Journal of Autonomous Intelligence (2024) Volume 7 Issue 4
doi: 10.32629/jai.v7i4.641

RESEARCH ARTICLE

A cross-item defect prediction method using adversarial learning

Jie Ma[1,2], Jasni Mohamad Zain[1,2], Dan Wang[3,4,*], Jilong Shi[5]

[1] *Institute for Big Data Analytics and Artificial Intelligence (IBDAAI), Universiti Teknologi MARA, 40450 Shah Alam, Selangor, Malaysia*

[2] *School of Computing Sciences, College of Computing, Informatics and Media, Universiti Teknologi MARA, 40450 Shah Alam, Selangor, Malaysia*

[3] *School of Mathematics and Statistics, Qiannan Normal University for Nationalities, Guizhou 558000, China*

[4] *Key Laboratory of Complex Systems and Intelligent Optimization of Guizhou, Guizhou 558000, China*

[5] *School of Computer Sciences, Baoji University of Arts and Sciences, Baoji 721007, China*

*** Corresponding author:** Dan Wang, wangdansci@aliyun.com

ABSTRACT

In today's information society, the rapid growth of information technology has resulted in software products being integrated into every aspect of people's lives. Consequently, the ability to accurately identify software modules that may cause problems within a specified time frame has become crucial for determining software development progress. This is because ensuring software dependability is a critical component of software development. In this paper, the enhanced abstract continuous generative adversarial network (AC-GAN) technique covers data processing and model construction. Three levels: (1) convert the code of the source project and the target project into the form of an abstract syntax tree (Unified Abstract Syntax Tree, UAST), then traverse the abstract syntax tree in a depth-first manner to obtain a node sequence, and then use continuous recursion to replace the nodes in the node sequence; (2) the processed numerical vectors are sent to the GAN-based model; (3) the GAN-based model generates the final word vectors. The network structure model is utilized for feature extraction and data transfer, and a binary classifier is then employed to determine if the target item code file is flawed. 15 sets of source-target item pairings are used to evaluate the AC-GAN approach. The experimental findings demonstrate the usefulness of the technique.

Keywords: AC-GAN; UAST; GAN

ARTICLE INFO

Received: 18 May 2023
Accepted: 2 June 2023
Available online: 19 February 2024

1. Introduction

In today's world, software advancements play an increasingly vital role in people's everyday lives. However, software flaws can pose significant risks to the quality and dependability of software. Identifying and fixing potential software flaws becomes more expensive the later they are discovered, and in extreme cases, can result in significant economic losses for businesses, and even loss of life[1]. To improve software quality assurance, researchers have developed Software Defect Prediction (SDP) technology. SDP is a research area within the software engineering discipline of data mining, aimed at identifying and improving erroneous program modules at an early stage of project development[2]. By enabling developers to identify faulty codes more quickly and reduce development time, as well as enhance test efficiency, SDP technology aims to improve overall software quality and reliability.

For defect prediction, traditional SDP research relies on manually derived static measurements. The source code of a program module, as a formal language, has rich structural and semantic information. Since manually-extracted static metrics primarily focus on the statistical properties of the program[3], it is impossible to capture this complex information from the source code. This restriction impacts the precision of defect prediction. Some academics employ deep learning to acquire important semantic and contextual information from source code[4–6] to circumvent this constraint. To preserve the semantics and context of the code throughout the training and prediction procedure, then use deep learning methods for the feature extraction step. In order to conduct deep learning-based SDP research, sufficient historical data must be collected to train the prediction model. However, in the actual software development process, the software project that must be forecasted may be brand new or have limited historical data. Consequently, cross-project defect prediction (CPDP)[7] was developed to compensate for the absence of historical data by borrowing data from other mature projects, where mature projects are considered the source project (training set) and will be forecasted. considered to be objective objects (test set). Consequently, throughout most projects, project defect prediction performance is inferior to intra-project defect prediction performance. To overcome this issue, several researchers have used machine learning techniques[8,9], such as transfer learning and feature selection. Nevertheless, these conventional machines frequently have learning techniques that are incapable of learning complex characteristics, and the design of loss functions is also quite hard. Experiments have shown that the generative adversarial network (GAN)[10] in adversarial learning may be used to minimize domain disparities. It is widely used in natural language processing and image identification[11–15]. Compared to conventional approaches, GAN networks provide the following benefits: as a training criterion, the GAN network training method uses two adversarial neural networks. The back-propagation approach is used for training, and the training does not depend on the inefficient Markov chain method or approximation inference, and there is no complicated variational lower limit, which substantially decreases the training difficulty and increases the training efficiency. GAN network instruction: the approach employs adversarial training, which generates clearer and more realistic samples, and the GAN network implements data transfer via the discriminator, so avoiding the challenge of loss function design in transfer learning. To improve the source project and target project solutions, this research employs the adversarial learning concept in a GAN network to train a neural network with domain adaptability, and it constantly modifies the distribution of target item attributes until they resemble the distribution of source item features.

This study introduces an augmented generative adversarial network (AC-GAN) technique that uses adversarial learning to tackle the problem of cross-item defect prediction. The method is as follows: first, the code of each module in the source project and the target project is parsed into an abstract syntax tree, and the node sequence is traversed according to the principle of depth first; secondly, the word vector is generated by the word embedding technology CBOW + RNN algorithm, and the node sequence is sorted according to the word vector table. Convert to a numerical vector; use the labeled numerical vector corresponding to the source item as input again to train the source feature extractor and source classifier; from then on, use the source feature as the real data, and use the target feature extracted by the target feature extractor as the fake data input discriminator play the game. Through the adversarial training of the GAN network, the distribution difference between the source data and the target data is reduced; finally, the target features whose distribution changes are input into the target classifier for defect prediction.

Specifically, the main contributions of this paper to cross-project defect prediction are as follows: code is represented in the form of an abstract syntax tree, word vectors are extracted using the CBOW + RNN algorithm, and features are extracted using a deep learning feature extractor, which greatly preserves the context and semantic information; according to the characteristics of the GAN network, an improved AC-GAN model with a similar structure to the GAN network is proposed. Data migration is performed through adversarial game training to solve the distribution difference between the source data and the target data, and the 15 pairs of the source-target project conducts comparative experiments to evaluate the performance of the proposed model. Experiments demonstrate that the enhanced generative adversarial network technique proposed in this study performs well in the entire data processing and model building stages.

2. Related work

This study discusses three research categories with the greatest correlation: software defect prediction; cross-project defect prediction; and adversarial learning.

