Cross-project software defect prediction based on generative adversarial network-enhanced graph neural network

Abstract—Cross-project software defect prediction (CPDP) is a method that mainly solves the problem of training software defect models, using only known software projects (source projects), when the project to be predicted (target project) is unknown. Because the source and target projects are developed for different requirements, their distribution is different. To solve this problem in cross-project software defect prediction, the method in this paper is as follows: firstly, the software in the source project and the target project are mapped into graph structures, and the community division algorithm is used to divide the graph structure into subgraphs. The next step is to improve the graph neural network using the generative adversarial method. This approach consists of three parts, the feature learning network, discriminator network, and classification network. The feature learning network is used to learn the representation vector of nodes, the discriminator network will eliminate the problem of inconsistent data distribution between source and target projects, and the classification network realizes classification. This model conducted one-to-one and many-to-one experiments on the Promise dataset, achieving better results in F1-measure and AUC compared to the benchmark model.

Keywords—software defect prediction; graph neural network; generative adversarial network

I. INTRODUCTION

Software is a major part of people's lives and is an intricate component of many applications and devices used in all aspects of everyday life. Software defects have always been a headache, seriously affecting the quality of software, and resulting in considerable economic losses. Therefore, software defect prediction has always been one of the hot areas of research by scholars [1]. Cross-project software defect prediction is an important branch of software defect prediction, and it is a new approach. The project focuses on real-life needs for software defect prediction in the developmental stages. This method uses the software project with known features and defect labels as the source project to train the software defect model. The trained model can then be used to predict the target project. The challenge is to build the model with a consistent distribution of data across projects. The applied methods to solve the above problems can be divided into two aspects: pre-filtering datasets and transforming data distribution.

A. Filter Related Datasets

Filtering related datasets involves using projects or instances in the source project with a degree of similarity to that of the target project, with the objective to use this as a training set. Some scholars design corresponding algorithms to calculate similarity, and the instances with high similarity are selected as the training set, primarily considering both the project and the instance.

There has been a great deal of research carried out on the statistical computational methods for the purpose of finding similar projects: Jureczko et al. [2] used hierarchical clustering and K-means to cluster projects and then used the Kohonen neural network to differentiate whether a cluster was for defect prediction. It was found that training on similar projects resulted in better prediction performance than training with all projects. Zimmermann et al. [3] believed that selecting source projects arbitrarily or based on the same domain was not practical, so they selected 40 features to describe the domains, processes, and data of each project and validated the similarity between projects through a series of steps. Herbold [4] proposed two strategies for selecting source projects using clustering and nearest neighbor approaches: EM-Clustering and Nearest Neighbor Selection. EM-Clustering divides projects into different clusters and selects those datasets in the same cluster as the target training data as the training set. Nearest Neighbor Selection sets a value k and calculates the Euclidean distance between other projects and the target project, selecting the closest k values as the training set.

For the purpose of finding similar instances: Nam et al. [5] proposed a new method called CLAMI (Clustering instances, Clustering LA-labeled instances in clusters, Metric selection, and Instance selection), which addresses the problem of unlabeled data using the first two steps. Then, the Measure Violation Score (MVS) is defined to find the best defect measurement element. After removing the instances with MVS exceptions from the total section of instances (instances that retain the selected defect measurement elements), the remaining instances are used as the training set. Seyedrebvar et al.[6] proposed a search-based instance selection method called GIS (Genetic Instance Selection), which uses genetic algorithms to search and determine the instance selection process based on convergence goals to match the distribution characteristics of the test set with more accuracy. The fitness function of each generation is evaluated based on the confirmation set obtained through the NN-filter. Moreover, based on these confirmation sets, the GIS method can solve both potential noise and instance selection problems in the data simultaneously. Turhan et al. [7] proposed the Burak filter method, which sets the k value of the knn algorithm around the instances of the target project, which can be used to find the nearest instances in terms of Euclidean distance from the source project. Peters et al. [8], on the contrary, begin with the source project itself, using the k-means clustering algorithm to divide all instances into n clusters, to select the cluster containing the instance of the target project, and finally find the instance closest to the test instance, from the cluster as the training dataset.

B. Transform the Data Distribution

The method of transforming data distribution involves transforming the data distribution of the measurement elements before establishing the model so that the source project and the target project have similar distributions. Scholars first thought of transforming the data of the source

and target projects. Zhang et al. [9] studied whether cross-project defect prediction would be affected by the application of different transformations (i.e., logarithmic and rank transformations, and Box-Cox transformations). First, they proposed the Multiple Transformation (MT) method for cross-project defect prediction using multiple transformations. Then, applied an enhanced method, MT+, which uses the parameters of the Box-Cox transformation to determine the most suitable training project for each target project. Li et al. [10] proposed a CPDP method called LSKDSA (Landmark Selection-based Kernelized Discriminant Subspace Alignment) based on landmark selection for kernelized discriminant subspace alignment. LSKDSA not only reduces the data distribution difference between the source and target projects but also characterizes complex data structures, increasing the probability of data being linearly separable. Xu et al. [11] considered the case of conditional distribution and used a novel transfer learning method based on Balanced Distribution Adaptation (BDA) to narrow this gap. BDA simultaneously considers both distribution differences and adaptively assigns them different weights. Through experimentation, the results found that this method does improve the performance of the model.

With the success of transfer learning, scholars have combined transfer learning with cross-project software defect prediction to map the original feature spaces of the source and target projects into the same space. Transfer Component Analysis (TCA) is the most commonly used in cross-project defect prediction models. Nam et al. [12] used the TCA method to map the source project defect metric and the target item defect metric to the same feature metric space and modified the normalization of TCA by experimenting on the ReLink and AEEEM datasets, and then proposed the TCA+ method. Cao et al. [13] proposed an effective software defect prediction method, namely Transfer Component Analysis Neural Network (TCANN). TCANN has three sections designed to solve each of these three issues. In the first section, a quartile range (IQR)-based method is proposed to eliminate noise in the dataset. The second stage, uses the transfer component analysis method, to reduce the difference in feature distribution between the source data and the target data. The third and final section proposes a dynamic sampling neural network to deal with the class imbalance problem of the training dataset. Based on classic open-source datasets collected by previous researchers, our experimental results show that TCANN improves the performance of intra-project and cross-project defect prediction compared to other methods. Wen et al. [14] conducted an empirical study on feature and project selection, and they proposed the MZTCA+ (median_zscore TCA+) method for cross-project models. First, mean_log, std_log, median_log, median_zscore, and TDS are used for source project selection. Then, TCA+ is used for transfer learning to solve the problem of consistent distribution of source and target projects. It was concluded that the source project selection method combined with TCA+ works better than using TCA+ alone. Liu et al. [15] also proposed a two-stage cross-project predict model. The method selects the two most appropriate source projects in the first stage. After that, TCA+ was applied to the two projects to train the model.

