0.114	0.112	0.161	0.148	0.133	0.16	0.127
Log4j-1.2	0.644	0.649	0.669	0.66	0.657	0.68	0.684	**0.689**	0.652
Synapse-1.2	0.511	0.504	0.517	0.451	0.491	0.511	0.502	0.47	**0.532**
Xalan-2.7	0.609	0.611	0.66	0.661	0.642	0.653	0.655	**0.675**	0.64
Xerces-1.4.4	0.643	0.617	0.611	0.556	0.57	0.645	0.659	0.676	**0.693**

The values in bold represent the most optimal values in a row of data.

$$\text{precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}. \tag{8}$$

(4) The *f*-measure value can be derived from precision and recall. This indicator contemplates both of them, and the *f*-measure is highest when both reach the equilibrium point. Its expression is shown in Equation (9).

$$f\text{-measure} = \frac{2 \times \text{precision} \times \text{recall}}{\text{precision} + \text{recall}}. \tag{9}$$

After introducing the above indicators used to assess the performance, and considering them comprehensively, the experiments use the relatively balanced *f*-measure as the indicator to evaluate the proposed model. *f*-measure is a widely used metric that combines both precision and recall, and it is especially useful when the distribution of positive and negative instances is imbalanced. Moreover, this indicator is often used to evaluate the effect of models in SDP. In the part of experimental results, we judge the performance of distinct models in CPDP by comparing the *f*-measure.

4.2. Model Comparison. It is common practice in machine learning [58, 59] to compare the performance of different types of models on a given task using evaluation metrics. We selected several common models, including three machine learning models and multiple network models [60, 61]. These models have shown significant effect in recent years and are becoming increasingly popular for CPDP tasks. Finally, we compared the *f*-measure indicators of these two types of models to determine which approach is most effective.

4.2.1. Machine Learning Models. LR: an algorithm for binary classification using metric features.

NNFilter: clustering algorithms are adopted to aggregate samples with high similarity and construct source samples based on target samples.

TCA: transfer component analysis, a classical method of learning transferable features.

4.2.2. Neural Network Models. DBN: the DBN model extracts features from code by training a constrained Boltzmann machine. The model consists of 10 hidden layers, each layer containing 100 hidden nodes.

DPDBN: the semantic features extracted by DBN are combined with metric features to train a classifier.

DPCNN: a method that abstracts the semantic information by CNN. Then, the metric features and semantic features are combined. In the parameter settings of CNN, we initialize the network with an embedded layer dimension of 30, batch size of 32, hidden layer node of 100, etc.

DPTCNN: combined semantic features and metric features on top of the transfer CNN.

TLSTM: the new model raised in this article is on the foundation of the LSTM model and adds transfer learning for CPDP. We set the initial parameters for the TLSTM network with hidden layers of 3 and hidden layer nodes of 64, etc.

DPTLSTM: based on the TLSTM, the semantic features and metric features are combined for defect prediction.

4.3. Analysis of Results. This section reveals the design and the experimental results. For the sake of guaranteeing the results are true and effective, we set a uniform random seed to generate the same random value for each experiment. At the same time, to prove the fairness of the experiments, we performed CPDP on the projects and averaged the results. The experimental results of six projects are selected. For each target project, the other source projects conducted a CPDP experiment for this project separately. The CPDP results are recorded in Table 4 and the results are shown below.

Due to space limitations, only the experimental average results of TLSTM, DPTLSTM, and other models are given here to testify the availability of our research approach. By observing the results in CPDP, it is concluded that the proposed TLSTM and DPTLSTM are better than other models in most projects. The results of these models are displayed in Figure 5. By analyzing the experimental results, it can be found that in six open-source Java projects, the *f*-measure value of the proposed model has achieved better results in four of them. These projects are Log4j-1.2, Synapse-1.2, Xalan-2.7, and Xerces-1.4.4. According to the calculation, the proposed model achieves the largest improvement on the Xerces-1.4.4 project, which is 5% higher than the other models. The remaining improvements are Log4j-1.2 (0.7%), Synapse-1.2 (2.9%), and Xalan-2.7 (2.1%). For example, DPTLSTM outperforms other models by 2.9% and 5% on Synapse-1.2 and Xerces-1.4.4 projects, respectively. Similarly, when using a single feature to build the prediction model, the TLSTM model also outperforms the other models on Log4j-1.2 and Xalan-2.7 projects. The improvements are 0.7% and 2.1%, respectively. Overall, the performance improvement ranges from 0.7% to 5%, proving that the proposed model outperforms traditional machine learning and