2.1. Core concepts Software Error Forecasting (SDP)

With an increase in people's reliance on software, the significance of software health has grown, and associated software defect prediction technology has also garnered increased interest. Chang[16] reported that early research on software defect prediction included mathematical and statistical approaches to discover faults. Behavior modification ideas are utilized to build low-defect and high-defect behaviors, whereas negative association rule mining approaches are employed to build prediction algorithms. Based on the fraction of used features, Singh et al.[17] established fuzzy criteria for selecting valuable characteristics for prediction. With the growth of the artificial intelligence sector, several machine learning techniques have been attempted to tackle the software defect prediction issue. Laradji et al.[18] coupled feature selection with ensemble learning and suggested a multiple classifier-combined average probability ensemble learning approach. As a final outcome, the average output defect probability is used. He et al.[19] introduced a learning approach dubbed extRF that augments the supervised random forest algorithm with a self-training model to create a more accurate prediction model. Yang et al.[20] suggested a technique for two-layer ensemble learning. The inner layer of the technique constructs a random forest model based on decision trees, whilst the outer layer uses random sampling to train multiple random forests for defect prediction. In Wu et al.[21], software flaws are detected by using a semi-supervised structured dictionary learning technique through learning a generic dictionary and numerous sub-dictionaries. As machine learning often relies on manually derived characteristics for prediction, its accuracy is restricted. Therefore, some researchers have started to use deep learning techniques. In the field of software fault prediction research, Yang et al.[4] used a deep belief network (DBN)[22] to traverse the token vector from the U AST of the program's source code and then extracted characteristics from the token vector to construct a software fault prediction model. Wang et al.[5] also assigned a unique integer identification to each token, transform the token vector to a numerical vector, and then input DBN to extract semantic characteristics. The survey of Qiao et al.[6] introduced a convolutional neural network (CNN)[23] software defect prediction framework based on neural networks. In this framework, they convert the label vector into a numerical vector using a word embedding app; roach and then use CNN to automatically discover semantic characteristics. The results of experiments indicate that these are retrieved using deep learning. All software fault prediction approaches based on program source code characteristics have excellent prediction performance. In order to improve the performance of defect prediction, this article uses deep learning for the feature extraction step.

2.2. Cross-project defect prediction (CPDP)

In actual software defect prediction applications, it is challenging to create an accurate prediction model for new projects owing to the absence of previous data. To circumvent this constraint of software defect prediction, researchers focus increasingly on using data from other mature projects to create predictions, CPDP method. Commonly used CPDP approaches may be categorized as supervised learning, semi-supervised learning, and unsupervised learning. The CPDP approach based on supervised learning is the most prevalent method among researchers. Nam et al.[24] employed transfer learning to improve the accuracy of the CPDP method. Transfer component analysis (TCA) transfers the training data under the assumption of conserving data properties, such that the distributions of the source data and target data are comparable. Long et al.[25] presented the joint distribution adaptation (JDA) technique, which jointly adjusts the marginal distribution and the conditional distribution in a principled dimensionality reduction process and builds a new feature representation that removes disparities in the data distribution. Turhan et al.[26] recommended the use of closest neighbor filtering, the survey of Xu et al.[27] offered a technique that takes marginal distribution and conditional distribution into account, and adaptive filtering was also proposed. Based on transfer learning, the balanced distribution adaptation (BDA) approach distributes various weights to different weights to balance the data distribution. Based on various candidate sources, Xia et al.[28] built multiple-source items for the semi-supervised learning-based CPDP approach. With the aid of genetic algorithms, a base classifier and an ideal combination (genetic algorithm, GA) classifier are generated. Using the Adaboost ensemble approach, the final model is created based on the GA classifier. Ryu et al.[29] proposed a transfer cost-sensitive improvement (transfer cost-sensitive boosting, TCSBooST) method, which combines the source item instances and a small number of labeled target item instances into a training set, iterates M times, builds M base classifiers, and then performs a weighted ensemble for the base classifiers. Wu et al.[30] introduced a cost-sensitive kernelized semisupervised dictionary learning (CKSDL) technique that employs a limited number of labels through semisupervised dictionary learning (SDL) technology. Source and target items are put in the same subspace to reduce distribution disparities between labeled and unlabeled data. For the CPDP technique based on unsupervised learning, Zhong et al.[31] chose typical modules by clustering and then used supplementary data and invited experts to annotate the document. Zhang et al.[32] separated all software modules into two groups using spectral clustering and then measured the total of the metric values in each category to determine if they were flawed. The acquired CPDP approach is expected to give a more objective look at how well the suggested method works.

2.3. Competitive learning

Presently, the concept of adversarial learning in GAN networks is commonly used in situations where data distribution disparities must be eliminated. Using the concept of adversarial learning, Choi et al.[14] presented a domain-adversarial neural network (DANN). Utilizing label prediction loss and domain discriminative loss, the network accomplishes feature selection across various domains. Engel et al.[15] created a cycle-consistent adversarial network (CycleGAN) with their CycleGAN proposal (CycleGAN). Consistency loss to ensure approximately the same mapping from domain X to domain Y, hence resolving the issue that unpaired training data cannot be used for image-to-image translation. Chang[16] developed a multi-content generative adversarial network (MC-GAN) model, which consists of Glyphnet for predicting rough glyphs and Ornanet for predicting final glyph color and texture. Using Glyphne to predict glyph masks and Ornanet for fine-tuning glyph color and decoration, they were able to transfer font. Singh et al.[17] developed the generative adversarial network in the form of a star and incorporated domain control information to enable style transfer across several picture domains. Laradji et al.[18] created an adversarial neural audio synthesis (GANsynth) model that produced high-fidelity audio by training a GAN network using a series of spectral samples. This research intends to apply the GAN network concept and structure to the CPDP field in order to address the source data and target data discrepancy issue.

Through dense embedding coding, the original AC-GAN translates picture labels to the same dimension as random noise, and then multiplies the two before feeding them into the generator. Following layer-by-layer transposed convolution processing, the effect of picture labels will diminish progressively. The original AC-GAN flattens the convolutional feature map of the last layer of the discriminator and then branches, which are completely linked to the output layer of distinguishing true from false and distinguishing categories, respectively. However, this approach has the following flaws: 1) the two-dimensional feature map must be expanded into one dimension from the convolution layer to the fully connected layer, which greatly increases the number of nodes in the fully connected layer and output layer, which is not conducive to model training; 2) the feature map does not distinguish between true and false. Both the branch and the discriminative category contain a single hidden layer, limiting the branch's capacity for fitting. The network topology of the enhanced AC-GAN generator is shown in **Figure 1**, where noise is white Gaussian noise, which is a collection of random values with a standard normal distribution. Transposed Conv is a convolution process with a 44 kernel size that is transposed. The size of a set of feature maps is 44512, where 44 represents the width and height of the feature map and 512 represents the number of channels on the feature map. DCGAN may include deep convolution into GAN and use trials to select parameters such as the number of convolution channels. Therefore, the size and number of channels of the convolution feature map of the experiment's generator and discriminator correspond to DCGAN. The strip defect picture utilized in the experiment is a grayscale image, with a channel count of 1. The picture labels are mapped to 128, 44, 88, 1616, 3232, and other dimensions using dense encoding embedding, and then multiplied with the feature map of the equivalent size for each layer of the generator network before being transposed.

