Wiley
IET Software
Volume 2024, Article ID 3946655, 34 pages
https://doi.org/10.1049/2024/3946655

Research Article

Software Defect Prediction Using Deep *Q*-Learning Network-Based Feature Extraction

Qinhe Zhang,[1] **Jiachen Zhang**,[1] **Tie Feng**,[1,2] **Jialang Xue**,[3] **Xinxin Zhu**,[1] **Ningyang Zhu**,[1] **and Zhiheng Li**[1]

[1]*College of Computer Science and Technology, Jilin University, Jilin 130012, China*
[2]*Key Laboratory of Symbolic Computation and Knowledge Engineering of Ministry of Education, Changchun, China*
[3]*College of Software, Jilin University, Jilin 130012, China*

Correspondence should be addressed to Tie Feng; fengtie@jlu.edu.cn

Received 15 September 2023; Revised 23 April 2024; Accepted 30 April 2024; Published 30 May 2024

Academic Editor: Siti Hafizah Ab Hamid

Machine learning-based software defect prediction (SDP) approaches have been commonly proposed to help to deliver high-quality software. Unfortunately, all the previous research conducted without effective feature reduction suffers from high-dimensional data, leading to unsatisfactory prediction performance measures. Moreover, without proper feature reduction, the interpretability and generalization ability of machine learning models in SDP may be compromised, hindering their practical utility in diverse software development environments. In this paper, an SDP approach using deep *Q*-learning network (DQN)-based feature extraction is proposed to eliminate irrelevant, redundant, and noisy features and improve the classification performance. In the data preprocessing phase, the undersampling method of BalanceCascade is applied to divide the original datasets. As the first step of feature extraction, the weight ranking of all the metric elements is calculated according to the expected cross-entropy. Then, the relation matrix is constructed by applying random matrix theory. After that, the reward principle is defined for computing the *Q* value of *Q*-learning based on weight ranking, relation matrix, and the number of errors, according to which a convolutional neural network model is trained on datasets until the sequences of metric pairs are generated for all datasets acting as the revised feature set. Various experiments have been conducted on 11 NASA and 11 PROMISE repository datasets. Sensitive analysis experiments show that binary classification algorithms based on SDP approaches using the DQN-based feature extraction outperform those without using it. We also conducted experiments to compare our approach with four state-of-the-art approaches on common datasets, which show that our approach is superior to these methods in precision, *F*-measure, area under receiver operating characteristics curve, and Matthews correlation coefficient values.

1. Introduction

Software quality has become a critical issue in the software engineering life cycle due to the increasing scale and complexity of modern software. Software defect prediction (SDP) is a technique used to anticipate which parts of a software system are likely to contain faults or errors [1], plays a vital role in minimizing extensive testing efforts and contributes to the delivery of a top-notch software system by pinpointing potential areas for improvement and error reduction [2, 3, 4].

Among various technologies of SDP, statistical techniques, machine learning methods, and DL architecture-based [5] approaches are absorbing more and more attention of researchers. Many researchers have formed training samples by extracting the software metrics and used Naive Bayes [6], support vector machine (SVM) [7], expectation-maximization clustering techniques [8], regression trees [9], and so on to conduct SDP. However, too many introductions of complexity metric attributes suffer from high-dimensional data. Furthermore, neglecting adequate feature reduction can weaken the interpretability and generalization capacity of machine learning models employed in SDP tasks. Therefore, effective feature reduction has now become one of the most prominent research areas of machine learning and DL-based SDP approaches.

Feature selection and feature extraction [10] are two typical approaches for addressing the high-dimensionality

problem and achieving a more accurate and robust performance when analyzing large and highly dimensional defect data. Feature extraction is a technique to eliminate redundant and irrelevant features by developing a transformation from a high-dimensionality space to another space, preserving most of the relevant information between different features.

On the other hand, deep reinforcement learning (RL) [11] combines the representational learning power of DL [12] with existing RL [13] methods and has played a crucial role in various fields, such as autonomous vehicle control, natural language processing, healthcare, and so on in the past decade. Deep RL fundamentally revolves around the concept of agents learning to make decisions through trial-and-error interactions within their environment, guided by a system of rewards. In particular, the deep Q-learning network (DQN) [14, 15], as a typical deep RL algorithm, constructs a Q-value table to select the optimal strategy, enabling the resolution of large-scale and intricate real-world challenges, such as the refinement of noisy data and the enhancement of features for classification problems.

DL has the ability of successfully processing high-dimensional information, while RL is good at performing continuous decision tasks in complex environments. Effective feature extraction needs to continuously select the most important feature subset for classification tasks according to data samples. This requirement is consistent to the application background of DL and RL. So, we try to apply DQN, a typical deep RL model, to conduct feature extraction and further cross-project SDP. The experimental results show the effectiveness of SDP using DQN-based feature extraction (see Sections 6.4 and 6.5).

With the consideration of preserving the relationship among the features, revealing dependence between one another of each metric pair, and leverage the capability of deep RL, this paper proposes an SDP approach using DQN-based feature extraction. In our approach, instead of selecting a subset of the original feature set, we construct a sequence of metric pairs (SMP) acting as the revised feature set to improve the performance of the SDP. As the first step of feature extraction, the weight ranking of all the metric elements is calculated according to the expected cross-entropy (ECE) for solving the overfitting problem. Then, the relation matrix is constructed by applying random matrix theory (RMT) [16] to explore the relation degree of metric elements. After that, the reward principle is defined for computing Q value of Q-learning based on weight rank, relation matrix, and error label, according to which a convolutional neural network (CNN) [17] model is trained on data sets in order to optimize the parameters of the CNN. The DQN algorithm integrates CNNs to construct a Q-value network, facilitating effective RL in complex, high-dimensional, and continuous state and action spaces. This network serves as the foundation for DQN's objective function, optimizing the CNN model iteratively to achieve efficient Q-learning. As the output of CNN, the SMP acts as the revised feature set, which is used by binary classification classifiers to predict software defects.

In addition, class imbalance [18], happening when the distribution of software defect data is highly skewed, widely influences the performance of SDP and always leads to unsatisfied results. In our study, we have used 11 NASA and 11 PROMISE repository datasets [19] in our experiments. In most of these defect datasets, the total number of positive classes (defective module) of data is far less than the total number of negative classes (nondefective module) of data. The most prevalent method to overcome imbalanced datasets of the software projects is sampling, in which the samples from majority classes are reduced (undersampling) or some synthetic samples are added to minority classes (oversampling). In this paper, the undersampling algorithm BalanceCascade [20], dividing majority classes into groups, is applied to address this problem. BalanceCascade is a method for handling imbalanced datasets by repeatedly performing random undersampling and training to balance the dataset. This approach can help alleviate training issues with DQN on imbalanced datasets. By dynamically adjusting the balance of the dataset in each training iteration, BalanceCascade ensures that each class's samples are adequately represented, thereby preventing the model from overly relying on minority class samples.

To assess the proposed DQN-based feature extraction (DQN-FE-SDP) approach, we explore the following four research questions (RQs):

RQ-1. How does feature ordering based on ECE remove nonphasic features?

RQ-2. How can we find the interrelated features and leverage the relation between features to generate the SMP?

RQ-3. Does the binary classification algorithm-based SDP using DQN have better performance comparing with those without using DQN?

RQ-4. Does the DQN-FE-SDP approach have better performance comparing with other state-of-the-art SDP approaches?

RQ-5. Does the feature extraction algorithm based on DQN outperform four state-of-the-art feature selection approaches in SDP?

The main contributions of this paper include the following aspects:

(1) We have proposed a binary classification algorithm-based machine learning approach using a DQN-based feature extraction framework for software defects prediction.

(2) A weight ranking algorithm based on ECE is provided to calculate the importance of different metrics and remove various redundant and irrelevant features.

(3) An RMT-based relation matrix construction model is presented to obtain the relationship between each of two different metric elements.

(4) A deep RL model, DQN, combining Q-learning and CNN is suggested to intelligently and dynamically generate the SMP for each dataset acting as a revised feature set, so that the precision, F-measure, the area

under the receiver operating characteristics curve (AUC), and Matthews correlation coefficient (MCC) values of SDP are obviously improved when binary classification classifiers work on it.

