




Adversarial domain adaptation for cross-project defect prediction

Hengjie Song¹ · Guobin Wu¹ · Le Ma² · Yufei Pan¹ · Qingan Huang¹ · Siyu Jiang³ 

Accepted: 24 July 2023 / Published online: 19 September 2023

© The Author(s), under exclusive licence to Springer Science+Business Media, LLC, part of Springer Nature 2023

Abstract

Cross-Project Defect Prediction (CPDP) is an attractive topic for locating defects in projects with little labeled data (target projects) by using the prediction model from other projects with sufficient data (source projects). However, previous models may not fully capture the semantic features of programs because of inappropriate feature extraction models. Besides, researchers may fail to consider the relationship between the decision boundary and target project data when matching two feature distributions by adopting transfer learning methods, which would lead to the misclassification of target samples that are near boundary. To handle these drawbacks, we propose a novel Adversarial Domain Adaptation (ADA) model for CPDP. Specifically, we leverage a Long Short-Term Memory network with attention mechanism to extract semantic features that better represent programs. Then, we train two classifiers to correctly categorize source samples and distinguish ambiguous target instances that influence prediction accuracy. Next, we treat the classifiers as a discriminator and feature extraction model as a generator, and train them based on adversarial learning methods to depict the desired relationship. As the classifiers know this relationship, they should attain better performance. Extensive experiments on two benchmark datasets are conducted to verify the effectiveness of the proposed ADA methods. Experimental and statistical results show that ADA significantly outperforms other state-of-the-art baseline methods.

Keywords Adversarial learning · Cross-project defect prediction · Domain adaptation · Software reliability

1 Introduction

With the development of modern software engineering, more and more resources are allocated in phase of software testing to keep software projects bug-free, and thus software testing has become one of the most important phases in the whole software lifecycle. As a result, how

Communicated by Leandro L. Minku.

✉ Siyu Jiang
jiangsiyu@gdufs.edu.cn

Extended author information available on the last page of the article

to guarantee software reliability through software testing attracts significant attention. Some researchers (He et al. 2015; Ni et al. 2019; Balogun et al. 2021; Bal and Kumar 2020) adopted Software Defect Prediction (SDP) methods to locate defects of software projects. A program that contains at least one bug (defect), and/or bugs that seriously interfere with its functionality, is said to be buggy (defective). Otherwise, the program is said to be clean (non-defective). Most existing SDP methods utilized the historical data of a software project to train their prediction models, and then employed it to find out future defects in the same project. However, in practice, it is hard for researchers or developers to build a satisfactory SDP model in the early stage of a given project because labeled data are hardly available. Therefore, Cross-Project Defect Prediction (CPDP) (Briand et al. 2002; Herbold et al. 2018; Hosseini et al. 2019; Khatri and Singh 2021) was proposed, making it feasible to build an SDP model in a new project. CPDP allows us to train defect prediction model based on mature projects with sufficient labeled data (source projects, and then apply it to new projects that lack labeled data (target projects). A typical CPDP method can be roughly split up into two modules: how to extract predictive features and how to effectively apply knowledge to target projects that is learned from source projects.

How to construct predictive features remains a challenge in the research of CPDP. As a binary classification problem, feature extraction is one of the most important components in CPDP models. In the early study of software defect prediction, most defect prediction methods (Watanabe et al. 2008; Turhan et al. 2009; Shepperd et al. 2014; Jing et al. 2014; Zhu et al. 2021; Li et al. 2019) depended on handcrafted features that represent the static characteristic of programs, including McCabe loop complexity (McCabe 1976), Halstead metric (Halstead 1977) and CK (Chidamber-Kemerer) metric (Chidamber and Kemerer 1994). However, these handcrafted metrics were often designed based on researchers' statistical analysis or experience of the software, which might not fully mine the contextual and semantic features of project source codes (Wang et al. 2016). If CPDP models are constructed based only on these handcrafted features, they may suffer from losing some prediction performance. Because of its prominence in feature representation, deep learning is strongly recommended by researchers (Wang et al. 2016; Li et al. 2017) to capture programs' hidden contextual and semantic features and feed them into prediction models for training. Several prevalent deep learning approaches have been adopted in CPDP, including Deep Bayes Network (DBN) (Tong et al. 2019; Wang et al. 2020), Long Short-Term Memory (LSTM) network (Li et al. 2019; Deng et al. 2020; Liang et al. 2019) and Convolutional Neural Network (CNN) (Li et al. 2017; Qiu et al. 2019a, d), etc. To some extent, source code in software project is a kind of standardized and formal languages which is closely related to NLP context (Huang et al. 2021), and we believe that the LSTM network is more suitable for extracting hidden semantic features in CPDP tasks. LSTM is a variant of recurrent neural networks, which processes sequential data by modeling units in sequence and "memorize" long-term dependencies by computations (Hochreiter and Schmidhuber 1997). In this paper, we extend the LSTM network with Attention Mechanism (AM) (Vaswani et al. 2017) as the feature generator for fully capturing the hidden semantic features of programs. AM is a method that enables us to focus on the most important parts to make further decisions by giving different weights to different parts of the input. Since it can bring significant improvement, AM recently has almost become an essential standard in many sequential tasks (Veličković et al. 2018; Zeng et al. 2021). In the field of CPDP, project source codes are also sequential data, and we believe we would benefit from LSTM network and AM, too.

Another challenging problem in CPDP tasks is how to effectively transfer knowledge learned from one project and apply it to other projects. Most fruitful CPDP methods (Yu et al. 2017; Ryu et al. 2017; Hosseini et al. 2018; Wu et al. 2018; Liu et al. 2019; Gong et al. 2020)

adopted transfer learning approaches to bridge the gap between feature distributions of two different projects. Although inspiring and straightforward, these CPDP models might miss out the relationship between the target project instances and decision boundary. Chen et al. (2015) put forward a CPDP method called Double Transfer Boosting, which re-calculated instances' weights and reduced the divergence of two distributions based on TrAdaBoost (Dai et al. 2007). Yu et al. (2017) proposed a feature matching and transfer method to perform CPDP. They designed an algorithm to match and transfer features in the source and target project with respect to their distance. Xu et al. (2019) came up with a transfer learning method, named Balanced Domain Adaptation, to assign different weights to marginal and conditional distributions of two projects. Through balancing these two kinds of distribution, their approach was able to effectively overcome the divergence of two projects. However, these methods might not be capable of mining predictive features since they did not take the relationship between target data and decision boundary into account when matching two distributions. From mathematical point of view, this relationship can be defined as the distance between the boundary and target data. As shown in Fig. 1, even if the two distributions are quite similar, the feature generator could generate ambiguous instances near the decision boundary if we do not take the distance into consideration. Hypothetically, we consider that these ambiguous targets samples could be misclassified and therefore result in poor prediction performance.

In our work, we propose a novel Adversarial Domain Adaptation (ADA) method for CPDP to handle the drawbacks mentioned above. The proposed approach, as shown in Fig. 1, adopts a domain adaptation method based on adversarial learning to rectify ambiguous classification data, which not only alleviates the gap in the feature distributions between different projects, but also takes the relationship between the target project instances and learned decision boundary into account to further enhance performance. The general framework of the ADA method is presented in Fig. 2. Specifically, we first compile project source files to generate corresponding abstract syntax trees (ASTs), and then convert ASTs into token vectors by traversing them. By certain mapping rules, we map the token vectors to integer vectors and feed them into the following feature extraction network. We adopt a bi-directional LSTM network with a self-attention layer to capture the programs' hidden semantic features. To avoid the overfitting problem and obtain more accurate features, we amalgamate the

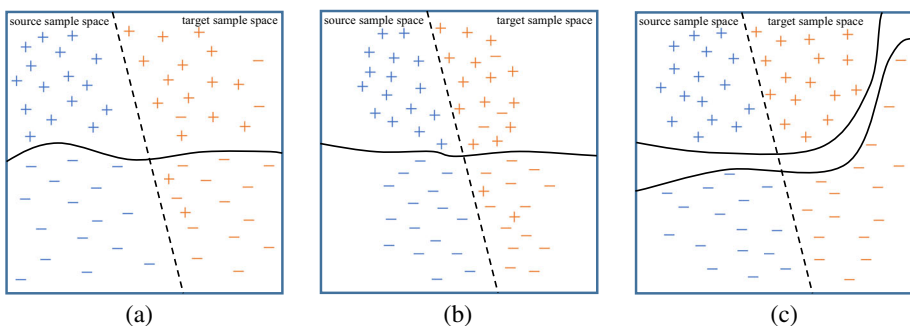


Fig. 1 (a) Knowledge learned from source project is directly transferred to target project. (b) Traditional transfer learning methods may fail to consider the relationship between the decision boundary and the target samples, which results in performance loss. (c) We train two classifiers simultaneously, which try to categorize source instances correctly and distinguish ambiguous target instances at the same time. In these figures, dashed lines represent the boundary between the source and target sample space while the solid lines represent the decision boundary learned by CPDP models

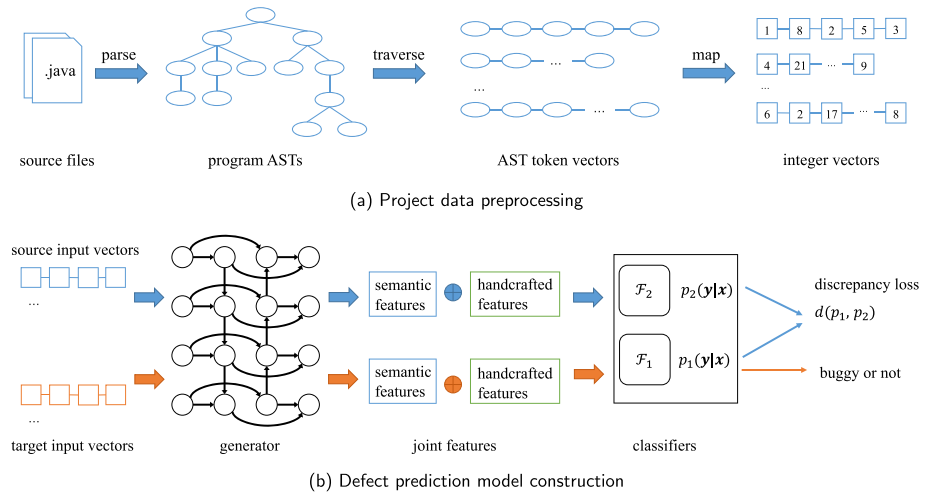


Fig. 2 (a) In the phase of data preprocessing, we first parse the project source files into abstract syntax trees (ASTs), and then traverse the ASTs to get token vectors. Lastly, we convert the token vectors into integer vector by pre-defined mapping rules. (b) In the phase of model construction, we first feed the integer vectors into the generator (LSTM network with AM) to extract preliminary semantic features. Then, we combine the semantic and handcrafted features to form joint features. Next, we utilize the joint features to simultaneously train two classifiers based on adversarial learning methods

generated features with the handcrafted features to construct joint features. Inspired by Generative Adversarial Networks (GAN) (Goodfellow et al. 2014) and an adversarial domain adaptation method proposed by Saito et al. (2018), we utilize a domain adaptation method and train it in the manner of adversarial learning to improve the discriminability of the joint features. In ADA, we simultaneously train two classifiers, and they take joint features as input, trying to correctly classify source samples and identify the target instances which are not close to the support of the source at the same time. We presume these target instances are non-discriminative and ambiguous since they are not categorized explicitly as negative or positive. Therefore, we see these two classifiers as a discriminator to discriminate whether the generator (LSTM network with an attention layer) creates discriminative features for the target instances. In this way, the disagreement of two classifiers can be used to further compute the distance between the decision boundary and the target instances. When we repeat this procedure, ADA can reduce the divergence of the feature distribution in two projects, and give information to the classifiers about the relationship mentioned above. As they know the relationship, the classifiers can correctly categorize those confusing target instances, and thereby improve defect prediction performance.

To validate the effectiveness of ADA method, we conduct multiple experiments on two public benchmark datasets, AEEEM (D’Ambros et al. 2010) and PROMISE (Jureczko and Madeyski 2010). In many experimental settings, the proposed model is superior to other the state-of-the-art methods by a significant margin. The contributions of this work are three-fold.

- We propose a novel CPDP model which integrates transfer learning and adversarial learning methods to not only bridge the gap between the feature distributions of different projects, but also fully think of the relationship between the target data and class boundary.
- We extend the LSTM network with Attention Mechanism to learn long-term dependencies in a software project context and extract more meaningful and semantic features of

programs. Together with AM, the LSTM network significantly improves the performance of defect prediction.

- We conduct extensive experiments on two benchmark datasets and the experimental and statistical results verify the effectiveness of the proposed method.

The rest of this paper consists of six parts. Firstly, we shortly report the related work in Section 2. Then, we thoroughly illustrate the ADA model in Section 3. In Section 4, we describe the details of experimental setups. Next, we present the experimental and statistical results in Section 5. More discussions on the proposed ADA method are in Section 6. Lastly, we conclude our work and talk about future work in Section 7.

2 Related Work

In this section, we briefly introduce related work on cross-project defect prediction, adversarial domain adaptation and attention mechanism.

2.1 Cross-Project Defect Prediction

Software Defect Prediction (SDP) (Li et al. 2018; Zou et al. 2018; Thota et al. 2020; Rathore and Kumar 2021) aims to find out defects in software modules before release so that developers can allocate limited resources optimally. According to the project data used in the training and evaluating phase, there are two branches in SDP, namely Within-Project Defect Prediction (WPDP) and Cross-Project Defect Prediction (CPDP). In WPDP, the data are from the same project in the training and evaluating phase. By contrast, CPDP uses different project data when training and evaluating. In our work, we concentrate on the problem of CPDP.

To evaluate the feasibility of CPDP, Zimmermann et al. (2009) carried out a total of 622 cross-project experiments on 12 software projects using logistic regression classifiers. Their experimental results showed that only 3.4% of them were successful. They came to conclusion that CPDP was a challenging problem because it was not transitive. Researchers hoped to improve software defect prediction performance by applying deep learning (LeCun et al. 2015) methods to mine semantic features of programs due to its prominent feature learning capability. Wang et al. (2016) utilized a DBN model to automatically extract semantic features from AST of programs, and bridged the gap between semantic and defect prediction features of different projects. Tong et al. (2019) adopted a transfer naive Bayes approach to consider both class-imbalance and feature importance problems, and used it to make two feature distributions as similar as possible. Li et al. (2017) applied a CNN to generate effective features of programs. Combined with traditional handcrafted features, the prediction model they trained achieved better performance than other baseline methods. Different from the deep learning methods that adopted in the above CPDP study, Pandey and Tripathi (2021) employed an LSTM network to mine programs' semantic features. In the context of CPDP, project source code is a kind of formal language containing rich structural and semantic information, which is more closely related to the context of NLP problem (Li et al. 2019). Therefore, we believe that Recurrent Neural Network (RNN) is more suitable since it excels at processing sequential input and we leverage a variant of RNN, the LSTM network to mine the structural and semantic information contained in programs.

