IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received November 29, 2019, accepted December 17, 2019, date of publication December 20, 2019,
date of current version December 31, 2019.

Digital Object Identifier 10.1109/ACCESS.2019.2961129

DeepCPDP: Deep Learning Based Cross-Project Defect Prediction

DEYU CHEN[1], XIANG CHEN[1], (Member, IEEE), HAO LI[2], JUNFENG XIE[3], AND YANZHOU MU[2]

[1]School of Information Science and Technology, Nantong University, Nantong 226019, China
[2]College of Intelligence and Computing, Tianjin University, Tianjing 300072, China
[3]School of Computer Science, Fudan University, Shanghai 200433, China

Corresponding author: Xiang Chen (xchencs@ntu.edu.cn)

This work was supported in part by the National Natural Science Foundation of China under Grant 61702041, Grant 61872263, and Grant 61202006, and in part by the Nantong Application Research Plan under Grant JC2018134 and Grant JC2019106.

ABSTRACT Cross-project defect prediction (CPDP) is an active research topic in the domain of software defect prediction, since CPDP can be applied to the following scenarios: the target project for software defect prediction is a new project or the target project does not have enough labeled modules. Most of the previous work tried to utilize the labeled dataset gathered from other projects (i.e., the source projects) and then proposed transfer learning based methods to reduce the data distribution difference between different projects. In this article, we propose a deep learning based CPDP method DeepCPDP. For this method, we represent source code of each extracted program module by using simplified abstract syntax tree (SimAST). For a node of SimAST, we only keep its node type, since this is project-independent, while we ignore the name of method and variable, since these information are project-specific. Therefore, SimAST is project-independent and especially suitable for the task of CPDP. Then, we extract the token vector from each module after it is modeled via SimAST. Moreover, we design a new unsupervised based embedding method SimASTToken2Vec to learn meaningful representation for these extracted token vectors. Later, we employ Bi-directional Long Short-Term Memory (BiLSTM) neural network to automatically learn semantic features from embedded token vectors. In addition, we use attention mechanism over the BiLSTM layer to learn the weight of the vectors from the learned semantic features. Finally, we construct CPDP models via Logistic regression classifier. To show the effectiveness of DeepCPDP, ten large-scale projects from different application domains are used and AUC measure is used to measure the prediction performance of trained models. By using Scott-Knott test, we can find DeepCPDP can significantly outperform eight state-of-the-art baselines. Moreover, we also verify that the usage of SimASTToken2Vec, BiLSTM and attention mechanism is competitive in our proposed method.

INDEX TERMS Software defect prediction, cross-project defect prediction, bi-directional long short-term memory, embedding method, attention mechanism.

I. INTRODUCTION

Software defect prediction (SDP) [15], [16] can assist developers to predict defective program modules in advance. Therefore, the limited testing resources can be more rational allocated to effectively testing these identified modules. In particular, SDP can train models after mining and analyzing software repositories, and then these trained models can be assisted to distinguish defective modules from non-defective modules in the project. The granularity of modules for extracting can be set to file, method, or even code change based on developer usage scenario. Most of previous work focus on the scenario of within-in project defect prediction (WPDP) scenario. In this scenario, developers train SDP models on labeled modules and then perform predict on the remaining unlabeled modules within the same project. However, in some cases, a target project may be a new project without any labeled modules or has a few labeled modules,

The associate editor coordinating the review of this manuscript and approving it for publication was Shuiguang Deng.

which is not enough to train a high-quality SDP model. However, if we directly use gathered labeled data from other projects (i.e., the source projects) and do not perform any data preprocessing, the performance of trained models may unsatisfactory, since the data distribution between different projects can not satisfy the similar distribution assumption in most cases. Therefore, researchers have designed different transfer learning based methods to alleviate this kind of data distribution difference in the scenario of cross-project defect prediction (CPDP) [21].

Source code, which is a specific kind of formal languages, can contain rich syntax and semantic information. However, previously proposed software metrics are designed in the manual manner. These metrics mainly focus on code complexity or characteristics of development process. Therefore, these metrics can not fully capture such complicated syntax information and semantic information in the source code [30], [55]. Recently, researchers resorted to deep learning (a powerful representation learning method) to learn meaningful semantic metrics from the extracted token vectors. These token vectors are extracted from the abstract syntax tree (AST) of the source code, therefore, these learned semantic features have a higher correlation with defects and their proposed methods are more promising. Wang *et al.* [55] leveraged DBN (deep belief network) to automatically learn semantic metrics from token vectors extracted from program modules' ASTs and then used these metrics to construct SDP models. Then, Li *et al.* [30] proposed CNN (convolutional neural network) based framework. In this framework, they encoded extracted token vectors as numerical vectors via word embedding and then used CNN to automatically learn semantic features. Finally they used both semantic features and traditional hand-crafted metrics to construct SDP models. Experimental results [30], [55] of these two studies showed that the semantic metrics learned by deep neural network (DNN) can significantly outperform the hand-crafted software metrics.

Motivated by these two deep learning based SDP studies [30], [55], we propose a new method DeepCPDP, which is a customization method for CPDP problem via deep learning. The process of DeepCPDP can be summarized as follows, we represent source code of each extracted program module via simplified abstract syntax tree (SimAST). For a node of SimAST, we only keep its node type, since this kind of information is project-independent. If we use full AST to model the source code of extracted program modules, some of gathered information (e.g., names of methods and variables) are project-specific. This will result in large distribution difference between the source project and the target project. It is not hard to find the SimAST is project-independent and especially suitable for the task of cross-project defect prediction. Then we extract token vectors from program modules' SimAST. In addition, since more meaningful representations of token vectors might be helpful to improve the prediction performance of trained SDP models, we propose a new unsupervised based embedding algorithm SimASTToken2Vec and use SimASTToken2Vec to automatically learn meaningful representation for token vectors. Finally, we use BiLSTM (Bi-directional Long Short-Term Memory) neural network [49] to learn contextual semantic features from vectorized token vectors. Compared to other deep learning models (such as CNN and RNN), BiLSTM has the advantage in capturing the long context relationships in the source code, where dependent code elements (such as try and catch in Java) are scattered with a long distance. Moreover, we use attention mechanism over the BiLSTM layer to learn the weight of the vectors from the learned semantic features. Finally, we construct CPDP models via Logistic regression classifier.

To show the competitiveness of the method DeepCPDP, we fist select ten large-scale projects in different application domains as experimental subjects. Then we adopt AUC measure to evaluate the performance of the trained models. The main findings of our empirical studies can be briefly summarized as follows: (1) Our proposed unsupervised based embedding method SimASTToken2Vec is helpful to improve the prediction performance of DeepCPDP. (2) Using BiLSTM neural network can significantly outperform CNN model in DeepCPDP. (3) Using attention mechanism can improve the prediction performance of DeepCPDP. (4) Our proposed method DeepCPDP can significantly outperform eight state-of-the-art baselines in CPDP (i.e., Li17-CNN [30], CamargoCruz09-DT [5], Turhan09-DT [54], Menzies11-RF [37], Watanabe08-DT [56], Ma12-DT [36], Panichella14-LR [43] and Amaski15-DT [1]). The average of the performance improvement is 6.18%, 21.17%, 12.13%, 18.30%, 12.34%, 5.07%, 4.52% and 5.62% respectively.

The main contributions can be summarized as follows:

- To our best knowledge, we are the first to propose a new CPDP method DeepCPDP, which can automatically learn project-independent semantic features by using deep learning. DeepCPDP first models source code of each extracted module via SimAST. Then it utilizes SimASTToken2Vec to learn meaningful representations for token vectors, since our method assumes that tokens appearing in the similar context should have the similar semantic information. Then it uses BiLSTM neural network to learn contextual semantic features from vectorized token vectors, Later it uses attention mechanism over the BiLSTM layer to further learn the weight of the vectors. Finally it constructs CPDP models via Logistic regression classifier.
- Large-scale empirical studies are designed to show the effectiveness of DeepCPDP by making a comparison with eight state-of-the-art baselines in CPDP. In addition, experimental results also show the usage of our proposed SimASTToken2Vec, BiLSTM and attention mechanism is more competitive in DeepCPDP when compared to other candidate settings.

The remaining part of this article is organized as follows. Section II analyzes the background of SDP and summarizes related work for CPDP. Section III first introduces the framework of our proposed CPDP method DeepCPDP and then shows the details of important steps in DeepCPDP. Section IV gives the details of our experimental setup, including experimental subjects, evaluation measure, and statistical analysis methods. Section V performs result analysis for four research questions. Section VI compares DeepCPDP in terms of effort-aware evaluation measures and analyzes the computational cost of DeepCPDP. Section VII discusses some potential internal, external and construct threats to the validity of our empirical studies. Section VIII summarizes this article and discusses several future directions for researchers to explore.

II. BACKGROUND AND RELATED WORK

A. BACKGROUND OF SOFTWARE DEFECT PREDICTION

In different phases of software development process (e.g., requirement analysis, software design and software coding), software defects may be introduced unconsciously. Hidden defects in the projects will output unexpected results and even result in economic loss that are difficult to estimate for enterprises after these software projects are deployed to real working environment.

Since available software quality assurance resources allocated for software testing (or code inspection) are usually limited, effective approaches are required by developers to identify all the defective modules in the early phase of development. This is helpful to optimize the allocation of limited testing resources. Software defect prediction [16], [34], [41] is one of such effective static methods. The target can be set to defect-proness or defect density of the program modules depending on the real application requirements. When the prediction target is set to defect-proness, the brief process can be summarized in Figure 1.[1] Firstly, software modules can be easily extracted from software repositories, such as version control systems (e.g., SVN, GIT, CVS), which store the source codes and commit messages, bug tracking systems (e.g., Jira, Bugzilla), which manage bug reports. Secondly, software metrics (i.e., features) are manually designed and these metrics are used to measure extracted program modules. These hand-crafted metrics are mainly designed based on the code complexity (such as Halstead features, McCabe features, CK features), development process (such as code churn based features), or developer experience. The type of the modules can be labeled after using SZZ method [12], [50], which links bug reports to their bug-fixing changes and considers modules related to bug-fixing changes as defective modules. Thirdly, SDP models can be trained based on the gathered labeled SDP datasets and these trained models are later utilized to distinguish defective modules from non-defective modules for new modules.

[1]In Figure 1, the red color is used to show the defective modules and the green color is used to show the non-defective modules.

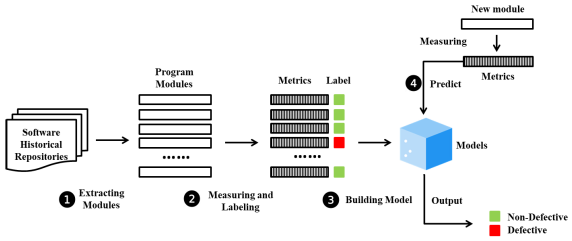


FIGURE 1. The brief process of software defect prediction when the prediction target of program modules is set to defect-proneness.

B. RELATED WORK ANALYSIS FOR CROSS-PROJECT DEFECT PREDICTION

When applying software defect prediction to real software development, a target project may be a new project or may have a few labeled modules to train a high-quality model. For this situation, a very simple solution is to use the labeled dataset gathered from other projects to train models. However, due to different application domains, development processes, used programming languages, developers' experience in different projects, a certain data distribution difference between two different projects exist in most cases.