(5) The proposed work outperforms three state-of-the-art SDP methods in terms of the performance of precision, *F*-measure, AUC, and MCC values.

The structure of this paper is as follows: In Section 2, we summarize the related work. In Section 3, the datasets of NASA and PROMISE repository, and software metric elements used as features are introduced. In Section 4, the whole methodology is illustrated. In Section 5, the performance measures of SDP are introduced. In Section 6, experiment results are discussed. In Section 8, we conclude this work and indicate future work.

2. Related Work

A large body of research has been conducted to turn learning-based SDP into a practical approach and described in the literature. Most of the approaches follow the procedure of sampling, optimizing datasets, and training and classification. Besides, many public datasets are created to train, predict, and evaluate the performance of SDP. In this section, we describe the most relevant research from five areas.

2.1. Machine Learning-Based SDP Methods. Many successful algorithms on SDP have been designed by applying machine learning algorithms, such as Naive Bayes [6], SVM [7], random forest [21], and so on. Bouguila et al. [22] introduced an unsupervised Bayesian algorithm based on finite Dirichlet mixtures employed to software prediction by categorizing modules into fault-prone and non-fault-prone. Kumar et al. [23] developed a fault prediction model by employing the least squares SVM (LSSVM) learning approach, which utilizes linear, polynomial, and radial basis function kernel functions. Khoshgoftaar and Seliya [9] presented a case study from their comprehensive evaluation of available regression-tree algorithms for software fault prediction. Wang et al. [24] conducted a comparative study of various ensemble methods with the perspective of taxonomy and concluded that voting and random forest have obvious performance superiority than others in all seven ensemble methods evolved. Zheng [25] studied three cost-sensitive boosting algorithms, one based on threshold-moving and two based on weight-updating, to boost neural networks for SDP. Owhadi-Kareshk et al. [26] suggested implementing a pretraining approach for an artificial neural network with a shallower architecture, which involves fewer hidden layers. Mahaweerawat et al. [27] introduced a fault prediction model employing supervised learning through a multilayer perceptron neural network. Following the classification results, classes with faults underwent in-depth analysis and were categorized based on specific fault types. This classification model was built upon clustering techniques utilizing the radial-basis function neural network.

Although the aforementioned methods have achieved some success in SDP, they exhibit certain limitations and shortcomings that warrant further consideration in ongoing research. First, certain traditional machine learning algorithms, such as Naive Bayes, SVMs, and random forest, while widely applied, may encounter performance bottlenecks when dealing with large-scale and high-dimensional data. Additionally, the effectiveness and accuracy of the unsupervised Bayesian algorithm introduced by Nizar Bouguila, based on finite Dirichlet mixtures for module classification, require further validation in practical applications. In the case of the LSSVM, as studied by Kumar [23], the impact of kernel function selection on the model's performance is highlighted. Furthermore, the evaluation of regression tree algorithms in the literature indicates that, despite their achievements in SDP, their adaptability to different datasets and features remains a subject requiring in-depth investigation. Concerning ensemble methods, although Wang's research suggests the superiority of voting and random forest among seven ensemble methods, the optimal choice for specific problems still merits further exploration. Additionally, Jun's three cost-sensitive boosting algorithms demand more empirical research to ascertain their robustness and performance across diverse application scenarios. In summary, despite significant progress, current SDP methods face challenges in handling large-scale, high-dimensional data and achieving generalization across diverse problem domains. Addressing these challenges necessitates further in-depth research and refinement of existing approaches.

2.2. DL in SDP. In response to the limitations of handcrafted features in achieving accurate SDP, several DL-based approaches have emerged. These methods aim to uncover intricate semantic and structural characteristics embedded within source code. They achieve this by extracting features from token vectors derived from program abstract syntax trees (ASTs), which are then utilized by classifiers to build robust SDP models. Viet Phan et al. [28] propose a novel approach that harnesses precise graphs representing program execution flows, leveraging deep neural networks to automatically capture defect-related features. Lu et al. [29] employ a deep belief network SDP model, utilizing deep belief nets composed of multilayer restricted Boltzmann machines to extract data features in a comprehensive manner. Albahli [30] devises an ensemble classifier that combines the outputs of three individual classifiers (random forest, XGBoost, multilayer perceptron) to create an efficient and state-of-the-art prediction model. However, it is worth noting that less emphasis is placed on examining the relationships between different software metrics in this approach.

In addressing the limitations associated with handcrafted features for accurate SDP, several DL-based approaches have been introduced. These methodologies strive to unveil intricate semantic and structural characteristics embedded within source code, primarily by extracting features from token vectors derived from program ASTs. While these techniques have shown promise, they are not without their shortcomings. One notable limitation lies in the method proposed by Viet Phan et al. [28], which utilizes precise graphs representing program execution flows [31]. Although this approach leverages deep neural networks to automatically capture

defect-related features, it may face challenges in scalability and generalization across diverse codebases due to the intricacies involved in accurately representing program execution flows. The approach employed by Lu et al. [29], utilizing a deep belief network for SDP, relies on multilayer restricted Boltzmann machines to comprehensively extract data features. However, the effectiveness of this model may be hindered by the potential overfitting to specific datasets and the inherent complexity of interpreting the learned representations, which could impede the model's transparency and interpretability. Albahli [30] ensemble classifier, combining outputs from random forest, XGBoost, and multilayer perceptron, represents a state-of-the-art prediction model. Nevertheless, the approach exhibits a notable drawback in its relative neglect of exploring the relationships between different software metrics. This omission may limit the model's ability to discern nuanced interdependencies among various metrics, potentially hindering its capacity to provide comprehensive defect predictions. In conclusion, while these DL-based approaches demonstrate advancements in SDP, they are not immune to challenges. Addressing these limitations is crucial for enhancing the robustness, scalability, and interpretability of such models, ultimately contributing to the continued evolution of effective SDP methodologies.

2.3. Feature Selection in SDP. Feature selection is also called feature subset selection or attribute selection. It refers to the selection of N features from the existing M features to optimize the performance of the system. At present, feature selection methods include term frequency [32], information gain (IG) [33], mutual information (MI) [34], statistics [35], cross-entropy [36], text weight of evidence [37], and so on. Khoshgoftaar and Seliya [9] used 16 software datasets to examine seven filter-based feature selection techniques for comparison, including chi-square (CS), IG, gain ratio, symmetric uncertainty (SU), Relief with two variables (RF and RFW) and signal-to-noise ratio [18]. In the research conducted by Ali et al. [38], they employed a predictive model that combines cost-sensitive logistic regression and decision tree ensemble methods. Their objective was to identify the most effective features for achieving precise defect prediction in software components. Meanwhile, Priyadarshini et al. [39] conducted a systematic examination of 12 automated feature selection techniques, considering factors such as consistency, correlation, performance, computational cost, and their impact on interpretability. The experimental outcomes revealed variations in the results across different feature selection methods and highlighted that the resulting subset of metrics included highly correlated measures [40].

Feature selection is crucial in SDP, but current methods have limitations. Fixed filter methods, lack of diversity due to feature correlation, high computational costs, inadequate handling of nonlinear relationships, and sensitivity to noise restrict their practical effectiveness [39]. Future research should focus on overcoming these limitations by combining methods, incorporating domain knowledge, or developing new algorithms to enhance robustness and applicability. Careful consideration of method performance in specific contexts is crucial when selecting feature selection methods.

2.4. Feature Extraction in SDP. Few feature extraction studies have been conducted on SDP over the past two decades. Pandey et al. [41] applied K-PCA to remove various redundant and irrelevant features and achieved satisfactory results. Linear discriminant analysis (LDA) [42] is a feature extraction method based on class label information of training samples aiming at maximizing the between-class scatter while simultaneously minimizing the within-class scatter. Malhotra and Khan [43] combined the principal component analysis (PCA), LDA, and kernel-based PCA with SVM to build a SDP model. Mahalanobis distance metric learning (MDML) is another feature extraction method based on pairwise cannot-link constraints (CC) and must-link constraints. The CC set is composed of the pairwise data that belong to the same class, and the must-link constraint set consists of pairwise data with heterogeneous class labels. Based on MDML, Xiang et al. [44] took advantage of the Mahalanobis learning matrix to maximize total distance in CC's pairwise and minimize total distance in must-link constraint's pairwise. MDML was also utilized in the works of NezhadShokouhi et al. [45] over the NASA datasets to overcome high-dimensional imbalanced problems. In a recent study conducted by Zhang et al. [46], they utilized an innovative deep neural network known as SSDAE. This network was employed to extract a set of novel combined features that excel in capturing robust deep semantic feature representations while eliminating irrelevant or redundant features within software projects.