In summary, cross-project software defect prediction needs to address the data differences between the source and target projects. However, it has been found that current methods all preprocess the data before the model, ignoring the fact that data preprocessing may affect the model prediction. Therefore, a graph neural network-based model for cross-project software defect prediction is proposed based on the idea of the generative network. First, a graph neural network is used to learn the representation vectors of nodes, and then a GAN network is used to address the distribution difference between datasets while using a classifier for defect prediction.

II. RELATED WORK

A. Cross-project Software Defect Prediction

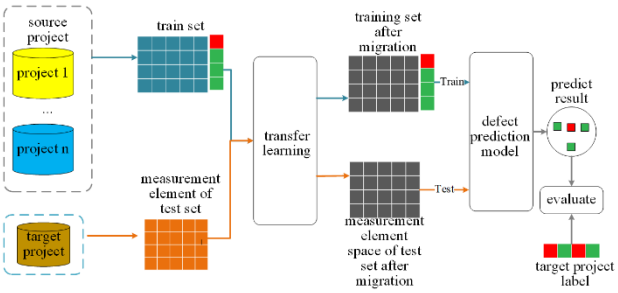

Fig.1. The basic framework for cross-project software defect prediction

As shown in Figure 1, data transfer or feature selection is conducted on the data of the source and target projects to eliminate differences in data distribution. The subsequent steps involve selecting the source project, which refers to the set of projects with known labels and measurement elements, as the training set, and the target project, meaning the set of projects with only defect measurement elements, as the test set. Transfer learning is then applied to eliminate distribution differences between the training and test sets. Subsequently, the transformed data from the source project (including labels) is used to train a defect prediction model, while the transformed test set is used to evaluate the performance of the model.

B. Graph Neural Network

GCNs based on spatial domain includes the following: GraphSAGE (Graph Sample and Aggregate) [16], GAT (Graph Attention Network) [17], and GIN (Graph Isomorphism Network) [18]. Spatial convolution aggregates the feature vectors of the first-order adjacent nodes of a node and then combines them with feature vectors of the current node to obtain a new feature vector. The graph convolution formula of GIN is as follows:

$$h_{v.}^{(k)} = MLP^{(k)}\left(\left(1+\varepsilon^{k}\right)\cdot h_{v}^{(k-1)} + \sum_{u\in N_{(v)}} h_{u}^{(k-1)}\right) \quad (1)$$

The first step is to define a graph G (V, E), in which $v \in V$, the feature vector of each node is X_v. $h_v^{(k)}$ denotes the representation vector of node v at kth layer, where k denotes the iteration level; $h_v^{(0)} = X_v$. $N_{(v)}$ represents a group of adjacent nodes of v. MLP is a multi-layer perceptron. ε is a learnable parameter or a fixed parameter.

C. Generative Adversarial Network

Generative adversarial network (GAN) is one of the main deep learning models, first proposed by Goodfellow et al. [19] in 2014. The concept of GAN has been widely utilized during the following years, and many derivative networks have been

developed. GAN has received considerable attention and research in the field of image synthesis [20] since traditional deep models only have a generator and cannot learn realistic data from limited training sets. However, the GAN network consists of both a generator and a discriminator, with the generator used to generate fake data sourced from the training set, and the discriminator used to distinguish between real and fake data. The generator and discriminator play off against each other, improving their respective performance, ultimately leading to the generation of data that is nearly indistinguishable from real data. Figure 2 shows the basic model of GAN. In the GAN model, it is important to consider the structure of the real data (image, text, graph structure, etc.) when setting the parameters of the generator network, and then use the discriminator to differentiate between real and fake data. Labels for real data and fake data are assigned as "real" and "fake", respectively, and used to train the generator and discriminator.

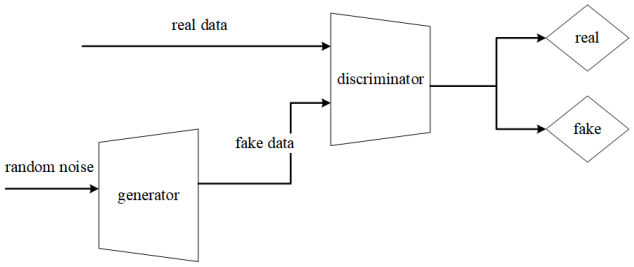

Fig.2. Generative adversarial network

The idea of the GAN network can not only be used in the field of image synthesis, but also has good potential development in data enhancement, image migration, image super-resolution, text image generation, and possibly many other fields. Among them, the successful application of image migration can be used for reference in cross-project software defect prediction, and by using the idea of adversarial learning to eliminate distribution differences.

III. RESEARCH METHOD

This model is mainly based on graph structure and uses graph neural networks based on the idea of generative adversarial learning to learn node feature vectors and eliminate data distribution differences between projects. The overall framework is shown in Figure 3. Firstly, multiple subgraphs are obtained by mapping the software through data preprocessing. Then, the graph structure information is learned by designing the feature learning network to obtain the feature vectors of nodes. The node feature vectors are input into the discriminator network and classification network respectively. The discriminator network distinguishes between the node feature vectors in the source project and those in the target project and trains the classification network using the node feature vectors and labels in the source project. Next, section 3.1 introduces the feature learning network in detail, section 3.2 introduces the discriminator network and classification network as well as the loss function.