Researchers conducted several large-scale empirical studies on software projects from real world to investigate the feasibility of CPDP. Zimmermann *et al.* [64] analyzed large-scale projects gathered from open-source communities and commercial corporation (i.e., Microsoft). After gathering results from more than 600 cross-project defect predictions, they found only 3% can achieve satisfactory performance. He *et al.* [18] only analyzed open-source projects. After running more than 160,000 cross-project defect predictions, they found only 0.3% to 4.7% CPDP pairs can achieve satisfactory performance when considering different classifiers. The above two empirical studies [18], [64] showed CPDP is a serious challenge problem. However, when considering effort-aware evaluation measures, Rahman *et al.* [46] surprisingly found the CPDP methods can perform significantly better than the random method and is no worse than the WPDP methods.

In the past years, a number of CPDP methods have been proposed to reduce data distribution difference between the source project and the target project. Detailed information of existing proposed CPDP methods can be found in a recent meta-analysis and systematic literature review [21]. In this article, we briefly classify existing CPDP methods into three categories: supervised homogeneous methods, supervised heterogeneous methods and semi-supervised methods. In this article, since we aim to propose novel supervised CPDP methods via deep learning, we only summarize related work of supervised homogeneous methods in this subsection.

Some methods focus on relevant source project selection. For example, He *et al.* [18] utilized 16 distribution characteristics, which can be used to measure the central tendency and dispersion of metric values, to select relevant projects. Krishna and Menzies [28] considered the concept of bellwether. In the context of SDP, the bellwether is the source

project whose training data can train the model with the best performance when compared to other source projects. Liu *et al.* [33] proposed a two-phase method TPTL. This method can automatically choose two source projects, which have highest distribution similarity to the target project. Then it leverage TCA+ method [40] to build two models for these two projects respectively and then combine their prediction results. Some methods focus on instance selection or instance weight setting for the source project. Two classical instance selection methods are Burak filter proposed by Turhan *et al.* [54] and Peter filter proposed by Peters *et al.* [44]. Li *et al.* [31] conducted an extensive comparison of instance-level filters. Ma *et al.* [36] proposed Transfer Naive Bayes (TNB) method. This method estimates the distribution of the target project, and transfers cross-project data information into the weight of the modules in the source project. Some methods focus on feature mapping and feature selection. Nam *et al.* [40] applied a classical transfer learning method TCA to make feature distribution between two project more similar. Moreover, they proposed TCA+, which supports selecting a suitable normalization method automatically. He *et al.* [17] analyzed the feasibility of the CPDP models trained by a simplified feature subset in three different scenarios. Ni *et al.* [42] proposed a cluster based method FeSCH. This method includes the feature clustering phase and the feature ranking phase. Yu *et al.* [61] performed an empirical study on the effectiveness of feature subset selection methods and feature ranking methods for CPDP. Other studies considered other machine learning methods (such as ensemble learning and class imbalanced learning). Panichella *et al.* [43] proposed a combined method CODEP by using ensemble learning. This method can effectively employ different classifiers. Ryu *et al.* [47], [48] designed CPDP methods by further considering the class imbalanced learning algorithms.

Since deep learning has succeeded in many areas (such as pattern recognition, natural language processing), researchers have also tried to apply deep learning to software defect prediction. Wang *et al.* [55] leveraged DBN (deep belief network) to learn semantic metrics from token vectors extracted from program modules' abstract syntax trees and then used these features to construct SDP models. Empirical results showed their method can outperform the TCA+ method [40]. Then, Li *et al.* [30] proposed CNN (convolutional neural network) based framework. In this framework, they encoded extracted token vectors as numerical vectors via word embedding and then used CNN to automatically learn semantic features. Finally they used both semantic features and traditional hand-crafted metrics to construct SDP models. Empirical results showed their method can outperform state-of-the-art SDP baselines, including DBN-based method [55].

Different from the above studies, we mainly focus on CPDP issue. We use SimAST to represent source code of extracted program modules, since this modeling method can ignore the project-specific information and is more suitable for CPDP issue. To train CPDP model, we employ BiLSTM neural network. BiLSTM is designed to effectively model the sequential data and can learn semantic features from token vectors. Moreover, since representing token vectors with more meaningful representation might further improve the prediction performance of software defect prediction, we design a new unsupervised based embedding method SimASTToken2Vec, which can automatically learn meaningful representation for token vectors. Moreover, we use attention mechanism over the BiLSTM layer to learn the weight of the vectors from the learned semantic features.

III. OUR PROPOSE METHOD DEEPCPDP

A. FRAMEWORK OF DEEPCPDP

The framework of DeepCPDP can be found in Figure 2. The first phase uses the labeled dataset in the source project for training and the granularity of the extracted modules is set to file in our empirical study. Since the majority (around 80%) of defects are contained in a small number (around 20%) of modules, most of the gathered datasets have a certain class imbalanced problem [6], [22], [35], [51]. In our study, we utilize a sampling based method (i.e., random under-sampling) to solve this problem. In the second phase, we first parse source code of extracted program modules and model it via SimAST. Then we extract token vectors from SimAST. In addition, we propose an unsupervised based embedding method SimASTToken2Vec, which is used to learn meaningful representation for token vectors by analyzing SimAST's natural structure. An embedding matrix is output by SimASTToken2Vec and we can generate vector representation of token vectors by retrieving this matrix. In the third phase, BiLSTM neural network is leveraged to learn semantic features from vectorized token vectors automatically. Moreover, we use the attention mechanism over the BiLSTM layer to learn the weight of the vectors from the learned semantic features. Finally we use Logistic regression classifier (i.e., Sigmoid layer) to train the CPDP model.

When predicting new program modules of the target project, we first use SimAST to model the source code of these modules. Then we extract token vector and perform word embedding. Finally we use the trained CPDP model to classify these modules as defective or non-defective. The details of important steps in DeepCPDP can be found in the remaining part of this section.

B. DETAILS OF DEEPCPDP

1) PERFORMING DATA PREPROCESSING

The labeled dataset is gathered by mining and analyzing the source project's software repositories (such as its version control system and bug tracking system). However, the majority (around 80%) of defects are contained in a small number (around 20%) of program modules [51] and this often results in a certain class imbalanced problem for the most of the gathered datasets. Notice in our study, we treat defective modules as the minority class and non-defective modules as the majority class. It is obvious the cost of misclassifying the

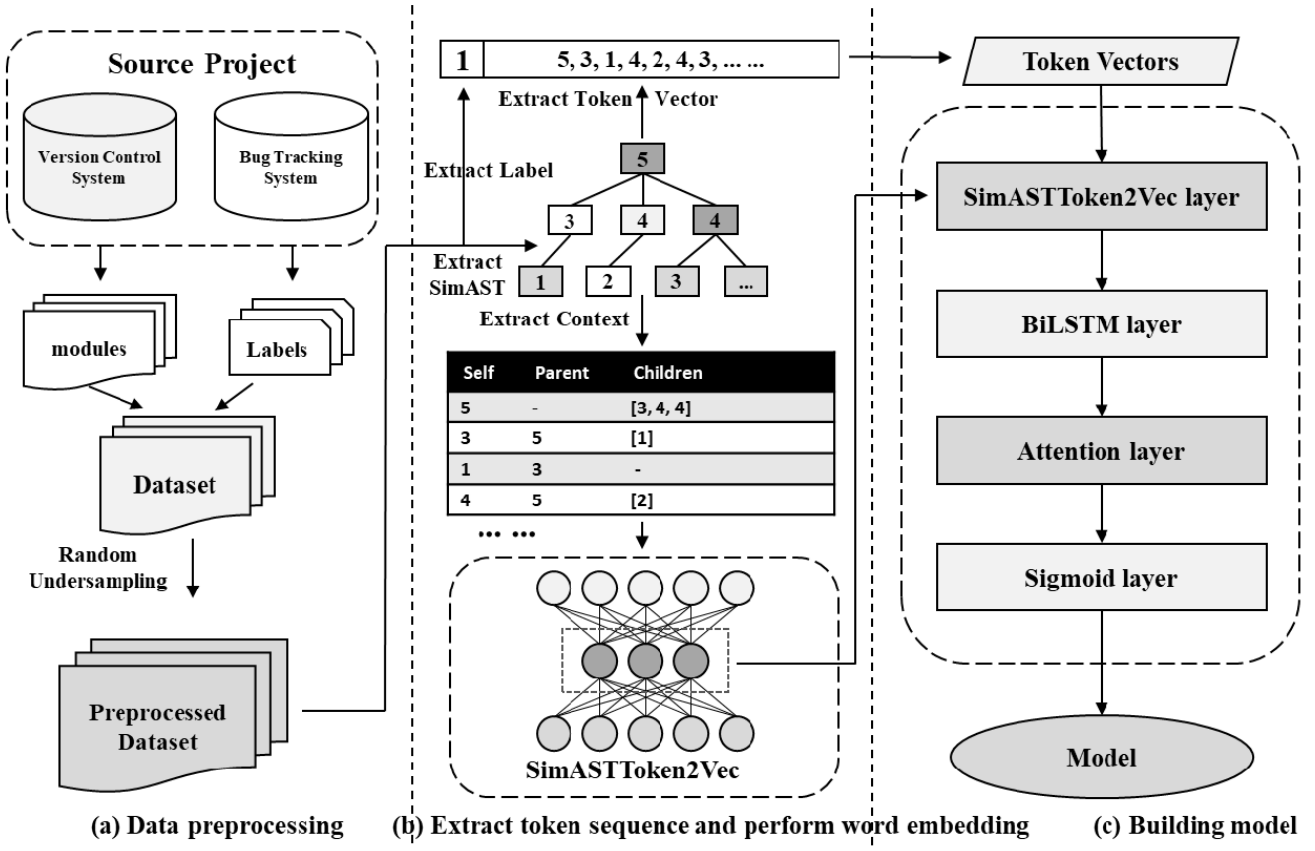


FIGURE 2. Framework of our proposed method DeepCPDP.

defective modules is higher than the cost of misclassifying the non-defective modules. When training models from these datasets with a certain class imbalanced problem, classical classifiers may not have satisfactory performance when predicting the instances in the minority class [51]. To solve this problem, we use a sampling based class imbalanced learning method (i.e., random under-sampling) to the training dataset. Random under-sampling is an instance-level strategy and this strategy has been successfully used in previous SDP studies for performing data preprocessing [14], [26]. This strategy randomly eliminates the majority class instances until the number of the majority class instances is the same as the number of the minority class instances. Assuming there are 400 non-defective modules and 100 defective modules, we will randomly remove 300 non-defective modules by using this strategy.

2) PARSING SOURCE CODE AND EXTRACTING TOKEN VECTOR

This phase parses source code of program modules and model it by using simplified abstract syntax tree (SimAST). For each node of SimAST, we only keep its node type, since this information is project-independent, while we ignore the name of method and variable,

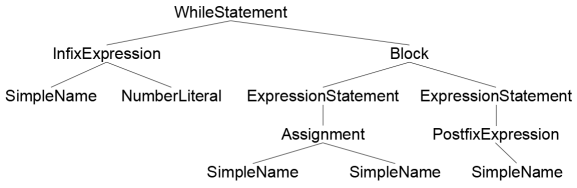


FIGURE 3. An SimAST of the simple Java code snippet.

since these information is project-specific (i.e., the name of method and variable has high diversity for different developers). Therefore, this modeling method is project-independent and especially suitable for the problem of CPDP. For a simple code snippet written by Java programming language, its SimAST after parsing the source code can be found in Figure 3.

```
while (j <= 1000)
{
    sumResult += j;
    j++;
}
```

According to the definition of Java AST, there are 92 types of AST nodes. However, based on the suggestions [30], [55], we only extract nodes, which belong to three categories,

TABLE 1. Selected AST node types and their corresponding categories.