However, despite the achievements of these feature extraction methods in SDP, there are still some limitations. First, the computational complexity of methods like K-PCA and MDML may increase with the scale of the dataset, limiting their practical application in large-scale software projects. Second, some methods rely on specific constraint conditions, which may make them less flexible for different types of data or problems. Additionally, although SSDAE has proven to be effective in extracting deep semantic features, its computational overhead may be relatively high, necessitating consideration in resource-constrained environments. Future research could focus on optimizing these feature extraction methods to improve their efficiency and applicability, taking into account the requirements of different application scenarios.

3. Datasets and Metrics

To predict software defects and evaluate the proposed DQN-FE-SDP, we selected 22 datasets from different open projects in the PROMISE repository and the NASA Metrics Data Program (MDP). The NASA MDP dataset used in this article is a cleaned version, from which all noise and dirty data have been removed. This data-cleaning process consists of two main parts: case handling and feature handling. In case handling, implausible or conflicting feature value cases are first identified and removed, followed by the elimination of

Table 1: NASA and PROMISE datasets.

Datasets	No. of metrics	No. of data	Defect rate (%)
CM1	38	344	12.209
JM1	22	9,591	18.340
KC1	22	2,095	15.513
MC1	39	8,737	2.778
MC2	40	125	35.2
MW1	38	263	10.266
PC1	38	735	8.299
PC2	37	1,493	1.072
PC3	38	1,099	12.557
PC4	39	1,378	12.980
PC5	39	16,962	2.960
ant-1.7	20	745	22.28
tomcat	20	848	8.97
xalan-2.4	20	723	12.21
xalan-2.5	20	803	48.19
xalan-2.6	20	885	46.44
camel	20	339	3.8
synampe	20	256	33.59
velocity-1.6	20	229	34.06
xerces-1.4	20	588	74.31
prop-6	20	660	10
poi-3.0	20	442	63.57

identical or inconsistent cases that may pose problems due to cross-validation strategies. In feature handling, cases with missing values are removed, and features that are constant or identical across the dataset are eliminated to improve the predictive performance of the model [47]. They are extensively adopted in previous empirical studies, and more than 20 studies show the representative and diversity of these two datasets [43].

3.1. Datasets. We have used 11 NASA MDP and 11 PROMISE repository datasets in our experiments listed in Table 1, in which CM1, JM1, KC1, MC1, MC2, MW1, PC1, PC2, PC3, PC4, and PC5 are from NASA MDP repository, and ant-1.7, tomcat, xalan-2.4, xalan-2.5, xalan-2.6, camel, synampe, velocity1.6, xerces-1.4, prop-6, and poi-3.0 are from PROMISE repository. It is obvious that the defect rates of most datasets are rather low, especially those of MC1, PC1, PC2, tomcat, and camel, which could cause a class imbalance problem. The strategy for optimizing class-imbalanced datasets will be described in Section 4.2.

3.2. The Software Metrics of NASA and PROMISE. A large amount of original software complexity measurements are adapted as features of SDP datasets, such as Chidamber and Kemerer's object-oriented metrics, McCabe metrics, Halstead's complexity metrics, and lines of code (LOC). The datasets of NASA consist of a total of 40 features such as the LOC, counts of a blank line, number of operators and operands, design complexity, essential complexity, etc., shown in Table 2. Following the traditional learning-based SDP, we adapt software complexity measurements as features and call them metric elements in this paper. We assigned a number to each metric element for reference when we explored the interrelation of them based on specific datasets in experiments (see Section 6.3).

The datasets of PROMISE include 20 metric elements such as depth of inheritance tree, cohesion among methods of class, and number of public methods, etc., shown in Table 3. We also assigned an order number to each metric element for reference when we explored the interrelation between them based on specific datasets in experiments (see Section 6.3).

4. Methodology

4.1. Approach Overview. Figure 1 depicts a high-level overview of DQN-FE, which is comprised of five major components, including preprocessing of datasets, ranking of metric elements, relation matrix construction, SMP determination, and training and classification, which are listed as follows:

(i) Preprocessing of datasets: BalanceCascade sampling technique is applied to split the majority class into smaller modules so that the optimal datasets are obtained and the positive classes and negative classes have equal quantity.

(ii) Metric elements ranking: ECE is adapted to calculate and rank the weight of metric elements in order to solve the overfitting problem.

(iii) Relation matrix construction: relation matrix is defined based on RMT to explore the relation degree of metric elements expressed as relation coefficient.

(iv) SMP determination: the combination of ranking of each individual metric elements and relation matrix representing the relation degree among them is used as a reward to optimize a CNN for obtaining SMP.

(v) Training and classification: decision tree, SVM, and K-nearest neighbor (KNN) are utilized on 11 NASA and 11 PROMISE datasets to train and classify, which are separated into two categories: train set (80%) and test set (20%).

4.2. Class Imbalance-Oriented Preprocessing of Datasets. The fact that the positive data in many datasets is less than 10% of the total data, such as PC2 and MC1, leads to a class imbalance problem, which could result in the unsatisfactory output of the training model.

We apply the BalanceCascade [20], which is a supervised learning method based on the Adaboost [48] algorithm and takes it as the base classifier. BalanceCascade is an undersampling method that divides a dataset into several subdatasets so that the number of positive classes is equal to that of negative classes in each subdatasets.

In each round of the loop, an AdaBoost classifier is trained by using the training set with the same number of majority classes and minority classes. Then, the classifier is used to decide the subdataset of majority classes, which could combine with minority classes as a new dataset, by controlling the false positive (FP) rate. After deleting the subdataset from majority classes, BalanceCascade reduces the quantity

Table 2: Software metrics of NASA datasets.

1. LOC BLANK	2. DESIGN DENSITY	3. HALSTEAD LEVEL
4. BRANCH COUNT	5. DESIGN COMPLEXITY	6. HALSTEAD PROG TIME
7. CALL PAIRS	8. EDGN COUNT	9. HALSTEAD VOLUME
10. LOC CODE AND COMMENT	11. ESSENTIAL COMPLEXITY	12. MAINTENANCE SEVERITY
13. LOC COMMENTS	14. ESSENTIAL DENSITY	15. DEFECTIVE
16. CONDITION COUNT	17. LOC EXECUTABLE	18. MULTIPLE CONDITION COUNT
19. HALSTEAD EFFORT	20. PARAMETER COUNT	21. NUM OPERATORS
22. CYCLONATIC DENSITY	23. GLOBAL DATA COMPLEXITY	24. NUM UNIQUE OPERANDS
25. DECISION COUNT	26. GLOBAL DATA DENSITY	27. NUM UNIQUE OPERTORS
28. DECISION DENSITY	29. HALSTEAD EFFORT	30. NUMBER OF LINES
31. HALSTEAD DIFFICULTY	32. NODE COUNT	33. PERCENT COMMENTS
34. CYCLONATIC COMPLEXITY	35. NORMALIZED CYLOMATIC COMPLEXITY	36. LOC TOTAL
37. HALSTEAD ERROR EST	38. NUM OPERANDS	39. MODIFIED CONDITION COUNT
40. HALSTEAD LENGTH		

Table 3: Software metrics of PROMISE datasets.

