2024 IEEE 24th International Conference on Software Quality, Reliability, and Security Companion (QRS-C)

Ada4DP: Adversarial Enhanced Dual-attentive Aggregation Learning for Defect Prediction

Yucheng Zhang[†], Chenqiang Shen[†*], Xiaolin Ju[†], Hao Shen[†], Xiang Chen[†]
[†]*School of Information Science and Technology*, *Nantong University*, China
Email: yc.zhang@stmail.ntu.edu.cn, grant@stmail.ntu.edu.cn, ju.xl@ntu.edu.cn, shenhyc@gmail.com, xchencs@ntu.edu.cn

Abstract—Software defect prediction (SDP) aims to assist practitioners for identifying defects quickly and efficiently, and to allocate limited software resources toward the most high-risk files to enhance software quality. Existing research mainly relies on manual feature engineering, metrics-based software quality assessment, machine learning for classification and identification, and other types. However, these methods are usually constrained, which can hinder the effectiveness of improving defect prediction. We propose a novel defect prediction method named Ada4DP, which combines deep semantic mining and data augmentation via adversarial learning. Ada4DP utilizes an encoder to extract semantic features from source code and employs adversarial learning (FGM) to perturb and augment the data, thereby mitigating the limitations of datasets. Moreover, we enhance the capture of source code context and explore additional semantics by integrating the Bi-GRU model with attention mechanisms. Additionally, Ada4DP can leverage the labels generated after prediction to enhance prediction accuracy through interactions between forward and backward classification by dual learning. Our experimental evaluation on 10 open-source projects from the PROMISE dataset demonstrates that Ada4DP is 11.6% higher than the baseline model in both within-project and cross-project defect prediction.

Keywords–deep learning; dual learning; data augmentation; semantic extraction

1. Introduction

With the growth of the software industry, software systems are becoming increasingly complex, which makes software quality and reliability a crucial issue [16]. For example, Facebook suffered serious economic losses due to code defects. Since the presence of defects in software development, defects are a key factor in code failure and are expensive to handle [8]. Therefore, software quality assurance(SQA) has become a hot topic in the field of defect handling. SQA [22] includes analyzing various data generated throughout the development phase to predict potential defects and to detect and resolve defects as early as possible. However, due to the current large and complex nature of software projects, it is not possible to carry out complete SQA activities on them.

Defect prediction techniques have been proposed to tackle potentially high-risk files with limited resources. In previous studies, manually designed features [30] have been primarily used as inputs for machine learning models [7], such as static features [2] and process features [5]. However, we rely solely on manually designed features for prediction [15], and it has limitations. In some cases, even if two Java code snippets are similar, they may differ in statement positions and semantic representation. Therefore, the semantic and structural characteristics in the code cannot be estimated by only manually designed features. Moreover, according to Wan et al.'s [25] description in the defect survey, they concluded code semantic information is far more useful for distinguishing defects than syntax information.

Recent research has focused mainly on the automatic learning of the semantic characteristics of code, and predict defects using deep learning networks [10]. Many researchers parse source code files by using abstract syntax trees (AST) in their methods, and they use these obtained node tokens to replace the structure of source files. Then, they adopt mainstream deep learning models (LSTM, DBN, CNN, etc) to learn and represent defects from node tokens of program files [20]. However, this kind of research has three main limitations as follows.

(1) **Size of the dataset is small.** The dataset used was relatively small in previous studies, and the semantic information learned by the model was insufficient, limiting its generalization ability. Moreover, the class imbalance problem caused by dataset defects is severe, which can lead to overfitting or information loss in the dataset.

(2) **Utilization of classification labels is more exploratory.** In downstream tasks, researchers focused more on training models for prediction using mainstream deep learning models, such as Bi-LSTM, TCA+, and GNN, etc. However, they rarely focused on the labels generated after classification, whereas they neglected the fact that classification labels can enhance the diversity of data.

(3) **Representation of the code is limited.** The bag-of-words models are directly used in vector conversion in traditional research methods. These models only count the frequency of words and lack a complete representation of the syntax and structure between words. Therefore, they may overlook contextual semantics.

Our goal is to fully leverage the semantic and structural information in the source code, explore the predicted classification labels, and expand the diversity of the original data. We propose a defect prediction method called Ada4DP, based

[*] Chenqiang Shen is the corresponding author.

on deep learning and dual learning models. This method combines the CodeT5 language pretraining model to extract semantic and structural information from the source code and incorporates the Bi-GRU model with attention and self-attention mechanisms to capture the contextual information in the source code to the maximum extent. Additionally, it utilizes adversarial perturbations with pre-prediction labels to augment the dataset and incorporates the generated labels after prediction. The interaction between the dual learning reverse classifier and forward classifier, enhances the accuracy of the predicted data to improve the overall defect prediction performance.

We conducted experiments by comparing Ada4DP with six state-of-the-art baseline methods and evaluated their performance on the Java open-source dataset, which has been widely used in previous studies. The experimental results show that Ada4DP can improve the performance of F1 measurement by an average of 11.6%.

The main contributions are as follows:

(1) We propose a novel method, Ada4DP, which utilize secondary prediction with classified labels. Using the predicted labels and applying a reversal operation, we employ both forward and backward classifiers to achieve multidimensional prediction.

(2) We propose the CBEncoder for converting source code into vectors. Combining information such as code structure, relationships, and semantics allows us to delve into the semantic aspects of source code. Then, Ada4DP captures a more comprehensive understanding of the code, including its finer details, which serves as a solid foundation for improved predictions in subsequent steps.

(3) We conducted extensive experiments on 10 open-source Java projects, demonstrating that our method, Ada4DP, outperforms compared baselines in both WPDP and CPDP.

2. Background

With the advancement of artificial intelligence (AI) technology, deep learning has emerged as the predominant approach to address the challenge in the Software defect prediction domain. Specifically, AI models such as CodeT5 [29] and Bi-GRU [33] are used to tackle the unique challenges posed by code text, and these models have been proven effective in capturing the inherent complexity and dependencies in software development. In addition, the Dual learning method has emerged as a promising technique to improve the accuracy and reliability of software defect prediction models.