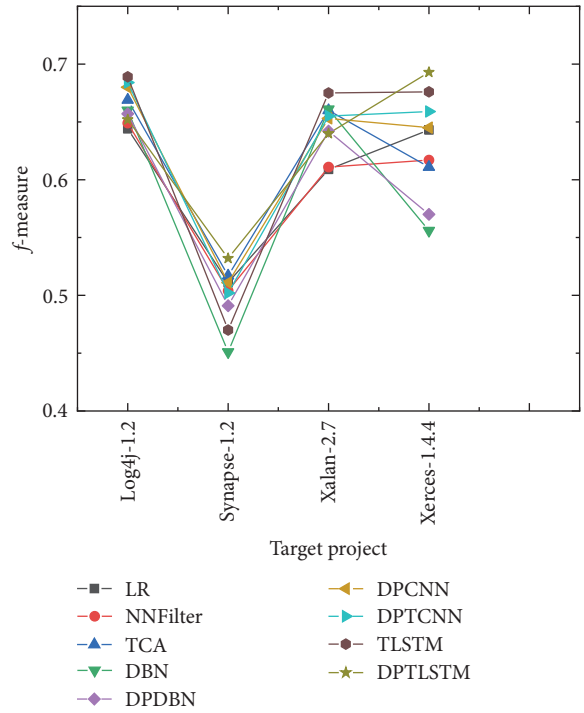

Figure 5: *f*-measure values of different models in CPDP.

Table 5: *f*-measure of a set of CPDP for different models when Xerces as the target project.

Source project	LR	NNFilter	TCA	DBN	DPDBN	DPCNN	DPTCNN	TLSTM	DPTLSTM
Camel-1.6	0.517	0.4	0.562	0.559	0.514	0.575	0.594	**0.747**	0.729
Forrest-0.8	0.336	0.355	0.332	0.418	0.347	0.487	0.482	**0.549**	0.415
Log4j-1.2	0.708	0.734	0.712	0.634	0.729	0.643	0.634	0.676	**0.762**
Synapse-1.2	0.579	0.508	0.692	0.519	0.542	0.702	0.73	0.749	**0.774**
Xalan-2.7	**0.782**	0.78	0.779	0.739	0.763	0.764	0.773	0.643	0.71

The values in bold represent the most optimal values in a row of data.

neural network models on most open-source software projects.

Figure 5 may be unable to represent the experiments in detail, so it is refined to one group of CPDP. We use Xerces as the target project and others as the source projects for CPDP separately. The outcomes are given in Table 5. Through comparison, it is found that the *f*-measure values of TLSTM and DPTLSTM are superior to other models in multiple CPDPs. A contrast between these models is presented in Figure 6.

To ensure the generality of the approach, we conducted the same experiments on six items from the NASA repository. The experimental results are recorded in Table 6, the corresponding box plots are shown in Figure 7. By observing the experimental results, it can be found that the performance of the proposed model is also better than that of other models. Therefore, this research method is valuable to the field of CPDP.

4.4. Research Questions. RQ1: What is the underlying factor for the superior performance of TLSTM compared to other network models?

Answer RQ1: For question 1, the reason is that the LSTM network has the advantage of long short-term memory, which can better retain useful information when processing long sequence inputs. When performing CPDP, the semantic features associated with defects can be extracted and irrelevant information can be forgotten. The ability of the LSTM to capture long-term dependencies and retain important information over time makes it well-suited for this task. This is difficult to be done by CNN and other neural networks, so the TLSTM works better in CPDP. In addition, TLSTM combines the benefits of LSTM with a transfer learning approach, where knowledge learned from source projects is transferred to the target project. Reducing the required training data can enhance model performance and improve generalization to new datasets.

RQ2: Why choose LR as the classifier of the experiments to build the CPDP model?

Answer RQ2: For question 2, by employing distinct classifiers for TLSTM and DPTLSTM in comparative experiments, it was observed that the *f*-measure values achieved under the LR classifier outperformed those under SVM and RF. The experimental findings are visually depicted in