1. Afferent couplings	2. Coupling between methods	3. Inheritance coupling
4. Measure of functional abstraction	5. LOCM3	6. Data access metric
7. Arithmetic mean value of cyclomatic complexity	8. Lines of code	9. Number of children
10. Cohesion among methods of class	11. Lack of cohesion in methods	12. Number of public methods
13. Depth of inheritance tree	14. Efferent couplings	15. Average method complexity
16. Coupling between object classes	17. Response for a class	18. Measure of aggregation
19. Greatest value of CC (Max_CC)	20. Weighted methods per class	

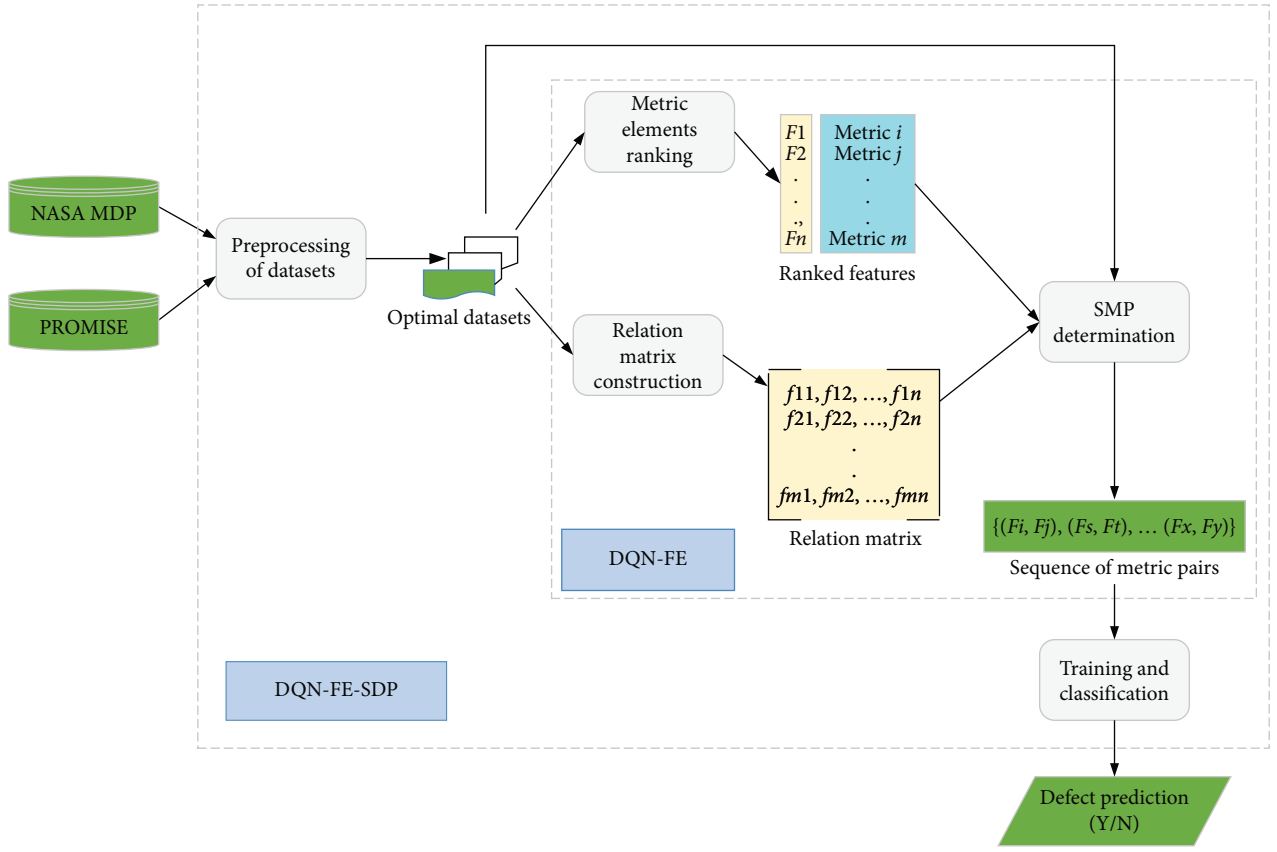

Figure 1: A high-level overview of DQN-FE-SDP.

Input: D, one dataset from NASA and PROMISE, such as PC1
P, a training set containing defective modules
N, a training set containing nondefective modules
T, the number of subsets to be sampled from N
S_i, the number of weak learners H_i
Output: A combined classifier $H(x)$
1. f $= \sqrt{T-1}\sqrt{\frac{|p|}{|N|}}$ //f is the FP rate (the error rate of misclassifying a nondefective module to the defective module) that H_i should achieve
2. For $i = 0$ to T do{
3. Randomly sample a subset N_i from N, $|N_i| = |P|$.
4. Learn H_i using P and N_i. H_i is an Adaboost ensemble with weak learners $H_{i,j}$ and corresponding weights $\alpha_{i,j}$,θ_i is the adjustment parameter of H_i.
$H_{\mathrm{i}} = \mathrm{sgn}(\sum_{j=1}^{s_i} \alpha_{i,j} h_{i,j}(x) - \theta_i)$
5. Adjust θ_i such that H_i's false positive (FP) rate is f
6. Remove from N all examples (the number is $|P| \times (1-f)$) that are correctly classified by H_i
7. }

Algorithm 1: Datasets preprocessing algorithm based on BalanceCascade.

of data in majority classes. At the same time, the classifier is optimized. By repeating the above procedure, the quantity of data in majority classes is kept being decreased until it equals to the quantity of data of minority classes, and the goal classifier is obtained. Algorithm 1 shows the detailed process of BalanceCascade.

The mathematical model of BalanceCascade is shown in Formula (1):

$$H(x) = \mathrm{sgn}\left(\sum_{i=1}^{\mathrm{T}} \sum_{j=1}^{s_i} \alpha_{i,j} h_{i,j}(x) - \sum_{i=1}^{T} \theta_i\right), \tag{1}$$

where T is the number of subsets extracted from majority class, $h_{i,j}(x)$ is the based classifier, the parameter θ_i is the coefficient for controlling the FP rate, $\alpha_{i,j}$ is the weight of $h_{i,j}(x)$.

The datasets preprocessing algorithm based on BalanceCascade is described in Algorithm 1. Based on the BalanceCascade, we have divided 22 datasets into 209 datasets, each of which includes positive classes and negative classes with equal quantity. We have also considered oversampling methods, such as Random oversampling [49] and so on. However, experiments show that oversampling methods generate low-quality classes, leading to unsatisfactory performance on practical datasets.

4.3. Ranking of Metric Elements Based on ECE. ECE [50], which is also called KL distance, has been widely used in feature selection of text classification. Compared with the IG, ECE improves the efficiency of classification by ignoring the seldom-appearing features and eliminating their interference. It measures the importance of a metric element in a software defect dataset. In the feature selection of SDP, the ECE can be used to reflect the probability distribution of whether there is a defect. The mathematical model of ECE is shown in Formula (2):

$$\mathrm{ECE}(t) = P(t) \sum_{i=1}^{|c|} P(c_i|t) \log \frac{P(c_i|t)}{P(c_i)}. \tag{2}$$

We define $P(t)$ as the probability associated with a metric element, where $P(c_i)$ represents the probability that a data point belongs to class c_i (which includes both positive and negative classes). $P(c_i|t)$ denotes the probability that metric element t is associated with class c_i. The symbol $|c|$ is used to denote the total number of classes across both categories. When $P(c_i|t)$ is significantly higher than the corresponding $P(c_i)$, it indicates a strong correlation between metric element t and the specific category c_i.

As shown in Tables 2 and 3, each dataset contains more than 20 metric elements, and the degree of correlation between some of the metric elements is relatively high. Suppose that the set C is the metric element set of a dataset, expressed as $C = \{c_1, c_2, \ldots, c_k\}$. F indicates the candidate feature subset, denoted as $F = \{f_1, f_2, \ldots, f_m\}$ and S represents the feature set that has been selected or determined, denoted as $S = \{s_1, s_2, \ldots, s_n\}$, f and s are the elements in the set F and S, respectively. For each candidate metric element f in F, the ECE (f) value is calculated, according to which a sequence is constructed in descending order.