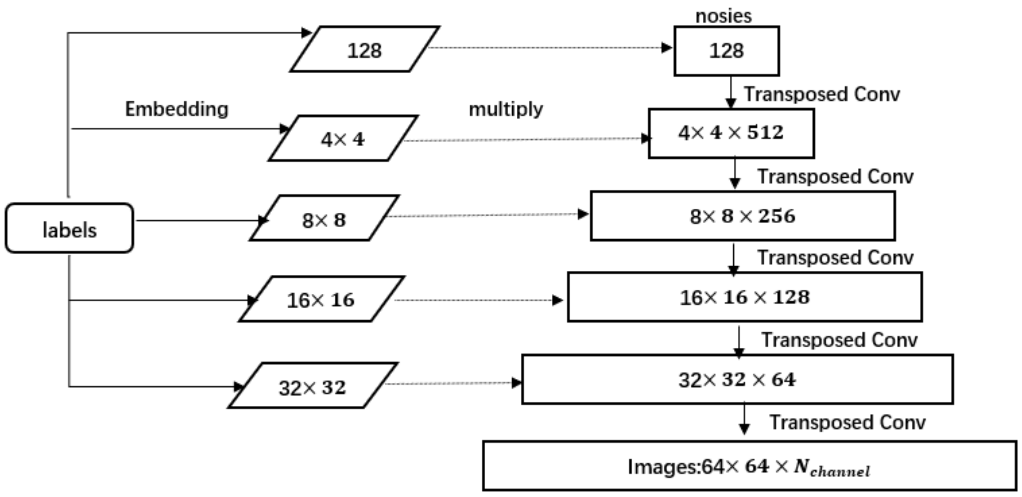


Figure 1. The structure of the generator network.

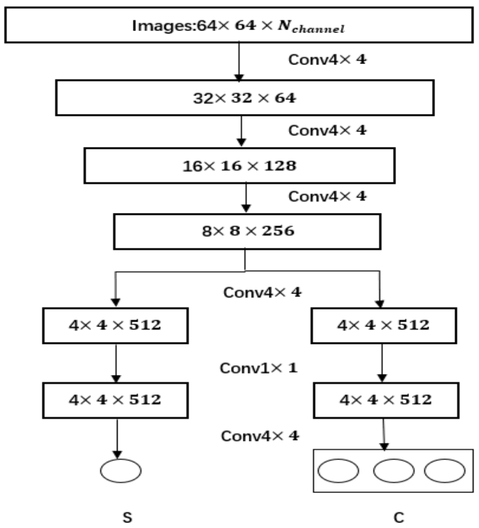


Figure 2. The structure of the discriminator network.

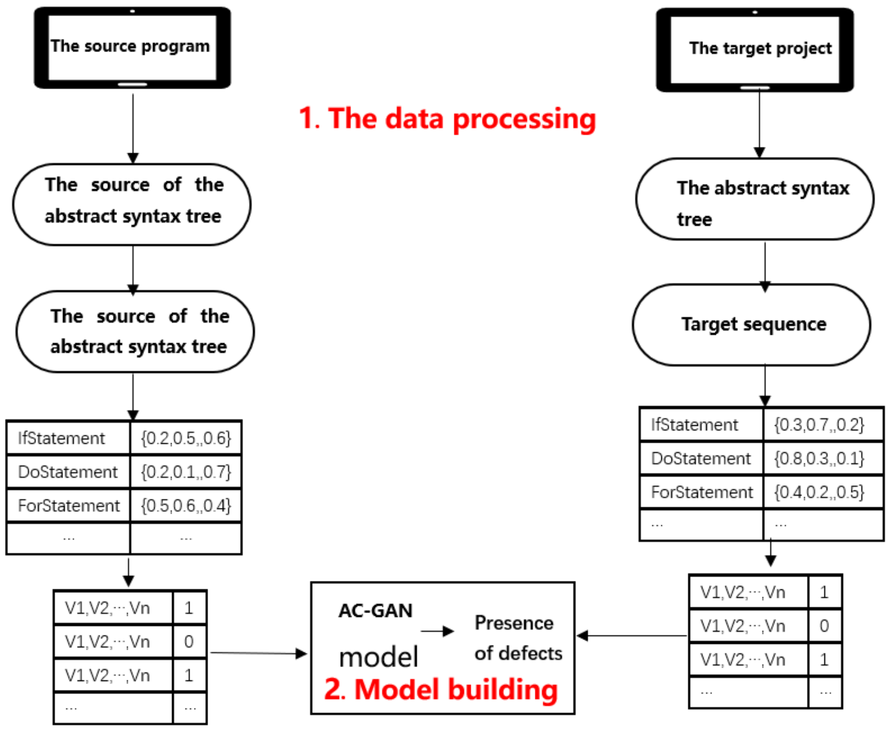
Figure 2 depicts the construction of the modified AC-GAN discriminator network, where Conv 44 denotes the convolution operation with a convolution kernel size of 44. After the third convolution of the discriminator yields an 8 × 8 × 256-dimensional feature map, branches are executed, and each branch is subjected to three convolutions to produce the discrimination results. The 11 convolution may enhance network depth without altering the size of the feature map and without introducing an excessive amount of processing.

The experiment thus employs 11 convolution to create a hidden layer to each branch. The upgraded discriminator lacks a completely linked layer, which prevents the anomalous growth and reduction of nodes, but the number of hidden layers in each branch grows, which not only ensures the sharing of weights across branches but also improves the independence of each branch and fitting ability.

3. Research methods

3.1. The AC-GAN method

Figure 3 shows the overall framework of the AC-GAN method proposed in this paper. Specifically, it consists of two stages: data processing and model building.



The source program
The target project
1. The data processing
The source of the abstract syntax tree
The abstract syntax tree
The source of the abstract syntax tree
Target sequence
IfStatement {0.2,0.5,,0.6}
DoStatement {0.2,0.1,,0.7}
ForStatement {0.5,0.6,,0.4}
IfStatement {0.3,0.7,,0.2}
DoStatement {0.8,0.3,,0.1}
ForStatement {0.4,0.2,,0.5}
V1,V2,···,Vn 1
V1,V2,···,Vn 0
V1,V2,···,Vn 1
AC-GAN model
Presence of defects
2. Model building

Figure 3. The framework of the AC-GAN method.

The data processing stage consists of three distinct phases.

(1) Code parsing and sequence creation (shown in **Figure 1**, the item-to-marker sequence procedure);
(2) Word vector extraction (in **Figure 1**, the process of tag sequence to word vector table);
(3) Data dimension normalization (in **Figure 1**, the word vector table to numerical vector process).