The First Category	The Second Category	The Third Category
CONSTRUCTOR_INVOCATION	ENUM_DECLARATION	IF_STATEMENT
METHOD_INVOCATION	TYPE_DECLARATION	WHILE_STATEMENT
SUPER_CONSTRUCTOR_INVOCATION	METHOD_DECLARATION	DO_STATEMENT
SUPER_METHOD_INVOCATION		FOR_STATEMENT
CLASS_INSTANCE_CREATION		ENHANCED_FOR_STATEMENT
ARRAY_CREATION		SWITCH_STATEMENT
		BREAK_STATEMENT
		CONTINUE_STATEMENT
		RETURN_STATEMENT
		THROW_STATEMENT

as tokens. The details of selected node types (i.e., 19 node types in total) and their corresponding category can be found in Table 1. From this table, the node types in the first category are related to class instance creation and method invocation. The node types in the second category are related to type declaration, method declaration, or enum declaration. The node types in the third category are related to control flow (such as if statement, while statement, catch statement, throw statement).

Since empirical subjects considered by our study are written by Java programming language, we utilize Eclipse JDT toolkits (i.e., ASTParser and ASTVisitor) to model SimASTs by parsing their source code and extracting token vectors from program modules. Specifically, we utilize ASTParser to parse source code of each module into SimAST. Then we utilize ASTVisitor by using preorder traversal strategy to extract nodes, which only belong to node types listed in Table 1. Notice the preorder traversal strategy is used to guarantee that the extracted tokens should be in the order, which keeps the consistence with the lexical reading order of the source code. In the end of this phase, each module can be represented as a token vector.

3) PERFORMING TOKEN EMBEDDING VIA SIMASTTOKEN2VEC

Since token vectors with more meaningful representation can be helpful to improve the prediction performance of trained SDP models, we propose a new unsupervised based embedding method SimASTToken2Vec. SimASTToken2Vec is used to learn meaningful vector presentation of tokens and then further construct vector presentation for token vectors.

Since our tokens are nodes extracted from SimAST, we only learn vector presentation for nodes in SimAST. In our proposed token embedding method, we assume that nodes appearing in the similar context should have the similar semantic information. Our method is motivated by the idea of continuous bag-of-words (CBOW) model of word2vec [38], since the assumption of CBOW model is also that words appearing in the similar context should have the similar semantic information. In our propose SimASTToken2Vec method, we aim to extract the context of node from SimAST's natural structure. That is to say, for a node of modeled SimAST, we treat nodes (its parent node and its children nodes) as the context of this node. In our method, this node is named as the central node, while its parent node and its children nodes are named as the context nodes. Then we train the SimASTToken2Vec model, which can be used to predict a central node when given its context nodes. This is similar to the CBOW model. We show an illustrative example in Figure 4. When given a node's context nodes (i.e., WhileStatement, NumberLiteral and SimpleName), our trained model can predict this central node as InfixExpression.

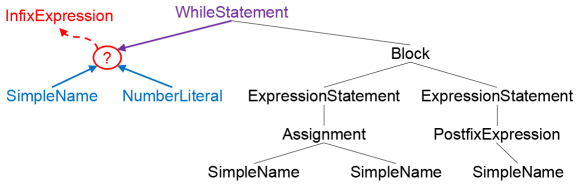


FIGURE 4. An illustrative example for predicting a central node when its context nodes is given.

Each node type in SimAST can be mapped to an unique integer i (its value range is from 1 to N). Here N is set to the sum of the total node types (i.e., 19) considered in SimAST. Then we further encode each node into a N-dimensional one-hot vector. For example, a node mapped to the integer i will be encoded into a one-hot vector. In this one-hot vector, only the value in i-th position is set to 1, while the value in the other positions of this vector is set to 0. We use $v_o(x)$ to denote the one-hot vector for the node x, where $v_o(x) \in R^N$. Then, a central node *central*, its parent node *parent* and its children nodes $\{child_1, child_2, \cdots, child_n\}$ can be denoted as $\{v_o(central), v_o(parent), v_o(child_1), v_o(child_2), \cdots, v_o(child_n)\}$. The primary objective function for the model training is:

$$v_o(central) \approx softmax\left\{\left[v_o(parent) + \sum_{k=1}^{n} v_o(child_k)\right] \cdot W_e \cdot W_h\right\} \quad (1)$$

where $W_e \in R^{N\times E}$ is the embedding matrix for learning; $W_h \in R^{E\times N}$ is the kernel of the output layer and is only used in the model training phase; E is the dimension of the vector presentation for learning.

Since the SimASTToken2Vec model can be regarded as a kind of neural network. To train SimASTToken2Vec model, we use Adam [27] for stochastic gradient descent and set cross-entropy loss to the loss function. After the model is trained, we can get the embedding matrix W_e. In this matrix, the vector in the i-th row is the vector representation of the node (i.e., token), which is mapped to the integer i. Therefore, we can find the vector representation by searching for the matrix W_e, and then generate the vector presentation for token vectors.

4) TRAINING CPDP MODEL VIA BILSTM NEURAL NETWORK AND ATTENTION MECHANISM

In this phase, we first leverage BiLSTM [49] to learn semantic features automatically from vectorized token vectors. Then we use attention mechanism over the BiLSTM layer to learn the weight of the vector from the learned semantic features. Finally, we use Logistic Regression classifier to train the CPDP model.

The BiLSTM [49] neural network is designed to effective capture long-term dependencies (i.e., a code element may depend on another code element, which are not immediately before it) from the sequential data and maintain the contextual features in both the past and the future. BiLSTM is a kind of recurrent neural network and its effectiveness has been verified in many application domains, such as text mining, machine translation, and speed recognition. Since code is also generated by developers, it shares similar properties with language texts. In our study, we want to investigate whether using BiLSTM can automatically learn semantic features from the source code and then use learned semantic features to perform CPDP. This model can transform a sequence of vectors as the input to a sequence vectors as the output. In our study, a BiLSTM layer is composed of two parallel LSTM layers, which can propagate in two different directions. Moreover, these two layers can memorize both the past information and the future information of each sequence from both the forward direction and the backward direction [49]. The forward pass and the backward passe of each layer are performed in the similar way of traditional neural networks. Finally, these two layers can memorize the sequence information from both directions.

The hidden features trained by the BiLSTM layer are fed into the attention layer. Attention mechanism can determine the importance of each node of the sequence, and weight each node when constructing the expression of the token vector. We can use the attention mechanism to enhance the impact of key nodes, and learn the advanced features through it. In this article, we adopt a simple approach utilized by both Bahdanau *et al.* [2] and Tang *et al.* [52].

Here we show the implementation details of this phase. In this phase, the structure is composed of an embedding layer, a BiLSTM layer, a max pooling layer, an attention layer, and a single unit output layer. For the embedding layer, we can encode token vectors into vector sequences and then feed these sequences into the BiLSTM layer. The embedding matrix W_e learned by SimASTToken2Vec in the previous phase is the kernel of this layer, and this embedding matrix will not be updatd during the training process of the DNN based CPDP model. Therefore, we can treat SimASTToken2Vec as a pre-training method for the embedding layer. For the BiLSTM layer, we can capture contextual features from both the past and the future in a vector and it is able to output a feature vector with the same length of its input vector, in which each feature is a contextual feature of a token. For the max pooling layer, we can employ this layer to pool an unfixed-size feature vector into a fixed-size feature vector in a multi-dimensional space. By using this layer, we can extract representative information from the original feature vector. For the attention mechanism, we use the implementations provided by Felbol *et al.* [13] to learn the weight of the vector. Finally, for the output layer, the input of this layer is the pooled contextual semantic features with the weight. This layer only has one unit and set its activation function to the sigmoid function. Therefore, this layer can be treated as a Logistic regression classifier. We choose this classifier since this classifier has also been used in previous deep neural network based software defect prediction studies [30], [55] and choosing the same classifier can guarantee a fair comparison.

In this phase, we use the L2 regularization to both the kernel of the BiLSTM layer and the output layer to alleviate the over fitting problem. Later, we set cross entropy loss to the loss function and use Adam [27] to perform stochastic gradient descent.

5) CLASSIFYING NEW MODULES IN THE TARGET PROJECT

This phase uses the trained CPDP model to classify unlabeled modules in the target project. First, source code of this module is parsed into SimAST and a token vector is extracted. Then, the token vector is encoded into a vector sequence by retrieving the embedding matrix W_e, which can be processed by the embedding layer. Later, the vector presentation of the sequence is put into the BiLSTM neural network. Finally, the trained model returns the defect-proneness of this unlabeled module. When the prediction value of the defect-proneness is smaller than a given threshold (i.e., 0.5), this new module can be predicted as non-defective, otherwise it can be predicted as defective.

IV. EMPIRICAL SETUP

A. EMPIRICAL SUBJECTS

In our study, PROMISE data sets [25] is chosen to show the effectiveness of DeepCPDP. This dataset is gathered by mining ten large-scale open source projects written by Java programming language. These project come from different application domains and have been widely used as experimental subjects in previous SDP studies [8], [9], [30], [55]. In addition, these datasets are publicly available online.[2]

[2]https://tiny.cc/seacraft

TABLE 2. The statistic information of experimental projects.

Project	Versions	# Modules	# Defective	% Defective
ant	1.7	741	166	22.4
camel	1.0, 1.2, 1.4, 1.6	2,717	562	20.7
jedit	3.2.1, 4.0, 4.1, 4.2, 4.3	1,695	303	17.9
log4j	1.0, 1.1, 1.2	417	257	61.6
lucene	2.0, 2.2, 2.4	750	437	58.3
poi	1.5, 2.0RC1, 2.5.1, 3.0	1,362	707	51.9
synapse	1.0, 1.1, 1.2	635	162	25.5
velocity	1.4, 1.5, 1.6.1	638	367	57.5
xalan	2.4, 2.5, 2.6, 2.7	3,280	1,806	55.1
xerces	init, 1.2, 1.3, 1.4.4	1,384	430	31.1

Therefore, the choice of these experimental subjects can make our comparison more reliable.

The statistic information of these experimental projects is summarized in Table 2. This table shows the name of the project, the versions of this project considered in our study, the number of program modules, the number (#) and the percentage (%) of defective modules. The hand-craft metrics used for some of baselines and the description of these metrics can be found in Table 3.

Since our proposed method DeepCPDP is based on DNN model, The effectiveness of DeepCPDP depends on the assumption that the training dataset has sufficient number of program modules. To satisfy this requirement, we first combine the modules of all the versions from the same project into a dataset. Then, we only choose the projects as the source projects if their corresponding datasets contain at least a certain number of modules after applying the random under-sampling strategy. In our study, the threshold for this number is set to 800 empirically.

Based on the above process, only four projects (i.e., camel, xalan, poi and xerces) can be set to the source projects in our study. For example, the project Camel has four versions (i.e., 1.0, 1.2, 1.4 and 1.6). After combining modules in these four versions, the final combined dataset contains 2,712 program modules in total, and 562 modules are defective modules (i.e., the percentage of defective modules is 20.7%). After applying the random under-sampling strategy, the preprocessed dataset still contains more than 800 modules. Therefore, the project Camel can be set to the source project in our study.