4.4. RMT-Based Relation Matrix Construction. The ECE considers the relationship between feature and category, but it ignores the relationship between features. Therefore, we calculate the relation matrix based on the RMT to express the relationship between different metrics elements. First, the sample covariance is calculated as follows. Suppose X and Y are two metric elements; then, the covariance of X and Y is calculated as Formula (3).

$$\mathrm{Cov}(X, Y) = \frac{\sum_{i=1}^{n} (X_i - \bar{X})(Y_i - \bar{Y})}{n-1}, \tag{3}$$

Input: A set of datasets with K elements, each of which is expressed as
$D_i = \{(x_{1,1,000}, \ldots, x_{1,n}, s_1), (x_{2,1}, \ldots, x_{2,n}, s_2), \ldots, (x_{m,1}, \ldots, x_{m,n}, s_m)\}$
An ranked feature set $F = \{f_1, f_2, \ldots, f_n\}$ from the ECE (see Section 4.3)
A $N^{\times}N$ relationship matrix from RMT (see Section 4.4)
Episode number K which is the number of datasets after BalanceCascade (see Section 4.2)
Output: The SMC of datasets, optimize the policy network, and updating
For episode $i = 1$ to K do {
  Randomly initialize the parameters φ and Θ of CNN
  Initialize experience reply memory M
  Shuffle the training data D
  Initialize state $s_1 = \{(x_1, x_2)\}, x_1 \in F$ and $x_2 \in F$
  Initialize action $a_1 = A(s_1, \text{agent})$ represents feeding s_1 to agent
  For $t = 1$ to n do {
    The reward principle $r_t = \text{sigmoid}(R_t)$, the R_t is determined by the $\{f_1, f_2, \ldots, f_n\}$ and the $n \times n$ relationship matrix (see Formula (7))
    Set $a_t = A(s_t, \text{agent})$
      Select a greedy policy to update the current state: $s_{t+1} = \pi_\theta(s_t)$
      Store (s_t, a_t, r_t, s_{t+1}) to M and update the Q network
      Select (s_j, a_j, r_j, s_{j+1}) from M using
      Set $y_j =$
      $\begin{cases} r_j & \text{if episodetermi nates at } s_{j+1} \\ r_j + \gamma \max_a Q^{\pi}(s_{j+1}, a', \varphi) & \text{otherwise} \end{cases}$
      Perform a gradient descent step on $L(\theta)$
      $\theta : L(\theta) = (y - Q(s_j, a_j, \theta))^2$
      If a_t appeared in M and $a_t = \arg\max_a Q^*(s, a, \theta)$ then break
      Update parameters φ
      }
}

Algorithm 2: The algorithm of training DQN for DQN-based feature extraction.

where X_i and Y_i are the ith values of metric elements X and Y, respectively. Then combined with the standard deviation, Pearson's correlation coefficient r is calculated as Formula (4).

$$r(X, Y) = \frac{(n-1)\text{Cov}(X, Y)}{\sqrt{\sum_{i=1}^{n}(X_i - \bar{X})^2 \sum_{i=1}^{n}(Y_i - \bar{Y})^2}}. \quad (4)$$

When the correlation coefficients are filled into an $n \times n$ matrix, we obtain the relation matrix of metric elements. The value of each element in it ranges from -1 to 1. The higher the value is, the closer the metric elements are.

4.5. SMP Determination Based on DQN. In this section, we explain the procedure of the SMP determination based on DQN using ranking of metric elements (see Section 4.3) and relation matrix (see Section 4.4). After defining critical components of DQN, such as state space, action space, reward function, and so on, we illustrate the proposed algorithm in detail. Four definitions are given as follows for further explanation.

Definition 1. Revised Feature Set. Give an original feature set X including feature $x_1, x_2, \ldots, x_n$, a feature set $Y = \{y_k = (x_i, x_j) \text{—} x_i \in X \wedge x_j \in X\}$ is called a revised feature set of X.

Definition 2. Metric Pair. Each element in feature set Y, $y_i = (x_m, x_n)$, is called a metric pair when x_m and x_n both belong to metrics listed in Tables 2 and 3.

It is obvious that Y is $X \times X$, and feature set X could have many different revised feature sets. In order to improve the performances of SDP, such as precision, to the greatest extent, we present the goal feature set selection strategy in Algorithm 2.

Definition 3. Larger than Relation $\geqslant$. Suppose Y is a revised feature set, $(x_i, x_j) \in Y$ and $(x_s, x_t) \in Y$. If correlation coefficient of (x_s, x_t) is larger than or equal to (x_s, x_t), i.e., $r(x_i, x_j) \geqslant r(x_s, x_t)$, then we call $(x_i, x_j) \geqslant (x_s, x_t)$.

Definition 4. SMP. An ordered list of metric pairs is called a sequence of metric pairs when it satisfies the following condition: for any $y_i = (x_s, x_t)$ and $y_j = (x_m, x_n)$, if $i < j$, then $(x_s, x_t) \geqslant (x_m, x_n)$.

4.5.1. Essential Elements of DQN for SMP. RL algorithms enriched with DL capabilities have achieved victories over world champions in the game of Go and have outperformed

human experts in various Atari video games. Today, we view the challenge of feature extraction as akin to a guessing game [11]. In this context, an agent utilizes a CNN to construct the policy network, transforming the problem into an exciting and effective approach. The state of the environment is acted by a sequence of metric pairs, which are also the input of the CNN; the reward is determined by calculating the ranking of metric elements, relation matrix, and the number of defects; the Q-value is calculated from the SMP, i.e., output of the CNN; action is directed by Q-value for deciding a new SMP as an input of the CNN in the new loop.

Assume that one of the software defect dataset is $D = \{(x_{1,1},\ldots, x_{1,n}, s_1), (x_{2,1},\ldots, x_{2,n}, s_2),\ldots,(x_{m,1},\ldots, x_{m,n}, s_m)\}$ where $x_{i,j}$ is the value of metric elements and s_i is the label of the class, indicating the number of the defect. In DQN-FE, a trained CNN acts as an agent to fulfill the task of feature extraction. The detailed concepts are illustrated as follows:

(A) State S: The state of the environment is determined by the training sample. At the beginning of training, the agent receives the first sample x_1 as initial state s_1. The state s_t of the environment corresponds to the sample x_t. When a new loop begins, the state of the environment, i.e., training samples in the training dataset, is updated. The state of the environment in the context is a sequence of metric pair $S_t = \{(\text{metric}_i, \text{metric}_j),\ldots, (\text{metric}_m, \text{metric}_n)\}$, where each metric element of metric pairs is the feature of the original datasets.

(B) Action A: An action is the behavior of delivering a sequence of metric pairs to the agent. The results of actions are cataloged into two sets, one of which involves those abandoning the current metric pairs and another of which involves those preserving the current metric pairs.

(C) Policy π: The policy $\pi(s|a)$ is a mapping: $S \times A \to S \times A$, where S is a set of states, and A is a set of actions. $\pi(s_t|a_t)$ leads to a new state and responding a new action. The policy π in DQN-FE is determined by comparing the Q-value with that outputted in the previous loop. The detailed procedure is shown in Algorithm 2.

(D) Discount factor γ: $\gamma \in [0, 1]$ is to balance the immediate and future reward.

(E) Reward R: A reward r_t is the feedback from the environment through which we measure the success or failure of an agent's actions. In order to guide the agent to train CNN for obtaining the optimal policy, the absolute reward value of a metric pair is determined by its relation degree, the ranking of each metric element of the metric pair, and the label (number of errors) of current class.

The Q-learning algorithm constructs an objective function, the CNN model in our approach, for DL. It generates the sequence of metric pairs, according to which Q-value is calculated. The Q-value is compared with that in the last loop for determining the new sequence of metric pairs. In our approach, a function $Q(s, a, \omega)$ is defined for calculating the Q-value instead of $Q(s, a)$ in the original Q-learning algorithm, where is the current state, is the action and is the parameter of CNN. $Q(s, a, \omega)$ takes state as the inputs and outputs the Q-value of each action with new ω'. In essence, the fundamental concept behind the DQN (deep Q-network) algorithm is to address RL problems when dealing with complex, high-dimensional, and continuous state and action spaces. To achieve this, DQN employs CNNs to construct a Q-value network. This network is crucial for solving the RL tasks effectively. As the Q-value network is developed, the loss function of the CNN is iteratively determined. This process leads to the formulation of the mathematical model for Q-learning, which is expressed in Formula (5).

$$Q(S_t, A_t, \theta) = Q(S_t, A_t, \theta) + \alpha[R_{t+1} + \gamma \max Q(S_{t+1}, A_{t+1}, \theta) - Q(S_t, A_t, \theta)]. \tag{5}$$

The loss function is defined as Formula (6).

$$L(\theta) = E[R_{t+1} + \gamma \max Q(S_{t+1}, A_{t+1}, \omega^-) - Q(S_t, A_t, \omega)^2], \tag{6}$$

where network parameter θ denotes the loss of mean square error. During the training of the CNN using random gradient descent methods, historical data, i.e., sequence of metric pairs, are continuously collected and stored in the experience pool of DQN. When CNN is finished being optimized on a small batch of the data in the experience pool, the Q-value network is finished being constructed.