During the stage of model building, the construction of the prediction model and the forecast of the target project are performed primarily. During this phase, the target project's code file may be classified as defect-prone (DP) or non-defect-prone (NDP).

In this part, we describe the design and implementation of our proposed UAST (Unified Abstract Syntax Tree) neural network model for classification of cross-project programs.

As seen in **Figure 1**, this model accepts various programming source codes as input before parsing them into ASTs. Afterwards, it employs a uniform language to execute path embedding and graph embedding on ASTs. Then, the route embedded vector and the graph embedded vector are separately input into separate sub-networks to capture code characteristics. The system then combines distinct learnt characteristics and performs the categorization job.

The cornerstone of cross-project program categorization is the understanding of the semantic and syntactic code characteristics of several programming projects. We propose a Sequence-based AST network for the extraction of syntactic structural information from code (SAST for short). The route sequence derived by a pre-order traversal of the unified AST may be considered a flattened representation of the AST. The route sequence provides the global information of the source code and also displays, to some degree, the syntactic structural features of the source code. We use the self-attention structure[33] to extract the relationships between nodes within a sequence. The self-attention mechanism is a potent mechanism in the transformer structure that is very successful at extracting internal linkages and may mitigate the issue of long-distance reliance. The formula for calculation is as follows:

$$Attention(Q,K,V) = SoftMax\left(\frac{QK^T}{\sqrt{d_k}}\right)V \tag{1}$$

where the three matrices $Q \in R^{l\times d}$, $K \in R^{l\times d}$, and $V \in R^{l\times d}$ are initialized and generated according to the embedded path sequence vector, and these three matrices are equal in the self-attention mechanism, which could reduce the parameters of the model and can train faster. d is the embedding dimension of the path sequence. l is the length of input path. Dot product is calculated between Q and K. $d_k \in R^d$ is the dimension of input vector. And *Attention* (Q, K, V) is the calculated attention score.

In writing code, the context of the code statement often reflects its intent. For example, it needs to declare a variable before using it in C++. The above-mentioned self-attention mechanism has captured the internal relationship of the code embedded vector. So, in addition to extracting the internal dependencies of the input source code, it also needs to capture the context dependency of the source code. Therefore, the Bidirectional Long Short-Term Memory (Bi-LSTM)[34] is introduced here. The Bi- LSTM could learn features of the input data from two directions, so it can infer the current information from the context of the code. In our proposed SAST network, Bi-LSTM is used to comprehensively consider all available input path information in the context to extract Semantic and logic features of the source code. Specifically, the hidden state of the LSTM at each position t of the input path is computed as:

$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i) \tag{2}$$

$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f) \tag{3}$$

$$o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o) \tag{4}$$

$$\tilde{c}_t = tan\,h(W_C \cdot [h_{t-1}, x_t] + b_c) \tag{5}$$

$$c_t = f_t \odot c_{t-1} + i_t \odot \tilde{c}_t \tag{6}$$

$$h_t = o_t \odot \tanh(c_t) \tag{7}$$

where σ is the sigmoid function, $tan\,h$ is the hyperbolic function, $X_t \in R^d$ presents the data at position t of the input path sequence.

After self-attention, c_t presents the hidden unit state of x_t, and $h_t \in R^h$ represents the hidden unit state of the learning layer, which is the final extracted code features. W_i, W_f, $W_o \in R^{h\times d}$ are the trainable weight matrices. $\odot$ is the element-wise matrix multiplication operator.

$$h_{SAST} = \overrightarrow{h_t} \oplus \overleftarrow{h_t} \tag{8}$$

After that, we concatenate the hidden state $\overrightarrow{h_t} \in R^h$ learned by the forward LSTM and the hidden state $\overleftarrow{h_t} \in R^h$ learned by the backward LSTM to obtain the $h_{SAST} \in R^{2\times h}$, which contains the context features of the code.

Sequence-based AST network has learnt the global structure and syntactic aspects of the code from the path sequence, but because the path sequence is a flattened representation, certain tree-like structural information of the code is ignored. We observe that, with the exception of project-specific library files or package files, the logic of various programming projects when developing particular functions is often identical.

The 1-hop aggregation can extract the direct relationship between code statements. As the number of hops increases, the "return statement" node can also indirectly aggregate the information of their second-order neighbors ("body, *identifier*$_1$, *identifier*$_2$"). Therefore, the "return statement" node could learn the local structural and semantic features of its neighbors. The right of **Figure 3** is the adjacency matrix of the left AST, and it is worth noting that$\tilde{A}$ adds an identity matrix to its adjacency matrix, indicating that the node could also learn the feature of itself. The graph convolution operation is defined as follows:

$$\tilde{A} = A + I \tag{9}$$

$$H_i^{(l+1)} = \sigma(\sum_{j\in N} D^{-\frac{1}{2}}\tilde{A}D^{-\frac{1}{2}}H_j^{(l)}W^{(l)}) \tag{10}$$

where $H_i^{(l+1)}$ is the feature of node i in the layer (l + 1), $H_j^{(l)}$ is the feature of all neighbor nodes of node i (including itself) in the layer l; N is the number of all neighbors of node i; $\tilde{A} \in R^{N\times N}$ is the adjacency matrix A of node i added with the identity matrix; $D \in R^{N\times N}$ is the degree matrix of $\tilde{A}$; $W^{(l)} \in R^{d_{in}\times d_{out}}$ is the trainable weight matrix in the layer l.

The aforementioned SAST has captured the global structure and logical characteristics of the code, and GAST has extracted the local structural and semantic feature of the code. In order to comprehensively consider the global and local code semantic feature, a fusion mechanism is further needed. We realize the enhancement of dimensional features through vector concatenation, which can be described as:

$$h_{code} = concat(h_{SAST}, h_{GAST}) \tag{11}$$

where $h_{code} =\in R^{2\times h+d_{out}}$ represents the feature vector after the unified AST feature fusion, $h_{SAST} =\in R^{2\times h}$ represents the global structural feature learned from the flattened sequence, and $h_{GAST} =\in R^{d_{out}}$ represents the local semantic feature learned from the graphlike AST.

After that, we obtain the feature vector which includes the global and local semantic features of the input code, and then we perform a fully connected layer for linear dimensional transformation, and eventually the probability pi is the output through the SoftMax layer. We use the Cross Entropy[35] as our loss function and adopt Adam optimizer[36] to minimize it. The loss is calculated as follows:

$$J = -\sum_{i=1}^{k} y_i log(p_i) \tag{12}$$

where k is the number of program categories, and y is the label of different programs, p_i is the output after the SoftMax layer.