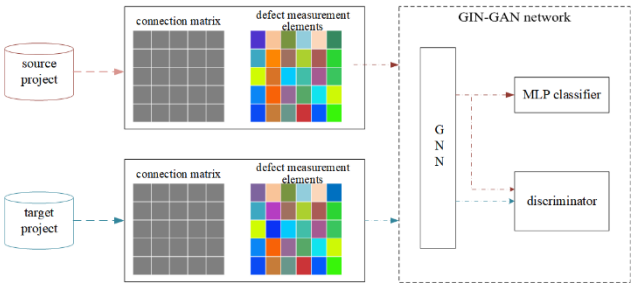

Fig.3. A holistic framework for cross-project defect prediction

A. Feature Learning Network

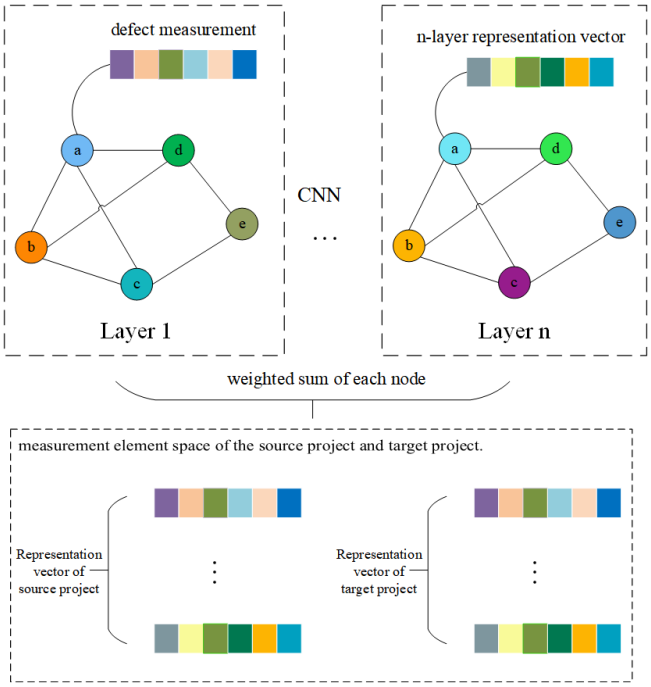

Fig.4. GIN-based graph convolution process

After data preprocessing, each project can be represented by $G=\{G_1,G_2,...,G_n\}$ and $G_i=(V_i,A_i,X_i)$. V_i, A_i, X_i respectively represent vertices, edges, and vertex attributes in the subgraph. Among them, Gi is input into the graph convolution process shown in Figure 4.

Taking node a in Figure 4 as an example, the representation vectors of the adjacent nodes b, c, and d are first added together, then added to the current representation vector of node a, and finally passed through an MLP to obtain a new representation vector h. Each node updates its representation vector using the following formula.

$$h_{v.}^{(k)}=MLP^{(k)}\left(h_v^{(k-1)}+\sum_{u\in N_{(v)}}h_u^{(k-1)}\right) \quad (2)$$

where v represents a node number, in which $v\in V_i$. the feature vector of each node is X_v. $h_v^{(k)}$ denotes the representation vector of node v at kth layer, where k denotes the iteration level. $N_{(v)}$ represents a group of adjacent nodes of v. then initialize $h_v^{(0)}=X_v$. MLP is a multi-layer perceptron.

Because each node goes through multiple graph convolutions, a node can be represented by multiple

representation vectors. In this chapter, all the representation vectors will be weighted and summed to obtain the node's representation vector. The final representation vector of a node is shown in the following formula:

$$h_v^{(rep)} = \sum_{k=0}^{n} w^{(k)} h_v^{(k)}, n = num\_gcn \quad (3)$$

where $w^{(k)}$ is a learnable parameter in kth layer, the initial value is set to $(1/len, 1/len)$, len is the length of representation vector, $h_v^{(k)}$ is the representation vector of node v in kth layer, num_gcn is the number of graph convolution layers, $h_v^{(rep)}$ represents the result of weighted summing of all feature vectors for node v.

B. Discriminative Network and Classification Network

The discriminative network and classification network are both fully connected networks, but they are constructed based on different objectives. Two issues should be noted. First, the setting of the loss function. Second, the discriminator network trains faster than the classification network.

1) Loss function *settings*

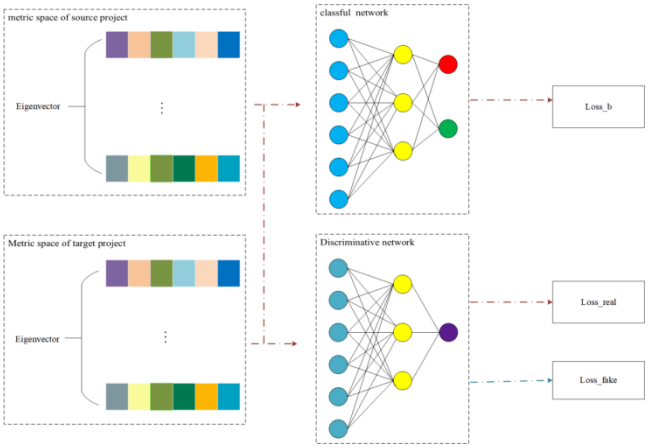

Fig.5. GAN-based target network.

After going through the above steps, the nodes in the source project and target project will obtain new vector representations. Combining all nodes in the source project together yields the source project feature vector space, denoted as X_source. Similarly, the target project feature vector space can be obtained and is denoted as X_target. Next, the new feature metric space is input into the MLP classifier and the discriminator, as shown in Figure 5. It can be seen from the figure that both classifiers and discriminators are implemented by a fully connected network, and there are a total of three loss functions according to the specific target that is used. First, the feature metric elements in X_source are passed through a classifier to obtain corresponding predicted labels pred_source. Then, their loss function value loss_b is calculated by taking the difference between the pred_source and the real labels y_source, as shown in formula 4.

$$loss\_b = loss\_fun\_1(pred\_source, y\_source) \quad (4)$$

Formula 4 is used to adjust the parameters of the graph neural network model so that the learned node representation vectors can be advantageous for defect classification. Next, we set the loss function for the discriminator. The discriminator is designed to address the problem of disparate data distributions between X_source and X_target. According to the concept of Generative Adversarial Network (GAN), the discriminator and feature learning network influence and progress each other. For example, if the discriminator network cannot distinguish whether the node representation vectors come from the source or target projects after training, it indicates that both the feature learning and discriminator networks have been well-trained. Therefore, we set the node labels in the source project to y_real=(1,1,…,1), and the node representation vectors are denoted as dis_source after passing through the discriminator. Similarly, we set the node labels in the target project to y_fake=（0,0,…,0）, and the node representation vectors are denoted as dis_target after passing through the discriminator. The two loss values, loss_real and loss_fake, are calculated by applying the corresponding loss function. The formulas are shown below:

$$loss\_real = loss\_fun\_2(dis\_source, y\_real) \quad (5)$$

$$loss\_fake = loss\_fun\_2(dis\_target, y\_fake) \quad (6)$$

The above uses a discriminator to adjust the parameters in the graph neural network so that it can eliminate the problem of different data distributions between source and target projects. The total loss function is as follows:

$$loss\_all = loss\_b + loss\_real + loss\_fake \quad (7)$$

This model regards graph neural network as generators in generative adversarial network, and designs corresponding discriminators to use three loss functions to achieve defect prediction and eliminate the problem of different data distributions. Unlike current cross-project software defect prediction models, this is an end-to-end training method that solves both problems simultaneously, rather than solving one problem after another, which can dynamically adjust the interaction between the two problems.