In summary, four projects can be set to the source projects, while all the ten projects can be set to the target projects. Since the same project can not be set to the source project and the target project simultaneously in the context of CPDP, there are 36 CPDP pairs in total.

B. EVALUATION MEASURE

Existing work have found that threshold-dependent performance measures (for example, accuracy, precision, recall or F1) are problematic. The reasons can be summarized as follows: First the computation of these measures depends on a selected threshold. The value of this threshold is set to 0.5 in most cases [10], [29], [45]. Second, these evaluation measures are sensitive to the datasets with class imbalanced issue [62].

Recently, researchers tend to use the AUC (Area Under the ROC Curve) to measure performance of the trained SDP models. This evaluation measure is also recommended in a recent benchmark study [29]. The AUC measure is a threshold-independent evaluation measure and it can be used to evaluate a SDP model's ability to discriminate the non-defective modules from the defective modules. AUC can be computed by computing the area under the curve. This curve can plots the false positive rate(fpr) against the true positive rate (tpr). The value range of AUC measure is [0,1]. Here the value 0 denotes the worst performance of the traind model, while the value of 1 denotes the best performance. It is obvious that the higher value of the AUC evaluation measure, the better the prediction performance of the trained SDP model.

C. STATISTICAL ANALYSIS METHODS

Statistical test can be used to analyze whether there exists a statistically significant difference between results of two different methods. Wilcoxon signed-rank test [57] is first used to perform statistical test. This statistical test method does not require the analyzed data should follow any distribution. Moreover, it can be used to compare pairs of results and is able to compare the difference against zero.

Then Cliff's delta [4] is used to measure the effect size between two different methods. Cliff's delta is a non-parametric effect size measure and can be used to quantify the amount of difference between two methods. Table 4 describes the value range of Cliff's delta values and its corresponding effectiveness level.

Finally we use Scott-Knott test [23] to rank our proposed method DeepCPDP and all the baselines in terms of AUC measure. Scott-Knott test performs the method grouping process in a recursive way. Firstly, s a hierarchical cluster analysis method is used to partition all the CPDP methods into two ranks by considering the mean performance (measured by AUC). Then, it is recursively conducted for each rank to further divide the ranks if the divided ranks are significantly different. The Scott-Knott test will terminate when the ranking can no longer be divided into statistically different rankings. For Scott-Knott test, a method $m1$ performs significantly better (or worse) than another method $m2$, if p value of Wilcoxon signed-rank test when the confidence level is set to 95% is less than 0.05 and the effectiveness level is not negligible (measured by Cliff's delta). While the difference between $m1$ and $m2$ is not significant, if (1) p-value is less than 0.05 and the effectiveness level is negligible, or (2) p value is not less than 0.05.

D. EXPERIMENTAL SETTINGS

1) SETTINGS FOR EMBEDDING METHOD SIMASTTOKEN2VEC

Based on Figure 2, the embedding matrix can be constructed by using the dataset gathered from a source project via SimASTToken2Vec. However, SimASTToken2Vec learns

TABLE 3. Name and description of the hand-crafted metrics used for measuring program modules in some baselines.

ID	Metric Name	Metric Description
1	WMC	Weighted methods in the given class
2	DIT	Depth of inheritance tree in the given class
3	NOC	The number of children of the given class
4	CBO	Coupling (i.e., efferent coupling and afferent coupling) between object classes
5	RFC	Response (i.e., the number of methods that can be executed) for a class
6	LCOM	Lack of cohesion in methods and measure by the method suggested by Henderson-Sellers [19]
7	CA	Afferent couplings (i.e., the number of classes that depend on the measured class)
8	CE	Efferent couplings (i.e., the number of classes that the measured class is depended on)
9	NPM	Number of public methods in a class
10	LCOM3	Lack of cohesion in methods and measured by the method suggested by Bansiy and Davis [3]
11	LOC	Lines of code in a class
12	DAM	Data access metric (i.e., the ratio of the number of private/protected attributes to the total number of attributes in the class)
13	MOA	Measure of aggregation (i.e., the number of class fields whose types are user-defined classes)
14	MFA	Measure of functional abstraction (i.e., the ratio of the number of methods inherited by a class to the number of methods, which are accessible by the member methods of this class.)
15	CAM	Cohesion among methods of class (i.e., the relatedness among methods of a class based on the parameter list of these methods)
16	IC	Inheritance coupling (i.e., the number of parent classes coupled by a given class)
17	CBM	Coupling between methods (i.e., the total number of new/redefined methods coupled by all the inherited methods.)
18	AMC	Average method complexity (i.e., the average method size for each class)
19	MAX_CC	Max value of CC (McCabe's cyclomatic complexity) among methods of the investigated class
20	AVG_CC	Average value of CC (McCabe's cyclomatic complexity) among methods of the investigated class

TABLE 4. The value range of Cliff's delta δ and its corresponding effectiveness level [4].

The value range of Cliff's delta	Effectiveness Level
$0.474 \leq \|\delta\|$	Large
$0.33 \leq \|\delta\| < 0.474$	Medium
$0.147 \leq \|\delta\| < 0.33$	Small
$\|\delta\| < 0.147$	Negligible

tokens' semantic information from the SimAST's natural structure. Moreover, both structure and tokens in SimAST are project-independent. Therefore, we do not need to train embedding matrix for every source project individually. Instead, we combine the modules from all the project into a single dataset and then train only one embedding matrix.

Based on the description of SimASTToken2Vec in Section III, we can find SimASTToken2Vec has two parameters: N and E. In our study, N is set to 19, since the sum of node types used in SimAST is 19. E is empirically set to 20. The batch size for training the embedding matrix is set to 1024. The epoch for training is set to 1000. Early-stopping criteria is set as follows: when the loss on the training dataset does not decrease any more, the training process will terminate in advance.

2) SETTINGS FOR BILSTM NEURAL NETWORK AND ATTENTION MECHANISM

For the embedding layer, we set the input dimension of this layer to 19, since this matches the dimension of the embedding matrix W_e learned by SimASTToken2Vec. Then we set the output dimension of this layer to 20. For the BiLSTM layer, we first set the dimension of this layer to 512, Then we set the output of the BiLSTM layer as the average value of the output of the both forward and backward LSTM layers. This setting can be implemented by Keras (i.e.,the merge mode of Bi-directional layer can be set to ave). The batch size for training is set to 64. To optimize the value of hyper-parameters in DeepCPDP, we further split the original training dataset by 80%:20% for training and validation respectively. The training epoch is set to 100, and early-stopping technique is utilized to stop training process automatically in advance when the performance on the validation set of the trained model can not be improved after 10 epochs. For each epoch, the DNN model along with the value of its hyper-parameters (i.e., kernels and bias) will be stored. Finally, the trained model, which can achieve the best performance on the validation set in terms of AUC measure will be output as our final trained CPDP model. Notice for other hyper-parameters (e.g., the activation function in the BiLSTM layer), we use the default value considered by Keras.

3) OTHER SETTINGS

Our proposed method DeepCPDP is implemented based on Keras[3] framework with Tensorflow backend. Our proposed method and baselines are run on a server: 2 Intel Xeon CPU E5-2640 @2.60GHz, 2 GPU (NVIDIA Titan X) with 12 GB memory, 32 GB RAM and Ubuntu 14.04.5 OS.

Since there exist the random factors when using random under-sampling to solve class imbalanced problem and training CPDP model, we run our experiments 10 times independently for each CPDP pair with different random seeds.

[3]https://keras.io/

TABLE 5. The cluster result after using *k*-means clustering.

Cluster No	Node Types
1	NullLiteral, BooleanLiteral, NumberLiteral, CharacterLiteral, TypeLiteral, StringLiteral
2	BlockComment, LineComment
3	TypeDeclaration, MethodDeclaration, FieldDeclaration
4	IfStatement, DoStatement, WhileStatement, ForStatement
5	BreakStatement, ThrowStatement, ContinueStatement

Finally, we show the average of these 10 values in terms of AUC measure.

V. RESULT ANALYSIS

A. RESULT ANALYSIS FOR RQ1

RQ1: Can our proposed new embedding method SimASTToken2Vec learn meaningful vector presentation of tokens?

Motivation: In RQ1, we first want to analyze whether our proposed embedding method SimASTToken2Vec can learn meaningful vector presentation for tokens. Then we want to investigate whether the learned vector presentation of tokens is helpful to improve the prediction performance of our proposed method DeepCPDP.

Approach: We evaluate vector representations learned by our proposed embedding method SimASTToken2Vec from qualitative perspective and quantitative perspective respectively.

Results: We first analyze the effectiveness of the embedding method SimASTToken2Vec from the qualitative perspective. Since similar tokens should share similar semantic information and they are assumed to have similar vector representations. To evaluate whether SimASTToken2Vec can achieve this goal, we select a subset of node types in SimAST. Then, we perform *k*-means clustering on their vector representations learned by SimASTToken2Vec and the value of *k* is set to 5. The clustering result can be found in Table 5. In this table, we can find: (1) cluster 1 includes node types mainly related to literal, (2) cluster 2 includes node types mainly related to loop, (3) cluster 3 includes node types mainly related to jump, (4) cluster 4 includes node types mainly related to declaration, (5) cluster 5 includes node types mainly related to comment. It is not hard to find that the cluster results after using *k*-means clustering is reasonable since it keeps the consistent with the domain knowledge. These results also suggest that our proposed method SimASTToken2Vec has achieved the goal. Therefore, the vector presentation learned by SimASTToken2Vec is meaningful.

Then we analyze the effectiveness of the embedding method SimASTToken2Vec from the quantitative perspective. We first set SimASTToken2Vec to our default embedding method in our proposed method DeepCPDP. Then we set a random embedding method in DeepCPDP as the baseline method for RQ 1. The random embedding method wa employed for CNN based SDP in the previous study [30]. For this baseline, the embedding matrix is randomly initialized in the embedding layer and then is tuned during the process of the CPDP model training. Figure 5 uses boxplot to show the performance comparison results (measured in terms of AUC) when DeepCPDP use two different embedding methods on all CPDP pairs. The detailed comparions results can be found in Table 6. On average, DeepCPDP using the embedding method SimASTToken2Vec can outperform DeepCPDP using the random embedding method by 3.62%. After conducting Wilcoxon signed-rank test, the computed *p*-value is 1.28E-3. This shows that the difference between these two embedding methods is statistically significant. After measuring the effect size, the computed value of Cliff's delta is 0.235. This shows that the effectiveness level between these two embedding methods is not negligible. Based on the above results, we can conclude that DeepCPDP using SimASTToken2Vec can significantly outperform DeepCPDP using the random embedding method. Therefore, the embedding method SimASTToken2Vec is helpful to improve the performance of our proposed method DeepCPDP.

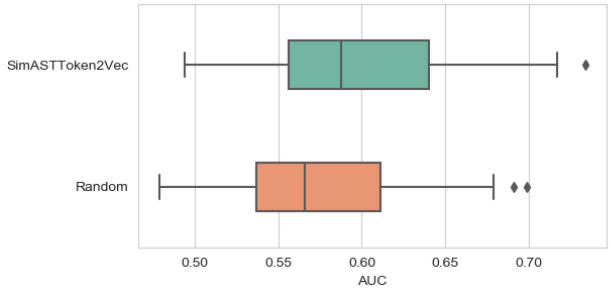


FIGURE 5. The performance value of DeepCPDP when using two different embedding methods in terms of AUC.