To further verify the feasibility of CPDP, He et al. (2012) carried out three experiments on 34 projects by manually selecting training data. They concluded that CPDP performance could be better than WPDP tasks in some cases. After further analysis, they concluded that

defect prediction results were related with the distributional characteristics which could be valuable for training data selection. Their research suggested that CPDP tasks were feasible if the data distributions across two different projects can be made similar. Taking their suggestions, Herbold (2013) put forward a distance-based algorithm for the training data selection according to the corresponding distributional characteristics. Concretely, they came up with characteristic vectors to represent each dataset, which consisted of mean and standard deviation, and they adopted the characteristic vectors to stand for the marginal distribution of each dataset. Herbold conducted experiments on 44 public software projects and witnessed a 9% improvement in performance over traditional CPDP methods. Apart from training data selection methods, transfer learning techniques (Nam et al. 2013; Qiu et al. 2019d; Liu et al. 2019; Jin 2021; Huang et al. 2021) were also adopted to make two distributions similar. Liu et al. (2019) came up with a CPDP method named Two-Phase Transfer Learning. Firstly, they chose two source projects that are most similar by means of a source project estimator. Secondly, they leveraged TCA+ (Nam et al. 2013) to construct two defect predictors based on the two selected project individually, and then combined their prediction probabilities to enhance performance. Qiu et al. (2019d) proposed a CPDP model called Transfer CNN (TCNN). They adopted CNN to extract the semantic features of project data and added a matching layer to align feature distribution of two different projects. When matching, they embedded the source and target data representations into a reproducing kernel Hilbert space and utilized a classic transfer learning method, Maximum Mean Discrepancy (MMD) (Borgwardt et al. 2006), to bridge the gap between two distributions. Next, they combined generated semantic features with handcrafted features and trained the predictor based on them. Jin (2021) learned domain adaptation model by a method called kernel twin support vector machine, trying to match the feature distributions between the source and target project as much as possible. He trained the feature generator that aimed to match distributions between two different projects and assumed that such target features were correctly categorized by the defect predictor for the reason that they were matched to source instances.

Compared to CPDP, heterogeneous defect prediction (HDP) (Nam and Kim 2015; Chen et al. 2021) relaxes the limitation of defect data used when predicting, allowing different metric sets to be contained in the source and target projects. Afterward, many researchers came up with fruitful work on HDP. Jing et al. (2015) proposed a unified metric representation for the defective data, and used them for further canonical correlation analysis (CCA) to reduce the gap between two domains. Based on CCA, Li et al. (2018) also proposed an HDP model called cost-sensitive transfer kernel canonical correlation analysis, which can not only make the data distributions of source and target projects much more similar in the nonlinear feature space, but also utilize the different misclassification costs for defective and non-defective classes to alleviate the class imbalance problem. Li et al. (2019) employed multiple sources to improve the performance of HDP model, putting forward a multi-source selection based manifold discriminant alignment approach. Their experimental results verified the performance gain. Bal and Kumar (2023) enhanced the data pre-processing for HDP by utilizing chi-square test to select the relevant metrics between source and target datasets. Finally, they performed experiments using their proposed approach with various machine learning algorithms to various the effectiveness of their model. Note that we only concentrate on "homogeneous" (non-heterogeneous) defect prediction.

In the above CPDP studies, most of them adopted different approaches (training data selection (Herbold 2013) or transfer learning (Qiu et al. 2019d; Liu et al. 2019; Jin 2021) to make the feature distributions of the two different projects as similar as possible. However, they might fail to take full consideration of the relationship between the decision boundary and target data when matching the distributions. They used different transfer learning approaches

that can match two distribution, hoping the target samples are correctly classified by the defect predictor. As we mentioned before, even though the two distributions are quite similar, the classifier would still be confused by the target samples with ambiguous features. In this paper, we put forward a domain adaptation method based on adversarial learning to fully consider the relationship between the target instances and decision boundary, which can eliminate the confusion for predictor.

2.2 Adversarial Domain Adaptation

Domain adaptation is a prevalent type of transfer learning, where the target task remains the same as the source whereas the domain is different (Pan and Yang 2010). Since generative adversarial network (GAN) was proposed by Goodfellow et al. (2014), many scholars have applied it into domain adaptation to solve specific tasks including image classification (Tzeng et al. 2017; Saito et al. 2018; Ma et al. 2019), object detection (Song et al. 2020; Su et al. 2020), machine translation (Wang et al. 2021), image semantic segmentation (Li et al. 2019; Yi et al. 2021). Tzeng et al. (2017) proposed an adversarial discriminative domain adaptation method to classify images. They learned discriminative representation based on source data at first. Then, they learned a separate encoding based on transfer learning to map target data to the same feature space as the source. Lastly, they trained the whole model by minimizing a domain-adversarial loss function. Song et al. (2020) proposed a method for the salient object detection problem based on adversarial domain adaptation. To evaluate the effectiveness of their model, they collected a new dataset and made comparison with other methods on the dataset. In the field of machine translation, Wang et al. (2021) came up with a counterfactual domain adaptation method to improve target domain translation. By adopting adversarial learning methods, they used the concatenations of texts in source domain and tags in target to construct counterfactual representations. Motivated by adversarial learning, Li et al. (2019) put forward a bidirectional learning model to solve the problem of image semantic segmentation. They separated their model into two submodules: image-to-image translation model and segmentation adaptation model, which would be motivated to promote each other alternatively and gradually reduced the domain gap. Saito et al. (2018) put forward an adversarial domain adaptation method, attempting to match distribution of source and target by using the task-specific decision boundaries. They have proven that this method outperformed other methods in the tasks of image classification and semantic segmentation.

Inspired by these fruitful work, we believe that CPDP models will benefit from adversarial learning since it also is an application of domain adaptation. Particularly, we assume that we can take advantage of the training pattern adopted by Saito et al. (2018), using multiple task-specific classifiers as discriminators and a feature generator which tries to "fool" them in order to generate more discriminative features. Different from previous CPDP methods that apply traditional domain adaptation methods (Jin 2021; Xu et al. 2018; Qiu et al. 2019b; Zou et al. 2021) which tried to align two distributions by training appropriate distance metrics, in this work, we develop the domain adaptation method based on adversarial learning like this to reduce the divergence between two feature distributions of different projects.

3 Proposed Method

In this section, we elaborate the ADA method in details. Firstly, we give the formal formulation of CPDP problem. Then, we put forward the overall framework of the proposed approach.

A Discussion about the data preprocessing of the software projects is placed in Section 3.2, including program parsing and data imbalanced learning. In Section 3.3, we elaborately present how we construct the CPDP model.

3.1 Problem Definition

Let the given source project with labelled data be $D_S = \{(x_{S_i}, y_{S_i})\}_{i=1}^n$, where $x_{S_i} = \{x_{S_{i1}}, \dots, x_{S_{id}}\} \in \mathbb{R}^{n*d}$ denotes the i -th source instance, and $y_{S_i} \in \{0, 1\}$ is the corresponding defect information (0 for non-defective and 1 for defective). Let the given target project without labelled data be $D_T = \{x_{T_i}\}_{i=1}^m$, where $x_{T_i} = \{x_{T_{i1}}, \dots, x_{T_{id}}\} \in \mathbb{R}^{m*d}$ denotes the i -th target instance. We assume that both source and target samples share the same feature space as they come from the same set of metrics (i.e. $x_S, x_T \in \mathbb{R}^d$ where d denotes the dimension of the feature). Let n and m be the numbers of instances in the source and target projects, respectively. Let $P_S(\mathbf{X}_S)$ and $P_T(\mathbf{X}_T)$ be the marginal probability distributions of $\mathbf{X}_S = \{x_{S_i}\}_{i=1}^n$ and $\mathbf{X}_T = \{x_{T_i}\}_{i=1}^m$ from the source and target projects, respectively. Generally, the distributions of two distinct projects are different, too, which implies $P_S(\mathbf{X}_S) \neq P_T(\mathbf{X}_T)$. Cross-project defect prediction aims to enhance the performance of the target predictor $f_T(\cdot)$ in target project D_T by utilizing the knowledge learned from source project D_S .

In this paper, we learn the optimal parameters θ^* of our method by solving the following minimization problem,

$$\begin{aligned} \theta^* &= \arg \min_{\theta \in \Theta} \sum_{(x,y) \in D_S} \frac{P_T(\mathbf{X}_T)}{P_S(\mathbf{X}_S)} P(\mathbf{X}_S) \ell(x, y, \theta) \\ &\approx \arg \min_{\theta \in \Theta} \sum_{i=1}^n \frac{P_T(x_{T_i})}{P_S(x_{S_i}, y_{S_i})} \ell(x_{S_i}, y_{S_i}, \theta), \end{aligned} \quad (1)$$

where Θ is the parameter space and $\ell(x, y, \theta)$ denotes the error function relying on θ . Therefore, by assigning distinct weights to each sample (x_{S_i}, y_{S_i}) with corresponding value $\frac{P_T(x_{T_i})}{P_S(x_{S_i}, y_{S_i})}$, we are able to build an accurate predictor for the target project.

3.2 Data Preprocessing

Figure 2 presents the general framework of the proposed ADA approach. We divide the whole framework into two steps: data preprocessing and model construction. In this subsection, we discuss the details of data preprocessing in ADA method.

3.2.1 Generating Input Vectors

An Abstract syntax tree (AST) is a representation of syntactic structure parsed from source code, contains rich semantic information of the software project, and is considered really useful in the field of program analysis. Alon et al. (2019); Compton et al. (2020) Early studies (Wang et al. 2016, 2020; Chen et al. 2016; Balog et al. 2017) have proven that ASTs can be mined and utilized in software defect prediction, so we choose AST as a high level representation of the project source codes. Specifically, we use an open source compiling tool, *Javalang*¹, to parse Java source files and generate corresponding ASTs. There are

¹ <https://pypi.org/project/javalang/0.13.0/>

many types of nodes in ASTs, but only a part of them are highly related to the defects of the code. Following previous work (Wang et al. 2020; Deng et al. 2020; Huang et al. 2021), we evaluate the AST nodes and select four kinds of nodes: 1) method invocation nodes, 2) declaration nodes, 3) control flow nodes and 4) other necessary nodes. Then, we employ depth-first traversal to generate sequence vectors from ASTs. Since the sequence vectors are a list of string tokens, which cannot be directly used as input of an LSTM network, we construct a mapping dictionary between nodes and integers. After this step, we convert the project source files into integer vectors.

3.2.2 Imbalanced Learning

The imbalanced learning problem is one of the challenging problems in machine learning, where the number of one kind of samples is much more than that of another kind of samples (He and Garcia 2009). This problem is also faced in the field of SDP (Jing et al. 2017; Tong et al. 2019; Bal and Kumar 2020), as we will discuss the details of datasets chosen in this paper in Section 4.1. Qiu et al. (2019c) and Song et al. (2019) carried out large-scale experiments to explore the characteristic of imbalanced learning problem and systematically evaluated multiple imbalanced learning methods. Their research proved that class imbalance was omnipresent and would significantly affect the performance of prediction models. Nevertheless, we can alleviate this problem by adopting appropriate imbalanced learning approaches.

There are two types of methods coping with class imbalance, over-sampling method and under-sampling method. The former over-samples the minority class and the latter under-samples the majority class, both of which make the numbers of instances in two class balanced. In our approach, we apply a Synthetic Minority Over-Sampling Technique (SMOTE) (Chawla et al. 2002), which is a hybrid method of under-sampling and over-sampling. SMOTE is capable of handling the skewed class distribution by introducing and learning a bias towards the minority class, thereby achieves better performance than minority over-sampling with replacement method.

3.3 Model Construction

In this section, we illustrate how we build the ADA model based on adversarial learning methods in details. Figure 1 provides a brief introduction to our approach. We first leverage a bi-directional LSTM network with AM as a feature generator which takes the integer vectors as input. Then, we concatenate the generated semantic features and the handcrafted features to construct the joint features. Next, we simultaneously train two classifiers as a discriminator by feeding the joint features into them. Two classifiers attempt to categorize source samples properly. In the meanwhile, they are also trained to find out which target instance is distance to the support of the source. We reckon that these target instances may confuse the classifiers as most of them are likely to be misclassified, and we consider these target samples as non-discriminative. Based on adversarial training process (Chen et al. 2020), the generator is forced to create discriminative target features near the support by considering discriminator prediction for target samples. In adversarial learning manner, the generator attempts to fool the discriminator, and the discriminator feedback to the generator whether the extracted features are wanted. Through this training pattern, we are able to align two feature distributions of the source and target project, and in the meanwhile, let the classifiers know how to correctly

predict ambiguous target instances. After training, the ADA model can predict whether a new project instance is defective or not.

3.3.1 Generator

Since source code of software project is a kind of standardized and formal languages (Huang et al. 2021), we believe CPDP tasks are more related to NLP context. LSTM networks (Hochreiter and Schmidhuber 1997) are more suitable for dealing with NLP tasks because they can learn long-term dependencies by sophisticated computations. To mine which code snippet is the most important in defect prediction, we also adopt attention mechanism to learn and assign different weights to the sequential data. Together with AM, we suppose the LSTM network is powerful to extract contextual and semantic features of project data.

The whole network architecture of the generator is described in Fig. 3. The feature generator is comprised of five parts: an input layer, an embedding layer, a bi-directional LSTM layer, an attention layer and an output layer. We discuss the latter four layer as follows.

Embedding Layer As a useful technique for encoding semantic information, word embeddings have been proven powerful as extra features in many NLP tasks (Almeida and Xexéo 2019). Therefore, we convert each integer vector into the corresponding embedding matrix by means of a word embedding layer,

$$\varphi : T \rightarrow W_e, \tag{2}$$

where $T \in \mathbb{R}^N$ represents the input integer vector after AST parsing, $W_e \in \mathbb{R}^{E*N}$ is the embedding matrix to be learned and φ is a mapping function between them. We randomly initialize the embedding matrix and it can be updated when training the whole network. We assume that the high-dimensional embedding matrix is able to capture more contextual information contained in ASTs. Encoded integer vectors represented by the embedding matrices are then fed into a bi-directional LSTM network to create preliminary semantic features.

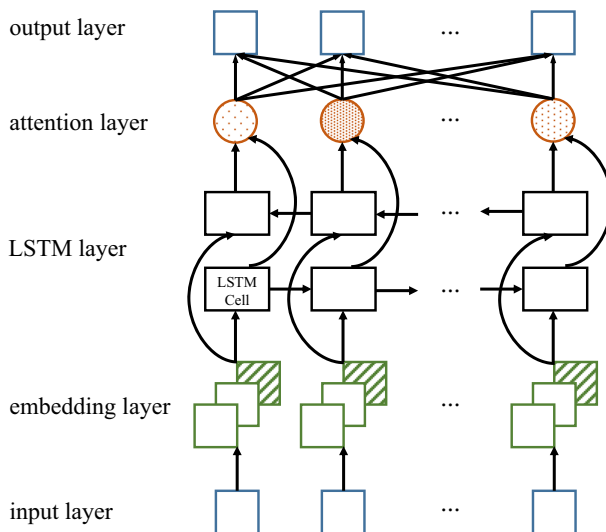


Fig. 3 The network architecture of the generator

LSTM Layer LSTM network is good at dealing with sequential data and explores long-term dependencies in the semantic context. To feed the embedding matrix into LSTM, we split it into N column vectors with E dimensions,

$$W_e = [e_1, e_2, \dots, e_N], \tag{3}$$

where $e_i \in \mathbb{R}^E (i = 1, 2, \dots, N)$. There are three types of gates, including input, output and forget gates, in LSTM networks. Based on these gates, LSTM can control how the information is processed and memorized in the cell states of LSTM units. The forget gates decide what information ought to be ignored from the previous moment, which can be described as

$$f_t = \sigma (W_{fg} \cdot [h_{t-1}, e_t] + b_{fg}), \tag{4}$$

where $\sigma(\cdot)$ denotes the sigmoid function used for neural activation, $h_i (0 \leq i \leq N)$ is the hidden state of the memory cell at moment i and W_{fg} and b_{fg} are two parameters of the forget gate. Different from forget gates, input gates decide what to memorize from the current moment. This includes two parts, one of which is the activation result of the input and the other of which is the tanh result of the input. We can formulate them as follows.

$$\begin{aligned} i_t &= \sigma (W_{in} \cdot [h_{t-1}, e_t] + b_{in}), \\ \tilde{C}_t &= \tanh (W_C \cdot [h_{t-1}, e_t] + b_C), \end{aligned} \tag{5}$$

where W_{in}, b_{in}, W_C, b_C are the corresponding weights and biases. \tilde{C}_t is called candidate value of moment t , which is a part of updating to the current cell state. We multiply the old state by f_t , trying to forget the information we decide to ignore, and add the candidate value scaled by the input gate's result:

$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t. \tag{6}$$

Final output is given by the output gates according to the output from last moment and cell state from current moment, namely

$$\begin{aligned} o_t &= \sigma (W_{out} \cdot [h_{t-1}, e_t] + b_{out}), \\ h_t &= o_t * \tanh(C_t), \end{aligned} \tag{7}$$

where W_{out}, b_{out} are the weights and bias of the output gates. Through the mechanisms of LSTM networks, we are able to learn the dependencies of defective codes in the context, which is greatly helpful for defect prediction.