2) Discriminative network trains too fast

When the discriminative network and the feature learning network are trained through game theory, the classification network also affects the feature learning network. If the discriminative network learns too quickly, a situation may arise where the discriminative network can distinguish whether the node's feature vector belongs to the source or target project before the classification network has completed its training. To prevent this situation, the following improvements are made to the model during training:

a) The learning rate of the discriminative network is set lower than that of the classification network;

b) When there is a large and non-zero difference between the dis_source and dis_target output by the discriminative network, gradient ascent is used to modify the network parameters before performing gradient descent again until dis_source and dis_target are both 0.5;

c) A stopping criterion is set such that training terminates when loss_b drops below a certain threshold and the output of the discriminator network is 0.5.

C. Pseudocode of Cross-project Software Defect Prediction

This section will introduce the pseudocode of cross-project software defect prediction. As mentioned in section III.A, the data is preprocessed first, then the representation vectors of the nodes in the graph are obtained through the

graph neural network, and finally, the two problems of cross-project software defect prediction are solved through the MLP network and discriminator network. The pseudocode for the algorithm is as follows:

Algorithm 1: Cross-project Software Defect Prediction

input: the graph structure $G=\{G_1,G_2,...,G_n\}$. $G_i=(A_i,X_i)$, A_i and X_i represent nodes, the adjacency matrix of the edges and the software defect measurement element of the nodes, respectively

output: the prediction result pred of the node

```
for i in num_layer do
// num_layer: the number of graph convolutional layers
// num_subgraph: the number of subgraphs
    for j in num_subgraph do
        Put the subgraph G_i = (A_i, X_i) into the graph convolution layer to learn the representation vector of the node;
    end for
    H += W*h;                    // Add up the weighted vectors of each layer
end for
if model_train = True:
    Train the MLP network using the representation vector space of the source project and the real defect label;
    Train the discriminator network using the representation vector space and the set discriminator label of the source and target projects;
    return the result of the source project after passing through the MLP network pred;
if model_test = True:
    Input the representation vector of the target project into the MLP network and obtain the result pred;
    return pred;
```

It can be noted from the above algorithm that it is improved from two aspects. 1) In the graph neural network, the learned node vectors are weighted and summed. 2) A discriminator network is added to eliminate the data distribution difference between the source and target projects.

IV. EXPERIMENTAL SETUP

A. Experimental Environment and Datasets

Experiment was conducted on Windows operating system using Python programming language. The graph neural network model was built using PyTorch and torch-geometric package. The comparison experiments were also implemented using Python.

The dataset used in the experiment was the PROMISE dataset which is a collection of open source software projects [2]. And the metrics used in this dataset are object-oriented metrics. For the experiment, we selected six projects from the dataset. The information is presented in the Table I..

TABLE I. PROJECT INTRODUCTION

Datasets	Version	Feature	Node	Edge	Defect	Defect rate
Ant	1.7.0	20	745	3961	166	0.2228
Camel	1.6.0	20	965	4215	188	0.1948
Lucene	2.4	20	340	1559	203	0.5970
Synapse	1.2	20	256	1162	86	0.3359
Velocity	1.6.1	20	229	1292	78	0.3406
Ivy	2	20	352	2063	40	0.1136

B. Comparison Method and Parameter Setting

The basic methods used in the experiment are Burak Filter [7], Peter Filter [8], and TCA+ [12]. The Burak Filter inputs the defect metrics of nodes into the KNN algorithm, and then selects the n nodes with similar Euclidean distance from the source project that match the target project nodes as the training set. BP is a basic machine learning classifier that can directly use software defect metrics for defect prediction. Peter Filter clusters all nodes using K-means and removes clusters that do not have any target project nodes. TCA+ is a classic model in cross-project software defect prediction, which uses transfer learning to map the target metric space and source project metric space into the same space, thereby solving the problem of inconsistent data distribution.

The hyperparameter settings in this chapter's experiments are shown in Table II. The learning rate of the discriminator is smaller than that of the graph neural network because overly large parameters will cause the MLP classifier to fail to converge. The discriminator will distinguish whether the nodes come from the source project or the target project, so in order to balance the training speed, it is necessary to adjust the learning rate. The settings of other parameters are default settings.

Accuracy refers to the proportion of correct classification to the total number, and the value range is [0, 1]. Higher values indicate better classifier performance. The formula is as follows:

$$Acc=\frac{TP+TN}{TP+FN+TN+FP} \tag{8}$$

TABLE II. TRAINING PARAMETERS

Parameters	Settings
epoch	5000
G_LR	0.0001
D_LR	0.00005
weight_decay	5e-4
loss_b	CrossEntropyLoss
loss_real，loss_fake	BCELoss
optimizer	Adam
activation	Relu
seed	55

The settings of the graph convolutional layer and classifier layer in the experiment are shown in Table III, and the

changes in vector dimension during the network process are also introduced. At first, the Burak filter, Peter filter and TCA+ method preprocess the dataset and then all classifiers are set as MLPs with the same parameters. The parameter setting for knn in Burak is n_neighbors=10. The parameter setting for k-means for the Peter filter is n_cluste. The parameters for both transfer learning and domain adaptation algorithms are set to default values without any specific modifications.

TABLE III. NETWORK PARAMETERS

Algorithm	Number of graph convolution layers	Change of vector dimension	Number of classifier layers	Change of vector dimension
Burak	0	None	3	20,10,2
Peter	0	None	3	20,10,2
TCA+	0	None	3	20,10,2
GIN_GAN	4	20,20,20,20,20	3	20,10,2

The dataset is sourced from the Promise dataset, and six projects were selected for experimentation. It includes defect metrics X={x1, x2, ..., xn} and their corresponding labels Y={y1,y2,…,yn}.