Answer to RQ1: our proposed embedding method SimASTToken2Vec can learn meaningful vector presentation of tokens both from qualitative perspective and quantitative perspective.

B. RESULT ANALYSIS FOR RQ2

RQ2: Can BiLSTM neural network achieve better performance than other DNN models in our proposed method DeepCPDP?

Motivation: Recently, Li *et al.* [30] applied CNN model, which has advantages in capturing local patterns more effectively, to software defect prediction and found CNN model can achieve better prediction performance than DBN model based on the analysis of their empirical results [55]. Therefore, in RQ2, we want to analyze the influence of different DNN models in our proposed method DeepCPDP and verify whether BiLSTM neural network can achieve the best performance.

Approach: In our proposed method DeepCPDP, we set BiLSTM neural network as our default DNN model. In RQ2, we want to compare the prediction performance of DeepCPDP method using BiLSTM neural network with DeepCPDP method using CNN model (i.e., the baseline in RQ2). For DeepCPDP using CNN model, we use 1D convolution

TABLE 6. The performance value of DeepCPDP when using the embedding method SimASTToken2Vec or the random embedding method.

Source Project	Target Project	SimASTToken2Vec	Random
camel	ant	0.717	0.691
camel	jedit	0.622	0.594
camel	log4j	0.566	0.536
camel	lucene	0.640	0.616
camel	poi	0.589	0.529
camel	synapse	0.633	0.611
camel	velocity	0.549	0.524
camel	xalan	0.555	0.551
camel	xerces	0.558	0.537
poi	ant	0.706	0.679
poi	camel	0.561	0.567
poi	jedit	0.671	0.668
poi	log4j	0.551	0.479
poi	lucene	0.645	0.643
poi	synapse	0.625	0.591
poi	velocity	0.552	0.561
poi	xalan	0.557	0.555
poi	xerces	0.581	0.589
xalan	ant	0.734	0.699
xalan	camel	0.569	0.565
xalan	jedit	0.653	0.660
xalan	log4j	0.498	0.503
xalan	lucene	0.624	0.64
xalan	poi	0.586	0.613
xalan	synapse	0.634	0.605
xalan	velocity	0.550	0.549
xalan	xerces	0.549	0.575
xerces	ant	0.678	0.604
xerces	camel	0.563	0.541
xerces	jedit	0.666	0.556
xerces	log4j	0.494	0.527
xerces	lucene	0.641	0.547
xerces	poi	0.619	0.531
xerces	synapse	0.600	0.592
xerces	velocity	0.551	0.518
xerces	xalan	0.559	0.527
Average		0.601	0.580
p-value		1.28E-3	-
Cliff's delta		0.235	-

layer to extract semantic features from token vectors instead of using a BiLSTM layer. This setting is also employed in the previous study [30]. For hyper-parameters used for the convolution layer, the dimension size and regularization method is consistent with that of our BiLSTM layer, and value of other hyper-parameters is consistent with the previous study [30] for a fair comparison.

Results: Figure 6 uses boxplot to show the performance comparison results (measured in terms of AUC) when DeepCPDP use two different DNN models on all CPDP pairs. The comparison results in detail can be found in Table 7. On average, DeepCPDP using BiLSTM neural network can outperform DeepCPDP using CNN model by 2.74%. After conducting Wilcoxon signed-rank test, the computed p-value is 1.07E-4. This shows that the difference between BiLSTM model and CNN model is statistically significant. After measuring the effect size, the computed value of Cliff's delta is 0.174. This shows that the effectiveness level between BiLSTM model and CNN model is not negligible.

TABLE 7. The performance value of DeepCPDP when using BiLSTM neural network or using CNN model.

Source Project	Target Project	BiLSTM	CNN
camel	ant	0.717	0.672
camel	jedit	0.622	0.594
camel	log4j	0.566	0.531
camel	lucene	0.640	0.62
camel	poi	0.589	0.566
camel	synapse	0.633	0.601
camel	velocity	0.549	0.537
camel	xalan	0.555	0.546
camel	xerces	0.558	0.552
poi	ant	0.706	0.691
poi	camel	0.561	0.557
poi	jedit	0.671	0.664
poi	log4j	0.551	0.500
poi	lucene	0.645	0.633
poi	synapse	0.625	0.611
poi	velocity	0.552	0.541
poi	xalan	0.557	0.559
poi	xerces	0.581	0.580
xalan	ant	0.734	0.703
xalan	camel	0.569	0.562
xalan	jedit	0.653	0.655
xalan	log4j	0.498	0.495
xalan	lucene	0.624	0.635
xalan	poi	0.586	0.609
xalan	synapse	0.634	0.618
xalan	velocity	0.550	0.549
xalan	xerces	0.549	0.577
xerces	ant	0.678	0.592
xerces	camel	0.563	0.547
xerces	jedit	0.666	0.613
xerces	log4j	0.494	0.503
xerces	lucene	0.641	0.596
xerces	poi	0.619	0.601
xerces	synapse	0.600	0.568
xerces	velocity	0.551	0.551
xerces	xalan	0.559	0.541
Average		0.601	0.585
p-value		1.07E-4	-
Cliff's delta		0.174	-

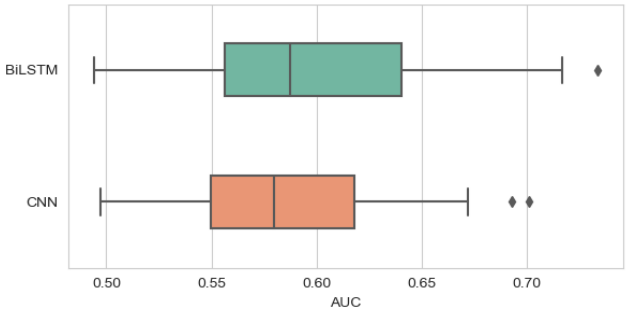


FIGURE 6. The performance value of DeepCPDP when using different DNN models in terms of AUC.

Answer to RQ2: DeepCPDP using BiLSTM neural network can significantly outperform DeepCPDP using CNN model.

C. RESULT ANALYSIS FOR RQ3

RQ3: Can using attention mechanism improve the performance of our proposed method DeepCPDP?

Motivation: Traditional BiLSTM does not consider the attention mechanism. Therefore, in this RQ, we want to

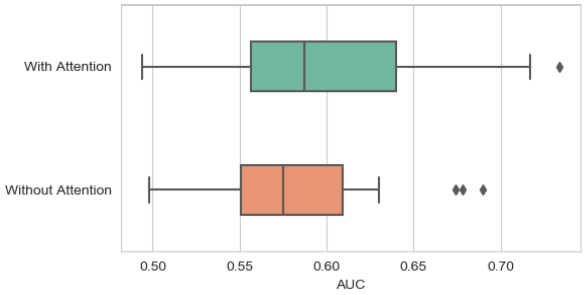


FIGURE 7. The performance value of DeepCPDP when using or not using the attention mechanism in terms of AUC.

investigate whether using attention mechanism can further improve the performance of our proposed method DeepCPDP.

Approach: We use attention mechanism over the BiLSTM layer to learn the weight of the vectors from the learned semantic features. Attention mechanism can determine the importance of each node of the vector, and weight each node when constructing the expression of the token vector. We can use the attention mechanism to enhance the impact of key nodes, and learn the advanced features through it. In this article, we use a simple approach utilized by Bahdanau *et al.* [2] and Tang *et al.* [52].

Results: Figure 7 uses boxplot to show the performance comparison results (measured in terms of AUC) between DeepCPDP with the attention mechanism and DeepCPDP without the attention mechanism on all CPDP pairs. The comparison results in detail can be found in Table 8. On average, DeepCPDP with the attention mechanism can outperform DeepCPDP without attention mechanism by 3.09%. After conducting Wilcoxon signed-rank test, the computed p-value is 3.47E-3. This shows that the difference between DeepCPDP with the attention mechanism and DeepCPDP without the attention mechanism is statistically significant. After measuring the effectiveness, the computed value of Cliff's delta is 0.174. This shows that the effectiveness level between DeepCPDP with the attention mechanism and DeepCPDP without the attention mechanism is not negligible.

Answer to RQ3:, DeepCPDP using the attention mechanism can achieve better performance with statistically significance than DeepCPDP without using the attention mechanism.

D. RESULT ANALYSIS FOR RQ4

RQ4: Can our proposed method DeepCPDP perform significantly better than state-of-the-art CPDP baseline method?

Motivation: In the previous three RQs, we thoroughly investigate the influence of embedding methods, DNN models, and whether using attention mechanism for our proposed method DeepCPDP. In RQ4, we want to investigate whether our proposed method DeepCPDP can advance the state of the art of CPDP domain by comparing CPDP baselines.

TABLE 8. The performance value of DeepCPDP when using the attention mechanism or not using the attention mechanism.

Source Project	Target Project	With Attention	Without Attention
camel	ant	0.717	0.624
camel	jedit	0.622	0.573
camel	log4j	0.566	0.592
camel	lucene	0.640	0.605
camel	poi	0.589	0.574
camel	synapse	0.633	0.596
camel	velocity	0.549	0.568
camel	xalan	0.555	0.541
camel	xerces	0.558	0.498
poi	ant	0.706	0.690
poi	camel	0.561	0.542
poi	jedit	0.671	0.678
poi	log4j	0.551	0.582
poi	lucene	0.645	0.613
poi	synapse	0.625	0.630
poi	velocity	0.552	0.576
poi	xalan	0.557	0.563
poi	xerces	0.581	0.560
xalan	ant	0.734	0.674
xalan	camel	0.569	0.544
xalan	jedit	0.653	0.620
xalan	log4j	0.498	0.546
xalan	lucene	0.624	0.603
xalan	poi	0.586	0.560
xalan	synapse	0.634	0.626
xalan	velocity	0.550	0.560
xalan	xerces	0.549	0.552
xerces	ant	0.678	0.591
xerces	camel	0.563	0.545
xerces	jedit	0.666	0.589
xerces	log4j	0.494	0.529
xerces	lucene	0.641	0.616
xerces	poi	0.619	0.608
xerces	synapse	0.600	0.574
xerces	velocity	0.551	0.535
xerces	xalan	0.559	0.523
Average		0.601	0.583
p-value		3.47E-3	-
Cliff's delta		0.174	-

Approach: In RQ4, we choose state-of-the-art CPDP baselines as follows. First, we consider a CNN model based SDP method proposed by Li *et al.* [30] as the first baseline. In their empirical studies, they found their proposed method has better performance than the DNN model based SDP method [55]. Then we consider other seven CPDP baseline method, which are based on hand-crafted metrics (the description of these metrics can be found in Table 3). These baseline methods are CamargoCruz09-DT [5], Turhan09-DT [54], Menzies11-RF [37], Watanabe08-DT [56], Ma12-DT [36], Panichella14-LR [43] and Amaski15-DT [1]. Since in a recent comparative study for comparing 24 different CPDP methods [20], these seven CPDP methods can achieve better performance when compared to other CPDP methods. The effectiveness of each CPDP method has been evaluated by considering six different classifiers and the suffix of method name represents the classifier with the best prediction performance in previous comparative study conducted by Herbold *et al.* [20] and Chen *et al.* [7]. In particular, the suffix DT denotes that the CPDP baseline uses decision tree as the classifier, the suffix RF denotes the CPDP baseline uses random forest as the

classifier, and the suffix LR denotes the CPDP baseline uses Logistic regression as the classifier.