Attention Layer The semantic features generated by the bi-directional LSTM network can be directly used by classifiers to train the prediction model. Though simple and effective, however, directly feeding these features into classification model may result in performance loss. As in natural language, different words have different importance, and people tend to put more emphasis on more significant words rather than less relevant ones. Since programming languages conform to the paradigm of natural languages to some extent, we believe different code snippets contribute distinctly to software defect prediction. To formally depict these differences, we introduce the attention mechanism (Vaswani et al. 2017) to assign weights for the generated semantic features. Firstly, we input the hidden state h_i and output feature o_{t-1} at moment $t - 1$ into an alignment model $\zeta(\cdot)$ to compute the alignment score,

$$a_{t,i} = \zeta (o_{t-1}, h_i). \tag{8}$$

We use a multi-layer perceptron as the alignment model, which evaluates how well the elements of the input sequence align with the current output at moment t . Then, we obtain the weights $\alpha_{t,i}$ by applying a softmax function to the previously computed alignment scores,

$$\alpha_{t,i} = \text{softmax} (a_{t,i}) = \frac{\exp (a_{t,i})}{\sum_{i=1}^N \exp (a_{t,i})}. \tag{9}$$

Finally, we obtain a context vector q_t at moment t by adding up the corresponding weights of all moments,

$$q_t = \sum_i^N \alpha_{t,i} h_i. \tag{10}$$

Output Layer The context vector produced by the attention mechanism then is concatenated with the handcrafted features to construct joint features for further prediction. We can see the generator as a mapping \mathcal{G} from the AST input integer vectors to the joint features,

$$\mathcal{G} : \mathcal{V} \rightarrow R_j, \tag{11}$$

where $\mathcal{V} \in \mathbb{R}^N$ represents the AST input integer vector and $R_j \in \mathbb{R}^{N_j}$ is the joint features vector with N_j dimensions extracted by the generator.

3.3.2 Training Steps

The aim of the ADA method is to minimize the misalignment between feature distributions of the source and target projects by classifiers which know the relationship about the target instances and decision boundary. To achieve this goal, we have to distinguish some target instances from others. Generally, those instances which are near the class boundary are more likely to be misclassified by classifiers and we say these instances are ambiguous. We exploit two distinct classifiers to predict whether a given target instance is defective or not, and then make use of their disagreement to find these target instances. Consider two distinct classifiers, \mathcal{F}_1 and \mathcal{F}_2 , which are initialized differently. Since the labeled source samples are available, we assume that \mathcal{F}_1 and \mathcal{F}_2 can correctly classify source samples after training. We see the two classifiers as a discriminator, telling the generator whether the target features are discriminative by maximizing the discrepancy of corresponding instances. By doing so, the two classifiers can be different and be capable of finding ambiguous target instances. Then, the generator is forced not to create such target features by minimizing the same discrepancy over the target instances. The discriminator and generator interact with each other, encouraging the generator to create more predictive features and the discriminator to classify more precisely. We repeat the above steps in adversarial learning manners.

$$\begin{aligned} & \min_{\mathcal{G}, \mathcal{F}_1, \mathcal{F}_2} \mathcal{L} (\mathbf{X}_S, Y_S), \\ \mathcal{L} (\mathbf{X}_S, Y_S) &= -\mathbb{E}_{(x_S, y_S) \sim (\mathbf{X}_S, Y_S)} [y_S \log p (y | \mathbf{x}_S) + (1 - y_S) \log (1 - p (y | \mathbf{x}_S))], \tag{12} \\ p (y | \mathbf{x}) &= \frac{p_1 (y | \mathbf{x}) + p_2 (y | \mathbf{x})}{2} \end{aligned}$$

To sum up, we have to train a generator and two classifiers first, both of which have to correctly categorize source instances. Then, we apply a discrepancy maximization problem to the two

classifiers for filtering out unwanted target instances, forcing the generator to create more predictive target features. We address this task in the following steps.

Step 1 Firstly, two classifiers and the generator are trained based on source project data. They have to properly categorize source instances. This step is important because subsequent derivations are based on this assumption. In our study, we select the Logistic Regression (LR) classifier as the base classifier. Since a CPDP task is a binary classification task, the binary cross entropy loss is applied to train the whole CPDP model, which can be written as (12), where $p_1(y | \mathbf{x})$ and $p_2(y | \mathbf{x})$ denote the output of the two classifiers over instance \mathbf{x} .

Step 2 Secondly, the discriminator (\mathcal{F}_1 and \mathcal{F}_2) is trained to find out those ambiguous target instances while the generator is fixed. Given a specific target instance, if one classifier categorizes it as negative while another classifiers think it as a positive instance, then this instance is more likely to be ambiguous. Therefore, in this step, we maximize the disagreement between two classifiers to find these target instances, that is,

$$\begin{aligned} & \min_{\mathcal{F}_1, \mathcal{F}_2} \mathcal{L}(\mathbf{X}_S, Y_S) - \lambda \mathcal{L}_{\text{dis}}(\mathbf{X}_T), \\ \mathcal{L}_{\text{dis}}(\mathbf{X}_T) &= \mathbb{E}_{\mathbf{x}_T \sim \mathbf{X}_T} [d(p_1(y | \mathbf{x}_T), p_2(y | \mathbf{x}_T))], \\ d(p_1(y | \mathbf{x}_T), p_2(y | \mathbf{x}_T)) &= (p_1^+ - p_2^+)^2 + (p_1^- - p_2^-)^2, \end{aligned} \tag{13}$$

where p_1^+/p_1^- and p_2^+/p_2^- denote the prediction probability of \mathcal{F}_1 and \mathcal{F}_2 for negative and positive class respectively, and $\lambda > 0$ is a weighting parameter for adjusting the influence of the discrepancy loss. The discrepancy loss equals to the expectation of the discrepancy over all target instances and we choose L_2 distance to calculate the discrepancy value of two classifiers.

Step 3 Finally, the generator is trained to create more discriminative features while the discriminator is fixed. When two classifiers do not update in this step, we are supposed to minimize the discrepancy above in order to generate discriminative features. Consider a target instance with the generated features. If the prediction results of \mathcal{F}_1 and \mathcal{F}_2 are the same, then this instance is unambiguous and more likely to be classified correctly. We formulate the objective of this step as follows.

$$\min_{\mathcal{G}} \mathcal{L}_{\text{dis}}(\mathbf{X}_T). \tag{14}$$

Ideally, the feature distribution of target data is well-aligned with the source after this step. As a result, the discriminator is able to achieve comparable performance to the source project data over target instances.

We repeat these three steps in training phase. Due to that classifiers can correctly categorize source samples (Step 1), we train them to detect desired target instances (Step 2) and force the generator to extract more discriminative features (Step 3). The pseudo-code of the proposed method ADA is described in Algorithm 1. Its approximate time complexity can be given as $\mathcal{O}(n \times N \times T)$, where n is the number of instances in source project, N is the number of neurons in LSTM network and T is the maximum training iteration. By applying the adversarial training method, we can improve the final prediction performance.

Algorithm 1 ADA training algorithm.

Required: Source project labeled data $\mathbf{X}_S = \{\mathbf{x}_{S_i}\}_{i=1}^m$, $Y_S = \{y_{S_i}\}_{i=1}^m$, target project unlabeled data $\mathbf{X}_T = \{\mathbf{x}_{T_i}\}_{i=1}^m$, parameters of two classifiers and generator $\theta_{\mathcal{F}_1}$, $\theta_{\mathcal{F}_2}$ and θ_G , learning rate η , maximum iteration T .

- 1: **procedure** ADA($\theta_{\mathcal{F}_1}$, $\theta_{\mathcal{F}_2}$, θ_G)
- 2: Initialize the parameters of two classifiers $\theta_{\mathcal{F}_1}$, $\theta_{\mathcal{F}_2}$ and generator θ_G randomly
- 3: **for** $t := 1$ *to* T **do**
- 4: /* Step 1 */
- 5: Compute the Step 1 loss \mathcal{L}_1 through (12)
- 6: $\theta_{\mathcal{F}_1} = \theta_{\mathcal{F}_1} - \eta \frac{\partial \mathcal{L}_1}{\partial \theta_{\mathcal{F}_1}}$
- 7: $\theta_{\mathcal{F}_2} = \theta_{\mathcal{F}_2} - \eta \frac{\partial \mathcal{L}_1}{\partial \theta_{\mathcal{F}_2}}$
- 8: $\theta_G = \theta_G - \eta \frac{\partial \mathcal{L}_1}{\partial \theta_G}$
- 9: /* Step 2 */
- 10: Compute the Step 2 loss \mathcal{L}_2 through (13)
- 11: $\theta_{\mathcal{F}_1} = \theta_{\mathcal{F}_1} - \eta \frac{\partial \mathcal{L}_2}{\partial \theta_{\mathcal{F}_1}}$
- 12: $\theta_{\mathcal{F}_2} = \theta_{\mathcal{F}_2} - \eta \frac{\partial \mathcal{L}_2}{\partial \theta_{\mathcal{F}_2}}$
- 13: /* Step 3 */
- 14: Compute the Step 3 loss \mathcal{L}_3 through (14)
- 15: $\theta_G = \theta_G - \eta \frac{\partial \mathcal{L}_3}{\partial \theta_G}$
- 16: Update learning rate η by Adam optimizer
- 17: **end for**
- 18: **end procedure**

Ensure: $\theta_{\mathcal{F}_1}$, $\theta_{\mathcal{F}_2}$ and θ_G

4 Experiment Setups

In this section, we describe our experimental setups in detail, including benchmark datasets, experimental settings, evaluation metrics, baseline methods, statistical analysis methods and research questions.

4.1 Benchmark Datasets

To assess the ADA model, we utilize two benchmark datasets, AEEEM (D’Ambros et al. 2010) and PROMISE (Jureczko and Madeyski 2010). These two datasets are readily available and widely used in recent CPDP research. The projects they contain are rather representative in the field of software engineering, which can better reflect the realities of defects in general software programs.

Table 1 5 projects chosen from the AEEEM dataset

Project name	Time period	#Instance	Defect rate
JDT	2005.01-2008.06	997	20.7%
PDE	2005.01-2008.09	1497	14.0%
Equinox	2005.01-2008.06	324	39.8%
Mylyn	2005.01-2009.03	1862	13.2%
Lucene	2005.01-2008.10	691	9.3%

Table 2 10 project chosen from PROMISE dataset

Project name	Project version	#Instance	Defect rate
Ant	1.7	745	22.3%
Camel	1.6	965	19.5%
Forrest	0.8	32	6.3%
Ivy	2.0	352	11.4%
Log4j	1.2	205	92.2%
Poi	3.0	442	63.6%
Synapse	1.2	256	33.6%
Velocity	1.6.1	229	34.1%
Xalan	2.7	909	98.8%
Xerces	1.4.4	588	74.3%

AEEM dataset consists of five open-source Java projects: Eclipse JDT Core (JDT), Eclipse PDE UI (PDE), Equinox framework (Equinox), Mylyn and Apache Lucene (Lucene). There are 61 metrics in it, including source code metrics, entropy-of-change metrics, entropy-of-source-code metrics, etc. Table 1 presents the main information of these projects.

PROMISE dataset is comprised of 48 releases of 15 public open-source projects, 27 releases of 6 proprietary projects and 17 releases of 17 academic projects. All of them are written in Java. In recent CPDP studies (Qiu et al. 2019d, b; Jin 2021), researchers prefer to use the open-source projects. In our work, we carefully select 10 projects (as shown in Table 2). PROMISE dataset consists of 20 handcrafted features (metrics), which mainly concentrate on the programs' complexity.

Under the conditions of the CPDP tasks, we need to select a source project and a target project. Therefore, we choose a project as the target project at first, and then treat the remaining projects as the source project respectively. In this way, we collect 20 and 90 project pairs in AEEM and PROMISE datasets, respectively. In the following sections, we conduct our experiments to perform the CPDP based on these project pairs.

4.2 Experimental Settings

There are many hyper parameters in the proposed ADA model. Different values of these parameters could have a different influence on the performance of defect prediction. We conduct the cross-validation analysis of these parameters to find the optimal ones. There are two main parameters in our model: the dimension of the embedding E and the penalty coefficient λ that balances the classification loss and the discrepancy loss in (13). We empirically set E to 48. More details about tuning λ are discussed in Section 6.2.

We implement the proposed ADA model with PyTorch (Paszke et al. 2019). The hidden state dimension of bi-directional LSTM is set to 256 and the batch size of training phase is set to 32. An extension to stochastic gradient descent, Adam Khatri and Singh (2017) is adopted to optimize the entire model. We set the momentum to 0.9 by default and initial learning rate to 0.001. The entire model is trained for 100 iterations. Our experimental environment is an Intel(R) Xeon(R) CPU E5-2618 at 2.20 GHz, 64 GB RAM and 8 GPU (NVIDIA 1080 Ti) of 80 GB memory server running Ubuntu 20.04.2 LTS.