C. Experimental Procedure

The experiments in this chapter are cross-project defect predictions and are divided into two scenarios: Firstly, a One-to-one experiment, where one project is selected as the target project and the rest are taken as source projects. For example, when Ant is the target project for defect prediction, Camel, Lucene, Synapse, Velocity, and Ivy are used as the source projects for training. The second scenario is a Many-to-one experiment, where one project is selected as the target project and all other projects are used as source projects. For example, when Ant is the target project for defect prediction, all graph structures of Camel, Lucene, Synapse, Velocity, and Ivy are used as the training set. The specific experimental steps are as follows:

1) Preprocessing of the target project and source projects yields graph structures G_source and G_target, which include, node defect measurement elements and edge information. Corresponding labels Y_source and Y_target are obtained based on the dataset. Input G_source, G_target, and Y_source into the GIN_GAN model. To prevent the discriminator network from training too fast, termination conditions are set. After the loop has ended, the pred_source obtained is compared with the Y_source to calculate the loss function and then return to adjust the network parameters.
2) Input G_target into the pre-trained GIN_GAN model to obtain predicted label pred_target, and finally evaluate the results using evaluation metrics.
3) Repeat the above steps for all target items in turn.

D. Experimental Metrics

To demonstrate the effectiveness of our model, we choose F1-measure and AUC as evaluation metrics, both of which are derived from the confusion matrix [21]. The confusion matrix is shown in the following table：

TABLE IV. CONFUSION MATRIX

Actual label	Predicted label	
	Defective	*Defective-free*
Defective	TP (true positive)	FN (false negative)
Defective-free	FP (false positive)	TN (true negative)

1) F1-measure

The F1-measure is the harmonic average of precision and recall, Precision P refers to the ratio of the number of correctly classified positive samples by the classifier to all positive samples of the classifier. Recall R is the ratio of the number of positive samples classified by the classifier to the total number of positive samples, with a value range of [0, 1]. The higher the R value, the better the classification effect. The formula is as follows:

$$P = \frac{TP}{TP + FP} \tag{8}$$

$$R = \frac{TP}{TP + FN} \tag{9}$$

$$F1 = \frac{2 \times P \times R}{P + R} \tag{10}$$

2) AUC

AUC is the area under the ROC curve with the coordinate axis of the ROC curve formed by Formula 11 and Formula 12.

$$FPR = \frac{FP}{FP + TN} \tag{11}$$

$$TPR = \frac{TP}{TP + FN} \tag{12}$$

FPR is the x-axis of the false positive rate, and TPR is the y-axis of the true positive rate. For example, if the probability of an instance obtained through prediction is pred = 0.6, and the threshold is set to threshold = 0.5, the instance has a defect if pred>threshold; if threshold = 0.7, the instance has no defect if pred<threshold. When different thresholds are taken, the false positive rate will change, and the ROC curve will change accordingly. The AUC differs from the above evaluation indicators (which only consider a single fixed threshold), the AUC considers different thresholds, so AUC can obtain more reasonable results when evaluating data with class imbalance.

V. RESULTS AND ANALYSIS

This section mainly conducts experiments from two aspects: on one hand, the source project trains a single project model, and then tests it on the target project; on the other hand, the source project trains a mixed project model and then tests it on the target project. The first group of experiments aims primarily to demonstrate that this method yields better performance than other baseline methods, while also comparing the differences between single-project and mixed-project training. The second group of experiments aims to verify the performance of various methods when the source project is a mixed-project and compare it with the performance of baseline methods. Next, the performance of the model proposed in this chapter will be analyzed mainly using evaluation metrics the F1-measure, and AUC. As introduced in the previous text, F1 comprehensively considers

precision and recall. AUC can make more reasonable judgments than accuracy under the condition of sample imbalance.

A. The Source Project as a Single Project

In this particular scenario, the model is trained using source projects other than the target project when selecting software projects. The F1 and AUC values obtained are shown in Table V and Table VI, respectively. The bold font in the table represents the maximum value. It can be observed from the table that this method performs well in most of the experiments. The GIN-GAN method improves the mean F1-measure of 11.5%, 13.2%, and 12.2% compared to the Burak Filter, Peter Filter, and TCA+ method, respectively. Here we found the mean value of AUC increased by 6.5%, 9.4%, and 6.2% respectively. In the next step, the analysis will be conducted on two evaluation metrics: F1-measure and AUC.

TABLE V. COMPARISON OF F1-MEASURE OF SOURCE PROJECT AS SINGLE PROJECTS

source project->target project	**Bruak**	**Peters**	**TCA+**	**GIN-GAN**
camel->ant	0.2479	0.3805	0.0652	**0.5359**
ivy->ant	0.1604	0.2200	0.0914	**0.3723**
lucene->ant	0.4448	0.3805	0.4755	**0.5202**
synapse->ant	0.5376	0.3805	0.5492	**0.5545**
velocity->ant	0.3241	0.2074	0.4738	**0.5851**
Avg	0.3429	0.3138	0.3310	**0.5136**
ant->camel	0.1633	0.1639	**0.2733**	0.2281
ivy->camel	0.0676	0.0106	0.0309	**0.1084**
lucene->camel	0.3926	0.3765	0.3728	**0.4149**
synapse->camel	0.2743	0.1038	0.3021	**0.3596**
velocity-camel	0.3111	0.2367	0.3537	**0.3829**
Avg	0.2417	0.1783	0.2665	**0.2987**
ant->ivy	0.4054	0.4468	**0.5176**	0.495
camel->ivy	0.2105	0.1387	0.1923	**0.4717**
lucene->ivy	0.2618	0.2677	0.2051	**0.2979**
synapse->ivy	0.3967	0.2222	0.4174	**0.4662**
velocity->ivy	0.3063	0.1356	0.3636	**0.4189**
Avg	0.3161	0.2422	0.3392	**0.4299**
ant->lucene	0.2251	0.2881	0.2594	**0.4573**
camel->lucene	0.2683	0.2481	0.1033	**0.3226**
ivy->lucene	0.0762	0.1370	0	**0.2137**
synapse->lucene	0.5959	0.6124	0.5497	**0.6284**
velocity->lucene	0.3590	0.2231	0.4126	**0.5987**
Avg	0.3049	0.3017	0.2650	**0.4441**
ant->synapse	0.3540	0.4684	0.5037	**0.5735**
camel->synapse	0.4000	**0.4920**	0.0833	0.4310
ivy->synapse	0.1489	0.2752	0.0889	**0.2963**
lucene->synapse	0.6119	0.5103	0.6025	**0.6408**
velocity->synapse	0.4729	0.4249	0.5125	**0.5481**
Avg	0.39754	0.4341	0.3581	**0.4979**
ant->velocity	0.1429	0.1957	**0.2772**	0.2174
camel->velocity	0.2549	0.3382	0.1379	**0.3415**
ivy->velocity	0.0988	0.1778	0.0256	**0.2045**
lucene->velocity	0.5662	0.4270	0.5022	**0.6261**
synapse->velocity	0.2881	0.3500	0.4030	**0.5000**
Avg	0.2701	0.2977	0.2691	**0.3779**

1) Based on the evaluation metric of F1-measure, both this method and TCA+ map the feature spaces of source and target projects into the same data distribution. However, this method also takes into account defect classification. The experimental results show our method to have a better and more stable performance. In most cases, there is a relatively skewed outcome when using the TCA+ method for transfer learning. For example, when ant is the target project , camel and ivy are the source projects, the resulting F1-measure is very low. Both the Bruak and Peter filter methods rely heavily on the algorithm for selecting the training data nodes to train based on the target project. However, this approach uses the method of generative adversarial networks to address the impact of different algorithms and is shown to be more versatile. It can be seen from the table that when Ivy is used as the source project, all the experimental results are very low. From the dataset itself, it can be found that the defect rate of Ivy is only 11%, indicating that using a single project as the source project has some drawbacks.