Baseline1 (Li17-CNN): Li *et al.* [30] proposed CNN (convolutional neural network) based framework. In this framework, they encoded extracted token vectors as numerical vectors via word embedding and then used CNN to automatically learn semantic features. Finally they used both semantic features and traditional hand-crafted metrics to construct SDP models. We use the same network structure and network parameter settings considered in the previous study [30] to implement this baseline method. Notice for a fair comparison, we use the same token vectors based on SimAST and the same training process as ours for this baseline. Finally, instead of using a fixed number of epochs (i.e., 15) for different source projects [30], we use early-stopping technique to automatically tune the number of epochs for better performance.

Baseline2 (CamargoCruz09-DT): Camargo and Ochimizu [5] applied power transformation to the value of the metric. This transformation method can improve symmetry and normality of the metric value. Moreover, this method can also effectively reduce the number of outliers.

Baseline3 (Turhan09-DT): Turhan *et al.* [54] proposed this method. This method first normalizes the metric data via the logarithm transformation. Then it applied a relevancy filter to the available training data of the source project by using k-nearest neighbor algorithm. After using this relevancy filter, the k nearest modules in the source project can be selected when given an module in the target project. The similarity of different modules is computed by using the Euclidean distance based on the metric value of these modules. Notice a module in the source project may be a nearest neighbours for different modules in the target project, they only choose this module once and store it in the filtered dataset. Moreover, the value of the parameter k is set to 10 in the previous study [54] and we follow this setting when implementing this baseline.

Baseline4 (Menzies11-RF): Menzies *et al.* [37] used the WHERE algorithm to construct local models by clustering the training dataset. Then they used WHICH rule learning algorithm to the classification of the results. Notice WHICH rules were generated for each cluster and then used for constructing local models.

Baseline5 (Watanabe08-DT): Watanabe *et al.* [56] proposed to alleviate the data distribution difference between projects via a standardization normalization formula. As a formula for standardization, they proposed to multiply each metric value of the target project with the mean value of the source project and divide this by the mean of the target project.

Baseline6 (Ma12-DT): Ma *et al.* [36] proposed Transfer Naive Bayes (TNB) method. Different from instance selection methods [37], this method assigns weights to the modules in the source project according to the similarity between the source project and the target project. Then it utilizes Naive Bayes classifier to construct CPDP models on these weighted program modules.

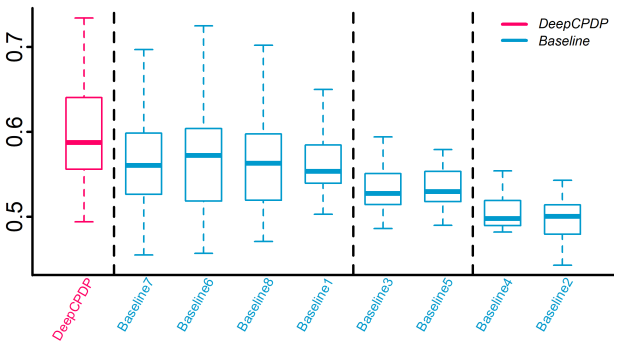


FIGURE 8. Comparison results between the method DeepCPDP and baseline methods in terms of AUC by using Scott-Knott test.

Baseline7 (Panichella14-LR): Panichella *et al.* [43] proposed a COmbined DEfect Predictor (CODEP) method. They investigated six different classifiers (belonging to four categories: regression function, neural network, decision tree and rule based model) in CPDP. Empirical results show these different classifiers could identify different defective program modules by using principal component analysis and overlap metrics, though they achieved similar performance. Then they proposed a combined model based on these six classifiers via Logisitic regression.

Baseline8 (Amaski15-DT): Amasaki *et al.* [1] investigated the data simplification by removing redundant information. In particular, they first identified redundant features from the dataset in the target project by using an unsupervised feature reduction method. Then, they removed the same features from the cross-project dataset.

For the last seven baselines, the setting of these methods is consistent with the settings that perform best in the recently comparative study on different CPDP methods [20]. Notice we follow the same process used by our proposed method DeepCPDP to evaluate these eight baselines for a fair comparison. For example, these methods all use random undersampling to handle the class imbalanced problem.

Results: Table 9 shows the comparison results (measured in terms of AUC) between our proposed method DeepCPDP and eight state-of-the-art baseline methods. Comparing with Li17-CNN, CamargoCruz09-DT, Turhan09-DT, Menzies11-RF, Watanabe08-DT, Ma12-DT, Panichella14-LR and Amaski15-DT, our method outperforms them by 6.18%, 21.17%, 12.13%, 18.30%, 12.34%, 5.07%, 4.52%, and 5.62% respectively on average. After conducting Wilcoxon signed-rank test, the computed p-value is 3.41E-05, 2.16E-07, 4.83E-06, 1.83E-07, 3.88E-07, 2.78E-02, 2.42E-02, and 1.88E-02 respectively. These results show that the difference between our proposed method DeepCPDP and these eight baselines is statistically significant. In addition, after measuring the effect size, the computed value of Cliff's delta is 0.402, 0.940, 0.704, 0.870, 0.718, 0.295, 0.279 and 0.31 respectively. These results show that the effectiveness level between our proposed method DeepCPDP and these baseline methods is not negligible. Figure 8 uses

TABLE 9. The performance value of our proposed method DeepCPDP and baseline methods in terms of AUC.

Source project	Target project	DeepCPDP	Baseline1	Baseline2	Baseline3	Baseline4	Baseline5	Baseline6	Baseline7	Baseline8
camel	ant	0.717	0.615	0.475	0.545	0.521	0.559	0.606	0.663	0.596
camel	jedit	0.622	0.601	0.443	0.505	0.530	0.532	0.708	0.697	0.702
camel	log4j	0.566	0.542	0.493	0.554	0.494	0.549	0.725	0.723	0.776
camel	lucene	0.640	0.550	0.480	0.502	0.487	0.518	0.588	0.528	0.551
camel	poi	0.589	0.576	0.455	0.539	0.483	0.526	0.584	0.562	0.588
camel	synapse	0.633	0.582	0.481	0.523	0.504	0.535	0.581	0.576	0.568
camel	velocity	0.549	0.531	0.504	0.516	0.491	0.509	0.522	0.455	0.524
camel	xalan	0.555	0.543	0.467	0.500	0.513	0.512	0.593	0.653	0.566
camel	xerces	0.558	0.518	0.497	0.513	0.494	0.513	0.719	0.715	0.680
poi	ant	0.706	0.680	0.524	0.594	0.543	0.622	0.656	0.681	0.694
poi	camel	0.561	0.554	0.512	0.589	0.498	0.545	0.539	0.526	0.513
poi	jedit	0.671	0.650	0.514	0.582	0.565	0.569	0.604	0.571	0.561
poi	log4j	0.551	0.575	0.543	0.607	0.486	0.542	0.590	0.587	0.599
poi	lucene	0.645	0.587	0.510	0.571	0.489	0.558	0.550	0.507	0.535
poi	synapse	0.625	0.621	0.489	0.542	0.517	0.574	0.532	0.545	0.517
poi	velocity	0.552	0.558	0.514	0.524	0.506	0.490	0.519	0.568	0.530
poi	xalan	0.557	0.571	0.511	0.533	0.522	0.533	0.606	0.577	0.604
poi	xerces	0.581	0.531	0.522	0.520	0.482	0.493	0.602	0.597	0.614
xalan	ant	0.734	0.667	0.467	0.588	0.565	0.573	0.510	0.525	0.518
xalan	camel	0.569	0.555	0.499	0.515	0.493	0.531	0.457	0.538	0.557
xalan	jedit	0.653	0.615	0.455	0.550	0.576	0.579	0.631	0.620	0.610
xalan	log4j	0.498	0.540	0.492	0.561	0.488	0.518	0.539	0.554	0.471
xalan	lucene	0.624	0.553	0.485	0.533	0.490	0.528	0.542	0.600	0.594
xalan	poi	0.586	0.560	0.479	0.552	0.482	0.559	0.639	0.595	0.616
xalan	synapse	0.634	0.619	0.448	0.549	0.554	0.568	0.604	0.632	0.575
xalan	velocity	0.550	0.550	0.502	0.516	0.498	0.507	0.517	0.512	0.499
xalan	xerces	0.549	0.539	0.479	0.528	0.488	0.522	0.578	0.518	0.561
xerces	ant	0.678	0.553	0.514	0.510	0.512	0.522	0.602	0.523	0.591
xerces	camel	0.563	0.503	0.508	0.510	0.488	0.510	0.533	0.537	0.565
xerces	jedit	0.666	0.551	0.506	0.514	0.524	0.529	0.511	0.551	0.503
xerces	log4j	0.494	0.522	0.513	0.509	0.498	0.519	0.518	0.517	0.526
xerces	lucene	0.641	0.526	0.514	0.520	0.493	0.530	0.494	0.565	0.484
xerces	poi	0.619	0.548	0.540	0.527	0.495	0.540	0.507	0.521	0.518
xerces	synapse	0.600	0.565	0.527	0.540	0.501	0.521	0.510	0.527	0.492
xerces	velocity	0.551	0.520	0.521	0.486	0.498	0.493	0.508	0.559	0.521
xerces	xalan	0.559	0.509	0.489	0.526	0.517	0.521	0.566	0.558	0.572
Average		0.601	0.566	0.496	0.536	0.508	0.535	0.572	0.575	0.569
p-value		-	3.41E-05	2.16E-07	4.83E-06	1.83E-07	3.88E-07	2.78E-02	2.42E-02	1.88E-02
Cliff's delta		- -	0.402	0.94	0.704	0.87	0.718	0.295	0.279	0.31

Scott-Knott test to show the performance comparison results when comparing our proposed method DeepCPDP with baselines in terms of AUC. From this figure, we can find DeepCPDP is in the top 1 group. Baseline7, Baseline6, Baseline8 and Baseline1 are in the top 2 group. Baseline3 and Baseline5 are in the top 3 group. While Baseline4 and Baseline 2 are in the final group.

Answer to RQ4:, our proposed method DeepCPDP can perform significantly better than eight state-of-the-art CPDP baseline methods.

VI. DISCUSSION

A. COMPARISON IN TERMS OF TWO EFFORT-AWARE EVALUATION MEASURES

In previous empirical studies, we mainly compare our proposed method DeepCPDP in terms of traditional non-effort-aware evaluation measure (i.e., AUC). In this subsection, we further compare DeepCPDP in terms of two effort-aware evaluation measures (i.e., *ACC* and P_{opt}). These two measures are more practical for practitioners, since it enable them for detecting more defective modules when given limited SQA efforts. In particular, *ACC* shows the percentage of defective modules identified by developers when expending 20% of the SQA efforts.[4] P_{opt} is the normalized version of the effort-aware performance indicator. The details of these two effort-ware evaluation measures can be found in recent effort-aware software defect prediction studies [11], [60]. Figure 9 and Figure 10 show the performance comparison results between our proposed method DeepCPDP and eight baseline methods in terms of *ACC* and P_{opt} by using Scott-Knott test respectively. Final results show DeepCPDP can also in the top 1 group when considering these two evaluation measures.