4.3 Evaluation Metrics

To measure the proposed ADA method, we employ the following three metrics, namely F_1 measure, balanced accuracy and geometric mean (G-Mean) which are widely used in the CPDP research. In binary classification analysis, we often measure a classifier by a confusion matrix (Table 3). According to its true label and classification result, an instance can be TP (True Positive), FP (False Positive), TN (True Negative) or FN (False Negative). We can derive the metrics we use in this work by means of confusion matrix. Sensitivity (or recall) evaluates the effectiveness of the classifier on the positive class while specificity assesses negative one. Precision measures how precise a model is, which is a widely-used indicator, too. The definitions of these metrics are as follows.

$$\begin{aligned} \text{Sensitivity} &= \frac{TP}{TP + FN}, \\ \text{Specificity} &= \frac{TN}{TN + FP}, \\ \text{Precision} &= \frac{TP}{TP + FP}. \end{aligned} \quad (15)$$

F_1 Measure According to their definition, sensitivity measures the ratio of samples that are underreported whereas precision measures the ratio of misreported samples. However, these two metrics usually are in conflict. One classifier with high sensitivity is more likely to perform poorly on precision, and vice versa. Therefore, we use F_1 measure to balance these two metrics, which equals to the harmonic average of them,

$$\begin{aligned} F_1 &= \frac{2 \times \text{Sensitivity} \times \text{Precision}}{\text{Sensitivity} + \text{Precision}} \\ &= \frac{2TP}{2TP + FN + FP}. \end{aligned} \quad (16)$$

Balanced Accuracy The most commonly used metric of a balanced classification problem is accuracy, which evaluates the overall effectiveness of a model. Accuracy equals to the number of correctly classified instances divided by the total number of all instances,

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}. \quad (17)$$

Nevertheless, when the data is skewed and imbalanced, accuracy may not be an appropriate metric (Bekkar et al. 2013). In CPDP tasks, a classifier that predicts all sample as the majority class is able to perform well on accuracy. Thus, we propose to use balanced accuracy to measure the performance of CPDP model. Balanced accuracy equals to the average of

Table 3 Confusion matrix for binary classification tasks

	Classified positive	Classified negative
Actual positive	TP	FN
Actual negative	FP	TN

specificity and sensitivity, which comprehensively considers the performance of a classifier both in the majority and minority classes. Balanced accuracy is formulated as follows.

$$\begin{aligned} \text{Balanced Accuracy} &= \frac{1}{2} (\text{Sensitivity} \times \text{Specificity}) \\ &= \frac{\text{TP} \times \text{TN}}{2 (\text{TP} + \text{FN}) (\text{TN} + \text{FP})}. \end{aligned} \quad (18)$$

Geometric Mean The geometric mean (G-Mean) (Kubat and Matwin 1997) of the sensitivity and specificity is another metric used in imbalanced classification problem. G-Mean tries to maximize the performance of both the majority and minority classes, and keep them balanced at the same time, which is defined as:

$$\begin{aligned} \text{G-Mean} &= \sqrt{\text{Sensitivity} \times \text{Specificity}} \\ &= \sqrt{\frac{\text{TP} \times \text{TN}}{(\text{TP} + \text{FN}) (\text{TN} + \text{FP})}}. \end{aligned} \quad (19)$$

4.4 Baseline Models

In order to validate that the proposed ADA model is valid and effective for CPDP tasks, we conduct extensive comparison experiments to show whether our model can outperform other state-of-the-art methods. We select 9 baseline models, and summarize them briefly in Table 4.

Please note that we re-implement all the baseline methods except TCNN (whose source codes are available online) with PyTorch. All baseline methods are re-trained under roughly the same experimental settings for fair comparisons.

4.5 Statistical Analysis Methods

To show the statistical significant difference between two models, we adopt a non-parametric test, Wilcoxon signed-rank test (Wilcoxon 1945) at a confidence level of 95%, which is commonly used in other defect prediction research (Ryu et al. 2016; Wang et al. 2020; Li et al. 2019). Wilcoxon signed-rank test relaxes the constraints on data distribution which does not require data to follow any distribution including the normal distribution. At the confidence level of 95%, we say that two methods are statistically different from each other if the p -value is less than 0.05. When the p -value equals or is larger than 0.05, we conclude that the difference between two methods is not statistically significant.

Besides, we utilize a variant of Scott-Knott Effect Size Difference (ESD) test (Tantithamthavorn et al. 2016) to measure the effect size between two models and rank all compared models. The Scott-Knott ESD test is a comparison method based on the mean of data. It partitions the set of means into statistically different groups with non-negligible difference by a hierarchical clustering algorithm. Table 5 describes the meanings of different Scott-Knott ESD (denoted by d).

Table 4 Baseline models for comparison

Model	Summary	Extracted semantic features	Imbalanced learning	Feature distribution mismatch	Relationship between instances and boundary
LR	Logistic Regression (LR) classifier is a classic SDP method, which only takes handcrafted features as input. LR is a simple but important baseline model. Plenty of researchers, including us, adopt LR as the base classifier.	No	No	No	No
TCA+	TCA+ (Nam et al. 2013) is a CPDP method based on Transfer Component Analysis (TCA) (Pan et al. 2010). TCA+ maps the source and target data into a high-dimensional space and calculated the geometric distance in the distribution. In the high-dimensional feature space, the two distributions are aligned by minimizing the distance.	No	No	Yes	No
TCNN	Transfer Convolutional Neural Network (TCNN) (Qiu et al. 2019d) uses the same data preprocessing approach to obtain the network input as we do. Additionally, they add a matching layer to reduce the divergence of two feature distributions. By minimizing the classification loss and distribution divergence, TCNN is trained for defect prediction under cross-project scenarios.	Yes	Yes	Yes	No
SemI	Liang et al. (2019) is an SDP method combining word embedding and deep learning techniques. They train an unsupervised word embedding model and leverage an LSTM network to extract the programs' semantic features. Then, they train a classifier that takes semantic features as input to predict defects in programs.	Yes	Yes	No	No
TPTL	Two-Phase Transfer Learning (TPTL) (Liu et al. 2019) is proposed to address the limitation of TCA+. Firstly, they adopt a project estimator to select two suitable projects by the estimation results. Secondly, they utilize TCA+ model to construct two prediction models and combine their results for final prediction.	No	No	Yes	No

Table 4 continued

Model	Summary	Extracted semantic features	Imbalanced learning	Feature distribution mismatch	Relationship between instances and boundary
DBN	Deep Belief Network (DBN) (Wang et al. 2020) is another deep learning method which can learn programs' semantic features. They propose a representation-learning algorithm to align the distributions of semantic and defect prediction features. Then, they build a classifier based on the proposed algorithm.	Yes	Yes	Yes	No
STF-NN	Stratification embedded in Nearest Neighbor (STF-NN) (Gong et al. 2020) is a CPDP model that utilizes an imbalance learning method based on Nearest Neighbor algorithm. They adopt TCA to bridge the gap between two feature distributions, and then conduct the imbalance learning method on transferred data for final defect prediction.	No	Yes	Yes	No
DMDA_JFR	DMDA_JFR (Zou et al. 2021) is a CPDP method which utilizes two auto-encoders to learn feature representations. One encoder captures the local features and the other one learns the global representations. For reducing distribution divergence, they introduce a pseudo-labels strategy and apply it into every training iteration.	No	Yes	Yes	No
MANN	Huang et al. (2021) propose a CPDP method by means of the multi-adaptation and nuclear norm (MANN), which adopts a multi-kernel MMD method to align feature distributions of different projects among multiple levels.	Yes	Yes	Yes	No

Table 5 Scott-Knott Effect Size Difference and corresponding effectiveness level

Effect Size Difference d	Effectiveness Level
$ d < 0.2$	Negligible
$0.2 \leq d < 0.5$	Small
$0.5 \leq d < 0.8$	Medium
$ d \geq 0.8$	Large

4.6 Research Questions

To assess the effectiveness and performance of the proposed ADA method, we discuss the following three research questions (RQs).

RQ1: Is our proposed ADA method better than other state-of-the-art CPDP models?

Motivation: To prove the effectiveness of the proposed ADA method, we need to make comparisons with other state-of-the-art CPDP models. We choose 9 baseline methods (Section 4.4), and compare the predictive performance of these baseline methods.

RQ2: How effective are the semantic features extracted by the generator that combines an LSTM network and attention mechanisms?

Motivation: To verify that our proposed feature generation model is more suitable for CPDP tasks, we need to confirm the effectiveness of our feature generation method.

RQ3: How effective is the adversarial domain adaptation method?

Motivation: Transfer learning is a powerful tool dealing with CPDP tasks. We need to verify the effectiveness of the proposed adversarial domain adaptation method compared to other transfer learning methods.

5 Experimental Results

In this section, we conduct extensive experiments and present the experimental and statistical results.

5.1 Rq1: Is our Proposed ADA Method Better than Other State-of-the-Art CPDP Models?

Tables 6, 7, 8, 9, 10 and 11 display the comparison results of the three evaluation metrics on PROMISE and AEEEM dataset, respectively.

On PROMISE dataset, there are 10 projects in this repository, which can form 90 project pairs. We can observe that, our proposed ADA method achieves 0.666, 0.722 and 0.713 on average in terms of F_1 measure, balanced accuracy and G-Mean, respectively. All of them are the best average values on the corresponding metrics compared to the other baseline methods. From the perspective of Win/Tie/Lose (W/T/L), our model wins 7 projects, ties 0 project and loses 3 projects on F_1 measure compared to TPTL method; and wins 9 projects, ties 0 project and loses 1 project on balanced accuracy compared to DMDA_JFR. To better illustrate that our method is better than other baseline model, we quantify the degree of improvement. Compared with DMDA_JFR method, ADA model attains average improvements of 12.36%, 5.01% and 3.94% in terms of F_1 measure, balanced accuracy and G-Mean, respectively; compared to DBN method, these improvements are 11.30%, 9.40% and 5.34%; compared to

Table 6 F_1 measure comparison results with 9 methods on PROMISE dataset

Target Project	LR	TCA+	TCNN	Seml	TPTL	DBN	STr-NN	DMDA_JFR	MANN	Ours
Ant	0.433	0.433	0.427	0.587	0.455	0.633	0.593	0.688	0.539	0.774
Camel	0.283	0.325	0.332	0.488	0.356	0.580	0.401	0.586	0.359	0.629
Forrest	0.385	0.473	0.553	0.620	0.604	0.722	0.636	0.713	0.749	0.777
Ivy	0.271	0.283	0.216	0.445	0.349	0.584	0.322	0.466	0.367	0.611
Log4j	0.584	0.604	0.458	0.485	0.606	0.452	0.554	0.517	0.662	0.495
Poi	0.610	0.626	0.677	0.576	0.787	0.636	0.682	0.512	0.672	0.685
Synapse	0.512	0.578	0.505	0.584	0.571	0.584	0.512	0.617	0.571	0.716
Velocity	0.472	0.473	0.521	0.583	0.568	0.602	0.606	0.676	0.554	0.639
Xalan	0.496	0.577	0.567	0.488	0.616	0.616	0.582	0.555	0.621	0.690
Xerces	0.307	0.262	0.294	0.530	0.690	0.579	0.619	0.600	0.391	0.649
Average	0.435	0.463	0.455	0.539	0.560	0.599	0.551	0.593	0.548	0.666
W/T/L	9/0/1	9/0/1	10/0/0	10/0/0	7/0/3	10/0/0	9/0/1	8/0/2	9/0/1	-
Improvement	53.09%	43.80%	46.45%	23.73%	18.96%	11.30%	20.87%	12.36%	21.50%	-
p-value	9.0E-3	9.0E-3	5.0E-3	5.0E-3	4.1E-2	5.0E-3	1.7E-2	1.3E-2	2.8E-2	-
ESD	2.222	1.806	1.243	1.776	0.983	0.896	1.179	0.889	1.044	-

The bold items indicate that the results are the best results on the corresponding target project data

Table 7 Balanced accuracy comparison results with 9 methods on PROMISE dataset

Target Project	LR	TCA+	TCNN	Seml	TPTL	DBN	STr-NN	DMDA_JFR	MANN	Ours
Ant	0.414	0.450	0.475	0.608	0.506	0.742	0.633	0.773	0.583	0.784
Camel	0.318	0.397	0.431	0.514	0.473	0.615	0.468	0.701	0.463	0.661
Forrest	0.409	0.488	0.563	0.643	0.654	0.661	0.695	0.740	0.738	0.776
Ivy	0.291	0.328	0.411	0.478	0.471	0.733	0.414	0.675	0.512	0.828
Log4j	0.518	0.548	0.513	0.518	0.513	0.527	0.576	0.503	0.534	0.556
Poi	0.596	0.633	0.687	0.601	0.713	0.649	0.777	0.708	0.689	0.713
Synapse	0.541	0.539	0.571	0.554	0.609	0.693	0.563	0.679	0.626	0.726
Velocity	0.586	0.491	0.564	0.581	0.633	0.651	0.624	0.706	0.623	0.739
Xalan	0.470	0.610	0.744	0.554	0.735	0.716	0.610	0.737	0.711	0.775
Xerces	0.287	0.383	0.403	0.578	0.648	0.609	0.641	0.650	0.558	0.658
Average	0.443	0.487	0.536	0.563	0.596	0.660	0.600	0.687	0.604	0.722
W/T/L	10/0/0	10/0/0	10/0/0	10/0/0	9/1/0	10/0/0	8/0/2	9/0/1	10/0/0	-
Improvement	62.92%	48.29%	34.61%	28.19%	21.18%	9.40%	20.33%	5.01%	19.54%	-
p-value	5.0E-3	5.0E-3	5.0E-3	5.0E-3	8.0E-3	5.0E-3	2.2E-2	3.7E-2	5.0E-3	-
ESD	2.775	2.617	1.886	2.399	1.415	0.852	1.313	0.449	1.388	-

The bold items indicate that the results are the best results on the corresponding target project data

Table 8 G-Mean comparison results with 9 methods on PROMISE dataset

Target Project	LR	TCA+	TCNN	Seml	TPTL	DBN	STr-NN	DMDA_JFR	MANN	Ours
Ant	0.449	0.490	0.483	0.598	0.522	0.759	0.593	0.773	0.615	0.782
Camel	0.303	0.418	0.426	0.535	0.467	0.682	0.440	0.702	0.499	0.717
Forrest	0.388	0.509	0.551	0.616	0.743	0.719	0.618	0.728	0.672	0.751
Ivy	0.475	0.364	0.444	0.481	0.536	0.739	0.351	0.675	0.548	0.817
Log4j	0.481	0.513	0.500	0.529	0.524	0.495	0.576	0.493	0.568	0.519
Poi	0.597	0.662	0.688	0.636	0.584	0.732	0.673	0.709	0.654	0.674
Synapse	0.539	0.615	0.582	0.609	0.675	0.661	0.512	0.680	0.692	0.720
Velocity	0.442	0.467	0.516	0.591	0.701	0.700	0.635	0.707	0.647	0.720
Xalan	0.502	0.552	0.705	0.638	0.666	0.693	0.626	0.738	0.726	0.753
Xerces	0.413	0.403	0.396	0.604	0.628	0.588	0.629	0.654	0.583	0.676
Average	0.459	0.499	0.529	0.584	0.605	0.677	0.565	0.686	0.620	0.713
W/T/L	10/0/0	10/0/0	9/0/1	9/0/1	9/0/1	9/0/1	9/0/1	9/0/1	9/0/1	-
Improvement	55.40%	42.78%	34.76%	22.17%	17.92%	5.34%	26.19%	3.94%	14.92%	-
p-value	5.0E-3	5.0E-3	7.0E-3	7.0E-3	7.0E-3	2.8E-2	1.3E-2	4.7E-2	1.7E-2	-
ESD	3.122	2.444	1.965	1.900	1.253	0.448	1.609	0.344	1.216	-

The bold items indicate that the results are the best results on the corresponding target project data

Table 9 F_1 measure comparison results with 9 methods on AEEEM dataset

Target Project	LR	TCA+	TCNN	Semi	TPTL	DBN	STr-NIN	DMDA_JFR	MANN	Ours
JDT	0.392	0.477	0.586	0.590	0.583	0.630	0.538	0.593	0.615	0.654
PDE	0.491	0.488	0.513	0.601	0.679	0.742	0.655	0.646	0.681	0.736
Equinox	0.252	0.324	0.487	0.580	0.544	0.492	0.418	0.563	0.562	0.600
Mylyn	0.378	0.455	0.508	0.588	0.692	0.664	0.586	0.706	0.650	0.717
Lucene	0.467	0.479	0.499	0.612	0.608	0.607	0.627	0.622	0.655	0.637
Average	0.396	0.445	0.519	0.594	0.621	0.627	0.565	0.626	0.625	0.669
W/T/L	5/0/0	5/0/0	5/0/0	5/0/0	5/0/0	4/0/1	5/0/0	5/0/0	4/0/1	-
Improvement	68.98%	50.43%	28.96%	12.57%	7.67%	6.70%	18.41%	6.86%	7.01%	-
p-value	4.3E-2	4.3E-2	4.3E-2	4.3E-2	4.3E-2	3.9E-2	4.3E-2	4.3E-2	3.9E-2	-
ESD	3.524	3.569	3.092	1.821	0.794	0.551	1.351	0.771	0.804	-