3.2. Improved AC-GAN model

During the data processing phase, the code files of the source project and the target project are transformed into numerical vectors of uniform length that can be fed into the feature extractor to extract features. This study investigates the original GAN network in light of cross-project defect prediction needs. The model's architecture was altered, and an enhanced AC-GAN model was presented. The approach is to use the output source features from the trained source feature extractor as actual samples and the output target features from the target feature extractor as a generative model. To play the game, the source feature and target feature are entered into the discriminator, or discriminant model, as a created false sample. When the discriminator is unable to discriminate between the source feature and the target feature, it demonstrates that the distribution of the two features is essentially same. Consequently, the difference in distribution between the source data and the target data is erased.

In the realm of natural language processing, the recurrent neural network (RNN) has gained notable success. The long short-term memory (LSTM)[37] is an RNN network improvement that resolves the issue of the issue of gradient disappearance of RNN networks in lengthy sequences may gain information about long-term dependencies. Since the code is a structure with logic and semantics and has a closely connected nature, the code fragments that causes faults are often associated with their respective contexts. The chosen characteristic extractor. The input sequence must be capable of being processed in both directions. This article employs a bidirectional long short-term memory (BLSTM)[38], i.e., a two-layer LSTM network, as the source feature extractor and target feature. Extractor is to produce a more accurate representation and higher resilience. The most often used binary classifier is the Logistic Regression Model (LR)[39]. In order to enable further comparisons with other classic CPDP algorithms, LR is chosen as the source and target classifier in this work.

As seen in **Figure 4**, the enhanced AC-GAN model is constructed using the following methods.

(1) Training source feature extraction and classification: before playing the game, the modified AC-GAN model underwent supervised training on the source feature extractor and source classifier, splitting the source item data into the training set and test set in a ratio of 8:2 to produce excellent results in defect prediction for the project;

(2) Training the discriminator and target feature extractor: according to the design of the GAN network, adversarial game training is done on the discriminator and target feature extractor. Since the target feature must ultimately resemble the source feature, the target feature extractor's parameters are originally set to. The source extractor for features is the same. If the output is not 0.5 after the discriminator determines the true and false, update the parameters using the gradient ascent technique, then update the target feature extractor parameters using the gradient descent method, change the output target features, and discriminate again. This cycle continues until the discriminator output reaches 0.5, at which point it cannot be evaluated as either true or false. When the discriminator output approaches 0.5, the target feature extractor's parameters will stop changing and training will cease;

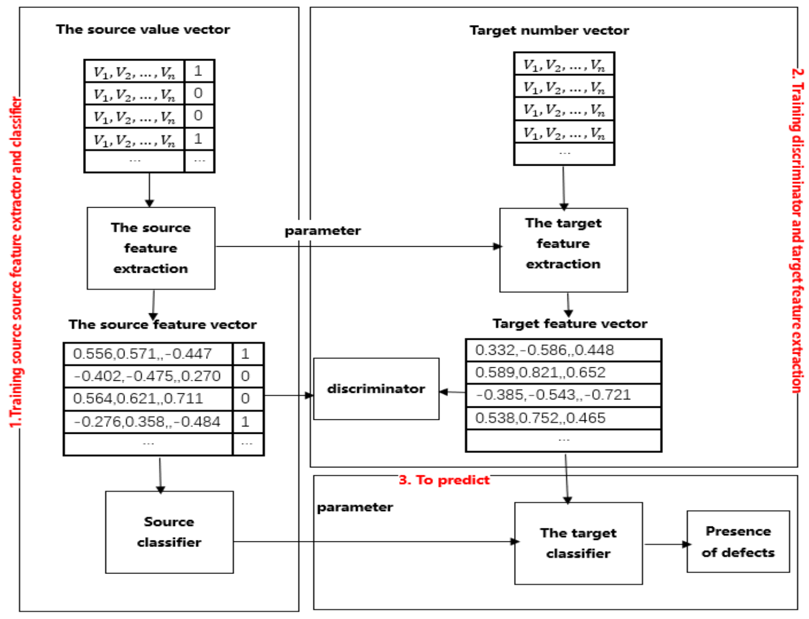


Figure 4. AC-GAN model architecture.

(3) Prediction: input the features extracted by the trained target feature extractor into the target classifier in order to forecast the presence or absence of faults. Since the source classifier has been trained, the target classifier's parameters should be identical to those of the learned source classifier.

3.3. Experimental design

The enhanced AC-GAN technique presented in this study adopts the GAN network structure, eliminates the disparity between the source item and the target item's data distribution via an adversarial game, and employs the BLSTM network as the feature extractor for feature extraction. To verify the effectiveness of the AC-GAN approach, this report asks the following two research questions about effectiveness:

RQ1: Are deep learning-based approaches more effective than conventional SDP methods?

RQ2: Is the improved AC-GAN method proposed in this paper superior to the conventional CPDP technique?

For RQ1, this study picks the conventional LR approach and the suggested AC-GAN method using a deep learning feature extractor for feature extraction in order to conduct comparative tests on the prediction of intra-item defects. Methods for defect categorization are based on metrics.

For RQ2, this article chooses various classic CPDP techniques and the AC-GAN approach based on the GAN network topology to perform comparative cross-project defect prediction studies. Several classic techniques are compiled using MATLAB R2018b, and the following serves as an introduction.

4. Results and discussion

4.1. Data set

As the experimental data set, this paper selects six commonly used CPDP projects from the open source database PROMISE[40]. According to **Table 1**, the magnitude and fault rate of these six projects are distinct, assuring the generalizability of the experimental findings. The Log4j project's failure rate is too high to be used as a training set.

Therefore, this paper selects the remaining five items as the training set and then randomly selects three items that are not included in the training set as the test set.

Table 1. The specific information of the project selected in this article.

Project name	**Version**	**Number of files**	**Defect rate (%)**
Camel _	1.6 _	9 35	2 0.1
Poi _	3.0 _	4 38	6 4.1
X erces	1.4 _	5 08	7 6.8
Lucene _	2.4 _	3 30	6 1.5
X alan	2.6 _	8 75	5 3.1
Log4j _	1.2 _	1 94	9 5.9

In addition to the source code, each item in the collection also includes static code metrics and defect annotations of the source code. **Table 2** displays the particular measurements of static code metrics. These measures may be used to compare comparable software fault prediction techniques to AC-GAN techniques.

Table 2. Abbreviations and names of 20 static code metrics.