To better analyze the overall experimental results, a box plot is used to visualize the F1-measure of the experimental results, as shown in Figure 6., the boxplots of all methods do not have outliers, but the boxplot of the TCA+ method is longer, indicating a significant difference in performance. However, both Burak filter, Peter filter, and the TCA+ methods, can find suitable training data. The box of this method is higher than other methods, and both the average and median are higher, indicating that this method is overall very good and more stable.

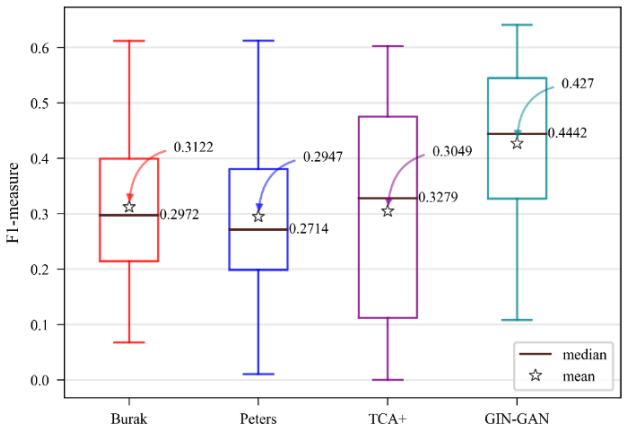

Fig.6. boxplots of F1-measure with the source project as a single project

2) The results of the analysis of the AUC evaluation index, shown in Table VI indicate that the performance of this method on the AUC index is better than that on F1 index, with only two non-optimal results. It can be seen from the table that in most cases, the effect of TCA+ and the application of the method shown in this article improves the resultant outcome. The Bruak and Peter filter find the closest theoretical node as training data through algorithms. However, when compared with the TCA+ method, it can be seen to perform even better, indicating that addressing data

distribution issues is a more reasonable approach for defect prediction.

Visualizing Table VI, we obtained Figure 7. It can be seen from the figure that the Peter filter method has resulting outliers, and the lower edge of the boxplot is below 0.5, this is because the Peter filter assumes that smaller test sets contain more information about defects, and therefore sacrifices more nodes. From the perspective of the mean value, the Burak filter, Peter filter, and the TCA+ method are similar, indicating that each of these methods has its advantages and disadvantages. This method performs better in mean, median, and box performance, as it takes into account the learning of graph structure information and the elimination of data distribution differences; it also completes a cross-project software defect prediction while learning feature vectors at a deeper level.

TABLE VI. COMPARISON OF AUC OF SOURCE PROJECT AS SINGLE PROJECTS

source project->target project	Burak	Peters	TCA+	GIN_GAN
camel->ant	0.5533	0.5929	0.5076	**0.6964**
ivy->ant	0.5399	0.5558	0.5232	**0.6103**
lucene->ant	0.6388	0.5929	0.6740	**0.7132**
synapse->ant	0.7087	0.5929	0.7198	**0.7311**
velocity->ant	0.5744	0.5180	0.6622	**0.7549**
Avg	0.6030	0.5705	0.6173	**0.7011**
ant->camel	0.5270	0.5013	**0.5533**	0.5479
ivy->camel	0.5101	0.5027	0.5000	**0.5264**
lucene->camel	0.5950	0.5700	0.5740	**0.6263**
synapse->camel	0.5469	0.5200	0.5394	**0.5966**
velocity-camel	0.5585	0.5050	0.5816	**0.6079**
Avg	0.5475	0.5198	0.5496	**0.5810**
ant->ivy	0.6569	0.7093	0.7379	**0.7544**
camel->ivy	0.5548	0.4548	0.5512	**0.7464**
lucene->ivy	0.6290	0.6347	0.5000	**0.6794**
synapse->ivy	0.7081	0.5585	0.7177	**0.7875**
velocity->ivy	0.6254	0.4871	0.6903	**0.7633**
Avg	0.6348	0.5688	0.6394	**0.7462**
ant->lucene	0.5531	0.5641	0.5542	**0.5756**
camel->lucene	0.5403	0.4890	0.5272	**0.5765**
ivy->lucene	0.5198	0.5296	0.5000	**0.5356**
synapse->lucene	0.6147	0.6006	0.6415	**0.6634**
velocity->lucene	0.5386	0.5180	0.5521	**0.6650**
Avg	0.5533	0.5402	0.5550	**0.6032**
ant->synapse	0.5952	0.6097	0.6525	**0.6936**
camel->synapse	0.6094	0.6018	0.5052	**0.6303**
ivy->synapse	0.5377	0.5631	0.5233	**0.5750**
lucene->synapse	0.6755	0.5743	0.6746	**0.7211**
velocity->synapse	0.5712	0.5396	0.6390	**0.6790**
Avg	0.5978	0.5777	0.5989	**0.6598**
ant->velocity	0.5356	0.5382	**0.5571**	0.5480
camel->velocity	0.5439	0.5277	0.5254	**0.5519**
ivy->velocity	0.5260	0.5351	0.5065	**0.5517**
lucene->velocity	0.6323	0.5339	0.5559	**0.6939**
synapse->velocity	0.5293	0.5620	0.5740	**0.5890**
Avg	0.5534	0.5393	0.5437	**0.5869**

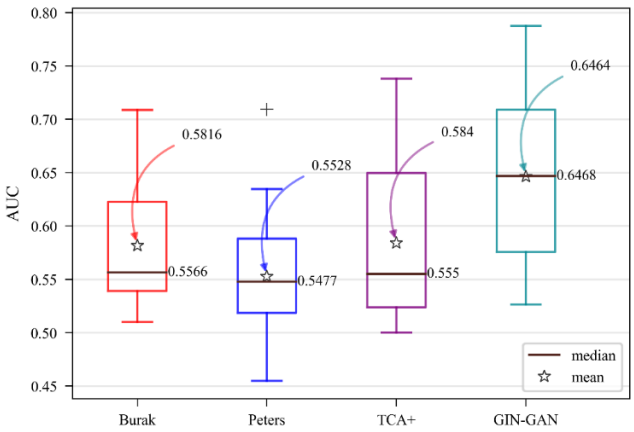

Fig. 7. The source project is the AUC boxplots for a single project.