B. COMPUTATIONAL COST ANALYSIS

In this subsection, we mainly compare our proposed DeepCPDP in terms of model construction time. The model construction time for DeepCPDP and baselines can be found in Table 10 and we report the average construction time when given the source project. From Table 10, we can find the model construction time of our proposed method is acceptable (i.e., training a model needs about 66 seconds

[4]Notice we use LOC (lines of code) to denote the efforts in our study, which is consistent with previous studies [11], [60].

TABLE 10. Comparison results with baselines for our proposed method DeepCPDP in terms of Computational Cost (Unit: Millisecond).

Source Project	DeepCPDP	Baseline1	Baseline2	Baseline3	Baseline4	Baseline5	Baseline6	Baseline7	Baseline8
camel	66852	14813	55	87	495	52	62	3361	54
poi	155473	17524	27	72	567	29	35	3874	30
xalan	355351	24237	98	121	1526	100	100	8728	104
xerces	122593	14351	17	47	379	18	20	2564	18

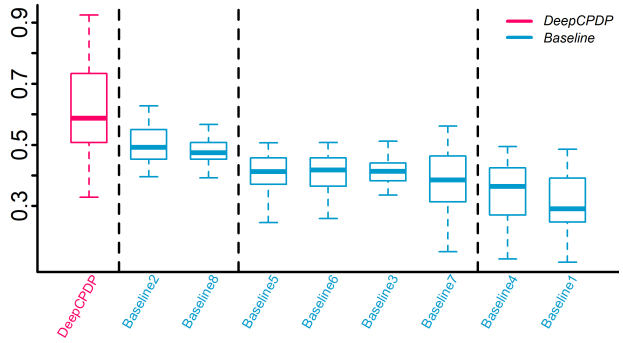


FIGURE 9. Comparison results with baselines for our proposed method DeepCPDP in terms of P_{opt} by using Scott-Knott test.

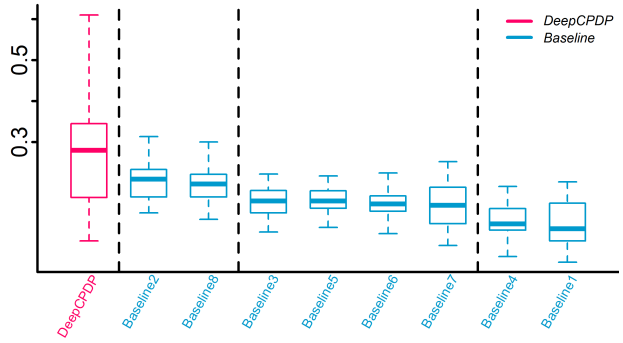


FIGURE 10. Comparison results with baselines for our proposed method DeepCPDP in terms of *ACC* by using Scott-Knott test.

to 355 seconds on average). Moreover, based on the results in terms of both the traditional evaluation measure and the effort-aware evaluation measures, our proposed DeepCPDP can achieve better performance than all the baselines with statistically significance. Finally, with the increasing labeled modules after gathering more datasets, the advantage of our proposed deep learning based method will be more obvious. Therefore, DeepCPDP is applicable in practical cross-project software defect prediction.

VII. THREATS TO VALIDITY

A. THREATS TO INTERNAL VALIDITY

The first internal threat is the faults during the implementations of these CPDP methods. To allevate this threat, we consider the implementations of supervised CPDP methods in CrossPare platform [20]. The second internal threat is the value of hyperparameter in the classifier used by our proposed method and baselines. In our study, we use the default value for these hyperparameters. In the future, we will investigate the influence of hyperparameter optimization [53] in our proposed method.

B. THREATS TO EXTERNAL VALIDITY

The first threat is that in our empirical studies, we only choose promise dataset. On the one side, the promise dataset has been widely used in previous deep learning based defect prediction studies [30], [55]. On the other side, our proposed method DeepCPDP is based on the analysis of the source code and we can only access the source code in the projects gathered by the promise dataset. The second threat is the choice of the baselines when answering for RQ4. In this article, we mainly focus on supervised homogeneous CPDP methods. These baselines are chosen since they either achieve competitive performance in recent benchmark studies [7], [20], or are often set to baselines in previous supervised CPDP studies. Moreover, based on the related work analysis for CPDP methods, we can find many recently proposed CPDP methods can not be chosen in our empirical studies. For example, some recent studies focused on heterogeneous defect prediction [24], [32], [39], which assume that the source project and the target project do not use the same metrics to measure the extracted program modules. However, our study does not have this issue, since our method use deep learning to learn project independent features from SimASTs. Other studies focused on semi-supervised CPDP methods [48], [58], [59], [63], which further use a certain number of labeled modules in the target project to construct models. However, these studies also do not satisfy our criteria for selecting baselines, since our study only uses the modules in the source project to construct the models.

C. THREATS TO CONSTRUCT VALIDITY

In previous SDP studies, many evaluation measures are used. However, some measures (such as precision, recall, F1) based on confusion matrix is threshold-dependent. The value of these measures depend on the predefined threshold to decide whether a new software module is defective or non-defective. Therefore, we consider AUC to evaluate the prediction performance of different CPDP methods. Moreover, we also compare the performance of our proposed method in terms of effort-aware evaluation measures.

VIII. CONCLUSION AND FUTURE WORK

In this article, we propose a novel deep learning based CPDP method DeepCPDP. We represent source code of each extracted program module by using simplified abstract syntax tree, since this modeling method ignore the project-specific information (such as the name of method and variable) and especially suitable for CPDP problem. Then we utilize embedding method SimASTToken2Vec, BiLSTM neural network and the attention mechanism to train the CPDP model.

Final empirical results on 10 open source projects from different application domains demonstrate the effectiveness of DeepCPDP when compared with state-of-the-art baselines.

In the future, we want to continue our research in three directions. First the generalization of our empirical results should be investigated by considering more datasets gathered from other large-scale commercial projects and open source projects. Second more state-of-the-art DNN models, embedding methods, and class imbalanced learning methods should be considered to further improve the performance of DeepCPDP method. Finally our proposed method DeepCPDP should be applied to the software development process and improve the quality of software projects.

For other researchers to follow our study, we provide a package[5] to replicate our proposed method DeepCPDP.

REFERENCES

[1] S. Amasaki, K. Kawata, and T. Yokogawa, "Improving cross-project defect prediction methods with data simplification," in *Proc. 41st Euromicro Conf. Softw. Eng. Adv. Appl.*, 2015, pp. 96–103.

[2] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," Sep. 2014, *arXiv:1409.0473*. [Online]. Available: https://arxiv.org/abs/1409.0473

[3] J. Bansiya and C. G. Davis, "A hierarchical model for object-oriented design quality assessment," *IEEE Trans. Softw. Eng.*, vol. 28, no. 1, pp. 4–17, Jan. 2002.

[4] Y. Benjamini and Y. Hochberg, "Controlling the false discovery rate: A practical and powerful approach to multiple testing," *J. Roy. Stat. Soc. B (Methodol.)*, vol. 57, no. 1, pp. 289–300, 1995.

[5] A. E. C. Cruz and K. Ochimizu, "Towards logistic regression models for predicting fault-prone code across software projects," in *Proc. 3rd Int. Symp. Empirical Softw. Eng. Meas.*, 2009, pp. 460–463.

[6] J. Chen, S. Liu, W. Liu, X. Chen, Q. Gu, and D. Chen, "A two-stage data preprocessing approach for software fault prediction," in *Proc. Int. Conf. Softw. Secur. Rel.*, Jun. 2014, pp. 20–29. [Online]. Available: https://ieeexplore.ieee.org/document/6895412

[7] X. Chen, Y. Mu, Y. Qu, C. Ni, M. Liu, T. He, and S. Liu, "Do different cross-project defect prediction methods identify the same defective modules?" *J. Softw., Evol. Process*, 2019. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/smr.2234

[8] X. Chen, D. Zhang, Z.-Q. Cui, Q. Gu, and X.-L. Ju, "Dp-share: Privacy-preserving software defect prediction model sharing through differential privacy," *J. Comput. Sci. Technol.*, vol. 34, no. 5, pp. 1020–1038, 2019.

[9] X. Chen, D. Zhang, Y. Zhao, Z. Cui, and C. Ni, "Software defect number prediction: Unsupervised vs supervised methods," *Inf. Softw. Technol.*, vol. 106, pp. 161–181, Feb. 2019. [Online]. Available: https://www.sciencedirect.com/science/article/abs/pii/S0950584918302076

[10] X. Chen, Y. Zhao, Z. Cui, G. Meng, Y. Liu, and Z. Wang, "Large-scale empirical studies on effort-aware security vulnerability prediction methods," *IEEE Trans. Rel.*, to be published.

[11] X. Chen, Y. Zhao, Q. Wang, and Z. Yuan, "MULTI: Multi-objective effort-aware just-in-time software defect prediction," *Inf. Softw. Technol.*, vol. 93, pp. 1–13, Jan. 2018.

[12] D. A. da Costa, S. McIntosh, W. Shang, U. Kulesza, R. Coelho, and A. E. Hassan, "A framework for evaluating the results of the SZZ approach for identifying bug-introducing changes," *IEEE Trans. Softw. Eng.*, vol. 43, no. 7, pp. 641–657, Jul. 2016.

[13] B. Felbo, A. Mislove, A. Søgaard, I. Rahwan, and S. Lehmann, "Using millions of emoji occurrences to learn any-domain representations for detecting sentiment, emotion and sarcasm," Aug. 2017, *arXiv:1708.00524*. [Online]. Available: https://arxiv.org/abs/1708.00524

[14] T. Fukushima, Y. Kamei, S. McIntosh, K. Yamashita, and N. Ubayashi, "An empirical study of just-in-time defect prediction using cross-project models," in *Proc. Work. Conf. Mining Softw. Repositories*, 2014, pp. 172–181.

[5] https://github.com/EzioQR/DeepCPDP

[15] D. Gray, D. Bowes, N. Davey, Y. Sun, and B. Christianson, "Software defect prediction using static code metrics underestimates defect-proneness," in *Proc. Int. Joint Conf. Neural Netw.*, 2010, pp. 1–7.

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

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

[18] Z. He, F. Shu, Y. Yang, M. Li, and Q. Wang, "An investigation on the feasibility of cross-project defect prediction," *Autom. Softw. Eng.*, vol. 19, no. 2, pp. 167–199, 2012.

[19] B. Henderson-Sellers, *Object-Oriented Metrics: Measures of Complexity*. Upper Saddle River, NJ, USA: Prentice-Hall, 1995.

[20] S. Herbold, A. Trautsch, and J. Grabowski, "A comparative study to benchmark cross-project defect prediction approaches," *IEEE Trans. Softw. Eng.*, vol. 44, no. 9, pp. 811–833, Sep. 2018.

[21] S. Hosseini, B. Turhan, and D. Gunarathna, "A systematic literature review and meta-analysis on cross project defect prediction," *IEEE Trans. Softw. Eng.*, vol. 45, no. 2, pp. 111–147, Feb. 2019.

[22] S. Huda, K. Liu, M. Abdelrazek, A. Ibrahim, S. Alyahya, H. Al-Dossari, and S. Ahmad, "An ensemble oversampling model for class imbalance problem in software defect prediction," *IEEE Access*, vol. 6, pp. 24184–24195, 2018.

[23] E. G. Jelihovschi, J. C. Faria, and I. B. Allaman, "Scottknott: A package for performing the Scott-Knott clustering algorithm in R," *TEMA (São Carlos)*, vol. 15, no. 1, pp. 3–17, 2014.