Abbreviated name	Full name
LOC	Lines of code
DIT	DIT Depth of inheritance tree
NOC	Number of children
RFC	Response for a class
C BO	Coupling between object classes
L COM	Lack of cohesion in methods
L COM3	Lack of cohesion in methods
N PM	Number of public methods
D AM	Data access metric
M OA	Measure of aggregation
MFA _	Measure of function abstraction
IC _	Inheritance coupling
C AM	Cohesion among methods of class
CBM _	Coupling between methods
AMC _	Average method complexity
Ca	Afferent couplings
C e	Efferent couplings
A vg(CC)	Average McCabe
Max (CC)	Maximum McCabe
W MC	Weighted methods per class

4.2. Experimental outcomes and discussion regarding RQ1

In general software defect prediction research, twenty manually extracted static metrics are used as feature vectors that are fed into the classifier to predict whether defects exist. However, static metrics often represent just the structural characteristics of code files and do not include any semantic characteristics. If there are bugs in the code's details, but they are not reflected in the structure, then the conventional method for predicting software defects cannot identify them. Consequently, this study utilizes the abstract syntax tree to explain the code structure while preserving semantic information. The CBOW algorithm is used to generate word vectors that reflect contextual information, while the BLSTM neural network-based feature extractor is utilized to extract semantic and contextual features and compare the outcomes of the experiments.

Table 3. F1-measure of AC-GAN method and LR method.

Title	AC-GAN method	LR method
Camel _	0.985 _	0.314
Poi _	0.986 _	0.715 _
X erces	0.986 _	0.901 _
Lucene _	0.991 _	0.570 _
X alan	0.995 _	0.560 _
Log4j _	0.979 _	0.949 _
Aver age	0.987 _	0.673 _
p-value	0.01553 _	

As seen in **Table 3**, the F1-measure of the AC-GAN technique is very near to 1, indicating that the method presented in this research may significantly enhance the accuracy of software fault prediction. The AC-GAN approach yields a higher average F1-measure value than the LR method. The average F1-measure value is 46.7% higher, indicating that the AC-GAN technique using an abstract syntax tree, CBOW algorithm, and BLSTM feature extractor is efficient and has a substantial increase in prediction performance. The *p*-value for the Wilcoxon signed-rank test is only 0.01553, which is significantly less than 0.05, showing that there is a statistically significant difference between the two approaches.

4.3. Experimental outcomes and discussion regarding RQ2

The AC-GAN approach uses the GAN network structure to constantly modify the target distribution to fit the source distribution until the issue of disparate data distributions is resolved. Unlike the conventional generative model, the training process of the GAN network employs two types of adversarial neural networks to play the game, which may be employed as a kind of reinforcement learning. The back-propagation strategy for training may minimize training difficulty and enhance training effectiveness. In contrast to classical transfer learning, the GAN network training approach employs adversarial training, which may yield more accurate and realistic samples, and the GAN network use a neural network. The discriminator is used to implement data transfer, which allows transfer learning to circumvent the problem of loss function construction. The experimental findings of comparing the AC-GAN approach to five classic CPDP methods are shown in **Tables 4** and **5**.

According to **Tables 4** and **5**, there are a total of fifteen sets of source-target item combinations. Overall, the AC-GAN technique has greater average F1-measure and average AUC than the other five methods. In the conventional technique, the BDA method's prediction is more accurate based on the F1-measure, while the CKSDL method's overall performance is superior based on the AUC. From the perspective of a single experiment, 86.7% of the experimental results for F1-measure indicate that the AC-GAN method outperforms the other 5 ways; similarly, 80.0% of the experimental data for AUC indicate that the AUC values of the AC-GAN method are all better. The following explains why the AC-GAN approach works better.

(1) The AC-GAN method preserves the semantic and contextual information of the code program during training and prediction, which can find bugs that cannot be identified by static metrics, resulting in more accurate predictions;

(2) The AC-GAN method uses the GAN network structure for data migration, and the target feature samples after the migration are clearer and more realistic, and the representation power is stronger while the distribution is similar to the source feature;

(3) The AC-GAN method preserves the semantic and contextual information of the code program during training and prediction, shown in **Tables 4** and **5**: in the Wilcoxon signed-rank test, the TCA, JDA, NNFilter, BDA and CKSDL's *p*-values of F-measure are 6.009×10^{-4}, 6.151×10^{-5}, 1.272×10^{-4}, 1.973×10^{-3} and 1.155×10^{-3}, which are far less than 0.05, which means that there is a significant difference between the AC-GAN method and other methods, proving that the AC-GAN method. The prediction results of the method are more accurate. The *p*-values of the AUCs of TCA, JDA, NNFilter, BDA and CKSDL are all less than 0.05, indicating that the overall performance of the AC-GAN method is better than other traditional methods.

Table 4. F1-measure of AC-GAN, TCA, JDA, NNFilter, BDA and CKSDL.

The source program	The target project	AC-GAN	TCA	JDA	NNFilter	BDA	CKSDL
Poi _	Camel _	0.784 _	0.496 _	0.453 _	0.449 _	0.370 _	0.303 _
Poi _	X erces	0.809 _	0.699 _	0.062 _	0.656 _	0.753 _	0.590 _
Poi _	X alan	0.797 _	0.755 _	0.489 _	0.573 _	0.649 _	0.450 _
Lucene _	Log4j _	0.778 _	0.626 _	0.351 _	0.594 _	0.759 _	0.579 _

Table 4. (*Continued*).

The source program	The target project	AC-GAN	TCA	JDA	NNFilter	BDA	CKSDL
Lucene _	X erces	0.762 _	0.640 _	0.344 _	0.664 _	0.736 _	0.643 _
Lucene _	X alan	0.843 _	0.598 _	0.444 _	0.579 _	0.649 _	0.628 _
X alan	Log4j _	0.686 _	0.461 _	0.405 _	0.459 _	0.625 _	0.629 _
X alan	Poi _	0.738 _	0.519 _	0.470 _	0.496 _	0.657 _	0.669 _
X alan	Lucene _	0.521 _	0.524 _	0.314 _	0.526 _	0.600 _	0.670 _
Camel _	Log4j _	0.797 _	0.246 _	0.125 _	0.289 _	0.722 _	0.383 _
Camel _	X alan	0.548 _	0.380 _	0.116 _	0.371 _	0.564 _	0.549 _
Camel _	Lucene _	0.806 _	0.324	0.201 _	0.333 _	0.615 _	0.651 _
X erces	Poi _	0.756 _	0.614 _	0.345 _	0.594 _	0.675 _	0.689 _
X erces	X alan	0.642 _	0.693 _	0.423 _	0.559 _	0.493 _	0.325 _
X erces	Lucene _	0.661 _	0.635 _	0.415 _	0.644 _	0.619 _	0.689 _
A level		0.728 _	0.547 _	0.329 _	0.520 _	0.634 _	0.545 _
p-value		-	6.009×10^{-4}	6.151×10^{-5}	1.272×10^{-4}	1.973×10^{-3}	1.155×10^{-3}

Table 5. AUC of AC-GAN, TCA, JDA, NNFilter, BDA and CKSDL.