The bold items indicate that the results are the best results on the corresponding target project data

Table 10 Balanced accuracy comparison results with 9 methods on AEEEM dataset

Target Project	LR	TCA+	TCNN	Seml	TPTL	DBN	STr-NN	DMDA_JFR	MANN	Ours
JDT	0.412	0.477	0.566	0.593	0.652	0.706	0.624	0.689	0.690	0.687
PDE	0.488	0.482	0.499	0.586	0.584	0.501	0.566	0.491	0.581	0.590
Equinox	0.300	0.360	0.484	0.601	0.571	0.582	0.481	0.571	0.603	0.648
Mylyn	0.399	0.434	0.466	0.601	0.697	0.630	0.572	0.673	0.628	0.708
Lucene	0.486	0.489	0.507	0.533	0.586	0.552	0.523	0.520	0.570	0.636
Average	0.417	0.448	0.504	0.583	0.618	0.594	0.553	0.589	0.614	0.654
W/T/L	5/0/0	5/0/0	5/0/0	5/0/0	5/0/0	4/0/1	5/0/0	4/0/1	4/0/1	-
Improvement	56.82%	45.85%	29.65%	12.25%	5.83%	10.05%	18.26%	11.08%	6.45%	-
p-value	4.3E-2	4.3E-2	4.3E-2	4.3E-2	4.3E-2	4.0E-2	4.3E-2	4.0E-2	4.0E-2	-
ESD	3.727	4.100	3.548	1.855	0.712	0.930	2.012	0.917	0.841	-

The bold items indicate that the results are the best results on the corresponding target project data

Table 11 G-Mean comparison results with 9 methods on AEEEM dataset

Target Project	LR	TCA+	TCNN	Seml	TPTL	DBN	STr-NN	DMDA_JFR	MANN	Ours
JDT	0.387	0.505	0.533	0.605	0.636	0.651	0.618	0.604	0.624	0.709
PDE	0.471	0.415	0.582	0.622	0.618	0.578	0.549	0.592	0.555	0.660
Equinox	0.392	0.383	0.543	0.596	0.600	0.597	0.637	0.617	0.572	0.685
Mylyn	0.367	0.400	0.561	0.545	0.629	0.696	0.642	0.637	0.629	0.724
Lucene	0.447	0.478	0.550	0.600	0.638	0.557	0.568	0.650	0.645	0.626
Average	0.413	0.436	0.554	0.594	0.624	0.616	0.603	0.620	0.605	0.681
W/T/L	5/0/0	5/0/0	5/0/0	5/0/0	4/0/1	5/0/0	5/0/0	4/0/1	4/0/1	-
Improvement	65.02%	56.07%	22.96%	14.69%	9.08%	10.61%	12.94%	9.83%	12.54%	-
p-value	4.3E-2	4.3E-2	4.3E-2	4.3E-2	3.9E-2	4.3E-2	4.3E-2	3.9E-2	3.9E-2	-
ESD	6.437	5.277	4.141	2.536	2.559	1.333	1.928	1.882	1.444	-

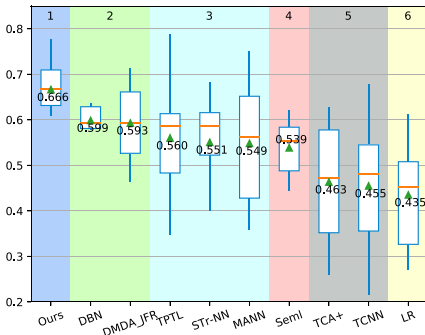
The bold items indicate that the results are the best results on the corresponding target project data

TPTL method, these improvements are 18.96%, 21.18% and 17.92%; compared to STr-NN method, these improvements are 20.87%, 20.33% and 26.19%. In the aspect of p -value given by Wilcoxon signed-rank test, all p -values are less than 0.05, meaning that the differences compared to ADA method are statistically significant at the 95% confidence level. From the perspective of Scott-Knott ESD test, all ESDs are larger than 0.8 on the metric of F_1 measure, which implies that the differences are large according to the Scott-Knott ESD effectiveness level as described in Table 5. On the metric of balanced accuracy, the ESD of DMDA_JFR is 0.449, indicating that the effectiveness level is small. The same is true for DBN and DMDA_JFR on the metric of G-Mean. Notably, our method achieves 0.495 in project Log4j on the metric of F_1 measure, which is only better than 3 other baseline methods. From Table 2 we can see that the defect rate of Log4j is 92.2%, resulting an imbalanced data distribution over positive samples. Even though we apply an imbalanced learning method to alleviate this problem, we may still get poor performance if there are insufficient data. This may be the reason why we perform poorly in this project. Consider another imbalanced project, Xalan, with 98.8% defect rate. Since there are far more samples (909 instances) in Xalan project, our method performs best among baseline models. Therefore, if there are ample samples in a project, ADA is able to attain considerable performance even though they are imbalanced. On AEEEM dataset, our model achieves best performance among baseline methods, too. From these tables we can observe that our method achieves 0.669, 0.654, 0.681 on average on the metrics of F_1 measure, balanced accuracy and G-Mean, respectively. Similarly, all of them are the best average values among all comparison methods. Experimental and statistical results on AEEEM dataset also indicate the same fact.

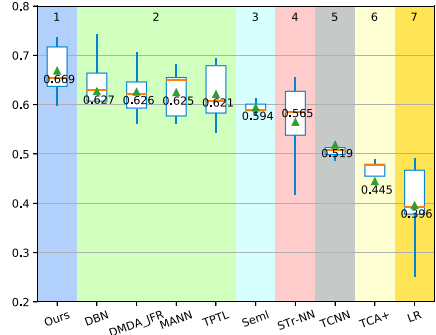
In order to intuitively show the differences among these methods, we draw box charts based on the Scott-Knott ESD test results, as shown in Fig. 4. These methods are ranked and grouped according to the results of Scott-Knott ESD test, and the methods in the same color box have little difference in performance. From the figures we can observe that both the medians and the means of our model on three different metrics are higher than other baseline models on two datasets. In the aspect of rankings, the proposed ADA model ranks first in all three metrics. The second tie models are DMDA_JFR and DBN, which rank second and third respectively in terms of balanced accuracy and G-Mean metrics on PROMISE dataset. TPTL, STr-NN and MANN methods rank after these two methods. On AEEEM dataset, TPTL ranks second on the metrics of balanced accuracy and G-Mean. By elaborate and comprehensive observation on the experimental results, we can answer the RQ1: Our proposed ADA method is better than other state-of-the-art baseline models.

5.2 Rq2: How Effective are the Semantic Features Extracted by the Generator that Combines an LSTM Network and Attention Mechanisms?

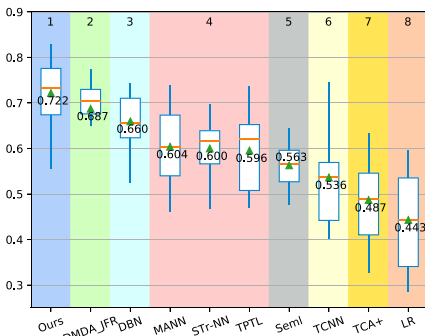
To answer the RQ2, we first compare different approaches adopted in CPDP research. Among the baseline models we choose, Seml Liang et al. (2019) uses an LSTM network to extract the semantic features of the programs while TCNN (Qiu et al. 2019d) utilizes CNN as the base feature generator. These two methods adopt roughly the same data preprocessing techniques, and train the model to perform CPDP tasks. From the empirical point of view, Seml performs better than TCNN. Referring to Fig. 4, Seml ranks sixth while TCNN ranks seventh or eighth, respectively in terms of F_1 measure, balanced accuracy and G-Mean on PROMISE dataset. As we discussed in Section 1, we are more inclined to consider the CPDP problem as an NLP problem since the programming language is a kind of standard, formal languages that follow certain language paradigms. Therefore, recurrent neural networks, which are good at



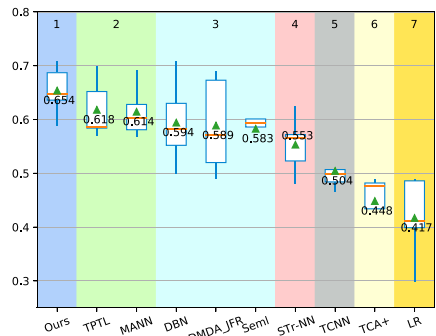
(a) F_1 measure on PROMISE dataset.



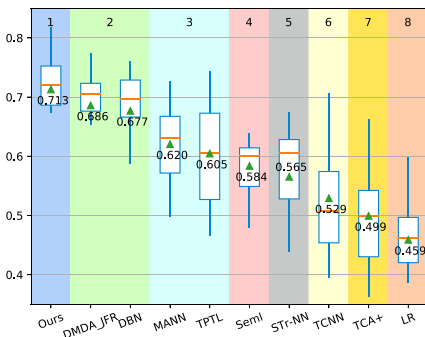
(b) F_1 measure on AEEEM dataset.



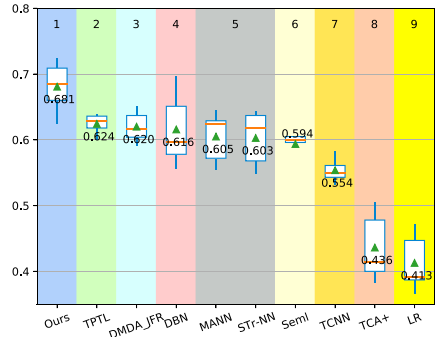
(c) Balanced accuracy on PROMISE dataset.



(d) Balanced accuracy on AEEEM dataset.



(e) G-Mean on PROMISE dataset.



(f) G-Mean on AEEEM dataset.

Fig. 4 Box charts of three evaluation metrics on PROMISE dataset (left-hand side) and AEEEM dataset (right-hand side). We eliminate the fliers for simplicity, so are that with the following box charts. These methods are ranked and grouped according to the Scott-Knott ESD test results. The orange lines represent the methods' average medians and the green triangles stand for the methods' means annotated by values. Methods in the same color box are grouped into the same cluster in Scott-Knott ESD test

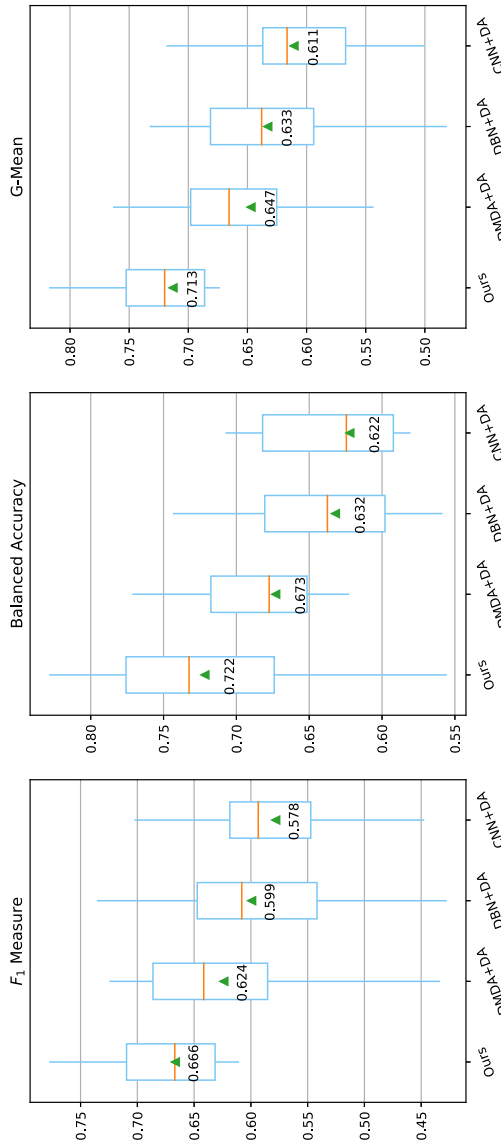


Fig. 5 The box charts of different feature generation models in terms of three evaluation metrics on PROMISE dataset. The orange lines represent the methods' average medians and the blue triangles stand for the methods' means annotated by values

Table 12 Ablation study of different components on the metric of F_1 measure on PROMISE dataset

Target Project	No-IL	No-HF	No-AM	No-DA	ADA
Ant	0.433	0.653	0.702	0.522	0.774
Camel	0.419	0.544	0.524	0.472	0.629
Forrest	0.367	0.681	0.661	0.310	0.777
Ivy	0.282	0.501	0.533	0.383	0.611
Log4j	0.193	0.453	0.413	0.248	0.495
Poi	0.622	0.576	0.623	0.424	0.685
Synapse	0.628	0.648	0.679	0.491	0.716
Velocity	0.512	0.546	0.601	0.363	0.639
Xalan	0.273	0.618	0.630	0.511	0.690
Xerces	0.422	0.533	0.600	0.502	0.649
Average	0.415	0.575	0.597	0.423	0.666
Improvement	60.57%	15.84%	11.67%	57.67%	-
p-value	2.53E-3	2.53E-3	2.53E-3	2.53E-3	-
ESD	2.135	1.164	0.826	2.751	-

The bold items indicate that the results are the best results on the corresponding target project data

processing sequential data, may perform better than other deep learning methods like CNN and DBN. The comparisons between SemI and TCNN, DBN and ADA partly prove this point of view.

To make this point more convincing, we further conduct a comparison experiment. We use different feature generator models, including CNN (adopted in TCNN (Qiu et al. 2019d)), DBN (adopt in Ref. Wang et al. (2020)), Double Marginalized Denoising Auto-Encoders (DMDA, adopted in DMDA_JFR (Zou et al. 2021)) and LSTM with AM (adopted in this paper) to extract the semantic features separately, and feed them into LR classifiers. We train these generators in the manner of adversarial learning, namely by means of adversarial domain adaptation method. We use the PROMISE dataset as it contains more projects than AEEEM repository. Figure 5 shows the performance differences between these feature generation models. Our model uses LSTM as the base model for generating features, which performs best. DMDA with DA ranks second among these methods and DBN or CNN with DA achieves the worst performance. Compared to DBN or CNN, DMDA uses two auto-encoders to capture the semantic features of program, which is also suitable for sequential data. DBN builds prediction model based on probability calculation and CNN utilizes convolution operation for feature extraction, which are better at dealing with computer vision tasks. We believe that this may be the main reason why our feature generation model outperforms other methods. We fully consider the context of CPDP problem, leveraging an LSTM network with AM to capture the semantic and contextual information contained in programs.

Based on the above analysis, we can answer the RQ2: Together with AM and LSTM networks, the feature generator of our proposed model is more effective than other approaches.