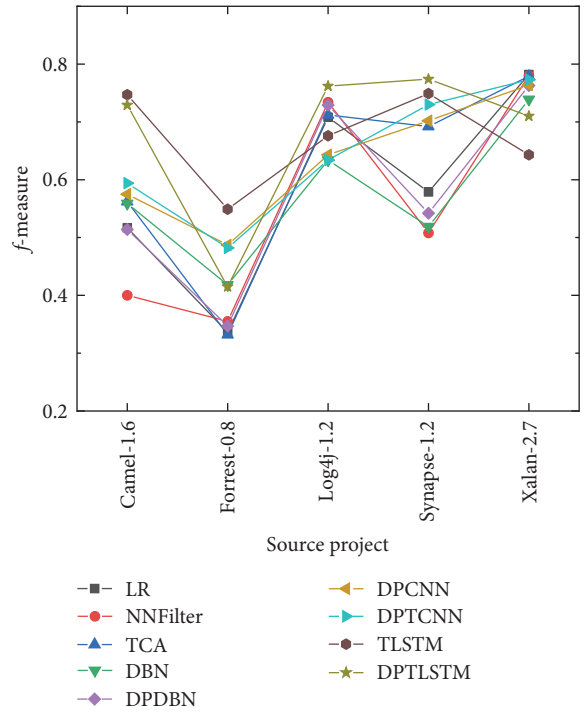

Figure 6: *f*-measure of a set of CPDP for the different models when Xerces as the target project.

Table 6: *f*-measure of different models on the NASA dataset.

Target project	LR	NNFilter	TCA	DBN	DPDBN	DPCNN	DPTCNN	TLSTM	DPTLSTM
CM1	0.288	0.29	0.229	0.261	0.287	0.325	0.36	0.369	**0.37**
JM1	0.403	0.365	0.351	0.446	0.461	0.528	0.533	0.591	**0.627**
KC1	0.608	0.606	0.609	0.626	0.64	0.651	0.68	0.677	**0.687**
KC3	0.604	0.513	0.732	0.733	0.769	0.778	0.786	**0.811**	0.8
MW1	0.379	**0.599**	0.377	0.423	0.433	0.457	0.485	0.575	0.585
PC2	0.603	0.61	0.611	0.566	0.58	0.605	0.611	0.713	**0.805**

The values in bold represent the most optimal values in a row of data.

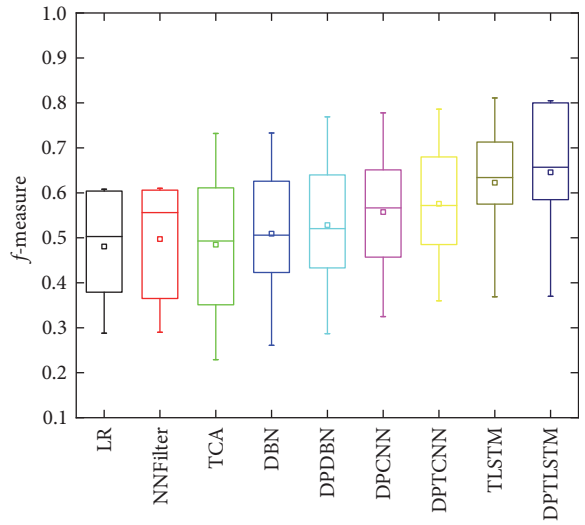

Figure 7: Box plots of different models on the NASA dataset.

Figure 8. Figures 8(a) and 8(b) shows the experimental comparison of TLSTM and DPTLSTM under different classifiers, respectively. The possible reason for this phenomenon is that the LR classifier focuses on the correlation of features between cross-projects during training, which has better results on CPDP compared to SVM and RF.

5. Discussion

In light of the results recorded in Table 4, it can be perceived that the TLSTM and DPTLSTM in most CPDPs are better than the other network models. The reasons for this situation are explained here.

5.1. Why Does TLSTM Work Well?

(1) The TLSTM network model has the function of memory retention and forgetting, which can effectively