[24] X. Jing, F. Wu, X. Dong, F. Qi, and B. Xu, "Heterogeneous cross-company defect prediction by unified metric representation and CCA-based transfer learning," in *Proc. Joint Meeting Found. Softw. Eng.*, 2015, pp. 496–507.

[25] M. Jureczko and L. Madeyski, "Towards identifying software project clusters with regard to defect prediction," in *Proc. Int. Conf. Predictive Models Softw. Eng.*, 2010, p. 9.

[26] 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 Softw. Eng.*, vol. 21, no. 5, pp. 2072–2106, 2016.

[27] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," Dec. 2014, *arXiv:1412.6980*. [Online]. Available: https://arxiv.org/abs/1412.6980

[28] R. Krishna and T. Menzies, "Bellwethers: A baseline method for transfer learning," *IEEE Trans. Softw. Eng.*, vol. 45, no. 11, pp. 1081–1105, Nov. 2019.

[29] S. Lessmann, B. Baesens, C. Mues, and S. Pietsch, "Benchmarking classification models for software defect prediction: A proposed framework and novel findings," *IEEE Trans. Softw. Eng.*, vol. 34, no. 4, pp. 485–496, Jul. 2008.

[30] J. Li, P. He, J. Zhu, and M. R. Lyu, "Software defect prediction via convolutional neural network," in *Proc. Int. Conf. Softw. Qual., Rel. Secur.*, 2017, pp. 318–328.

[31] Y. Li, Z. Huang, Y. Wang, and B. Fang, "Evaluating data filter on cross-project defect prediction: Comparison and improvements," *IEEE Access*, vol. 5, pp. 25646–25656, 2017.

[32] Z. Li, X.-Y. Jing, X. Zhu, H. Zhang, B. Xu, and S. Ying, "Heterogeneous defect prediction with two-stage ensemble learning," *Automated Softw. Eng.*, vol. 26, no. 3, pp. 599–651, 2019. [Online]. Available: https://link.springer.com/article/10.1007/s10515-019-00259-1

[33] C. Liu, D. Yang, X. Xia, M. Yan, and X. Zhang, "A two-phase transfer learning model for cross-project defect prediction," *Inf. Softw. Technol.*, vol. 107, pp. 125–136, Mar. 2019. [Online]. Available: https://www.sciencedirect.com/science/article/abs/pii/S0950584918302416

[34] S. Liu, X. Chen, W. Liu, J. Chen, Q. Gu, and D. Chen, "FECAR: A feature selection framework for software defect prediction," in *Proc. IEEE 38th Annu. Comput. Softw. Appl. Conf.*, Jul. 2014, pp. 426–435.

[35] W. Liu, S. Liu, Q. Gu, J. Chen, X. Chen, and D. Chen, "Empirical studies of a two-stage data preprocessing approach for software fault prediction," *IEEE Trans. Rel.*, vol. 65, no. 1, pp. 38–53, Mar. 2016.

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

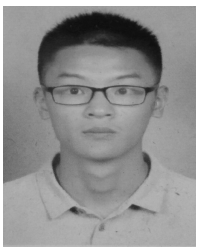
[37] T. Menzies, A. Butcher, A. Marcus, T. Zimmermann, and D. Cok, "Local vs. global models for effort estimation and defect prediction," in *Proc. IEEE/ACM Int. Conf. Automated Softw. Eng.*, Nov. 2011, pp. 343–351.

[38] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations *in vector* space," Jan. 2013, *arXiv:1301.3781*. [Online]. Available: https://arxiv.org/abs/1301.3781
[39] J. Nam, W. Fu, S. Kim, T. Menzies, and L. Tan, "Heterogeneous defect prediction," *IEEE Trans. Softw. Eng.*, vol. 44, no. 9, pp. 874–896, Sep. 2018.
[40] J. Nam, S. J. Pan, and S. Kim, "Transfer defect learning," in *Proc. Int. Conf. Softw. Eng.*, 2013, pp. 382–391.
[41] C. Ni, X. Chen, F. Wu, Y. Shen, and Q. Gu, "An empirical study on Pareto based multi-objective feature selection for software defect prediction," *J. Syst. Softw.*, vol. 152, pp. 215–238, Jun. 2019. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0164121219300573
[42] C. Ni, W.-S. Liu, X. Chen, Q. Gu, D.-X. Chen, and Q.-G. Huang, "A cluster based feature selection method for cross-project software defect prediction," *J. Comput. Sci. Technol.*, vol. 32, no. 6, pp. 1090–1107, 2017.
[43] A. Panichella, R. Oliveto, and A. De Lucia, "Cross-project defect prediction models: L'Union fait la force," in *Proc. Softw. Evol. Week-IEEE Conf. Softw. Maintenance, Reeng., Reverse Eng.*, Feb. 2014, pp. 164–173.
[44] F. Peters, T. Menzies, and A. Marcus, "Better cross company defect prediction," in *Proc. Work. Conf. Mining Softw. Repositories*, 2013, pp. 409–418.
[45] F. Rahman and P. Devanbu, "How, and why, process metrics are better," in *Proc. Int. Conf. Softw. Eng.*, 2013, pp. 432–441.
[46] F. Rahman, D. Posnett, and P. Devanbu, "Recalling the imprecision of cross-project defect prediction," in *Proc. Int. Symp. Found. Softw. Eng.*, 2012, p. 61.
[47] D. Ryu, O. Choi, and J. Baik, "Value-cognitive boosting with a support vector machine for cross-project defect prediction," *Empirical Softw. Eng.*, vol. 21, no. 1, pp. 43–71, 2016.
[48] D. Ryu, J.-I. Jang, and J. Baik, "A transfer cost-sensitive boosting approach for cross-project defect prediction," *Softw. Qual. J.*, vol. 25, no. 1, pp. 235–272, 2017.
[49] M. Schuster and K. K. Paliwal, "Bidirectional recurrent neural networks," *IEEE Trans. Signal Process.*, vol. 45, no. 11, pp. 2673–2681, Nov. 1997.
[50] J. Sliwerski, T. Zimmermann, and A. Zeller, "When do changes induce fixes?" *ACM Sigsoft Softw. Eng. Notes*, vol. 30, no. 4, pp. 1–5, 2005.
[51] Q. Song, Y. Guo, and M. Shepperd, "A comprehensive investigation of the role of imbalanced learning for software defect prediction," *IEEE Trans. Softw. Eng.*, vol. 45, no. 12, pp. 1253–1269, Dec. 2019.
[52] D. Tang, F. Wei, N. Yang, M. Zhou, T. Liu, and B. Qin, "Learning sentiment-specific word embedding for Twitter sentiment classification," in *Proc. Annu. Meeting Assoc. Comput. Linguistics*, 2014, pp. 1555–1565.
[53] C. Tantithamthavorn, S. McIntosh, A. E. Hassan, and K. Matsumoto, "Automated parameter optimization of classification techniques for defect prediction models," in *Proc. Int. Conf. Softw. Eng.*, 2016, pp. 321–332.
[54] B. Turhan, T. Menzies, A. B. Bener, and J. Di Stefano, "On the relative value of cross-company and within-company data for defect prediction," *Empirical Softw. Eng.*, vol. 14, no. 5, pp. 540–578, 2009.
[55] S. Wang, T. Liu, and L. Tan, "Automatically learning semantic features for defect prediction," in *Proc. Int. Conf. Softw. Eng.*, 2016, pp. 297–308.
[56] S. Watanabe, H. Kaiya, and K. Kaijiri, "Adapting a fault prediction model to allow inter languagereuse," in *Proc. Int. Workshop Predictor Models Softw. Eng.*, 2008, pp. 19–24.
[57] F. Wilcoxon, "Individual comparisons by ranking methods," *Biometrics Bull.*, vol. 1, no. 6, pp. 80–83, 1945.
[58] W. Fei, X.-Y. Jing, S. Ying, S. Jing, and Y. Sun, "Cross-project and within-project semisupervised software defect prediction: A unified approach," *IEEE Trans. Rel.*, vol. 67, no. 2, pp. 581–597, Jun. 2018.
[59] X. Xia, D. Lo, S. J. Pan, N. Nagappan, and X. Wang, "HYDRA: Massively compositional model for cross-project defect prediction," *IEEE Trans. Softw. Eng.*, vol. 42, no. 10, pp. 977–998, Oct. 2016.
[60] Y. Yang, Y. Zhou, J. Liu, Y. Zhao, H. Lu, L. Xu, B. Xu, and H. Leung, "Effort-aware just-in-time defect prediction: Simple unsupervised models could be better than supervised models," in *Proc. Int. Symp. Found. Softw. Eng.*, 2016, pp. 157–168.
[61] Q. Yu, J. Qian, S. Jiang, Z. Wu, and G. Zhang, "An empirical study on the effectiveness of feature selection for cross-project defect prediction," *IEEE Access*, vol. 7, pp. 35710–35718, 2019.
[62] X. Yu, J. Liu, Z. Yang, X. Jia, Q. Ling, and S. Ye, "Learning from imbalanced data for predicting the number of software defects," in *Proc. Int. Symp. Softw. Rel. Eng.*, 2017, pp. 78–89.
[63] Z. Zhang, X. Jing, and T. Wang, "Label propagation based semi-supervised learning for software defect prediction," *Automated Softw. Eng.*, vol. 24, no. 1, pp. 47–69, Mar. 2017.
[64] T. Zimmermann, N. Nagappan, H. Gall, E. Giger, and B. Murphy, "Cross-project defect prediction: A large scale experiment on data vs. domain vs. process," in *Proc. Eur. Softw. Eng. Conf. ACM Symp. Found. Softw. Eng.*, 2009, pp. 91–100.

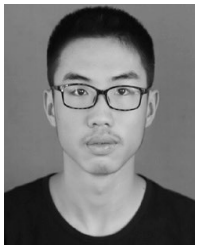
DEYU CHEN is currently an Associate Professor with the School of Information Science and Technology, Nantong University. His research interest includes software quality assurance.

XIANG CHEN received the B.Sc. degree from the School of Management, Xi'an Jiaotong University, China, in 2002, and the M.Sc. and Ph.D. degrees in computer software and theory from Nanjing University, China, in 2008 and 2011, respectively. He is currently an Associate Professor with the Department of Information Science and Technology, Nantong University. His research interest includes software engineering. Particularly, he is interested in software maintenance and software testing, such as software defect prediction, combinatorial testing, regression testing, and fault localization. In these areas, he has published over 60 articles in refereed journals or conferences, such as the IEEE TRANSACTIONS ON SOFTWARE ENGINEERING, IEEE ACCESS, *Information and Software Technology*, *Journal of Systems and Software*, IEEE TRANSACTIONS ON RELIABILITY, *Journal of Software: Evolution and Process*, *Software Quality Journal*, *Journal of Computer Science and Technology*, ASE, ICSME, SANER, and COMPSAC. He is a Senior Member of CCF, China, and a member of ACM. He is currently serving as an Associate Editor for IEEE ACCESS.

HAO LI is currently pursuing the master's degree with the College of Intelligence and Computing, Tianjin University. His research interests include software defect prediction and data mining.



JUNFENG XIE is currently pursuing the master's degree with the School of Computer Science, Fudan University. His research interests include software defect prediction and data mining.



YANZHOU MU is currently pursuing the master's degree with the College of Intelligence and Computing, Tianjin University. His research interests include software defect prediction and data mining.