2.1 Pre-trained model CodeT5

CodeT5 is a powerful natural language processing model proposed by Wang et al. [29]. It is based on the T5 encoder-decoder model, specifically designed for processing code text. When faced with special syntax structures and symbols in code, it uses multiple strategies to generate more accurate and meaningful vector representations(code block division, AST representation, special symbol processing). In defect prediction tasks, the semantic information of the code greatly affects the prediction results. CodeT5 uses the ”SubWordTextEncoder” tokenizer module to convert source code to tokens. It gradually splits source code words and fully preserves the structure and semantic information of the source code. Therefore, it is widely used in various defect prediction tasks.

2.2 Dual learning

Dual learning [9] is an emerging machine learning approach that aims to address real-world challenges such as imbalanced dataset distributions, and noisy, and unlabeled data. In contrast to traditional data processing methods, Dual learning utilizes bidirectional classifiers for supervised learning to generate samples with diverse characteristics. This method effectively addresses the issue of data imbalance and enhances the diversity of the original dataset. Additionally, dual learning leverages techniques such as error correction and knowledge extraction to detect and learn from classification problems, thus improving classification accuracy, improving overall data quality, and providing a more balanced and stable dataset for subsequent predictions. In summary, dual learning is a powerful machine learning method that offers effective solutions to dataset challenges in the field of software engineering.

2.3 Research Motivation

motivation 1: Since the PROMISE dataset is of smaller capacity, using data augmentation strategies can effectively increase the data volume and improve the predictive performance of the model. Specifically, the severe class imbalance problem in the dataset is much of a concern. Most studies address this issue through sampling, which may lead to overfitting or loss of information in the dataset. Therefore, we propose that data augmentation can effectively enhance data diversity and reduce prediction errors.

motivation 2: As Bi-GRU is a simplified version of LSTM that optimizes both the computational cost and performance of the Bi-LSTM model, it is more advantageous to use Bi-GRU. In previous research, Chanathip and his colleagues took full advantage of this feature for defect prediction and localization. In this research question, we aim to demonstrate that when self-attention is incorporated into Bi-GRU, it can better capture the long-term dependency relationships in the code and grasp more comprehensive contextual information.

motivation 3: Considering that some data in the dataset lack corresponding labels, inspired by the idea of dual learning in the field of image classification as proposed by Xu et al. [31], prognostic labels are used as pseudo-labels. These pseudo-labels are input into another sub-model for mutual learning, improving prediction accuracy. Additionally, the credibility of pseudo-labels is validated to ensure prediction quality, even in cases where the predicted labels are incorrect, the sub-models can continuously correct each other through mutual learning. Therefore, this study aims to demonstrate the effectiveness of dual learning in the field of defect prediction.

3. Our method

In this section, we introduce the architecture of Ada4DP. As shown in Figure 1, Ada4DP consists of three main parts: Data

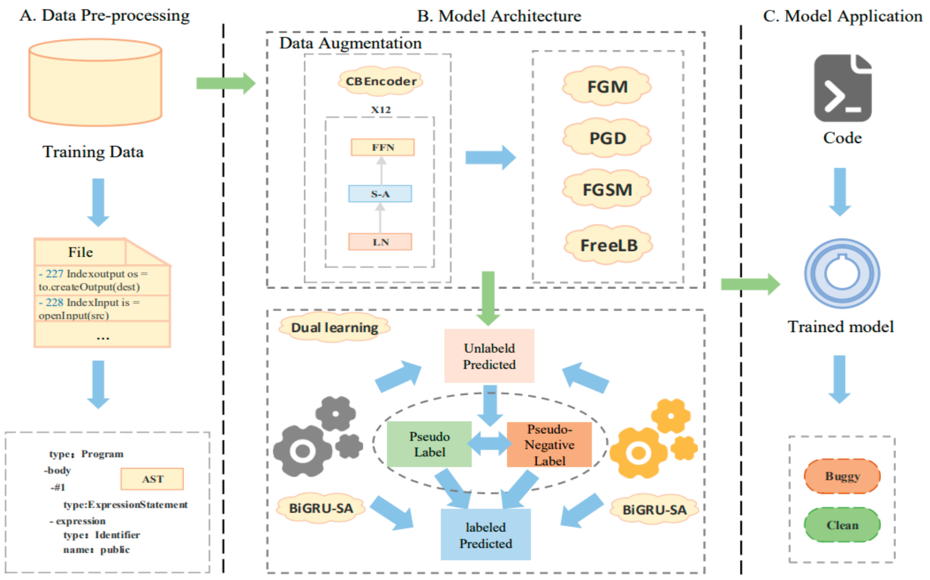


Figure 1. The overall framework of Ada4DP

Pre-processing, Model Architecture, and Model Application. Next, we will provide detailed explanations for each part.

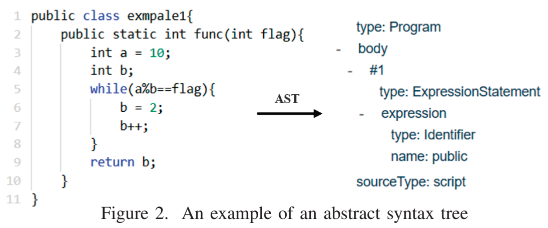
3.1 Approach Overview

Our approach begins with data collection and preparation. It aims to generate data that includes source code files with file-level and function-level labels (e.g., files containing post-release defects or lines modified to fix defects). First, for each source code file, we perform several preprocessing steps, including code abstraction and vocabulary management. Second, we generate vectors to represent the source code. Thirdly, we employ a bidirectional Bi-GRU network with self-attention mechanisms [24] to conduct deep semantic mining. Then we capture the intrinsic characteristics of the source code for defect prediction. Finally, using dual learning, we leverage the interaction between the forward classifier and the backward classifier based on the predicted labels to obtain the final prediction results.

3.2 Source Code Preprocessing

The goal of data pre-processing is to extract the syntax, semantics and contextual code features from each file, and then we generate the corresponding tokens.