The source program	The target project	AC-GAN	TCA	JDA	NNFilter	BDA	CKSDL
Poi _	Camel _	0.882 _	0.497 _	0.542 _	0.403 _	0.615 _	0.802 _
Poi _	X erces	0.839 _	0.606 _	0.435 _	0.535 _	0.665 _	0.584 _
Poi _	X alan	0.756 _	0.659 _	0.579 _	0.582 _	0.664 _	0.759 _
Lucene _	Log4j _	0.796 _	0.591 _	0.565 _	0.573 _	0.529 _	0.855 _
Lucene _	X erces	0.793 _	0.612 _	0.624 _	0.685 _	0.625 _	0.749 _
Lucene _	X alan	0.829 _	0.600 _	0.655 _	0.635 _	0.748 _	0.678 _
X alan	Log4j _	0.787 _	0.469 _	0.529 _	0.452 _	0.479	0.529 _
X alan	Poi _	0.711 _	0.562 _	0.655 _	0.553 _	0.569 _	0.521 _
X alan	Lucene _	0.674	0.575 _	0.655 _	0.600 _	0.569 _	0.603 _
Camel _	Log4j _	0.839 _	0.516 _	0.552 _	0.579 _	0.669 _	0.769 _
Camel _	X alan	0.718 _	0.674 _	0.517 _	0.669 _	0.500 _	0.915 _
Camel _	Lucene _	0.840 _	0.615 _	0.668 _	0.629 _	0.629 _	0.596 _
X erces	Poi _	0.573 _	0.402 _	0.459 _	0.456 _	0.596 _	0.689 _
X erces	X alan	0.812 _	0.494 _	0.252 _	0.412 _	0.356 _	0.749 _
X erces	Lucene _	0.600 _	0.465 _	0.428 _	0.485 _	0.449 _	0.603 _
A level		0.769 _	0.552 _	0.549 _	0.539 _	0.577 _	0.699 _
p-value		-	6.096×10^{-5}	6.102×10^{-5}	6.108×10^{-5}	1.223×10^{-4}	3.528×10^{-2}

Tables 6 and **7** display the ranking of F1-measure and AUC for the six approaches according to the Scott-Knott test. **Table 6** divides the F1-measure of the six approaches into four categories: AC-GAN method, the highest ranking; BDA method; CKSDL, TCA, and NNFilter techniques, which are grouped together; JDA method, the lowest rating. **Table 7** classifies the AUCs of the six approaches into three categories: the best approach is still the AC-GAN method, followed by the CKSDL method and the other four ways. The AC-GAN approach ranks top in both categories and has the greatest performance, whereas the F1-measure of BDA is superior. The AUC of the CKSDL approach is second only to the AUC of the AC-GAN method when

compared to other classic CPDP methods.

Table 6. Ranking of F1-measure.

A C-GAN	B DA	C KSDL	T CA	N NF ilter	J DA
0.719 _	0.615 _	0.573 _	0.549 _	0.525 _	0.321 _

Table 7. Ranking of AUC.

A C-GAN	C KSDL	B DA	T CA	N NF ilter	J DA
0.779 _	0.706	0.586 _	0.563 _	0.551 _	1.485 _

5. Conclusion

This paper proposes a novel Cross-Item Defect Prediction (CIDP) method that leverages the power of adversarial learning. The proposed CIDP method aims to overcome the limitations of traditional defect prediction methods, which often suffer from poor performance due to the difficulty of accurately identifying defects in highly diverse software systems. This method offers two sets of comparison experiments utilizing F1-measure and AUC as assessment metrics to demonstrate the efficacy of the AC-GAN algorithm. The experimental findings demonstrate that the strategy presented in this study performs well throughout both the data processing and model construction phases. Not only can the AC-GAN approach collect semantic and contextual information, but it can also effectively reduce data disparities between distinct objects. Due to the paper's exclusive usage of the LR classifier and the abstract syntax tree as the representation technique, the accuracy of predictions is limited. Using the data, this work will use the ensemble approach or a more appropriate classifier in the future. In order to gain better experimental findings, the approach presented in this research is confined to the one-to-one prediction of source project data pairs. However, the many-to-one prediction method will also be investigated in the future.

Author contributions

Conceptualization, JM and DW; methodology, JS; software, JS; validation, JM and DW; formal analysis, JM and DW; investigation, JM and DW; resources, JM and DW; data curation, JM and DW; writing—original draft preparation, JM and DW; writing—review and editing, JM and DW; visualization, JMZ and DW; supervision, JMZ; project administration, JMZ; funding acquisition, JMZ. All authors have read and agreed to the published version of the manuscript.

Acknowledgments

This work was supported by the National Natural Science Foundation of China (No.61862051), the Science and Technology Foundation of Guizhou Province (No.[2019]1299, No.ZK[2022]449), the Top-notch Talent Program of Guizhou province (No.KY[2018]080), the Natural Science Foundation of Education of Guizhou province(No.[2019]203) and the Funds of Qiannan Normal University for Nationalities (No. qnsy2019rc09).

Conflict of interest

The authors declare no conflict of interest.

References

1. Gray J. Why do computers stop and what can be done about it? In: Symposium on Reliability in Distributed Software and Database Systems; 13–15 January1986; Los Angeles, USA. pp. 3–12.