4.5.2. The Selection of Sequence of Metric Pairs Based on the Improved DQN Algorithm

(1) Reward Principle. The reward principle of our DQN-based feature extraction is established according to the ranking of the weight of metric elements (see Section 4.3) and relationship matrix (see Section 4.4). In order to obtain a revised feature set expressed as a sequence of metric pairs, we defined the reward function as Formula (7).

$$R_i = \text{Sigmoid}\left(\sum_{t=0}^{n} \text{Rank}(x_t) + \sum_{i=0,j=0}^{m} \text{Rel}(x_i, x_j) + |\,\text{Label}\,|\right), \tag{7}$$

where the Rank function gives the weight of different metrics obtained by ECE, the Rel function returns the correlation coefficient of different metrics obtained by RMT, and |Label| indicates the number of errors in the current class, the Sigmoid function normalizes the sum of the three components so that the computing results are ranged from 0 to 1 which makes it convenient to compare the value of them.

(2) Deep Network Application. In our DQN-based feature extraction, the policy π receives a sequence of metric pairs and returns a new one with the highest possibility of being selected as input for the next loop. The policy can be obtained by Formula (8).

$$\pi_\theta(s_i, a_i) = P(s_i|a_i). \tag{8}$$

The agent can get a positive reward when it correctly recognizes a metric pair. Rewards from different loops have to be accumulated so as to correctly generate the sequence of metric pairs. The cumulative rewards g_t can be obtained by Formula (9).

$$g_t = \sum_{k=0}^{\infty} \gamma^k r_t + k. \tag{9}$$

In RL, the Q function is responsible for calculating the quality of a state-action combination, which is shown in Formula (10).

$$Q^\pi(s, a, \theta) = E_\pi([g_t|s_t = s, a_t = a]). \tag{10}$$

According to Formula (5), the Q function can be obtained by Formula (11).

$$Q^\pi(s, a, \theta) = E_\pi[r_t + \gamma Q^\pi(s_{t+1}, a_{t+1}, \theta)]. \tag{11}$$

The agent can maximize the cumulative rewards by the Q^* function in which the greedy policy is applied. Optimal policy π^* is obtained according to cumulative rewards shown in Formula (12).

$$\pi^*(s|a) = \begin{cases} 1, & \text{if } a = \text{argmax}_a Q^*(s, a, \theta) \\ 0, & \text{else} \end{cases}. \tag{12}$$

In our DQN-based feature extraction, the tuples (s, a, r, s') involving temporary data generated during each loop are stored in the experience pool M. The agent randomly samples a mini-batch of set B composed of those tuples from M and performs a gradient descent method according to the loss function. The mathematical model is shown in Formula (13).

$$L(\theta_k) = \sum_{(s,a,r,s')\in B} (y - Q(s, a, \theta_k))^2, \tag{13}$$

where y is the target estimate of the Q function. The mathematical model of y is shown in Formula (14).

$$y = r + (1 - t)\gamma \max Q(s', a', \theta_{k-1}), \tag{14}$$

where s' is the state of environment after s, a' is the action performed by agent in state s'. The optimal Q^* function is obtained by minimizing the loss function. The maximum cumulative rewards are obtained by the greedy policy adapted in the optimal Q^* function. So the optimal policy π^*: $S \to S$ for DQN-based feature extraction is achieved. The procedure is shown in Figure 2.

(3) Training Details. The training process based on DQN feature extraction includes several steps: experience replay, target network update, and policy network update. During experience replay, the DQN algorithm saves the information (current state-action pair, next state-action pair, current state reward, and next state) in an experienced pool and randomly selects a batch of samples from the pool for the training of the policy network. This process is mainly to reduce the correlation between data and improve the efficiency of training.

In the target network update process, after every four iterations of selecting state-action pairs, the parameters of the current network are copied to the target network to obtain a set of updated target Q-values. Then, these target Q-values are used to calculate the error and update the parameters of the current network during training. In this way, the parameters of the target network remain unchanged during the training process, while the parameters of the current network are updated after each iteration, allowing DQN to learn a more accurate Q-value function.

The policy network of DQN adopts Q-learning for updates. During each decision-making process, the policy network estimates the value function of each possible action based on the current state and selects the action with the highest value as the current policy. The policy network is updated by minimizing the mean square error between the policy network and the target network. Specifically, every time the agent executes an action and observes a new state and reward, it stores this experience tuple in an experience replay buffer. Then, the agent randomly selects a certain number of experience tuples from the replay buffer to update the policy network.

The DQN feature extraction network structure is trained with an initial learning rate of 0.001, a reward discount factor of 0.99, and an experience pool size of 10,000. Meanwhile, the greedy epsilon value decreases from 1 to 0.01 during the training process. The purpose of choosing an initial learning rate of 0.001 is to make the loss function decrease faster in the initial stage of training. As training proceeds, the learning rate gradually decreases until the model converges to a smaller loss. The change in the greedy epsilon value is to facilitate more exploration in the initial stage of training and gradually increase the probability of utilizing the optimal policy. The DQN feature extraction process uses the NASA MDP storage library dataset and PROMISE storage library dataset as input and outputs an optimal state-action pair sequence to replace all state-action pairs. More precisely, the Q network is a neural network model that excludes a final softmax layer. We configure its architecture and hyperparameters, such as the quantity of convolution layers, the dimensions of convolution kernels, the number of hidden layers, and the neuron count, with the goal of enhancing the model's capacity for generalization and fitting.

The architecture of the CNN utilized in our study is meticulously tailored to address the unique challenges posed by the dataset's diverse feature dimensions across various projects. In particular, the PROMISE dataset, with its 20-dimensional feature vectors, required a thoughtful approach to ensure the effective application of convolutional operations. Our methodology involved expanding these features to a 25-dimensional space by adding five columns of zeros. This process, detailed in Figure 3, is designed to maintain