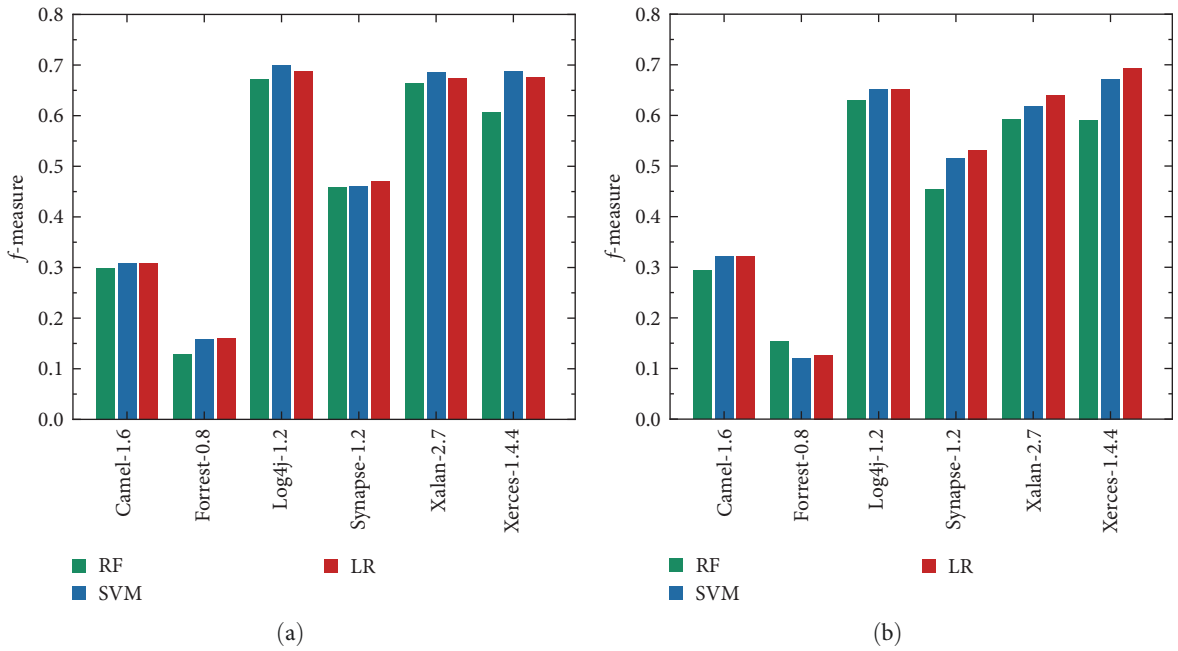

Figure 8: Comparison of TLSTM and DPTLSTM with different classifiers: (a) TLSTM with different classifiers and (b) DPTLSTM with different classifiers.

analyze the semantic features of the code, retain useful information and forget invalid information. In CPDP, this could be important because code changes and bug reports are typically related to previous actions, and TLSTM can effectively capture these dependencies. Moreover, it can parse the content of integer vectors and discover the connection between defects, so it has better experimental results compared with other network models. In addition, a transfer learning algorithm is added to the LSTM, and the iterative training of the network model is used to reduce the MMD distance between the different features. By enhancing the transfer learning ability of features, the function of the CPDP model will be upgraded. Although CNN can similarly extract semantic features, the features discovered by TLSTM can explain the causes of defects.

(2) In contrast with the formerly CPDP models that only apply deep learning to generate features, the TLSTM model considers the divergence in feature distribution. We perform the transfer learning approach by mapping the features to the RKHS, and transfer features with high similarity. By directly learning from source and target project data, TLSTM can better capture the divergence in feature distributions, which is important in the context of CPDP. In other words, TLSTM can identify and focus on the features that are most relevant for predicting bugs in the target project, even if they are different from those significant in the source project. In addition, we discovered that the combined features can effectively promote the function of the CPDP model in some projects.

5.2. Threats to Validity. We explore the validity threat of the experiments as follows. Three aspects of internal validity, external validity, and construct validity are presented and solutions are given.

5.2.1. Threats to Internal Validity. The network parameters of TLSTM lack research. In the experiments, the TLSTM network model is set with an initialization parameter, which does not consider the influence of different parameters on the experimental effect. The settings of different parameters, such as the number of hidden layer units, can be investigated in future studies. In the next stage, we determine to explore whether parameter optimization can contribute to the CPDP results. For the optimization of parameters in the neural network model, we can combine hyperparameter optimization algorithms to explore the use of different hyperparameter optimization methods, such as Bayesian optimization, genetic algorithm, etc., to adjust the key parameters of the neural network. In addition, different learning rate adjustment strategies can be investigated, including learning rate decay, dynamic learning rate adjustment, etc. Determine the most effective learning rate adjustment method during training to accelerate convergence and avoid getting stuck in a local optimal solution. By selecting and optimizing these parameters, we can improve the accuracy and robustness of the models, and ensure that they are effective for a wide range of tasks and datasets.

5.2.2. Threats to External Validity. In the experiments, the LR classifier in machine learning was used in the model construction. Although it was compared with two classifiers, RF and SVM, there was a lack of comparative research on other

classification models. Aiming at the problem of classifier selection for building software defect prediction models, the selection range of classifiers can be extended, including but not limited to decision tree, naive Bayes, k-nearest neighbor, etc. In the next study, the performance of these classifiers on the software defect prediction task will be comprehensively compared. In addition, considering the performance differences of different classifiers in defect prediction, the base classifiers can be integrated to make full use of the advantages of each classifier to improve the comprehensive performance of the model. Each of these models has its strengths and weaknesses, and may be better suited to certain types of tasks. It is unknown that the choice of other classifiers affects the experimental results, so contrast experiments with different classifiers can be added in subsequent studies.