2. Hall T, Beecham S. Bowes D, et al. A systematic literature review on fault prediction performance in software engineering. IEEE Transactions on Software Engineering 2012; 38(6): 1276–1304. doi: 10.1109/TSE.2011.103
3. Punitha K, Chitra S. Software defect prediction using software metrics-A survey. In: 2013 International Conference on Information Communication and Embedded Systems (ICICES); 21–22 February2013; Chennai. pp. 555–558.
4. Yang X, Lo D, Xia X, et al. Deep learning for just-in-time defect prediction. In: 2015 IEEE International Conference on Software Quality, Reliability and Security; 3–5 August 2015; Vancouver. pp. 17–26.
5. Wang S, Liu T, Tan L. Automatically learning semantic features for defect prediction. In: 2016 IEEE/ACM 38th International Conference on Software Engineering (ICSE); 14–22 May 2016; Austin. p. 297.
6. Qiao L, Li G, Yu D, Liu, H. Deep feature learning to quantitative prediction of software defects. In: 2021 IEEE 45th Annual Computers, Software, and Applications Conference (COMPSAC); 12–16 July 2021; Madrid. pp. 1401–1402.
7. Gray D, Bowes D, Davey N, et al. Software defect prediction using static code metrics underestimates defect-proneness. In: The 2010 International Joint Conference on Neural Networks (IJCNN); 18–23 July 2010; Barcelona. pp. 1–7.
8. Hosseini S, Turhan B, Gunarathna D. A systematic literature review and meta-analysis on cross project defect prediction. IEEE Transactions on Software Engineering 2019; 45(2): 111–147. doi: 10.1109/TSE.2017.2770124
9. Jin C. Cross-project software defect prediction based on domain adaptation learning and optimization. Expert Systems with Applications 2021; 171(1): 114637. doi: 10.1016/j.eswa.2021.114637
10. Wang K, Gou C, Duan Y, et al. Generative adversarial networks: Introduction and outlook. IEEE/CAA Journal of Automatica Sinica 2017; 4(4): 588–598. doi: 10.1109/JAS.2017.7510583
11. Ganin Y, Ustinova E, Ajakan, et al. Domain-adversarial training of neural networks. The Journal of Machine Learning Research 2020; 17(1): 2096–2030. doi :10.1109/TNNLS.2020.3025954
12. Zhu JY, Park T, Isola P, Efros AA. Unpaired image-to-image translation using cycle consistent adversarial networks. In: Proceedings of the IEEE International Conference on Computer Vision; 22–29 October 2017; Venice. pp. 2223–2232.
13. Azadi S, Fisher M, Kim VG, et al. Multi-content GAN for few-shot font style transfer. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition; 18–23 June 2018; Salt Lake. pp. 7564–7573.
14. Choi Y, Choi M, Kim M, et al. Stargan: Unified generative adversarial networks for multi-domain image-to-image translation. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition; 18–23 June 2018; Salt Lake. pp. 8789–8797.
15. Engel J, Agrawal KK, Chen S, et al. Gansynth: Adversarial neural audio synthesis. arXiv:1902.08710; 2019. doi: 10.48550/arXiv.1902.08710
16. Chang CP. Integrating action-based defect prediction to provide recommendations for defect action correction. International Journal of Software Engineering and Knowledge Engineering 2012; 23(2): 147–172. doi: 10.1142/S0218194013500022
17. Singh P, Pal NR, Verma, S, Vyas OP. Fuzzy rule-based approach for software fault prediction. IEEE Transactions on Systems, Man, and Cybernetics: Systems 2017; 47(5): 826–837. doi: 10.1109/TSMC.2016.2521840
18. Laradji IH, Alshayeb M, Ghouti L. Software defect prediction using ensemble learning on selected features. Information and Software Technology 2015; 58: 388–402. doi: 10.1016/j.infsof.2014.07.005
19. He Q, Shen B, Chen Y. Software defect prediction using semi-supervised learning with change burst information. In: 2016 IEEE 40th Annual Computer Software and Applications Conference (COMPSAC); 10–14 June 2016; Atlanta. pp. 113–122.
20. Yang X, Lo D, Xia X, Sun J. TLEL: A two-layer ensemble learning approach for just-in-time defect prediction. Information and Software Technology 2017; 87(87): 206–220. doi: 10.1016/j.infsof.2017.03.00
21. Wu F, Jing XY, Dong X, et al. Cross-project and within-project semi-supervised software defect prediction problems study using a unified solution. In: 2017 IEEE/ACM 39th International Conference on Software Engineering Companion (ICSE-C); 20–28 May 2017; Buenos. pp. 195–197.
22. Hinton GE, Osindero S, Teh YW. A fast learning algorithm for deep belief nets. Neural Computation 2006; 18(7): 1527–1554. doi: 10.1162/neco.2006.18.7.1527
23. Li J, He P, Zhu J, Lyu MR. Software defect prediction via convolutional neural network. In: 2017 IEEE International Conference on Software Quality, Reliability and Security (QRS); 25–29 July 2017; Prague. pp. 318–328.
24. Nam J, Pan SJ, Kim S. Transfer defect learning. In: 2013 35th International Conference on Software Engineering (ICSE); 18–26 May 2013; San Francisco. pp. 382–391.
25. Long M, Wang J, Ding G, et al. Transfer feature learning with joint distribution adaptation. In: Proceedings of the IEEE International Conference on Computer Vision; 1–8 December 2013; Sydney. pp. 2200–2207.
26. Turhan B, Menzies T, Bener AB, Di Stefano J. On the relative value of cross-company and within-company data for defect prediction. Empirical Software Engineering 2009; 14(5): 540–578. doi: 10.1007/s10664-008-9103-7
27. Xu Z, Pang S, Zhang T, et al. Cross project defect prediction via balanced distribution adaptation based transfer learning. Journal of Computer Science and Technology 2019; 34(5): 1039–1062. doi: 10.1007/s11390-019-1959-z

28. Xia X, Lo D, Pan SJ, et al. Hydra: Massively compositional model for cross-project defect prediction. IEEE Transactions on Software Engineering 2016; 42(10): 977–998. doi: 10.1109/TSE.2016.2543218
29. Ryu D, Jang JI, Baik J. A transfer cost-sensitive boosting approach for cross-project defect prediction. Software Quality Journal 2017; 25(1): 235–272. doi: 10.1007/s11219-015-9287-1
30. Wu F, Jing XY, Sun Y, et al. Cross-project and within-project semisupervised software defect prediction: A unified approach. IEEE Transactions on Reliability 2018; 67(2): 581–597. doi: 10.1109/TR.2018.2804922
31. Zhong S, Khoshgoftaar TM, Seliya N. Unsupervised learning for expert-based software quality estimation. In: HASE; 25–26 March 2004; Tampa. pp. 149–155.
32. Zhang F, Zheng Q, Zou Y, Hassan AE. Cross-project defect prediction using a connectivity-based unsupervised classifier. In: 2016 IEEE/ACM 38th International Conference on Software Engineering (ICSE); 14–22 May 2016; Austin. pp. 309–320.
33. Vaswani A, Shazeer N, Parmar N, et al. Attention is all you need. In: 31st Conference on Neural Information Processing Systems; 4–9 December 2017; Long Beach. pp. 6000–6010.
34. Graves A, Schmidhuber J. Framewise phoneme classification with bidirectional LSTM and other neural network architectures. Neural Networks 2005; 18(5–6): 602–610. doi: 10.1109/IJCNN.2005.1556215
35. De Boer PT, Kroese DP, Mannor S, Rubinstein RY. A tutorial on the cross-entropy method. Annals of Operations Research 2005; 134(1): 19–67. doi: 10.1007/s10479-005-5724-z.
36. Jiang L, Su Z. Automatic mining of functionally equivalent code fragments via random testing. In: Proceedings of the Eighteenth International Symposium on Software Testing and Analysis; 19 July 2009; Chicago. pp. 81–92.
37. Hochreiter S, Schmidhuber J. Long short-term memory. Neural Computation 1997; 9(8): 1735–1780. doi: 10.1162/neco.1997.9.8.1735
38. Ray A, Rajeswar S, Chaudhury S. Text recognition using deep BLSTM networks. In: 2015 Eighth International Conference on Advances in Pattern Recognition (ICAPR); 4–7 January 2015; Kolkata. pp. 1–6.
39. Cramer JS. The origins of logistic regression. Tinbergen Institute Working Paper 2002; 119. doi: 10.2139/ssrn.360300
40. Degtyarenko KN, North AC, Findlay JB. PROMISE: A database of bioinorganic motifs. Nucleic Acids Research 1999; 27(1): 233–236. doi: 10.1093/nar/27.1.233