B. The Source Project as a Hybrid Project

When the source project is a single project, there are two situations where some results are good and some results are very poor, which makes it difficult to choose the source project in a real environment. Therefore, hybrid projects should be selected as source projects. Table VII and Table VIII respectively show the F1-measure and AUC evaluation results obtained by using mixed projects. The bold font in the table represents the maximum value of the row. The results tables below show that this method can still improve performance. Compared with the Burak filter, Peter filter and TCA+ methods, the GIN_GAN method has increased the F1 evaluation metric mean by 16.8%, 15.7% and 13.0%, respectively, and increased the AUC evaluation metric mean by 5.6%, 11.5%, and 4.5%, respectively.

TABLE VII. F1-MEASURE FOR SOURCE PROJECT AS A HYBRID PROJECT

Target project	Burak	Peters	TCA+	GIN_GAN
ant	0.4014	0.3805	0.3719	**0.5567**
camel	0.2448	0.3460	0.3182	0.2909
ivy	0.3564	0.0597	0.3810	**0.5122**
lucene	0.1905	0.3346	0.2137	**0.5083**
synapse	0.3273	0.3910	0.4844	**0.5000**
velocity	0.1778	0.2524	0.1591	**0.3390**

The F1-measure evaluation index indicates that in most cases, when the source project is a mixed project, this method still attains the highest value. The evaluation values of the Burak filter and Peter filter still vary, this is due to the increase in instances in the hybrid projects and the fact that the parameters of knn and kmeans are default, thus resulting in less training data being selected. However, this method doesn't require special parameter adjustments, which reduces human involvement and produces better results than other basic method project.

The AUC evaluation index shows that GIN_GAN results have improved in most projects compared to the basic method project. And the AUC evaluation index performs better than the F1 value evaluation index because the projects used for testing are inherently class imbalanced.

TABLE VIII. AUC FOR SOURCE PROJECT AS A HYBRID PROJECT

Target project	Burak	Peters	TCA+	GIN_GAN
ant	0.6175	0.5929	0.6085	**0.7056**
camel	0.5485	0.5104	**0.5649**	0.5611
ivy	0.6556	0.325	0.6548	**0.7286**
lucene	0.5281	0.5613	0.5356	**0.6004**
synapse	0.5866	0.5879	0.6471	**0.6559**
velocity	0.5351	0.5405	0.5319	**0.5589**

For a more intuitive analysis of the experimental results, the F1-measure and AUC evaluation metrics were represented by box plot graphs, as shown in Figures 8 and 9, showing the GIN_GAN method in AUC box plot graph has the highest mean and median. There are no outliers shown in both the AUC and F1 box plots except for the Peter filter method, indicating that the predicted results are within the normal range. Moreover, in both box plot graphs, the upper and lower edges of this method are the highest. As can be seen in the results figure, this method shows an obvious performance improvement compared to the basic method, but the data are more concentrated in the middle and lower reaches, which indicates that there is still room for improvement.

The Peter filter has a lower outlier value, but the box is also more concentrated. However, it is necessary to modify the Kmeans parameters for different target projects. The TCA+ box plot, shows the upper and lower edges with the largest difference, indicating that the feature vector space, after transfer learning, eliminates the problem of data distribution differences. However, the subsequent obtained vectors do not improve performance. The box plot results for the AUC method, are always the best, with the Peter filter method as the only exception; all other methods are relatively stable, indicating that the dataset cannot be arbitrarily removed during training.

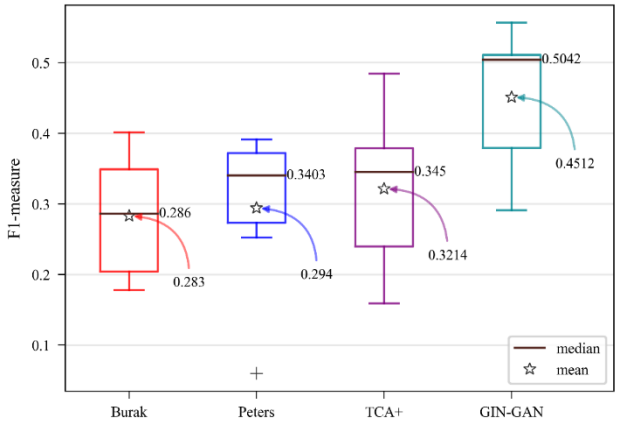

Fig. 8. F1-measure boxplots with source project as hybrid projects

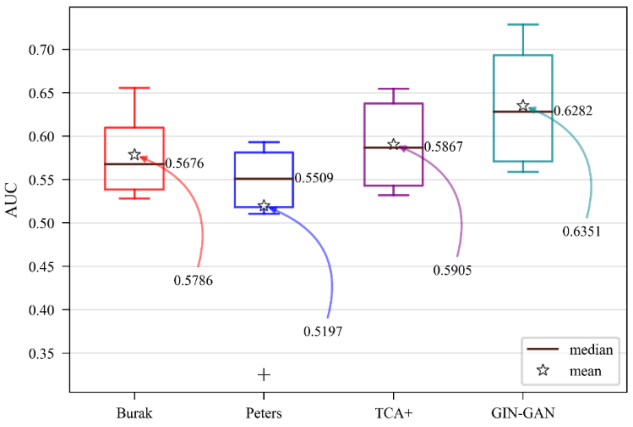

Fig. 9. AUC boxplots with the source project as hybrid project

VI. CONCLUSION

This paper describes how the graph neural network learns useful feature vectors while eliminating the difference in data distribution between the source and target projects. And, how the application of the graph neural network idea, is advantageous for future software defect detection by applying this novel generative adversarial learning methodology.

The experimental results show that the cross-project software defect prediction model proposed in this paper is more effective compared to the baseline model. The model achieves through the application of three essential component network parts: the feature learning network, the discriminator network, and the classification network. This can be described as an end-to-end model; the feature learning network is mainly constructed by graph convolutional layers to learn graph structure information and obtain new feature vectors for nodes. The discriminator network is used to solve the problem of data distribution differences, whilst the classification network achieves the final defect prediction.