Abstract Syntax Tree (AST). [3] For each Java file, we use an abstract syntax tree to parse the program file and obtain an AST token sequence. In this approach, as shown in Figure 2, we extract various types of tokens, such as method invocation nodes, class definition nodes, various declaration nodes, control flow nodes, and so on. Based on these extracted tokens, we construct a token set.



```
public class expale1{
    public static int func(int flag){
        int a = 10;
        int b;
        while(a%b==flag){
            b = 2;
            b++;
        }
        return b;
    }
}
```

```
type: Program
- body
  - #1
     type: ExpressionStatement
   - expression
       type: Identifier
       name: public
sourceType: script
```

Figure 2. An example of an abstract syntax tree

CBEncoder. To match the input of our model, we represent the token set extracted from Java files as vectors. For vector representation of the extracted tokens, we utilize the CodeT5 architecture and design the CBEncoder module for encoding representation. The CBEncoder module consists of 12 encoders, as illustrated in Figure 1. Each encoder includes multi-head self-attention mechanisms and feed-forward neural networks to process the input sequence and obtain higher-level representations.

1) Convert the set of tokens into textual representations. We use the AST traversal algorithm. Then the token set is transformed into text representations by converting the nodes in order. These text representations are pushed into the word embedding layer in the CBEncoder. The AST nodes represented as text are further processed using the SubWordTextEncoder, which employs the BPE (Byte Pair Encoding) algorithm for subword tokenization. It initializes the input word embedding matrix based on the statistics of the token set, which is named X.

2) Multi-head Self-Attention Mechanism [24]. For the embedding matrix of the input word $X = |x_1, x_2, ..., x_n|$, the

multihead self-attention mechanism is introduced as the core mechanism of the encoder. Each head of self-attention involves linear transformations of queries Q_i, keys K_i, and values V_i, as shown in formula 1.

$$\begin{cases} Q_i = XW_i^q \\ K_i = XW_i^k \\ V_i = XW_i^v \end{cases} \tag{1}$$

where W_i^q, W_i^k, and W_i^v are the linear transformation matrices corresponding to head i. Through dot product and scaling operations on Q, K, and V, attention weights Z_i are obtained, which can be represented as:

$$Z_i = Attention(Q_i, K_i, V_i) = softmax(\frac{Q_i k_i^T}{\sqrt{d_k}})V_i, \tag{2}$$

where d_k represents the dimension of the key vectors, and K^T denotes the transpose of matrix K. Finally, the outputs of all heads are linearly transformed and concatenated to obtain the final representation Y, which can be represented as:

$$Y = MultiHead(Q, K, V)W^o = concat(Z_1, Z_2, ..., Z_n)W^o \tag{3}$$

3) Feedforward Neural Network. After the generation of Y through the multi-head self-attention mechanism, the encoder utilizes a feed-forward neural network for further processing and combining the representations. This layer is composed of two fully connected layers, with the intermediate step employing the non-linear activation function ReLU. Its purpose is to map the input Y to a higher-dimensional feature space, thereby transforming the complexity and contextual information of the original code into more expressive representation vectors Y.
4) Residual connection and normalization. After obtaining the representation vectors, residual connections and normalization operations are introduced to improve the overall performance and stability of the encoder. This involves connecting the obtained representation vectors Y_f with the original input through a residual connection.

$$Y_f = \sigma(Y + X + b_f), \tag{4}$$

where b_f represents the learned parameters and $\sigma(\cdot)$ denotes the activation function. In addition, for each residual connection, layer normalization is introduced to normalize the outputs of individual sub-layers as :

$$Y_o = LayerNorm(Y_f + c_f) = \frac{\gamma(Y_f - \bar{Y_f})}{\sqrt{S^2 + \epsilon}} + \beta, \tag{5}$$

where c_f represents the learned parameters, γ and β represent the learned scaling and shifting parameters, ϵ represents a parameter for numerical stability, and S^2 represents the variance of Y_o. The process is trained iteratively to generate the final vector representation, maximizing the preservation of the source code's semantics and structural information for subsequent predictions.
Adversarial learning methods. [23] Considering the final representations generated by the dataset, four adversarial learning methods (FGM, PGD, FGSM, and FreeLB) were selected to explore enhancement effects to achieve higher training effectiveness of the model. To verify the augmentation effect, experiments were conducted on the four adversarial learning methods to determine the best one for use in this approach.

3.3 Learning the Deep Structure of Source Code

We employ a self-attention mechanism to learn the deeper structure of the source code. Specifically, we utilize a two-layer Bi-GRU network consisting of two layers of forward and backward GRU, as shown in Figure 3 Assume that a source code file F has a sequence of tokenized code $Y_O = |y_{i1}, y_{i2}, ..., y_{in}|$, where y_{ik} represents the code token at function level, for $i \in [1, n]$.

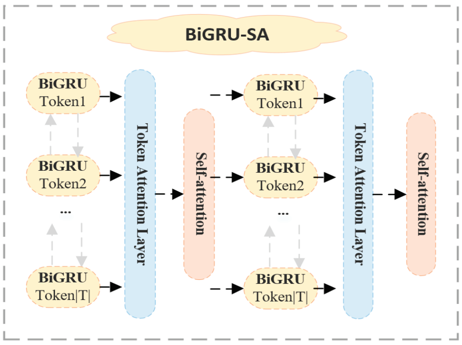


Figure 3. The framework of BIGRU-SA module

We utilize a two-layer Bi-GRU to summarize the contextual information of tokens in both directions. Through multiple layers of stacking, we progressively capture higher-level semantics and contextual information, while addressing the issues of gradient explosion and vanishing. To capture the contextual information, Bi-GRU includes both a forward GRU and a backward GRU, where the forward GRU reads from left to right, as:
$GRU \overrightarrow{h}_{it} = \overrightarrow{GRU}(y_{it}), t \in [1, n]$, the backward GRU reads from right to left, as:
$GRU \overleftarrow{h}_{it} = \overleftarrow{GRU}(y_{it}), t \in [1, n]$ By connecting the forward hidden states with the backward hidden states, as:
$h_{it} = [\overrightarrow{h}_{it} \oplus \overleftarrow{h}_{it}]$ so we obtain the sequence of hidden states for a given token.
Self-Attention. Based on empirical evidence, it is evident that not all representations contribute equally to the hidden state sequence. We employ the self-attention mechanism to emphasize the internal correlations and dependencies within the sequence. Firstly, We create a query vector $Q \in R^{T \times 2D}$, key vector $K \in R^{T \times 2D}$, and value vector $V \in R^{T \times 2D}$, where T represents the number of time steps and $2D$ represents the dimension of the hidden states of the forward and backward GRUs. Then, we calculate the attention matrix $A = softmax(\frac{QK^T}{\sqrt{D}})$, which represents the weighted combination of the value vector V, resulting in the final context vector C_1.
Fully-Connected Layer(FC). Utilizing a two-layer Bi-GRU for processing, we perform the aforementioned operations