5.2.3. Threats to Construct Validity. Although the *f*-measure can be used as an indicator to estimate the predictive ability of a model, it is not excluded that there are better indicators worth studying. Such as Matthews correlation coefficient (MCC), the area under roc curve (AUC), etc. MCC is a metric that comprehensively considers TP, TN, FP, and FN in binary classification problems. In addition, AUC is a metric used to evaluate the performance of a model under different classification thresholds, which is also widely used to evaluate binary classification models. These indicators are not considered to assess the function of the CPDP model in this article, they are worth studying in the future.

6. Conclusion and Future Work

In conclusion, our proposed TLSTM model represents a significant advancement in addressing the challenges of CPDP. Firstly, the model combines transfer learning and LSTM network to improve the transfer ability of features across items. Secondly, the innovative combination of semantic features and metric features in CPDP is a key factor in improving the predictive ability of the model. Unlike existing machine learning models and traditional neural network approaches, TLSTM demonstrates superior performance.

Furthermore, by leveraging deep learning for feature generation, TLSTM incorporates a matching layer that effectively associates semantic features between source and target projects. This novel approach minimizes classification errors and mitigates distribution diversity among projects, resulting in a substantial improvement in overall performance.

Importantly, experimental results consistently highlight the superiority of TLSTM and its variant, DPTLSTM, across various scenarios in the CPDP task. The outperformance of these models underscores their effectiveness in handling the intricacies of CPDP.

Several issues are worth investigating in future research. First, more approaches will be applied to measure the distribution dispersion between features and find better transfer learning methods to match features. Second, we will conduct more experiments on SDP datasets constructed in other programing languages (C, C++, etc.) to further explore the generalizability of our approach. Finally, the neural network model proposed in other studies can be used to study feature extraction, and explore whether other network models can further improve the prediction effect.

Abbreviations

WPDP:	Within-project defect prediction
CPDP:	Cross-project defect prediction
TLSTM:	The transfer long short-term memory network
LSTM:	The long short-term memory network
AST:	Abstract syntax trees
TCA:	Transfer component analysis
LR:	Logical regression
SDP:	Software defect prediction
CCDP:	Cross-company defect prediction
CNN:	Convolutional neural network
RNN:	Recurrent neural network
RKHS:	Reproducing kernel Hilbert space
MMD:	Maximum mean discrepancy
SVM:	Support vector machine
RF:	Random forest
CGCN:	Convolutional-graph convolutional network
3SW-MSTL:	Three-stage weighting framework for multi-source transfer learning
KTSVMs:	Kernel twin support vector machines
TrAdaBoost:	Transfer AdaBoost
HYDRA:	A hybrid model reconstruction approach
GA:	Genetic algorithm
DBN:	Deep belief network
TCNN:	Transfer convolutional neural network
NB:	Naive Bayes
DT:	Decision tree
KNN:	K-nearest neighbors
SGD:	Stochastic gradient descent
TP:	True positive
FP:	False positive
FN:	False negative
TN:	True negative
AUC:	Area under roc curve
MCC:	Matthews correlation coefficient.

Data Availability

PROMISE repository link: https://github.com/feiwww/PROMISE-backup

Conflicts of Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Authors' Contributions

Hongwei Tao contributed in the supervision, writing–review & editing, funding acquisition, and validation. Lianyou Fu contributed in the writing–original draft, software, investigation, and validation. Qiaoling Cao, Xiaoxu Niu, Haoran Chen, Songtao Shang, and Yang Xian contributed in the writing–review & editing.

Acknowledgments

This work was financially supported by the National Natural Science Foundation of China (61906175), Henan Province Higher Education Teaching Reform Research and Practice Key Project (2021SJGLX292), Doctoral Research Fund of Zhengzhou University of Light Industry (2020BSJJ067) and Science and Technology Project of Henan Province (222102210096, 232102210014).

References

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

[2] X. Yu, M. Wu, Y. Jian, K. E. Bennin, M. Fu, and C. Ma, "Cross-company defect prediction via semi-supervised clustering-based data filtering and MSTrA-based transfer learning," *Soft Computing*, vol. 22, no. 10, pp. 3461–3472, 2018.

[3] X. Jing, F. Wu, X. Dong, and B. Xu, "An improved SDA based defect prediction framework for both within-project and cross-project class-imbalance problems," *IEEE Transactions on Software Engineering*, vol. 43, pp. 321–339, 2017.