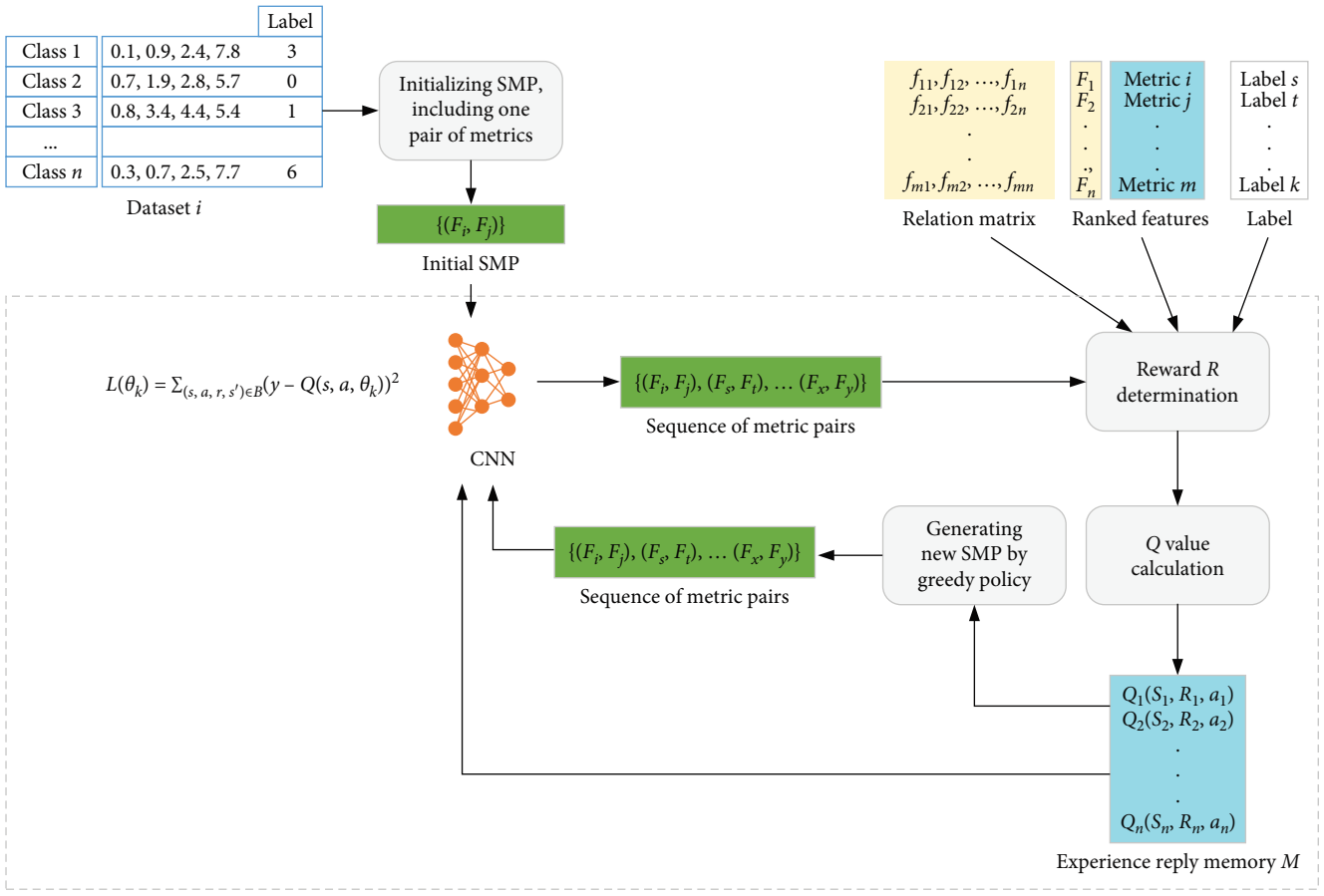

Figure 2: Sequence of metric pairs determination based on DQN.

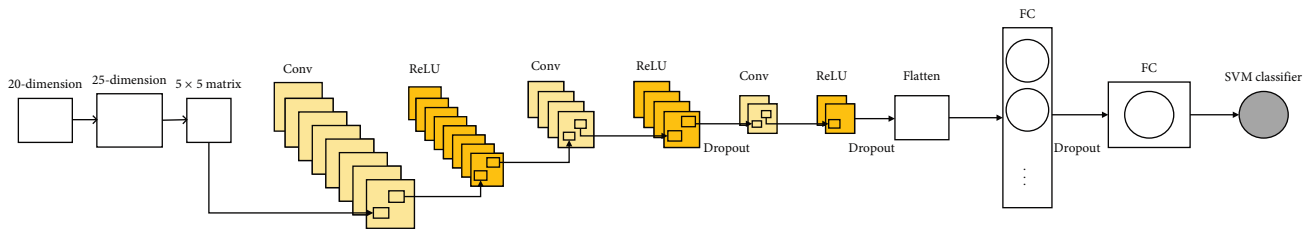

Figure 3: CNN parameter details in feature extraction based on DQN.

spatial resolution post-convolution [51]. The augmented features are then restructured into a 5×5 matrix format, a decision influenced by our comprehension of feature space and guided by successful precedents in the literature for handling similar data structures [52]. This restructuring is crucial as it allows the network to be more aligned with two-dimensional convolutional operations, thereby enhancing feature extraction capabilities.

The mentioned reshaping process enhances the network's ability to extract and represent features more effectively. Then, we start with a single-channel feature matrix, and the first convolutional layer has an input channel of 1 and an output channel of 64. The role of this layer is not a typical convolution operation but rather introduces nonlinearity through pointwise convolution, transforming the input 5×5 matrix into a 64-channel feature map, with each channel having a spatial size of 5×5. Dropout is not applied in this layer, and its main purpose is to introduce nonlinearity. The second convolutional layer takes the 64-channel input from the previous layer and reduces it to 32 channels using a 3×3 convolutional kernel, further compressing the feature map to a size of 3×3. A Dropout layer is added to mitigate overfitting. The third convolutional layer further reduces the 32-channel input to 16 channels with a 4×4 convolutional kernel, forming a 4×4 feature map. Another Dropout layer is added for regularization.

Our CNN comprises an input layer followed by two convolutional layers, each paired with the ReLU activation function, a choice substantiated by extensive empirical evidence. The ReLU function is widely recognized for its performance benefits and computational efficiency across various network architectures [53].

The selection of convolutional layers is based on a balance between model complexity and computational expense, reflecting common configurations for analogous tasks [54]. This setup is adept at capturing the intricate inter-feature relationships, providing a nuanced understanding of the dataset, which is vital due to the variable dimensionality of features across projects. The dual-layer configuration was chosen not only for its empirical success in delineating spatial hierarchies within the data [55] but also because it represents a strategic balance. There is substantial evidence to suggest that additional layers may not consistently yield performance improvements and could potentially lead to overfitting or unnecessary computational complexity without commensurate accuracy gains, especially in datasets with a certain level of intricacy. Therefore, our selection of a two-layer model is a deliberate compromise to leverage depth while preserving computational efficiency and mitigating the risk of overfitting.

By providing these additional details, we aim to offer a thorough understanding of our CNN architecture, specifically the in and out channels of each convolution layer, the shape of the input and output of these layers, and the incorporation of dropout, thereby enhancing the clarity and replicability of our proposed method.

Algorithm 2 depicts the procedure of transforming the metric elements into a sequence of metric pairs.

4.6. Training and Classification through Binary Classification Algorithms. In essence, DQN-FE is a binary classification algorithm-based machine learning model applied in SDP, whose prominent difference with other SDPs is the application of DQN-based feature extraction. So, it is unavoidable to apply some binary classification classifier to train and classify over special datasets. In our approach, a simple but widely adopted classifier, decision tree, is chosen to accomplish our goals. On the other hand, in order to further verify the effectiveness of DQN-based feature extraction, another two classifiers, SVM and KNN, are also chosen in experiments for sensitive analysis in Section 6.4.

SVM: SVM transforms the classification problem into the problem of finding the classification plane and realizes the classification by maximizing the distance between the classification boundary point and the classification plane.

K-Nearest Neighbor: The KNN classification algorithm is one of the most straightforward techniques in data mining for classification. It categorizes each sample based on the values of its k closest neighbors.

Decision Tree: In this classification algorithm, we create a decision tree using training data to categorize unknown data. Within the decision tree structure, each internal node signifies a test conducted on an attribute; each branch signifies the outcome of that test, and each leaf node stores a class label.

5. Experimental Setup

5.1. Effectiveness Validation Measures of the SDP. To comprehensively demonstrate the effectiveness of our proposed feature extraction method based on DQN, we consider four widely adopted performance metrics commonly used in the field of SDP to evaluate the model's capabilities: the AUC [56], precision [57], F-measure [58], and MCC [59]. We use the abovementioned confusion matrix to describe predictions as true positives (TP), FPs, true negatives (TN), or false negatives (FN) (Table 4).

Table 4: Confusion matrix.

		Predicted	
		Positive	Negative
Real results	True	TP	TN
	False	FP	FN

"Precision," also referred to as "positive predictive value," is a metric primarily focused on determining the proportion of relevant instances among the retrieved instances. The fundamental formula for precision is as follows:

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}. \tag{15}$$

Recall, also known as true positive rate or sensitivity, is an evaluation metric commonly used in machine learning and statistics to measure the ability of a model to correctly identify positive instances from a dataset. It is defined as the ratio of TP (i.e., the number of positive instances correctly predicted by the model) to the sum of TP and FN (i.e., the number of positive instances incorrectly predicted as negative by the model). Mathematically, recall can be expressed as follows:

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}}. \tag{16}$$

The F-measure, sometimes referred to as the F_1 score, provides a balanced assessment by combining precision and recall into a single metric. It offers insights into both precision and recall simultaneously and yields values within the range of [0, 1]. A value of 0 indicates all incorrect predictions, while a value of 1 represents all correct predictions. The basic formula for calculating the F-measure is as follows:

$$F - \text{measure} = 2 \times \frac{\text{recall} \times \text{precision}}{\text{recall} + \text{precision}}, \tag{17}$$

MCC is a binary classification metric that accounts for true and FPs and negatives. It ranges from −1 to 1, with higher values indicating better prediction performance:

$$\text{MCC} = \frac{(\text{TP} \times \text{TN}) - (\text{FP} \times \text{FN})}{\sqrt{(\text{TP} + \text{FP})(\text{TP} + \text{FN})(\text{TN} + \text{FP})(\text{TN} + \text{FN})}}. \tag{18}$$

5.2. Statistical Analysis: Scott-Knott Effect Size Difference (ESD) Test. In this paper, we initially utilized the Scott-Knott ESD test to rank multiple feature selection or feature

Table 5: Settings for DQN hyperparameters.