for the context vector C_1 generated from one layer. Firstly, we derive hidden state representations from the forward and backward GRUs. Then, we apply the self-attention mechanism to further capture and generate contextual information at a higher level. The final vector representation C_2 is fed into a multilayer perceptron (MLP) to generate a prediction score $h = W_f \times C_2 + b_h$, where W_f represents the weight matrix connecting the layers and hidden layers, and b_h represents the bias value. Finally, the prediction score is used to determine the predicted labels based on a threshold.

Dual learning(DL). After generating predictions, we further utilize the generated labels to explore the correlation between the predicted labels and the original labels. We augment the original dataset by applying dual learning, wherein the generated labels are inverted, i.e., 0 becomes 1 and 1 becomes 0, resulting in the creation of a new dataset. We then input this new dataset into the constructed prediction model for reverse training, aiming to generate the final prediction results. This process allows us to optimize the predictions by leveraging the association between the predicted and original labels.

4. Experiment Set

This study aims to evaluate our method Ada4DP through extensive experiments on inter-project prediction (WPDP) and cross-project prediction (CPDP) in software projects. To ensure that the training effect of the experiments does not exhibit significant bias, each implementation is repeated 10 times, and the measurement results of the average F1 are taken.

4.1 Evaluation Metrics

We used the following four widely used performance metrics for evaluation:

Accuracy: $Accuracy = \frac{TP+TN}{TP+TN+FN+FP}$. Accuracy refers to the proportion of correctly predicted defects to all defects. TP refers to the number of true positive instances. Similarly, TN refers to the number of true negative instances; FN refers to the number of false negative instances; FP refers to the number of false positive instances. TP+TN+FN+FP means the total number of defects.

Precision: $Precision = \frac{TP}{TP+FP}$. Precision refers to the proportion of relevant negative defects among the retrieved defects that are predicted as negative.

Recall: $Recall = \frac{TP}{TP+FN}$. Recall refers to the proportion of relevant defects that are predicted as negative among the retrieved negative defects.

F1 score: $F1score = 2 \times \frac{Precision \times Recall}{Precision+Recall}$. F1 score refers to the geometric mean of precision and recall representing a balance between the two.

4.2 Dataset

To compare and reproduce experiments with more mainstream defect prediction methods, we have selected publicly available historical datasets from the PROMISE repository. These datasets are widely used in many current defect prediction methods. We have chosen 10 open-source JAVA projects from the repository, which include project names, project details, project versions, and project class information. Data collection was performed based on version numbers and types. We obtained the source code for each file from the open-source website GitHub and have made the details of these projects publicly available as represented in TABLE I.

TABLE I
DATA DESCRIPTION

DataSet	version	Instances	FP instances	Targets
ant	1.5	293	32	37
	1.6	351	92	37
	1.7	745	166	37
camel	1.2	608	216	38
	1.4	872	145	38
	1.6	965	188	38
jEdit	3.2	272	90	37
	4	306	75	37
	4.1	312	79	37
log4j	1	135	34	39
	1.1	109	37	39
lucene	2	195	91	39
	2.2	247	144	39
	2.4	340	203	39
xalan	2.4	823	110	38
	2.5	803	387	38
xerces	1.2	440	71	39
	1.3	453	69	39
ivy	2	352	40	39
synapse	1	157	16	39
	1.1	222	60	39
	1.2	256	86	39
poi	1.5	237	141	38
	2.5	385	248	38
	3	442	281	38

4.3 baselines

To evaluate the performance of the proposed Ada4DP model for defect prediction, we employed a combination of manual and automated feature extraction techniques, specific to the current stage, for both within-project and cross-project defect prediction. Additionally, we replicated the baseline models we selected, following the experimental details mentioned in the respective papers, to conduct a comprehensive comparison against the baselines.

For WPDP (Within-Project Defect Prediction) settings, we compared against the following baseline models:

DBNs [27]: An advanced deep learning method that automatically learns program semantic representations from source code. Extracts the source code into an AST and utilizes Deep Belief Networks (DBN) to automatically learn semantic features for defect prediction.

PROMISE-DP [28]: A classic prediction method that utilizes the traditional set of 20 metrics from the PROMISE dataset for defect prediction, employing ADTree for the prediction task.

DP-LSTM [6]: A prediction method based on deep neural networks based on long- and short-term memory (LSTM), which utilizes AST tokens to represent source files and uses this network for feature extraction and prediction.

DP-CNN [14]: It is a prediction method based on Convolutional Neural Networks (CNN), which utilizes token sequences extracted from AST. It employs CNN to extract defect features and combines manual and automatic features for prediction.

GH-LSTM [26]: A method that combines gate units to automatically extract features and utilizes the gate unit mechanism for prediction.

DTL-DP [4]: A method based on deep transfer learning, which employs visual representation by converting source files into images for prediction processing.

For CPDP (Across-Project Defect Prediction) settings, since some models like PROMISE-DP and GH-LSTM are not applicable for CPDP prediction, we compared against the following baseline models:

DBNs-CP [27]: It is a semantic variation of source code information that, after being trained by DBN, generates corresponding semantic feature information for projects.

TCA+ [17]: It is a method of deep transfer learning that effectively reduces the differences in distribution between data from the same project. It is a relatively novel technique for CPDP.

To obtain training and testing data for WPDP and CPDP, we followed the rules proposed by Wang et al. For WPDP, we performed predictions using consecutive versions of the same project from the dataset, using the older version as training data and the newer version as testing data. For CPDP, we selected random versions from each project, following the selection of test project pairs as described by Wang et al.