[4] C. Liu, D. Yang, X. Xia, M. Yan, and X. Zhang, "A two-phase transfer learning model for cross-project defect prediction," *Information and Software Technology*, vol. 107, no. 107, pp. 125–136, 2019.

[5] L. Chen, B. Fang, Z. Shang, and Y. Tang, "Negative samples reduction in cross-company software defects prediction," *Information and Software Technology*, vol. 62, pp. 67–77, 2015.

[6] L. N. Gong, S. J. Jiang, L. L. Bo, L. Jiang, and J. Y. Qian, "A novel class-imbalance learning approach for both within-project and cross-project defect prediction," *IEEE Transactions on Reliability*, vol. 69, pp. 40–54, 2020.

[7] S. Amasaki, "Cross-version defect prediction: use historical data, cross-project data, or both?" *Empirical Software Engineering*, vol. 25, no. 2, pp. 1573–1595, 2020.

[8] Y. Zhou, Y. Yang, H. Lu et al., "How far we have progressed in the journey? An examination of cross-project defect prediction," *ACM Transactions on Software Engineering and Methodology*, vol. 27, no. 1, pp. 1–51, 2018.

[9] L. Gong, S. Jiang, and L. Jiang, "An improved transfer adaptive boosting approach for mixed-project defect prediction," *Journal of Software: Evolution and Process*, vol. 31, no. 10, Article ID e2172, 2019.

[10] K. Weiss, T. M. Khoshgoftaar, and D. D. Wang, "A survey of transfer learning," *Journal of Big Data*, vol. 3, no. 1, Article ID 9, 2016.

[11] S. J. Pan, I. W. Tsang, J. T. Kwok, and Q. Yang, "Domain adaptation via transfer component analysis," *IEEE Transactions on Neural Networks*, vol. 22, no. 2, pp. 199–210, 2011.

[12] J. Li, P. He, J. Zhu, and M. R. Lyu, "Software defect prediction via convolutional neural network," in *2017 IEEE International Conference on Software Quality, Reliability and Security (QRS)*, pp. 318–328, IEEE, Prague, Czech Republic, 2017.

[13] T. Zimmermann and N. Nagappan, "Predicting defects using network analysis on dependency graphs," in *The 30th International Conference on Software Engineering*, pp. 531–540, ACM, 2008.

[14] Z. Xu, S. Li, J. Xu et al., "LDFR: learning deep feature representation for software defect prediction," *Journal of Systems and Software*, vol. 158, Article ID 110402, 2019.

[15] P. Wei, Y. Ke, and C. K. Goh, "Feature analysis of marginalized stacked denoising autoencoder for unsupervised domain adaptation," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 30, no. 5, pp. 1321–1334, 2019.

[16] K. Shi, Y. Lu, G. Liu, Z. Wei, and J. Chang, "MPT-embedding: an unsupervised representation learning of code for software defect prediction," *Journal of Software: Evolution and Process*, vol. 33, no. 4, Article ID e2330, 2021.

[17] A. Majd, M. Vahidi-Asl, A. Khalilian, P. Poorsarvi-Tehrani, and H. Haghighi, "SLDeep: statement-level software defect prediction using deep-learning model on static code features," *Expert Systems with Applications*, vol. 147, Article ID 113156, 2020.

[18] H. Tong, B. Liu, and S. Wang, "Kernel spectral embedding transfer ensemble for heterogeneous defect prediction," *IEEE Transactions on Software Engineering*, vol. 4, no. 9, pp. 1866–1906, 2021.

[19] A. E. C. Cruz and K. Ochimizu, "Towards logistic regression models for predicting fault-prone code across software projects," in *2009 3rd International Symposium on Empirical Software Engineering and Measurement*, pp. 460–463, IEEE, Lake Buena Vista, FL, USA, 2009.

[20] R. Krishna and T. Menzies, "Bellwethers: a baseline method for transfer learning," *IEEE Transactions on Software Engineering*, vol. 45, pp. 1081–1105, 2018.

[21] N. Limsettho, K. E. Bennin, J. W. Keung, H. Hata, and K. Matsumoto, "Cross project defect prediction using class distribution estimation and oversampling," *Information and Software Technology*, vol. 100, pp. 87–102, 2018.

[22] S. Wang and X. Yao, "Using class imbalance learning for software defect prediction," *IEEE Transactions on Reliability*, vol. 62, no. 2, pp. 434–443, 2013.