5.3 RQ3: How Effective is the Adversarial Domain Adaptation Method?

Tables 12, 13 and 14 and Fig. 6 show the results of ablation study. We examine the influence on the prediction performance of each component, including imbalanced learning techniques

Table 13 Ablation study of different components on the metric of balanced accuracy on PROMISE dataset

Target Project	No-IL	No-HF	No-AM	No-DA	ADA
Ant	0.392	0.643	0.732	0.488	0.784
Camel	0.431	0.590	0.583	0.392	0.661
Forrest	0.358	0.688	0.712	0.423	0.776
Ivy	0.289	0.700	0.538	0.482	0.828
Log4j	0.203	0.496	0.462	0.278	0.556
Poi	0.641	0.591	0.685	0.432	0.713
Synapse	0.664	0.666	0.711	0.423	0.726
Velocity	0.612	0.588	0.643	0.352	0.739
Xalan	0.298	0.629	0.692	0.500	0.775
Xerces	0.399	0.636	0.623	0.537	0.658
Average	0.429	0.623	0.638	0.431	0.722
Improvement	68.29%	15.90%	13.07%	67.50%	-
p-value	2.53E-3	2.53E-3	2.53E-3	2.53E-3	-
ESD	2.326	1.409	1.000	3.726	-

The bold items indicate that the results are the best results on the corresponding target project data

(IL), handcrafted features (HF), attention mechanisms (AM) and adversarial domain adaptation methods (DA). We now focus on adversarial domain adaptation methods (we discuss more details about ablation study in Section 6.1). No-DA stands for the model that the feature generator is not trained by means of adversarial domain adaptation. We train the generator in an SDP way, that is, we directly feed the output of the generator into the LR classifier without applying any transfer learning or domain adaptation method. From the tables we can see that, the performance drops significantly. On the metric of F_1 measure, ADA achieves

Table 14 Ablation study of different components on the metric of G-Mean on PROMISE dataset

Target Project	No-IL	No-HF	No-AM	No-DA	ADA
Ant	0.413	0.594	0.712	0.493	0.782
Camel	0.471	0.569	0.635	0.462	0.717
Forrest	0.377	0.653	0.662	0.452	0.751
Ivy	0.304	0.690	0.554	0.529	0.817
Log4j	0.245	0.485	0.413	0.352	0.519
Poi	0.628	0.575	0.612	0.466	0.674
Synapse	0.673	0.657	0.673	0.482	0.720
Velocity	0.653	0.571	0.669	0.392	0.720
Xalan	0.344	0.609	0.642	0.532	0.753
Xerces	0.428	0.597	0.597	0.492	0.676
Average	0.454	0.600	0.617	0.465	0.713
Improvement	57.17%	18.87%	15.57%	53.21%	-
p-value	0.00253	0.00253	0.00253	0.00253	-
ESD	2.141	1.604	1.161	3.544	-

The bold items indicate that the results are the best results on the corresponding target project data

0.666 on average while No-DA only achieves 0.423 on average, losing more than one third of the performance; on the metric of balance accuracy, ADA achieves 0.722 on average while No-DA only achieves 0.431 on average, dropping 37.45% performance; on the metric of G-Mean, ADA achieves 0.713 on average while No-DA only achieves 0.465, losing 34.78% performance. We can observe the obvious differences between ADA and No-DA in the box charts (Fig. 6).

To further validate how effective our proposed adversarial domain adaptation method is used in ADA model, we conduct comparison experiments with other transfer learning methods. We compare our domain adaptation method with the transfer learning method adopted in TCNN (Qiu et al. 2019d). TCNN leverages a standard CNN to extract the features of programs and add a matching layer to the CNN model and then embeds both source and target data representation into a reproducing kernel Hilbert space. Next, TCNN adopts Maximum Mean Discrepancy (MMD) to reduce the distribution divergence between the source and target projects. In the comparison experiments, we also add a same matching layer to our proposed feature generation model, trying to align the two distributions by means of MMD. Table 15 shows the experimental comparison results. The comparative model is denoted by MMD in the table and figure. From the table and figure, we can observe that ADA is better than MMD, which surpasses MMD by 15.29%, 9.34% and 8.23% respectively in terms of three metrics. Compared to MMD or other transfer learning methods adopted in CPDP (Jin 2021; Nam et al. 2013), ADA fully consider the relationship between the decision boundary and the target instances while reducing the distribution divergence, forcing the feature extraction model to generate more discriminative features. We believe this is the main reason why the adversarial domain adaptation method we use is more practical than others.

Through the above discussions, we are able to answer the RQ3: The adversarial domain adaptation method we utilize is the main reason accounting for the performance improvement. We can say that it is more effective than other transfer learning methods according to the experimental results.

6 Discussions

In this section, we further discuss the proposed ADA method with several questions, and talk about threats of validity of this work.

6.1 Why does ADA Work?

From the empirical point of view, the experimental and statistical results on two benchmark datasets of 15 software releases witness that our proposed ADA method is generally superior to other baseline methods in terms of three different evaluation metrics including F_1 measure, balanced accuracy and G-Mean. In this section, we discuss the effectiveness of ADA in aspects of problem modeling and ablation study.

From the perspective of problem modeling, we believe our model is superior to other models for two main results. Firstly, we exploit the semantic and contextual features of programs in a more appropriate way. Because project source code is a kind of standardized, formalized language that follows certain specifications, we believe that the CPDP problem is more in line with the context of NLP. Therefore, we extend an LSTM network with attention mechanism as the feature generator since it is better at coping with sequential data

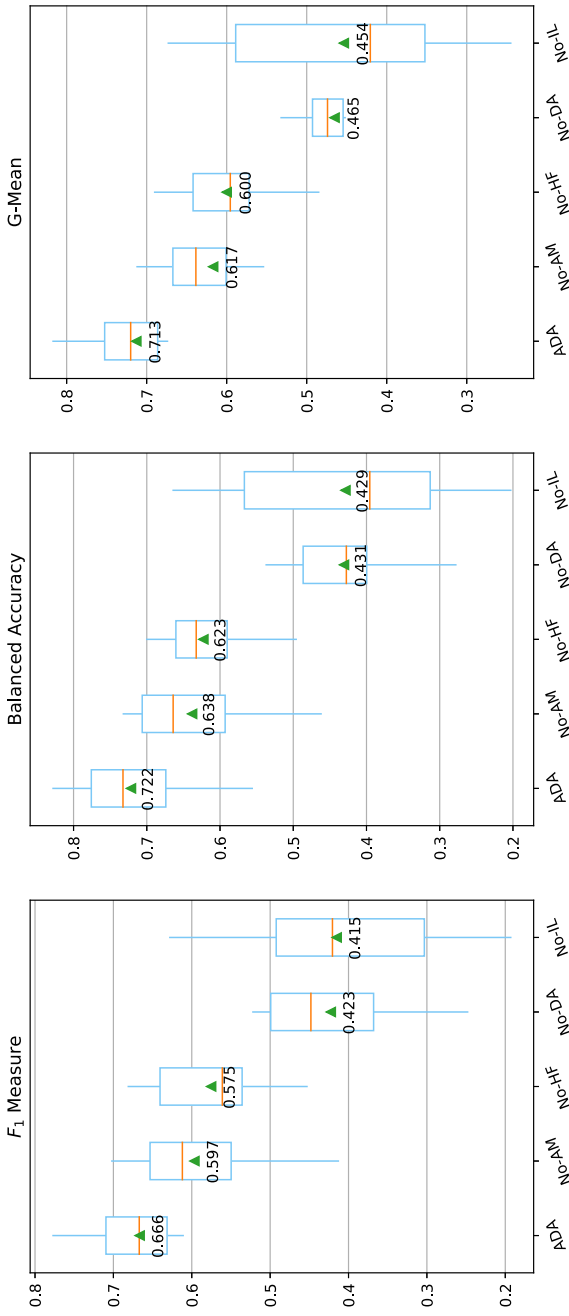


Fig. 6 The box charts of ablation study results in terms of three evaluation metrics on PROMISE dataset. The orange lines represent the methods' average medians and the blue triangles stand for the methods' means annotated by values

Table 15 Comparison results with MMD transfer learning method

Metric	MMD	ADA	Improvement
Avg. F1-Measure	0.578	0.666	15.29%
Avg. Balanced Accuracy	0.660	0.722	9.34%
Avg. G-Mean	0.659	0.713	8.23%

The bold items indicate that the results are the best results on the corresponding target project data

like programming languages. AM enables us to assign different weights to distinct tokens. Just like natural language in daily life, the importance of each word in a single sentence is different, and we believe that the contribution of different tokens in a program to software defects is also different. In this way, the feature extraction model we use can produce better semantic features for prediction. Secondly, we take an effective measure to transfer knowledge learned from the source project and apply it to the target project. Previous CPDP methods (Qiu et al. 2019d; Ryu et al. 2017; Liu et al. 2019) utilized transfer learning methods to reduce the divergence between the feature distributions of different projects. Though simple and effective, most of them might fail to consider the substantial information about feature space. In the proposed method, we adopt an adversarial domain adaptation method. By training the feature generator and defect predictor in the manner of adversarial learning, we reduce the divergence of the two feature distributions of the source and target project and fully take the relationship between the decision boundary and target project instance into consideration. As described in Section 3.3, we train the whole model in three steps. In order to observe how the adversarial learning method works, we investigate the trend in the value of the discrepancy at each iteration. Take project pair Xerces \rightarrow Ivy (i.e., Xerces is the source project and Ivy is the target project) as an example. Concretely, we record the discrepancy value between two classifiers (denoted by d_1) as described in (13), and the difference loss of the feature generator (denoted by d_2) as described in (14) when two classifiers are fixed. d_1 reflects the degree of disagreement of the predictions of the two classifiers on the target project data. If the value is large, it means that the current target instance is near the class boundaries and considered as ambiguous one. By maximizing d_1 , the classifiers are more likely to distinguish these ambiguous target instances and correctly classify non-ambiguous instances. Once we know about this relationship between decision boundaries and target instances, we can tell the generator not to generate such ambiguous instances. d_2 reflects the difference loss of the feature generator on the target project. The smaller the value is, it means that the features generated by the feature generator can be more accurately predicted by the predictor. We improve the discriminability of the features generated by the feature generator by minimizing the difference loss d_2 , so that the features of ambiguous instances are generated in regions far away from the decision boundary. We repeat this procedure in an adversarial training manner until convergence. We draw a line chart to show the trends of d_1 , d_2 and the loss value in (12), as shown in Figs. 7 and 8. From the line charts we can see that d_1 is increasing and d_2 is decreasing in general, which is in line with our expectation. In this example, the loss is stabilized and does not decrease any more after 15 iterations, meaning that the model is done training and our algorithm has converged.

In order to examine the effectiveness of each component of the proposed model, we design four ablation models. The first model removes the imbalanced learning techniques (No-IL) described in Section 3.2. The second model only takes semantic features extracted by the generator to train the prediction model without combining the handcrafted features. We define the second model as No-HF. The third model erases the attention mechanisms

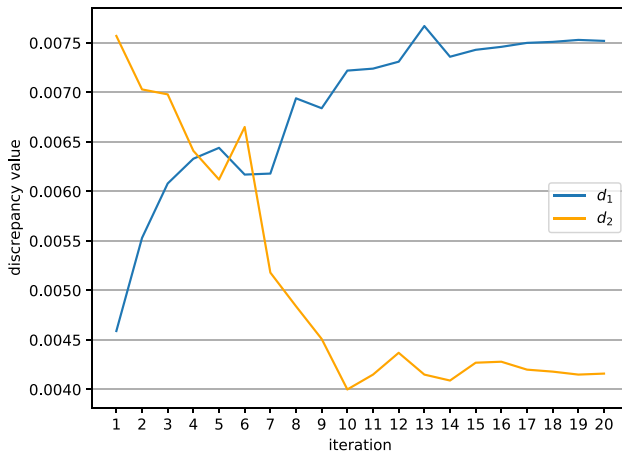


Fig. 7 The changes of discrepancy d_1 and d_2

adopted in the generator, namely assigning the same weights to different tokens. The third model is called No-AM. The final model is designed without adversarial domain adaptation methods (No-DA). We train both the feature generator and a LR classifier (as the prediction model) on source project data, and directly apply the model to the target project data without matching the distribution difference between them. Tables 12-14 show the experimental results of these models on the metrics of F_1 measure, balanced accuracy and G-Mean. Figure 6 visualizes the table results. We can draw some conclusions based on the observation from the tables and figures. Firstly, the performance of our model drops significantly if we do not apply techniques to tackle the data imbalance problem. Compared to ADA method, the performance of No-IL drops from 0.666 to 0.415, from 0.722 to 0.429 and from 0.713 to 0.454 in terms of three metrics. Secondly, handcrafted features and attention mechanisms play important roles in generating crucial features for software prediction. On average, handcrafted

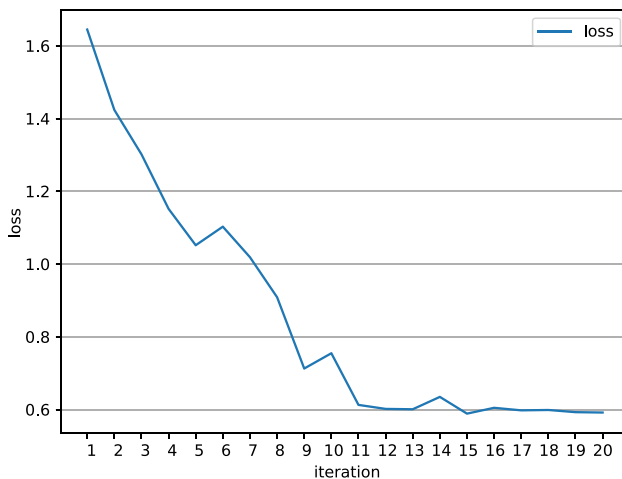


Fig. 8 The change of loss value

features can improve the performance by 15.84%, 15.90% and 18.87% on the three metrics. By assigning more weights to important tokens and less weights to irrelevant ones, attention mechanisms can bring 11.67%, 13.07% and 15.57% improvement to the ADA model in the three performance indicators respectively. Last but not least, adversarial domain adaptation plays a significant role in defect prediction. If we do not take any actions on transferring knowledge learned from the source projects and utilizing it in target projects, we could suffer from poor performance in all aspects. By adopting the domain adaptation method, we can enhance the performance from 0.423 to 0.666, from 0.431 to 0.722 and from 0.465 to 0.713 in terms of F_1 measure, balanced accuracy and G-Mean.

6.2 How does the Hyper Parameter λ Affect the Performance of ADA?

Hyper parameter λ controls the weight ratio between the classification loss of the prediction model and the discrepancy loss between two classifiers in (13). If λ is small, we tend to think less of the discrepancy loss and put more emphasis on the classification loss; if λ is large, we care more about the discrepancy loss and take less of the classification into consideration. We set $\lambda = 0.8$ in our research through cross-validation experiments. We select 14 different values of λ (from 0.1 to 1.3 with step 0.1), and see how they influence the prediction performance of the ADA model.

Figure 9 shows the experimental results of the performance of ADA with the selected 14 λ values in terms of F_1 measure, balanced accuracy and G-Mean on PROMISE dataset. From the figure, we can draw the following two conclusions. First, when λ is relatively small (less than 0.7), the performance on the three metrics is quite low. This is consistent with our early discussions. If we think less of the discrepancy, we may not be able to maximize the discrepancy loss between two classifiers, and thereby they could not distinguish ambiguous target samples. Thus, the ADA is less effective. Second, when λ is in the interval $[0.8, 1.0]$, the performance of ADA is the best. If λ is larger than 1.0, the performance starts to drop. Based on these observations, we choose $\lambda = 0.8$. Note that other hyper parameters in the ADA model are also selected in this way.