5. RESULT

5.1 RQ1: How does Ada4DP in comparison with other existing baseline models in WPDP?

We compared Ada4DP with six current baseline methods in WPDP. Following the guidance from previous experiments, we selected 14 sets of WPDP experiments, where each set utilized different versions of the same project, with historical versions as training data and newer versions as testing data.

Table II displays the F1 of the WPDP defect prediction experiments, with the highest F1 score highlighted for each experiment. To mitigate the effects of randomness, we conducted 10 experiments and averaged the results to reduce significant value fluctuations. On average, our Ada4DP model achieved an F1 of 68.86%, while the baseline methods DBNs, PROMISE-DP, DP-LSTM, DP-CNN, GH-LSTM, and DTL-DP achieved F1 scores of 59.99%, 49.76%, 52.96%, 57.26%, 57.69%, and 64.14%, respectively. Our method performed the best among the selected WPDP experiments, indicating that Ada4DP is competitive and significantly improves the performance of WPDP tasks compared to the mainstream baseline models. Furthermore, we employed the Wilcoxon signed rank test for further comparison. Specifically, if our method outperformed a specific baseline model, and the p-value of the Wilcoxon signed-rank test was less than 0.05, the difference was statistically significant and could not be ignored. In our case, all p-values were less than 0.05, validating that our method outperformed other baseline models in terms of performance. In addition, we visualize the data in a violin plot, as shown in Figure 4. The Ada4DP method demonstrates a concentrated probability density in the range of 65%-75% during WPDP, while other methods exhibit sparse probability densities. This indicates the higher reliability of our method's predictions. Furthermore, both the maximum and minimum values of our method surpass those of other methods, suggesting that our method outperforms others in prediction.

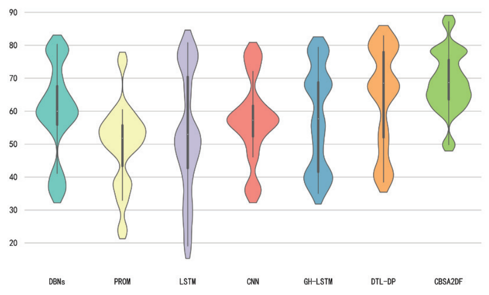


Figure 4. The probability density of F1 in all violins in WPDP

5.2 RQ2: How to compare the performance of different methods used in the data augmentation phase?

To address RQ2, this study aims to explore the enhancement effect of adversarial learning on the Ada4DP method and evaluate its effectiveness. It is worth noting that in adversarial learning techniques, there are four common ways to effectively expand and adjust datasets. To assess the effectiveness of these techniques for this method, different comparative experiments were conducted in the WPDP experiment, setting FGM, PGD, FGSM, and FreeLB methods, as shown in Table III, where the best results are highlighted in bold.

In contrast to different adversarial learning techniques, FGM achieved the best results in experiments across nine projects, with an average F1 score improvement of 0.28%, 0.98%, and 0.59%. Analysis of this situation indicates that FGM is more suitable for this method. Moreover, in terms of its technical operation, FGM has advantages over FGSM, PGD, and FreeLB methods in calculating adversarial perturbations, with lower computational costs. Therefore, the FGM adversarial learning method is more suitable for the Ada4DP method, further enhancing its effectiveness.

Additionally, to verify the performance of FGM in software defect prediction, ablation experiments were conducted on WPDP, as shown in Figure 5. Overall, using FGM technology can effectively improve performance. In the WPDP project, where the dataset distributions are relatively similar, the F1 score improved by 2.16%.

5.3 RQ3:Does the integration of the self-attention mechanism within the Bi-GRU module higher efficacy?

To address RQ3, this study aims to explore the effectiveness of integrating the self-attention mechanism into the Bi-GRU as the base model and evaluate its efficacy. It is worth noting

TABLE II
F1 OF ADA4DP AND OTHER BASELINE METHODS FOR WPDP

project	version	DBNs	PROM	LSTM	CNN	GH-LSTM	DTL-DP	Ada4DP
camel	1.2->1.4	**78.5**	37.3	32.3	46.1	41.7	40.6	59.3
	1.4->1.6	37.4	39.1	40.0	50.8	45.4	38.5	**62.4**
jedit	3.2->4.0	57.4	55.6	47.2	56.4	56.2	49.4	**61.4**
	4.0->4.1	61.5	54.6	49.0	58.0	63.5	**68.7**	64.7
log4j	1.0->1.1	70.1	58.7	53.9	63.2	74	**68.8**	64.8
lucene	2.0->2.2	65.1	50.2	75.1	76.1	73.7	**78.3**	73.6
	2.2->2.4	77.3	60.5	75.1	72.1	74.6	77.6	**78.7**
xalan	2.4->2.5	59.5	51.8	65.7	60.1	66.1	**79.4**	78.5
xereces	1.2->1.3	41.1	23.8	26.8	37.4	35	**82**	68.7
ivy	1.4->2.0	35	32.9	19.1	34.7	39.6	**82.9**	49.8
synapse	1.0->1.1	54.4	47.6	45.4	53.9	48.1	54.7	**66.3**
	1.1->1.2	58.3	53.3	53.3	55.6	60.9	65.9	**71.1**
poi	1.5->2.5	64	55.8	80.8	58.9	**84.4**	68.5	77.5
	2.5->3.0	80.3	75.4	77.7	78.4	82.3	42.6	**87.2**
average		60.0	49.8	53.0	57.3	60.4	64.1	**68.9**

TABLE III
F1 OF FOUR ADVERSARIAL LEARNING METHODS FOR WPDP

project	version	FGM	PGD	FGSM	FreeLB
camel	1.2->1.4	59.3	**64.5**	59	63.9
	1.4->1.6	**62.4**	59.3	59.3	59.5
jedit	3.2->4.0	61.4	**64.5**	63.3	63.6
	4.0->4.1	**64.7**	64.6	62.9	63.9
log4j	1.0->1.1	64.8	**65.6**	64.6	64.9
lucene	2.0->2.2	**73.6**	71.4	71.7	71.7
	2.2->2.4	**78.7**	76.7	75.4	75.9
xalan	2.4->2.5	**78.5**	76.8	77	76.7
xereces	1.2->1.3	**68.7**	67.2	68.4	67.5
ivy	1.4->2.0	49.8	**52**	51.5	51.1
synapse	1.0->1.1	66.3	**67.8**	66.6	67.3
	1.1->1.2	**71.1**	70.3	70.6	69.8
poi	1.5->2.5	**77.5**	77.2	76.2	76.7
	2.5->3.0	**87.2**	82.4	83.8	83.3
average		**68.86**	68.58	67.88	68.27