[23] S. Feng, J. Keung, X. Yu et al., "COSTE: complexity-based oversampling technique to alleviate the class imbalance problem in software defect prediction," *Information and Software Technology*, vol. 129, Article ID 106432, 2021.

[24] D. Ryu, J.-I. Jang, and J. Baik, "A transfer cost-sensitive boosting approach for cross-project defect prediction," *Software Quality Journal*, vol. 25, no. 1, pp. 235–272, 2017.

[25] N. Seliya and T. M. Khoshgoftaar, "The use of decision trees for cost-sensitive classification: an empirical study in software quality prediction," *WIREs Data Mining and Knowledge Discovery*, vol. 1, no. 5, pp. 448–459, 2011.

[26] J. Zheng, "Cost-sensitive boosting neural networks for software defect prediction," *Expert Systems with Applications*, vol. 37, no. 6, pp. 4537–4543, 2010.

[27] Y. Zhang, D. Lo, X. Xia, and J. Sun, "An empirical study of classifier combination for cross-project defect prediction," in *Proceedings of the IEEE 39th Annual Computer Software and Applications Conference (COMPSAC)*, vol. 2, pp. 264–269, IEEE, Taichung, Taiwan, 2015.

[28] D. Ryu, O. Choi, and J. Baik, "Value-cognitive boosting with a support vector machine for cross-project defect prediction," *Empirical Software Engineering*, vol. 21, no. 1, pp. 43–71, 2016.

[29] K. Gao, T. M. Khoshgoftaar, and A. Napolitano, "The use of ensemble-based data preprocessing techniques for software defect prediction," *International Journal of Software Engineering and Knowledge Engineering*, vol. 24, no. 9, pp. 1229–1253, 2014.

[30] R. Subramanyam and M. S. Krishnan, "Empirical analysis of CK metrics for object-oriented design complexity: implications for software defects," *IEEE Transactions on Software Engineering*, vol. 29, no. 4, pp. 297–310, 2003.

[31] S. Wang, T. Liu, and L. Tan, "Automatically learning semantic features for defect prediction," in *2016 IEEE/ACM 38th International Conference on Software Engineering (ICSE)*, pp. 297–308, ACM, 2016.

[32] M. Vikas and K. Prabhpreet, "Lung cancer detection using chi-square feature selection and support vector machine algorithm," *International Journal of Advanced Trends in Computer Science and Engineering*, vol. 10, no. 3, pp. 2050–2060, 2021.

[33] S. Hosseini, B. Turhan, and M. Mäntylä, "A benchmark study on the effectiveness of search-based data selection and feature selection for cross project defect prediction," *Information and Software Technology*, vol. 95, pp. 296–312, 2018.

[34] Y. Kamei, T. Fukushima, S. McIntosh, K. Yamashita, N. Ubayashi, and A. E. Hassan, "Studying just-in-time defect prediction using cross-project models," *Empirical Software Engineering*, vol. 21, no. 5, pp. 2072–2106, 2016.

[35] C. López-Martín, Y. Villuendas-Rey, M. Azzeh, A. Bou Nassif, and S. Banitaan, "Transformed k-nearest neighborhood output distance minimization for predicting the defect density of software projects," *Journal of Systems and Software*, vol. 167, Article ID 110592, 2020.

[36] C. Zhou, P. He, C. Zeng, and J. Ma, "Software defect prediction with semantic and structural information of codes based on graph neural networks," *Information and Software Technology*, vol. 152, Article ID 107057, 2022.

[37] J. Bai, J. Jia, and L. F. Capretz, "A three-stage transfer learning framework for multi-source cross-project software defect prediction," *Information and Software Technology*, vol. 150, Article ID 106985, 2022.

[38] C. Jin, "Cross-project software defect prediction based on domain adaptation learning and optimization," *Expert Systems with Applications*, vol. 171, Article ID 114637, 2021.

[39] Y. Chen and X. Ding, "Research on cross-project software defect prediction based on transfer learning," *AIP Conference Proceedings*, vol. 1955, Article ID 040083, 2018.

[40] J. Nam, S. J. Pan, and S. Kim, "Transfer defect learning," in *2013 35th International Conference on Software Engineering (ICSE)*, pp. 382–391, IEEE, San Francisco, CA, USA, 2013.

[41] X. Xia, D. Lo, S. J. Pan, N. Nagappan, and X. Wang, "HYDRA: massively compositional model for cross-project defect prediction," *IEEE Transactions on Software Engineering*, vol. 42, no. 10, pp. 977–998, 2016.