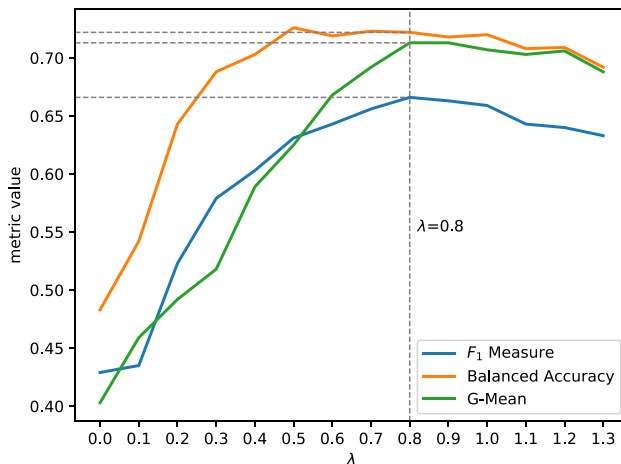


Fig. 9 Experimental results of the three metric values of ADA with different λ values on PROMISE dataset

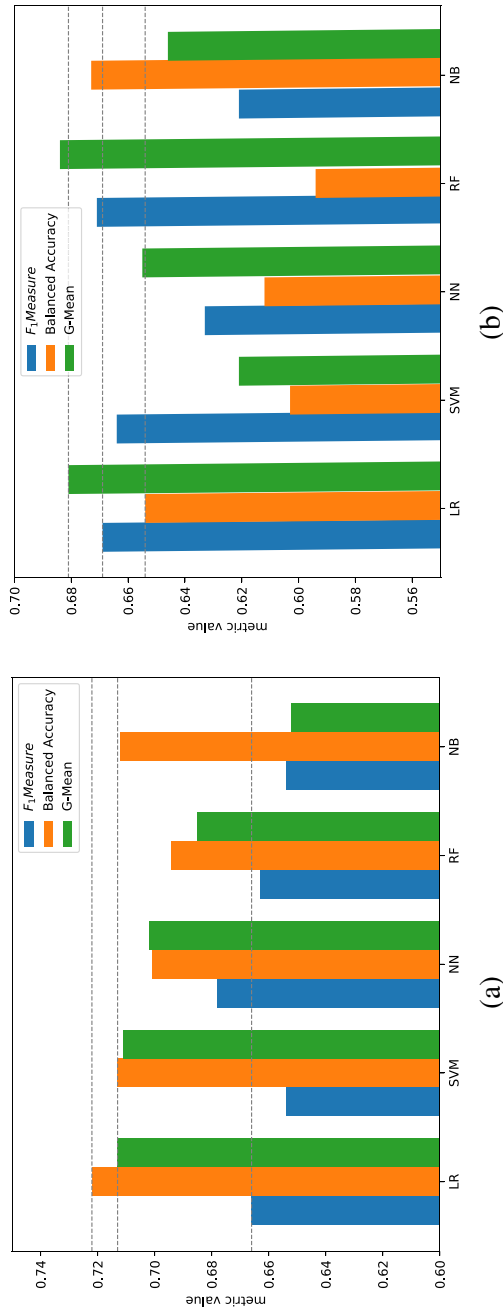


Fig. 10 Experimental results of ADA model with five different classifiers on the three evaluation metrics on PROMISE (left) and AEEEM (right) dataset

6.3 How Different Classifiers Affect the Performance of ADA?

We talk about why we choose the Logistic Regression classifier as the base classification model in this subsection. To evaluate the impact of different classifiers on the prediction performance, we choose five classic machine learning classifiers, including Logistic Regression (LR), Support Vector Machine (SVM), Neural Network (NN), Random Forrest (RF) and Naive Bayes (NB). For SVM, we use the Gaussian radial basis function as the kernel function. There is one hidden layer in NN, and the number of hidden neural is twice the input feature dimension.

Figure 10 displays the results of the performance of the proposed method with five different classifiers on the three evaluation metrics on PROMISE and AEEEM dataset. From the figure we can observe that the performance of these models is not very different. On PROMISE dataset, they all achieve the performance of about 0.65 in the F_1 metric; LR achieves the best performance among five models in terms of balanced accuracy and G-Mean. On AEEEM dataset, the performance of SVM, NN and RF in the balanced accuracy is relatively low compared to LR and NB; RF achieves comparable, even better performance to LR in terms of F_1 measure and G-Mean, but it fails to compete with LR on the metric of balanced accuracy.

To sum up, all kinds of classifiers we choose do not differ greatly in performance indicators. Among them, the Logistic Regression classifier is able to consistently achieve the best performance in terms of the three metrics on two benchmark datasets.

6.4 Threats to Validity

Threats to Construct Validity We carefully choose three commonly used performance metrics, including F_1 measure, balanced accuracy and G-Mean, as our evaluation criteria. F_1 measure equals to the harmonic average between the sensitivity and precision, seeking a balance between the underreporting rate and misreporting rate. Balanced accuracy and G-Mean are two commonly used metrics when the data distribution is imbalanced and skewed. However, these metrics might not be the only appropriate metrics. There are other measures (e.g., Area under the ROC (receiver operating characteristic) Curve (AUC) and Matthews Correlation Coefficient (MCC)) that can be used for performance measurement in binary classification tasks.

Threats to Internal Validity We implement some baseline models that are with open-access source code (such as TCNN) by utilizing the provided source code to reduce the potential impact of the improper implementations. For those models whose source code is not provided, we cautiously implement the models following the details described in the corresponding papers. However, our implementation might not fully reveal the details in the baseline methods. For a fair comparison, we apply consistent implementations of overlapped parts including data preprocessing, LR implementation, etc.

Threats to External Validity We try to reduce the bias by carefully selecting two public benchmark datasets of 15 open-source projects. These projects may not represent all software projects. Additionally, all selected projects are written in Java programming language. The results of our proposed method in some commercial software projects or projects that are written in other programming languages may be better or worse. Validating the effectiveness of ADA on more diversity and datasets with more projects is needed.

7 Conclusions

The main challenges of CPDP problem lie in how to construct more meaningful and contextual features that can represent programs and how to effectively transfer knowledge learned from source projects and apply it to target projects. In this paper, we proposed a novel ADA method to tackle these two challenges. For the feature generation, we extend a variant of Recurrent Neural Networks, Long Short-Term Memory networks with Attention Mechanism. Compared to other deep learning methods, LSTM is better at dealing with sequential data like project source codes. With attention mechanism, the feature generator can capture more important parts and further improve the prediction performance. Moreover, to effectively transfer knowledge learned from source projects to target projects, we propose an adversarial domain adaptation method to bridge the gap between two feature distributions. Besides, our proposed method takes full consideration of the relationship between the target instances and class boundary when aligning the distributions. We treat the feature extraction model as the generator, and train two classifiers as the discriminator. By training the whole model in the manner of adversarial learning, we first maximize the discrepancy of two classifiers over target samples to distinguish the ambiguous ones, and then train the generator to create more discriminative features according to the information about the relationship between the target instances and class boundary.

We conduct extensive experiments on two benchmark datasets of 15 open-source Java projects. The classification performance of ADA is measured on the evaluation metric of F_1 measure, balanced accuracy and G-Mean. We compare ADA with the state-of-the-art CPDP models by using Wilcoxon signed-rank test and Scott-Knott Effect Size Difference test. Experimental and statistical results show that ADA is effective and outperforms other baseline models by a significant margin.

There are several problems needed to be investigated in the future. Firstly, our proposed method may not generalize well on other datasets. We will conduct experiments on more projects and extend our method to other programming language to make ADA more generalizable. Secondly, transfer learning is a powerful tool in dealing with CPDP tasks. More precise and appropriate transfer learning methods which can align two or more feature distributions without losing the information of the source domain is needed to be explored in the future.

Acknowledgements This work was supported by the Science and Technology Planning Project of Guangzhou (Grant No. 202102020637).

Data Availability Statements The PROMISE dataset we use in this work is available at <https://doi.org/https://doi.org/10.1145/1868328.1868342>. The AEEEM dataset we use in this work is available at <https://doi.org/10.1109/MSR.2010.5463279>. The implementation of the proposed ADA model is available from the corresponding author on a reasonable request.

Declarations

Conflict of interest The authors declare that they have no conflict of interest.

References

Almeida F, Xexéo G (2019) Word embeddings: A survey

- Alon U, Zilberstein M, Levy O, Yahav E (2019) Code2vec: Learning distributed representations of code. Proc ACM Program Lang 3 (POPL). <https://doi.org/10.1145/3290353>
- Bal PR, Kumar S (2020) Wr-elm: Weighted regularization extreme learning machine for imbalance learning in software fault prediction. IEEE Trans Reliab 69(4):1355–1375. <https://doi.org/10.1109/TR.2020.2996261>
- Bal PR, Kumar S (2023) A data transfer and relevant metrics matching based approach for heterogeneous defect prediction. IEEE Trans Softw Eng 49(3):1232–1245. <https://doi.org/10.1109/TSE.2022.3173678>
- Balog M, Gaunt AL, Brockschmidt M, Nowozin S, Tarlow D (2017) Deepcoder: Learning to write programs. In: Proceedings of the 5th International Conference on Learning Representations (ICLR). <https://openreview.net/forum?id=ByldLrqlx>
- Balogun AO, Basri S, Capretz LF, Mahamad S, Imam AA, Almomani MA, Adeyemo VE, Kumar, G (2021) An adaptive rank aggregation-based ensemble multi-filter feature selection method in software defect prediction. Entropy 23(10). <https://doi.org/10.3390/e23101274>
- Bekkar M, Djemaa HK, Alitouche TA (2013) Evaluation measures for models assessment over imbalanced data sets. J Inf Eng Appl 3(10):27–38. https://eva.fing.edu.uy/pluginfile.php/69453/mod_resource/content/1/7633-10048-1-PB.pdf
- Borgwardt KM, Gretton A, Rasch MJ, Kriegel HP, Schölkopf B, Smola AJ (2006) Integrating structured biological data by kernel maximum mean discrepancy. Bioinformatics 22(14):e49–e57. <https://doi.org/10.1093/bioinformatics/btl242>
- Briand LC, Melo WL, Wust J (2002) Assessing the applicability of fault-proneness models across object-oriented software projects. IEEE Trans Softw Eng 28(7):706–720. <https://doi.org/10.1109/TSE.2002.1019484>
- Chawla NV, Bowyer KW, Hall LO, Kegelmeyer WP (2002) Smote: Synthetic minority over-sampling technique. J Artif Intell Res 16:321–357. <https://doi.org/10.1613/jair.953>
- Chen H, Jing XY, Li Z, Wu D, Peng Y, Huang Z (2021) An empirical study on heterogeneous defect prediction approaches. IEEE Trans Softw Eng 47(12):2803–2822. <https://doi.org/10.1109/TSE.2020.2968520>
- Chen L, Fang B, Shang Z, Tang Y (2015) Negative samples reduction in cross-company software defects prediction. Inf Softw Technol 62:67–77. <https://doi.org/10.1016/j.infsof.2015.01.014>
- Chen L, Li J, Peng J, Xie T, Cao Z, Xu K, He X, Zheng Z (2020) A survey of adversarial learning on graphs
- Chen X, Liu C, Shin R, Song D, Chen M (2016) Latent attention for if-then program synthesis. In: Proceedings of the 30th International Conference on Neural Information Processing Systems, NIPS'16, pp 4581–4589. Red Hook, NY, USA. <https://dl.acm.org/doi/pdf/10.5555/3157382.3157609>
- Chidamber SR, Kemerer CF (1994) A metrics suite for object oriented design. IEEE Trans Softw Eng 20(6):476–493. <https://doi.org/10.1109/32.295895>
- Compton R, Frank E, Patros P, Koay A (2020) Embedding java classes with code2vec: Improvements from variable obfuscation. In: Proceedings of the 17th International Conference on Mining Software Repositories, pp 243–253. Association for Computing Machinery, New York, NY, USA. <https://doi.org/10.1145/3379597.3387445>
- Dai W, Yang Q, Xue GR, Yu Y (2007) Boosting for transfer learning. In: Proceedings of the 24th International Conference on Machine Learning, ICML '07, pp 193–200. Association for Computing Machinery, New York, NY, USA. ewblock <https://doi.org/10.1145/1273496.1273521>
- D'Ambros M, Lanza M, Robbes R (2010) An extensive comparison of bug prediction approaches. In: 2010 7th IEEE Working Conference on Mining Software Repositories (MSR 2010), pp 31–41. <https://doi.org/10.1109/MSR.2010.5463279>
- Deng J, Lu L, Qiu S (2020) Software defect prediction via lstm. IET Software 14(4):443–450. <https://doi.org/10.1049/iet-sen.2019.0149>
- Gong L, Jiang S, Bo L, Jiang L, Qian J (2020) A novel class-imbalance learning approach for both within-project and cross-project defect prediction. IEEE Trans Reliab 69(1):40–54. <https://doi.org/10.1109/TR.2019.2895462>
- Goodfellow IJ, Pouget-Abadie J, Mirza M, Xu B, Warde-Farley D, Ozair S, Courville A, Bengio Y (2014) Generative adversarial nets. In: Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 2, NIPS'14, pp 2672–2680. MIT Press, Cambridge, MA, USA. <https://proceedings.neurips.cc/paper/2014/file/5ca3e9b122f61f8f06494c97b1afccf3-Paper.pdf>
- Halstead MH (1977) Elements of Software Science (Operating and Programming Systems Series). Elsevier Science Inc
- He H, Garcia EA (2009) Learning from imbalanced data. IEEE Trans Knowl Data Eng 21(9):1263–1284. <https://doi.org/10.1109/TKDE.2008.239>
- He P, Li B, Liu X, Chen J, Ma Y (2015) An empirical study on software defect prediction with a simplified metric set. Inf Softw Technol 59:170–190. <https://doi.org/10.1016/j.infsof.2014.11.006>