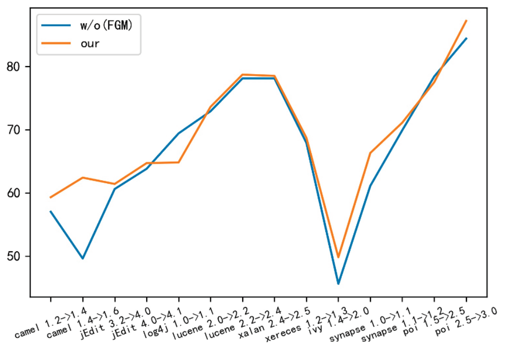


Figure 5. Performance of FGM on WPDP

that the Bi-GRU model exhibits significant effectiveness in capturing the characteristics of long sequences in source code, but lacks the ability to capture internal correlations and dependencies. To evaluate the effectiveness of the self-attention mechanism, corresponding ablation experiments were conducted on WPDP, with results shown in Figure 6, where w/o SA denotes the use of the Ada4DP method without the self-attention mechanism.

Comparative experiments on all datasets indicate that the model improved with the self-attention mechanism achieves the best results. It is evident that compared to the original projects, the mean F1 scores achieved improvements of 3.69%, indicating certain advantages of the self-attention mechanism in exploring internal correlations and dependencies in source code semantics.

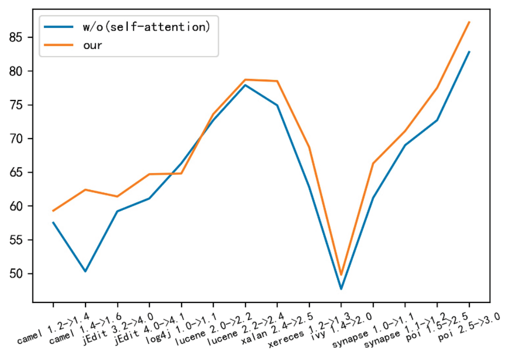


Figure 6. Performance of self-attention mechanism on WPDP

5.4 RQ4:Does the use of Dual Learning result in greater overall performance of the model?

To address RQ4, this method aims to explore the effect of using prognostic labels in conjunction with dual learning on Ada4DP and evaluate its effectiveness. It is worth noting that dual learning can utilize predicted labels to generate pseudo-labels, and after reversing them, input them into sub-models for mutual learning, thereby improving prediction performance. To assess the effectiveness of dual learning, ablation experiments were conducted in WPDP, and the results are shown in Figure 7.

Comparative experiments on all datasets indicate that models improved with dual learning modules achieve the best results. It is evident that compared to the original projects, there are improvements in F1 score mean values of 5.13%, demonstrating the advantage of using prognostic labels as pseudo-labels for overall prediction accuracy enhancement.

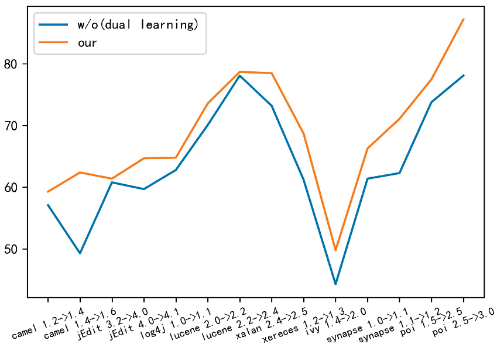


Figure 7. Performance of Dual Learning on WPDP

6. Discussions

6.1 Analysis on the CPDP

In this chapter, we will be analyzing the performance of this method in cross-project defect prediction. Following the rules proposed by Wang et al., we selected 22 cross-project experiments consistent with the same five popular baseline models. Each experiment picks 2 versions from different projects, one as the training version and the other as the testing version. All experimental settings are consistent with the RQ1 processing method.

As shown in Figure 8, our method exhibits stable performance throughout the entire cross-project experiment process, with a smaller degree of data dispersion. The quartile distribution of this plot is relatively even, meaning that the difference in experimental effects is relatively small, and the performance is better than other methods.

Furthermore, to ensure the effectiveness of the proposed components, experiments were also conducted on CPDP to integrate them, with the representation of W/O consistent with the aforementioned experimental section. According to the performance shown in Table IV, the proposed components still demonstrate effectiveness in CPDP experiments, achieving improvements of 3.1%, 3.7%, and 3.4%, respectively.

6.2 Threats to Validity

Internal validity: The main challenges are experimental errors and the selection of baseline methods. One major challenge