[42] M. Long, Y. Cao, Z. Cao, J. Wang, and M. I. J. Jordan, "Transferable representation learning with deep adaptation networks," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 41, no. 12, pp. 3071–3085, 2019.

[43] S. Wang, T. Liu, J. Nam, and L. Tan, "Deep semantic feature learning for software defect prediction," *IEEE Transactions on Software Engineering*, vol. 46, pp. 1267–1293, 2020.

[44] S. Qiu, H. Xu, J. Deng, S. Jiang, and L. Lu, "Transfer convolutional neural network for cross-project defect prediction," *Applied Sciences*, vol. 9, no. 13, Article ID 2660, 2019.

[45] J. Deng, L. Lu, and S. Qiu, "Software defect prediction via LSTM," *IET Software*, vol. 14, no. 4, pp. 443–450, 2020.

[46] R. Moser, W. Pedrycz, and G. Succi, "A comparative analysis of the efficiency of change metrics and static code attributes for defect prediction," in *2008 30th International Conference on Software Engineering(ICSE)*, pp. 181–190, IEEE, 2008.

[47] K. Zhu, N. Zhang, S. Ying, and X. Wang, "Within-project and cross-project software defect prediction based on improved transfer naive bayes algorithm," *Computers, Materials & Continua*, vol. 63, pp. 891–910, 2020.

[48] Y. Ma, G. Luo, X. Zeng, and A. Chen, "Transfer learning for cross-company software defect prediction," *Information and Software Technology*, vol. 54, no. 3, pp. 248–256, 2012.

[49] M. Gashler, C. Giraud-Carrier, and T. Martinez, "Decision tree ensemble: small heterogeneous is better than large homogeneous," in *Seventh International Conference on Machine Learning and Applications*, pp. 900–905, IEEE, San Diego, CA, USA, 2008.

[50] M. J. Siers and M. Z. Islam, "Software defect prediction using a cost sensitive decision forest and voting, and a potential solution to the class imbalance problem," *Information Systems*, vol. 51, pp. 62–71, 2015.

[51] Z. M. Zain, S. Sakri, and N. H. A. Ismail, "Application of deep learning in software defect prediction: systematic literature review and meta-analysis," *Information and Software Technology*, vol. 158, Article ID 107175, 2023.

[52] T. Hall, S. Beecham, D. Bowes, D. Gray, and S. Counsell, "A systematic literature review on fault prediction performance in software engineering," *IEEE Transactions on Software Engineering*, vol. 38, no. 6, pp. 1276–1304, 2012.

[53] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting," *Journal of Machine Learning Research*, vol. 15, no. 56, pp. 1929–1958, 2014.

[54] M. Jureczko and L. Madeyski, "Towards identifying software project clusters with regard to defect prediction," in *The 6th International Conference on Predictive Models in Software Engineering*, pp. 1–10, ACM, 2010.

[55] M. D'Ambros, M. Lanza, and R. Robbes, "Evaluating defect prediction approaches: a benchmark and an extensive comparison," *Empirical Software Engineering*, vol. 17, no. 4-5, pp. 531–577, 2012.

[56] M. Shepperd, Q. Song, Z. Sun, and C. Mair, "Data quality: some comments on the NASA software defect datasets," *IEEE Transactions on Software Engineering*, vol. 39, no. 9, pp. 1208–1215, 2013.

[57] R. Wu, H. Zhang, S. Kim, and S.-C. Cheung, "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*, pp. 15–25, ACM, 2011.

[58] F. Huang, Z. Cao, J. Guo, S.-H. Jiang, S. Li, and Z. Guo, "Comparisons of heuristic, general statistical and machine learning models for landslide susceptibility prediction and mapping," *CATENA*, vol. 191, Article ID 104580, 2020.

[59] Z. Chang, F. Huang, J. Huang et al., "An updating of landslide susceptibility prediction from the perspective of space and time," *Geoscience Frontiers*, vol. 14, no. 5, Article ID 101619, 2023.

[60] F. Huang, J. Zhang, C. Zhou, Y. Wang, J. Huang, and L. Zhu, "A deep learning algorithm using a fully connected sparse autoencoder neural network for landslide susceptibility prediction," *Landslides*, vol. 17, no. 1, pp. 217–229, 2020.

[61] F. Huang, H. Xiong, C. Yao, F. Catani, C. Zhou, and J. Huang, "Uncertainties of landslide susceptibility prediction considering different landslide types," *Journal of Rock Mechanics and Geotechnical Engineering*, vol. 15, no. 11, pp. 2954–2972, 2023.