- He Z, Shu F, Yang Y, Li M, Wang Q (2012) An investigation on the feasibility of cross-project defect prediction. *Autom Softw Eng* 19(2):167–199. <https://doi.org/10.1007/s10515-011-0090-3>
- Herbold S (2013) Training data selection for cross-project defect prediction. In: Proceedings of the 9th International Conference on Predictive Models in Software Engineering, pp 1–10. <https://doi.org/10.1145/2499393.2499395>
- Herbold S, Trautsch A, Grabowski J (2018) A comparative study to benchmark cross-project defect prediction approaches. *IEEE Trans Softw Eng* 44(9):811–833. <https://doi.org/10.1109/TSE.2017.2724538>
- Hochreiter S, Schmidhuber J (1997) Long short-term memory. *Neural Comput* 9(8):1735–1780. <https://doi.org/10.1162/neco.1997.9.8.1735>
- Hosseini S, Turhan B, Gunarathna D (2019) A systematic literature review and meta-analysis on cross project defect prediction. *IEEE Trans Softw Eng* 45(2):111–147. <https://doi.org/10.1109/TSE.2017.2770124>
- Hosseini S, Turhan B, Mäntylä M (2018) A benchmark study on the effectiveness of search-based data selection and feature selection for cross project defect prediction. *Inf Softw Technol* 95:296–312. <https://doi.org/10.1016/j.infsof.2017.06.004>
- Huang J, Guan X, Li S (2021) Software defect prediction model based on attention mechanism. In: 2021 International Conference on Computer Engineering and Application (ICCEA), pp 338–345. IEEE. <https://doi.org/10.1109/ICCEA53728.2021.00073>
- Huang Q, Ma L, Jiang S, Wu G, Song H, Jiang L, Zheng C (2021) A cross-project defect prediction method based on multi-adaptation and nuclear norm. *IET Softw* pp 1–14. <https://doi.org/10.1049/sfw2.12053>
- Jin C (2021) Cross-project software defect prediction based on domain adaptation learning and optimization. *Expert Syst Appl* 171:114637. <https://doi.org/10.1016/j.eswa.2021.114637>
- Jing X, Wu F, Dong X, Qi F, Xu B (2015) Heterogeneous cross-company defect prediction by unified metric representation and cca-based transfer learning. In: Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering, pp 496–507. Association for Computing Machinery. <https://doi.org/10.1145/2786805.2786813>
- Jing XY, Wu F, Dong X, Xu B (2017) An improved sda based defect prediction framework for both within-project and cross-project class-imbalance problems. *IEEE Trans Softw Eng* 43(4):321–339. <https://doi.org/10.1109/TSE.2016.2597849>
- Jing XY, Ying S, Zhang ZW, Wu SS, Liu J (2014) Dictionary learning based software defect prediction. In: Proceedings of the 36th International Conference on Software Engineering, ICSE 2014, pp 414–423. Association for Computing Machinery, New York, NY, USA. <https://doi.org/10.1145/2568225.2568320>
- Jureczko M, Madeyski L (2010) Towards identifying software project clusters with regard to defect prediction. In: Proceedings of the 6th International Conference on Predictive Models in Software Engineering, pp 1–10. <https://doi.org/10.1145/1868328.1868342>
- Khatri Y, Singh SK (2021) Cross project defect prediction: A comprehensive survey with its swot analysis. *Innovations Syst Softw Eng* pp 1–19. <https://doi.org/10.1007/s11334-020-00380-5>
- Kingma DP, Ba J (2017) Adam: A method for stochastic optimization
- Kubat M, Matwin S (1997) Addressing the curse of imbalanced training sets: One-sided selection. In: In Proceedings of the 14th International Conference on Machine Learning, pp 179–186. Morgan Kaufmann. <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.43.4487&rep=rep1&type=pdf>
- LeCun Y, Bengio Y, Hinton G (2015) Deep learning. *Nature* 521(7553):436–444. <https://doi.org/10.1038/nature14539>
- Li H, Li X, Chen X, Xie X, Mu Y, Feng Z (2019) Cross-project defect prediction via astoken2vec and blstm-based neural network. In: 2019 International Joint Conference on Neural Networks (IJCNN), pp 1–8. <https://doi.org/10.1109/IJCNN.2019.8852135>
- Li J, He P, Zhu J, Lyu MR (2017) Software defect prediction via convolutional neural network. In: 2017 IEEE International Conference on Software Quality, Reliability and Security (QRS), pp 318–328. <https://doi.org/10.1109/QRS.2017.42>
- Li Y, Yuan L, Vasconcelos N (2019) Bidirectional learning for domain adaptation of semantic segmentation. In: 2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), pp 6929–6938. <https://doi.org/10.1109/CVPR.2019.00710>
- Li Z, Jing XY, Wu F, Zhu X, Xu B, Ying S (2018) Cost-sensitive transfer kernel canonical correlation analysis for heterogeneous defect prediction. *Autom Softw Eng* 25:201–245. <https://doi.org/10.1007/s10515-017-0220-7>
- Li Z, Jing XY, Zhu X (2018) Progress on approaches to software defect prediction. *IET Softw* 12(3):161–175. <https://doi.org/10.1049/iet-sen.2017.0148>
- Li Z, Jing XY, Zhu X, Zhang H, Xu B, Ying S (2019) Heterogeneous defect prediction with two-stage ensemble learning. *Autom Softw Eng* 26:599–651. <https://doi.org/10.1007/s10515-019-00259-1>

- Li Z, Jing XY, Zhu X, Zhang H, Xu B, Ying S (2019) On the multiple sources and privacy preservation issues for heterogeneous defect prediction. *IEEE Trans Softw Eng* 45(4):391–411. <https://doi.org/10.1109/TSE.2017.2780222>
- Liang H, Yu Y, Jiang L, Xie Z (2019) S eml: A semantic lstm model for software defect prediction. *IEEE Access* 7:83812–83824
- Liu C, Yang D, Xia X, Yan M, Zhang X (2019) A two-phase transfer learning model for cross-project defect prediction. *Inf Softw Technol* 107:125–136. <https://doi.org/10.1016/j.infsof.2018.11.005>
- Ma X, Mou X, Wang J, Liu X, Geng J, Wang H (2021) Cross-dataset hyperspectral image classification based on adversarial domain adaptation. *IEEE Trans Geosci Remote Sens* 59(5):4179–4190. <https://doi.org/10.1109/TGRS.2020.3015357>
- McCabe TJ (1976) A complexity measure. *IEEE Trans Softw Eng SE-2*(4):308–320. <https://doi.org/10.1109/TSE.1976.233837>
- Nam J, Kim S (2015) Heterogeneous defect prediction. In: *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*, pp 508–519. Association for Computing Machinery. <https://doi.org/10.1145/2786805.2786814>
- Nam J, Pan SJ, Kim S (2013) Transfer defect learning. In: *2013 35th International Conference on Software Engineering (ICSE)*, pp 382–391. <https://doi.org/10.1109/ICSE.2013.6606584>
- Ni C, Chen X, Wu F, Shen Y, Gu Q (2019) An empirical study on pareto based multi-objective feature selection for software defect prediction. *J Syst Softw* 152:215–238. <https://doi.org/10.1016/j.jss.2019.03.012>
- Pan SJ, Tsang IW, Kwok JT, Yang Q (2010) Domain adaptation via transfer component analysis. *IEEE Trans Neural Networks* 22(2):199–210. <https://doi.org/10.1109/TNN.2010.2091281>
- Pan SJ, Yang Q (2010) A survey on transfer learning. *IEEE Trans Knowl Data Eng* 22(10):1345–1359. <https://doi.org/10.1109/TKDE.2009.191>
- Pandey SK, Tripathi AK (2021) Dnnattention: A deep neural network and attention based architecture for cross project defect number prediction. *Knowl-Based Syst* 233:107541. <https://doi.org/10.1016/j.knosys.2021.107541>
- Paszke A, Gross S, Massa F, Lerer A, Bradbury J, Chanan G, Killeen T, Lin Z, Gimelshein N, Antiga L, Desmaison A, Kopf A, Yang E, DeVito Z, Raison M, Tejani A, Chilamkurthy S, Steiner B, Fang L, Bai J, Chintala S (2019) Pytorch: An imperative style, high-performance deep learning library. In: *Advances in Neural Information Processing Systems*, 32, pp 8024–8035. <https://proceedings.neurips.cc/paper/2019/file/bdbca288fee7f92f2bfa9f7012727740-Paper.pdf>
- Qiu S, Lu L, Cai Z, Jiang S (2019a) Cross-project defect prediction via transferable deep learning-generated and handcrafted features. In: *The 31st International Conference on Software Engineering and Knowledge Engineering*, pp 431–436. <https://doi.org/10.18293/SEKE2019-070>
- Qiu S, Lu L, Jiang S (2019) Joint distribution matching model for distribution-adaptation-based cross-project defect prediction. *IET Softw* 13(5):393–402. <https://doi.org/10.1049/iet-sen.2018.5131>
- Qiu S, Lu L, Jiang S, Guo Y (2019) An investigation of imbalanced ensemble learning methods for cross-project defect prediction. *Int J Pattern Recogn Artif Intell* 33(12):1959037. <https://doi.org/10.1142/S0218001419590377>
- Qiu S, Xu H, Deng J, Jiang S, Lu L (2019) Transfer convolutional neural network for cross-project defect prediction. *Appl Sci* 9(13):2660. <https://doi.org/10.3390/app9132660>
- Rathore SS, Kumar S (2021) An empirical study of ensemble techniques for software fault prediction. *Appl Int* 51:3615–3644. <https://doi.org/10.1007/s10489-020-01935-6>
- Ryu D, Choi O, Baik J (2016) Value-cognitive boosting with a support vector machine for cross-project defect prediction. *Empir Softw Eng* 21(1):43–71. <https://doi.org/10.1007/s10664-014-9346-4>
- Ryu D, Jang JI, Baik J (2017) A transfer cost-sensitive boosting approach for cross-project defect prediction. *Softw Qual J* 25(1):235–272. <https://doi.org/10.1007/s11219-015-9287-1>
- Saito K, Watanabe K, Ushiku Y, Harada T (2018) Maximum classifier discrepancy for unsupervised domain adaptation. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2018). <https://doi.org/10.1109/CVPR.2018.00392>
- Shepperd M, Bowes D, Hall T (2014) Researcher bias: The use of machine learning in software defect prediction. *IEEE Trans Softw Eng* 40(6):603–616. <https://doi.org/10.1109/TSE.2014.2322358>
- Song Q, Guo Y, Shepperd M (2019) A comprehensive investigation of the role of imbalanced learning for software defect prediction. *IEEE Trans Softw Eng* 45(12):1253–1269. <https://doi.org/10.1109/TSE.2018.2836442>
- Song S, Yu H, Miao Z, Fang J, Zheng K, Ma C, Wang S (2020) Multi-spectral salient object detection by adversarial domain adaptation. *Proceedings of the AAAI Conference on Artificial Intelligence* 34:12023–12030. <https://doi.org/10.1609/aaai.v34i07.6879>


- Su JC, Tsai YH, Sohn K, Liu B, Maji S, Chandraker M (2020) Active adversarial domain adaptation. In: 2020 IEEE Winter Conference on Applications of Computer Vision (WACV), pp 728–737. <https://doi.org/10.1109/WACV45572.2020.9093390>
- Tantithamthavorn C, McIntosh S, Hassan AE, Matsumoto K (2016) An empirical comparison of model validation techniques for defect prediction models. *IEEE Trans Softw Eng* 43(1):1–18. <https://doi.org/10.1109/TSE.2016.2584050>
- Thota MK, Shajin FH, Rajesh P (2020) Survey on software defect prediction techniques. *Int J Appl Sci Eng* 17:331–344. [https://doi.org/10.6703/IJASE.202012_17\(4\).331](https://doi.org/10.6703/IJASE.202012_17(4).331)
- Tong H, Liu B, Wang S, Li Q (2019) Transfer-learning oriented class imbalance learning for cross-project defect prediction
- Turhan B, Menzies T, Bener AB, Di Stefano J (2009) On the relative value of cross-company and within-company data for defect prediction. *Empir Softw Eng* 14(5):540–578. <https://doi.org/10.1007/s10664-008-9103-7>
- Tzeng E, Hoffman J, Saenko K, Darrell T (2017) Adversarial discriminative domain adaptation. In: 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp 2962–2971. <https://doi.org/10.1109/CVPR.2017.316>
- Vaswani A, Shazeer N, Parmar N, Uszkoreit J, Jones L, Gomez AN, Kaiser L, Polosukhin I (2017) Attention is all you need. In: Proceedings of the 31st International Conference on Neural Information Processing Systems, NIPS'17, pp 6000–6010. Curran Associates Inc., Red Hook, NY, USA. <https://proceedings.neurips.cc/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf>
- Veličković P, Cucurull G, Casanova A, Romero A, Lió P, Bengio Y (2018) Graph attention networks. In: Proceedings of the 6th International Conference on Learning Representations (ICLR). <https://openreview.net/forum?id=rJXMpikCZ>
- Wang K, Chen G, Huang Z, Wan X, Huang F (2021) Bridging the domain gap: Improve informal language translation via counterfactual domain adaptation. In: Proceedings of the AAAI Conference on Artificial Intelligence, 35, pp 13970–13978. <https://ojs.aaai.org/index.php/AAAI/article/view/17645>
- Wang S, Liu T, Nam J, Tan L (2020) Deep semantic feature learning for software defect prediction. *IEEE Trans Softw Eng* 46(12):1267–1293. <https://doi.org/10.1109/TSE.2018.2877612>
- Wang S, Liu T, Tan L (2016) Automatically learning semantic features for defect prediction. In: 2016 IEEE/ACM 38th International Conference on Software Engineering (ICSE), pp 297–308. <https://doi.org/10.1145/2884781.2884804>
- Watanabe S, Kaiya H, Kaijiri K (2008) Adapting a fault prediction model to allow inter languagereuse. In: Proceedings of the 4th International Workshop on Predictor Models in Software Engineering, PROMISE '08, pp 19–24. Association for Computing Machinery, New York, NY, USA. <https://doi.org/10.1145/1370788.1370794>
- Wilcoxon F (1945) Individual comparisons by ranking methods. *Biom Bull* 1(6):80–83. <https://doi.org/10.2307/3001968>
- Wu F, Jing XY, Sun Y, Sun J, Huang L, Cui F, Sun Y (2018) Cross-project and within-project semisupervised software defect prediction: A unified approach. *IEEE Trans Reliab* 67(2):581–597. <https://doi.org/10.1109/TR.2018.2804922>
- Xu Z, Pang S, Zhang T, Luo XP, Liu J, Tang YT, Yu X, Xue L (2019) Cross project defect prediction via balanced distribution adaptation based transfer learning. *J Comput Sci Technol* 34(5):1039–1062. <https://doi.org/10.1007/s11390-019-1959-z>
- Xu Z, Yuan P, Zhang T, Tang Y, Li S, Xia Z (2018) Hda: Cross-project defect prediction via heterogeneous domain adaptation with dictionary learning. *IEEE Access* 6:57597–57613. <https://doi.org/10.1109/ACCESS.2018.2873755>
- Yi L, Gong B, Funkhouser T (2021) Complete & label: A domain adaptation approach to semantic segmentation of lidar point clouds. In: 2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), pp 15363–15373. <https://doi.org/10.1109/CVPR46437.2021.01511>
- Yu Q, Jiang S, Zhang Y (2017) A feature matching and transfer approach for cross-company defect prediction. *J Syst Softw* 132:366–378. <https://doi.org/10.1016/j.jss.2017.06.070>
- Zeng J, Wu S, Yin Y, Jiang Y, Li M (2021) Recurrent attention for neural machine translation. In: Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing, pp 3216–3225. <https://doi.org/10.18653/v1/2021.emnlp-main.258>
- Zhu K, Ying S, Zhang N, Zhu D (2021) Software defect prediction based on enhanced metaheuristic feature selection optimization and a hybrid deep neural network. *J Syst Softw* 180:111026. <https://doi.org/10.1016/j.jss.2021.111026>
- Zimmermann T, Nagappan N, Gall H, Giger E, Murphy B (2009) Cross-project defect prediction: A large scale experiment on data vs. domain vs. process. In: Proceedings of the 7th Joint Meeting of the European

- Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering, pp 91–100. <https://doi.org/10.1145/1595696.1595713>
- Zou Q, Lu L, Yang Z, Gu X, Qiu S (2021) Joint feature representation learning and progressive distribution matching for cross-project defect prediction. *Inf Softw Technol* 137:106588. <https://doi.org/10.1016/j.infsof.2021.106588>
- Özakıncı R, Tarhan A (2018) Early software defect prediction: A systematic map and review. *J Syst Softw* 144:216–239. <https://doi.org/10.1016/j.jss.2018.06.025>

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Springer Nature or its licensor (e.g. a society or other partner) holds exclusive rights to this article under a publishing agreement with the author(s) or other rightsholder(s); author self-archiving of the accepted manuscript version of this article is solely governed by the terms of such publishing agreement and applicable law.

Authors and Affiliations

Hengjie Song¹ · Guobin Wu¹ · Le Ma² · Yufei Pan¹ · Qingan Huang¹ · Siyu Jiang³ 

Hengjie Song
sehjsong@scut.edu.cn

Guobin Wu
wugb2020@163.com

Le Ma
80796487@qq.com

Yufei Pan
yufpan2021@163.com

Qingan Huang
huangqa1001@126.com

- ¹ School of Software Engineering, South China University of Technology, Guangzhou, China
- ² Guangzhou City University of Technology, Guangzhou, China
- ³ Guangzhou Key Laboratory of Multilingual Intelligent Processing, School of Information Science and Technology, Guangdong University of Foreign Studies, Guangzhou, China