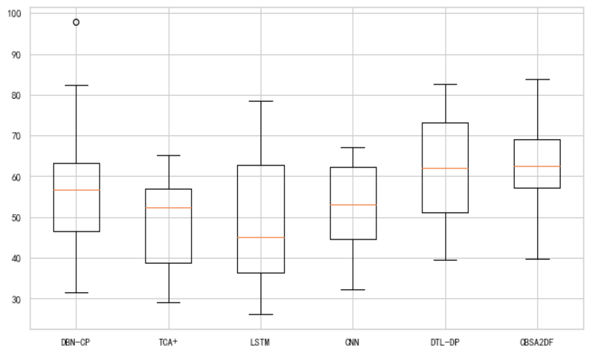


Figure 8. The probability density of F1 in all boxes in CPDP

TABLE IV
ABLATION EXPERIMENTS FOR CPDP

Source	Target	w/o FGM	w/o SA	w/o DL	Ada4DP
ant1.6	**camel1.4**	36.9	37	36	**39.8**
jEdit4.1	**camel1.4**	55.2	55.6	52.6	**57.7**
camel1.4	**ant1.6**	62.6	65.7	63.5	**68.4**
poi3.0	**ant1.6**	57.5	57.6	58.8	**60.2**
camel1.4	**jEdit4.1**	58.9	61.3	59.2	**64.8**
log4j1.1	**jEdit4.1**	60.9	65.5	65.6	**68.3**
jEdit4.1	**log4j1.1**	74.7	72.8	73.7	**75.4**
lucene2.2	**log4j1.1**	74.5	71.1	71.6	**75.6**
lucene2.2	**xalan2.5**	65.0	**65.4**	62.1	64.7
xerces1.3	**xalan2.5**	62.1	63.1	62.6	**69.8**
xalan2.5	**lucene2.2**	77.2	74.9	75.4	**79**
log4j1.1	**lucene2.2**	60.1	59.4	60.9	**62.1**
xalan2.5	**xerces1.3**	39.4	37.4	39.5	**40.3**
ivy2.0	**xerces1.3**	46.9	46.7	46.8	**47.9**
xerces1.3	**ivy2.0**	40.8	43.0	**44.8**	43.9
synapse1.2	**ivy2.0**	41.7	41.3	40.4	**42.3**
ivy1.4	**synapse1.1**	56.7	53.2	55.1	**56.9**
poi2.5	**synapse1.1**	60.3	59.5	60.1	**62.5**
ivy2.0	**synapse1.2**	55.6	55.7	54.4	**59.8**
poi3.0	**synapse1.2**	64.1	63.7	64.9	**68.1**
synapse1.2	**poi3.0**	78.9	80.5	76.0	**83.9**
ant1.6	**poi3.0**	76.2	62.8	75.5	**82.9**
AVG		59.4	58.8	59.1	**62.5**

is the inherent randomness in deep learning models and the unknown details of training. Despite repeating each implementation ten times and averaging the results, there may still be significant fluctuations. Furthermore, to ensure the effectiveness of the proposed Ada4DP method, we strictly adhered to the implementation details mentioned in the original papers for baseline methods. However, new implementations may not fully reflect all original.

External validity: The main limitation stems from selecting the PROMISE dataset. This method achieved good performance only on the PROMISE dataset, and if tested on datasets from different repositories, it may yield different experimental results. Furthermore, our model was only tested on the JAVA programming language and lacked experiments on other languages such as C or Python. This requires validation in future work.

Construct validity: The main limitation arises from the evaluation metric, F1. F1, being the primary evaluation metric in this experiment, has been widely used in previous software defect prediction studies.

7. Related Works

In this section, we analyze the related work on software defect prediction and data augmentation. After examining the relevant studies, we emphasize the novelty of our research.

7.1 Semantic mining

In traditional software defect prediction, most researchers have focused on handcrafted features [18]. These features are used as inputs to machine learning models and are typically categorized into static features (e.g., lines of code, code complexity [21]) and process features (e.g., instantaneous behaviors during development) [12].

Considering the limitations of handcrafted features, which may not fully capture the semantic information of source code, Xu et al [30] proposed a new software defect prediction method that extracts structured semantic information from source code through AST and incorporates contextual information. Based on these handcrafted or automatically extracted features [32], researchers have directed their attention toward downstream classification tasks. For instance, Jiaojiao et al. [1] proposed a three-stage weighting framework with multiple sources for defect prediction. This method involves implementing source selection strategies in the first stage, applying transfer techniques in the second stage, and utilizing multi-source data utilization scheme in the third stage.

Our approach differs from previous research. Instead of using traditional word embedding or bag-of-words models for mapping data to vectors, we have designed an encoding module for high-quality vector transformation. Additionally, we incorporate a dual-layer attention mechanism that takes a global perspective and explores all the information in the text, maximizing the capture of the original code semantics.

7.2 Data Augmentation

Data augmentation has been widely applied in the field of Natural Language Processing (NLP), particularly in software defect prediction [19]. Xiao Song et al. [11] utilized unsupervised domain adaptation techniques to leverage pseudo-labeling and effectively enhance information using limited data for data augmentation, aiming to alleviate the problem of data scarcity. Currently, researchers mostly concentrate on addressing class imbalance issues in software defect prediction through methods such as oversampling, undersampling [13], and SMOTE [23]. However, our approach differs from previous approaches. Instead of focusing solely on the data imbalance problem, we leverage the predicted labels after training and employ a dual learning mechanism that incorporates both reverse classification and forward classification. Additionally, considering the limited size of the selected dataset, we performed data augmentation using the FGM method to perturb the feature vectors of the original dataset, generating new instances.

8. Conclusion

Our work focus on two key aspects. Firstly, we introduce Ada4DP method which is the first approach to utilize labeled data for secondary prediction and to be enabled to enhance learning of defect features. Secondly, we extend Ada4DP method through experiments on WPDP and CPDP, demonstrating its superiority over existing methods. Our Ada4DP incorporates FGM perturbation expansion on the existing dataset while leveraging the CBEncoder, a novel semantic mining encoder, for vector transformation. Moreover, by incorporating a two-layer attention mechanism into the deep learning BiGRU model, we facilitated semantic defect learning and prediction. Ada4DP achieves an overall improvement of 4.8% and 0.7% in performance.

References

[1] Jiaojiao Bai, Jingdong Jia, and Luiz Fernando Capretz. A three-stage transfer learning framework for multi-source cross-project software defect prediction. *Information and Software Technology*, 150:106985, 2022.

[2] V.R. Basili, L.C. Briand, and W.L. Melo. A validation of object-oriented design metrics as quality indicators. *IEEE Transactions on Software Engineering*, 22(10):751–761, 1996.

[3] Ziyi Cai, Lu Lu, and Shaojian Qiu. An abstract syntax tree encoding method for cross-project defect prediction. 7:170844–170853. Conference Name: IEEE Access.

[4] Jinyin Chen, Keke Hu, Yue Yu, Zhuangzhi Chen, Qi Xuan, Yi Liu, and Vladimir Filkov. Software visualization and deep transfer learning for effective software defect prediction. In *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering*, pages 578–589. ACM.

[5] Shyam R Chidamber and Chris F Kemerer. A metrics suite for object oriented design. *IEEE Transactions on software engineering*, 20(6):476–493, 1994.

[6] Hoa Khanh Dam, Trang Pham, Shien Wee Ng, Truyen Tran, John Grundy, Aditya Ghose, Taeksu Kim, and Chul-Joo Kim. A deep tree-based model for software defect prediction.