This research has outlined the complex factors involved in predictive defect detection and for the cross-project software defect prediction model, we have considered both one-to-one and many-to-one experiments. Moreover, we have described in great detail a novel methodology for improving defect detection and also highlighted the many complex factors that offer future challenges.

The potential for further research and advancement is great and we propose that greater learning can begin from the subgraphs of software and selecting similar subgraphs as training data. One aspect of the research has highlighted concerns with the levels of differentiation between the selection of multiple project sources, and it has been shown that when selecting multiple projects as source projects, the similarity between each project and the source project will display a level of variance. In the future, a model could apply different weights, which can be assigned from the graph's representation vector for learning, as well as considering the Euclidean distance between instances. This study reveals significant improvements compared with the basic model, and as a team, we are excited to consider methods to improve the speed of the model whilst improving the accuracy of future predictive software defect detection.

REFERENCES

[1] M. Cui, S. Long, Y. Jiang, and X. Na, “Research of Software Defect Prediction Model Based on Complex Network and Graph

Neural Network," *Entropy*, vol. 24, no. 10, Oct. 2022, doi: 10.3390/e24101373.

[2] M. Jureczko and L. Madeyski, "Towards identifying software project clusters with regard to defect prediction," in *Proceedings of the 6th international conference on predictive models in software engineering*, 2010, pp. 1–10.

[3] T. Zimmermann, N. Nagappan, H. Gall, E. Giger, and B. Murphy, "Cross-project defect prediction: A large scale experiment on data vs. domain vs. process," in *ESEC-FSE'09 - Proceedings of the Joint 12th European Software Engineering Conference and 17th ACM SIGSOFT Symposium on the Foundations of Software Engineering*, 2009. doi: 10.1145/1595696.1595713.

[4] S. Herbold, "Training data selection for cross-project defect prediction," in *ACM International Conference Proceeding Series*, 2013. doi: 10.1145/2499393.2499397.

[5] J. Nam and S. Kim, "CLAMI: Defect Prediction on Unlabeled Datasets (T)," in *30th IEEE/ACM International Conference on Automated Software Engineering, ASE 2015, Lincoln, NE, USA, November 9-13, 2015*, M. B. Cohen, L. Grunske, and M. Whalen, Eds., IEEE Computer Society, 2015, pp. 452–463. doi: 10.1109/ASE.2015.56.

[6] S. Hosseini, B. Turhan, and M. Mäntylä, "A benchmark study on the effectiveness of search-based data selection and feature selection for cross project defect prediction," *Inf Softw Technol*, vol. 95, 2018, doi: 10.1016/j.infsof.2017.06.004.

[7] B. Turhan, T. Menzies, A. B. Bener, and J. Di Stefano, "On the relative value of cross-company and within-company data for defect prediction," *Empir Softw Eng*, vol. 14, no. 5, 2009, doi: 10.1007/s10664-008-9103-7.

[8] F. Peters, T. Menzies, and A. Marcus, "Better cross company defect prediction," in *IEEE International Working Conference on Mining Software Repositories*, 2013. doi: 10.1109/MSR.2013.6624057.

[9] F. Zhang, I. Keivanloo, and Y. Zou, "Data Transformation in Cross-project Defect Prediction," *Empir Softw Eng*, vol. 22, no. 6, 2017, doi: 10.1007/s10664-017-9516-2.

[10] Z. Li, J. Niu, X.-Y. Jing, W. Yu, and C. Qi, "Cross-Project Defect Prediction via Landmark Selection-Based Kernelized Discriminant Subspace Alignment," *IEEE Trans. Reliab.*, vol. 70, no. 3, pp. 996–1013, 2021, doi: 10.1109/TR.2021.3074660.

[11] Z. Xu *et al.*, "Cross Project Defect Prediction via Balanced Distribution Adaptation Based Transfer Learning," *J. Comput. Sci. Technol.*, vol. 34, no. 5, pp. 1039–1062, 2019, doi: 10.1007/s11390-019-1959-z.

[12] J. Nam, S. J. Pan, and S. Kim, "Transfer defect learning," in *Proceedings - International Conference on Software Engineering*, 2013. doi: 10.1109/ICSE.2013.6606584.

[13] Q. Cao, Q. Sun, Q. Cao, and H. Tan, "Software defect prediction via transfer learning based neural network," in *Proceedings of 2015 the 1st International Conference on Reliability Systems Engineering, ICRSE 2015*, 2015. doi: 10.1109/ICRSE.2015.7366475.

[14] W. Wen, B. Zhang, X. Gu, and X. Ju, "An empirical study on combining source selection and transfer learning for cross-project defect prediction," in *2019 IEEE 1st International Workshop on Intelligent Bug Fixing (IBF)*, 2019, pp. 29–38.

[15] C. Liu, D. Yang, X. Xia, M. Yan, and X. Zhang, "A two-phase transfer learning model for cross-project defect prediction," *Inf. Softw. Technol.*, vol. 107, pp. 125–136, 2019, doi: 10.1016/j.infsof.2018.11.005.

[16] W. Hamilton, Z. Ying, and J. Leskovec, "Inductive representation learning on large graphs," *Adv Neural Inf Process Syst*, vol. 30, 2017.

[17] P. Velickovic, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio, "Graph Attention Networks," in *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*, OpenReview.net, 2018. [Online]. Available: https://openreview.net/forum?id=rJXMpikCZ

[18] K. Xu, W. Hu, J. Leskovec, and S. Jegelka, "How Powerful are Graph Neural Networks?," in *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*, OpenReview.net, 2019. [Online]. Available: https://openreview.net/forum?id=ryGs6iA5Km

[19] I. J. Goodfellow *et al.*, "Generative Adversarial Nets," in *Advances in Neural Information Processing Systems 27: Annual Conference on Neural Information Processing Systems 2014, December 8-13 2014, Montreal, Quebec, Canada*, Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, Eds., 2014, pp. 2672–2680. [Online]. Available: https://proceedings.neurips.cc/paper/2014/hash/5ca3e9b122f61f8f06494c97b1afccf3-Abstract.html

[20] J. Agnese, J. Herrera, H. Tao, and X. Zhu, "A survey and taxonomy of adversarial neural networks for text-to-image synthesis.," *WIREs Data Mining Knowl. Discov.*, vol. 10, no. 4, 2020, doi: 10.1002/widm.1345.

[21] Q. Zhang and J. Ren, "Software-defect prediction within and across projects based on improved self-organizing data mining," *J Supercomput*, vol. 78, no. 5, pp. 6147–6173, 2022.