[7] Geanderson Esteves, Eduardo Figueiredo, Adriano Veloso, Markos Viggiato, and Nivio Ziviani. Understanding machine learning software defect predictions. *Automated Software Engineering*, 27(3-4):369–392, 2020.

[8] N.E. Fenton and M. Neil. A critique of software defect prediction models. *IEEE Transactions on Software Engineering*, 25(5):675–689.

[9] Yaroslav Ganin, Evgeniya Ustinova, Hana Ajakan, Pascal Germain, Hugo Larochelle, François Laviolette, Mario March, and Victor Lempitsky. Domain-adversarial training of neural networks. *Journal of machine learning research*, 17(59):1–35, 2016.

[10] Görkem Giray, Kwabena Ebo Bennin, Ömer Köksal, Önder Babur, and Bedir Tekinerdogan. On the use of deep learning in software defect prediction. 195:111537.

[11] Xiaosong Huang, Yifan Wu, Hongyi Liu, Ying Li, Hao Yu, Dadi Guo, and Zhonghai Wu. UDA-DP: Unsupervised domain adaptation for software defect prediction. In *2023 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*, pages 308–318. ISSN: 2640-7574.

[12] Chaiyakarn Khanan, Worawit Luewichana, Krissakorn Pruktharathikoon, Jirayus Jiarpakdee, Chakkrit Tantithamthavorn, Morakot Choetkiertikul, Chaiyong Ragkhitwetsagul, and Thanwadee Sunetnanta. JITBot: an explainable just-in-time defect prediction bot. In *Proceedings of the 35th IEEE/ACM International Conference on Automated Software Engineering*, pages 1336–1339. ACM.

[13] Dae-Kyoo Kim and Yeasun K Chung. Addressing class imbalances in software defect detection. *Journal of Computer Information Systems*, pages 1–13, 2023.

[14] Jian Li, Pinjia He, Jieming Zhu, and Michael R Lyu. Software defect prediction via convolutional neural network. In *2017 IEEE international conference on software quality, reliability and security (QRS)*, pages 318–328. IEEE, 2017.

[15] Jingyu Liu, Jun Ai, Minyan Lu, Jie Wang, and Haoxiang Shi. Semantic feature learning for software defect prediction from source code and external knowledge. 204:111753.

[16] Tim Menzies, Zach Milton, Burak Turhan, Bojan Cukic, Yue Jiang, and Ayşe Bener. Defect prediction from static code features: current results, limitations, new approaches. 17(4):375–407.

[17] Jaechang Nam, Sinno Jialin Pan, and Sunghun Kim. Transfer defect learning. In *2013 35th International Conference on Software Engineering (ICSE)*, pages 382–391. ISSN: 1558-1225.

[18] Chao Ni, Wei Wang, Kaiwen Yang, Xin Xia, Kui Liu, and David Lo. The best of both worlds: integrating semantic features with expert features for defect prediction and localization. In *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pages 672–683. ACM.

[19] Chao Ni, Xin Xia, David Lo, Xiaohu Yang, and Ahmed E. Hassan. Just-in-time defect prediction on JavaScript projects: A replication study. 31(4):1–38.

[20] Safa Omri and Carsten Sinz. Deep learning for software defect prediction: A survey. In *Proceedings of the IEEE/ACM 42nd International Conference on Software Engineering Workshops*, pages 209–214. ACM.

[21] Chanathip Pornprasit and Chakkrit Kla Tantithamthavorn. DeepLineDP: Towards a deep learning approach for line-level defect prediction. 49(1):84–98. Conference Name: IEEE Transactions on Software Engineering.

[22] Dilini Rajapaksha, Chakkrit Tantithamthavorn, Jirayus Jiarpakdee, Christoph Bergmeir, John Grundy, and Wray Buntine. SQAPlanner: Generating data-informed software quality improvement plans. 48(8):2814–2835. Conference Name: IEEE Transactions on Software Engineering.

[23] Stefan Richter, Colin Neil Jones, and Manfred Morari. Computational complexity certification for real-time MPC with input constraints based on the fast gradient method. 57(6):1391–1403. Conference Name: IEEE Transactions on Automatic Control.

[24] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc.

[25] Zhiyuan Wan, Xin Xia, Ahmed E Hassan, David Lo, Jianwei Yin, and Xiaohu Yang. Perceptions, expectations, and challenges in defect prediction. *IEEE Transactions on Software Engineering*, 46(11):1241–1266, 2018.

[26] Hao Wang, Weiyuan Zhuang, and Xiaofang Zhang. Software defect prediction based on gated hierarchical lstms. *IEEE Transactions on Reliability*, 70(2):711–727, 2021.

[27] Song Wang, Taiyue Liu, Jaechang Nam, and Lin Tan. Deep semantic feature learning for software defect prediction. *IEEE Transactions on Software Engineering*, 46(12):1267–1293, 2018.

[28] Song Wang, Taiyue Liu, and Lin Tan. Automatically learning semantic features for defect prediction. In *Proceedings of the 38th International Conference on Software Engineering*, pages 297–308. ACM.

[29] Yue Wang, Weishi Wang, Shafiq Joty, and Steven C.H. Hoi. CodeT5: Identifier-aware unified pre-trained encoder-decoder models for code understanding and generation. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 8696–8708. Association for Computational Linguistics.

[30] Jiaxi Xu, Fei Wang, and Jun Ai. Defect prediction with semantics and context features of codes based on graph representation learning. 70(2):613–625. Conference Name: IEEE Transactions on Reliability.

[31] Jiaxi Xu, Fei Wang, and Jun Ai. Defect prediction with semantics and context features of codes based on graph representation learning. *IEEE Transactions on Reliability*, 70(2):613–625, 2020.

[32] Meng Yan, Xin Xia, Yuanrui Fan, Ahmed E Hassan, David Lo, and Shanping Li. Just-in-time defect identification and localization: A two-phase framework. *IEEE Transactions on Software Engineering*, 48(1):82–101, 2020.

[33] Jinxiong Zhao, Sensen Guo, and Dejun Mu. DouBiGRU-A: Software defect detection algorithm based on attention mechanism and double BiGRU. *Computers & Security*, 111:102459, 2021.