DQN hyperparameter	Setting
Learning rate	0.0005
Discount factor	0.98
Replay buffer size	10,000
Batch size	64
Target network update frequency	5,000
Initial ε for ε-greedy policy	1.0
Decay rate for ε-greedy policy	0.995

extraction methods, as well as multiple defect predictors, based on four evaluation indicators. Subsequently, we presented boxplots with the Scott-Knott ESD test to visually illustrate the performance differences among these feature selection or feature extraction methods and defect predictors. The Scott-Knott ESD test is a statistical approach that extends the Scott-Knott test, which is a mean comparison approach that employs hierarchical clustering to partition multiple methods into statistically distinct groups with significant differences, taking into consideration the effect size (i.e., magnitude of difference) among multiple methods within a group and between groups.

5.3. Configuration of DQN Training. In our experiment, we utilized the DQN algorithm and configured its hyperparameters, as depicted in Table 5. These hyperparameter settings play a crucial role in determining the performance and training efficacy of the DQN algorithm. Specifically, we set the learning rate to 0.0005, which governs the step size taken by the model during each update. The discount factor was set to 0.98, influencing the model's emphasis on future rewards. The replay buffer size was configured as 10,000, determining the capacity for storing experience replay data during training. A batch size of 64 was employed, indicating the number of samples drawn from the replay buffer for each update. The target network update frequency was established as 5,000, representing the interval at which the target network is updated. Furthermore, the initial ε value for the ε-Greedy policy was set to 1.0, affecting the degree of exploration by the model in the early stages of training. The decay rate for the ε-Greedy policy was defined as 0.995, determining the rate at which the ε value decreases over training steps. The rational selection of these hyperparameters is critical for achieving optimal performance in the experiment.

6. Experiment

Each of our experiments runs on the 16 GB RAM of the NVIDIA GPU server repeatedly for 10 times, and we took the mean value of the results as the final result. In this section, we will present comprehensive results from our experiments and assess the model's performance by addressing the following five RQs. We will show the results of the ranking of metric elements (see Section 6.2), the interrelation of metric elements, and the sequence of metric pairs (see Section 6.3). In order to verify the effectiveness of the DQN-based feature extraction, we use the three binary classification algorithms (SVM, KNN, and decision tree) without the DQN-based feature extraction as the baseline for comparison. The sensitivity analysis shows that the performances (precision, *F*-measure, AUC, and MCC) are sensitive to the DQN-based feature extraction in that they have been improved, respectively, comparing the binary classification algorithms based on SDPs using the DQN-based feature extraction and those without using it (see Section 6.4). At the same time, we conduct experiments to compare our approach with four state-of-the-art methods (K-PCA-ELM [41], CFIW-TNB [60], MDA-O [61], and weighted ensemble model (WEM) [62]), and experimental results show that the SDP approach using the proposed DQN-based feature extraction is superior to the state-of-the-art methods in the aspects of precision, *F*-measure, AUC, and MCC (see Section 6.5). Finally, we compared four state-of-the-art feature selection algorithms on four metrics.

6.1. Parameter Settings. Following the identical practice principle with Zhu et al.'s [63] research, our feature extraction is exclusively performed on the training dataset. To ensure consistent feature dimensions between them, we adjust the test dataset based on the features extracted from the training dataset, thereby achieving dimensional uniformity.

6.2. RQ−1: How Feature Ordering Based on ECE Removes Nonphasic Features? As we have discussed in Section 4.3, we use ECE to obtain the importance of metric elements and remove irrelevant, redundant, and noisy features in 22 datasets. In order to compare the weight of various metric elements, this section compared the results of the 22 datasets. The statistical distribution of each group of data is shown in Figures 4 and 5: The weight of various metric elements of PROMISE datasets is shown in Figure 4, and the weight of various metric elements of NASA datasets is shown in Figure 5. Table 6 shows the irrelevant, redundant, and noisy features of the NASA dataset.

In Figure 4, we have aggregated the bar graph over the weight of the top six metric elements in PROMISE datasets. The *x*-axis indicates the various metric elements concerning PROMISE datasets, whereas the *y*-axis indicates various weight values of corresponding metric elements. In the datasets of ant-1.7, tomcat, xalan-2.4, xalan-2.5, xalan-2.6, synampe, xerces-1.4, prop-6, and poi-3.0, the metric element of response for a class has the highest weight. Especially in the datasets of ant-1.7, xerces-1.4, and prop-6, the weight of the metric element of response for a class is higher than 70%. Therefore, when there is the metric element of response for a class, the SMP will obtain a higher reward in our SDP approach. In the dataset of velocity, the metric element of the lines of code has the highest weight and the metric element of the response for a class ranks second. In the dataset of camel, the metric element of coupling between object classes has the highest weight and the metric of the response for a class ranks 6th.

In Figure 5, we present the weight of the top six metric elements in NASA datasets. The *x*-axis indicates the various metric elements concerning NASA datasets, whereas the *y*-axis indicates various weight values of corresponding metric elements. It shows the weight of various metric elements,

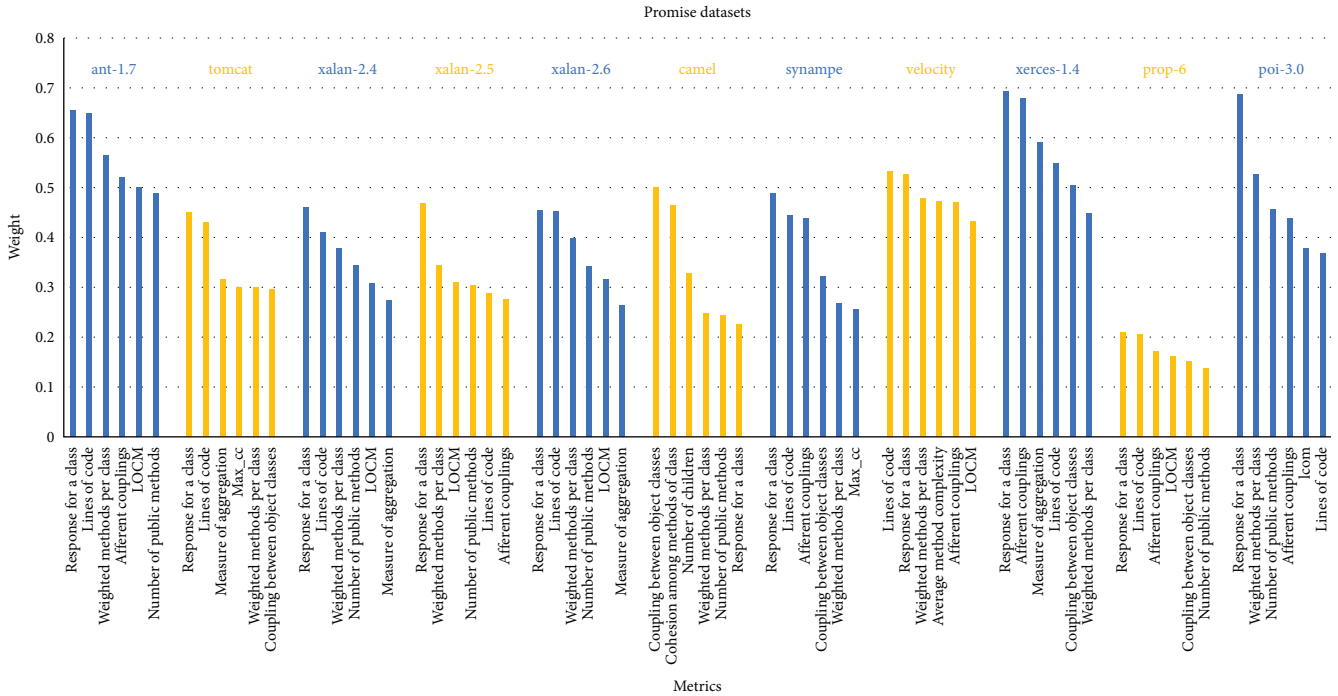

FIGURE 4: The weight of metric elements PROMISE datasets.

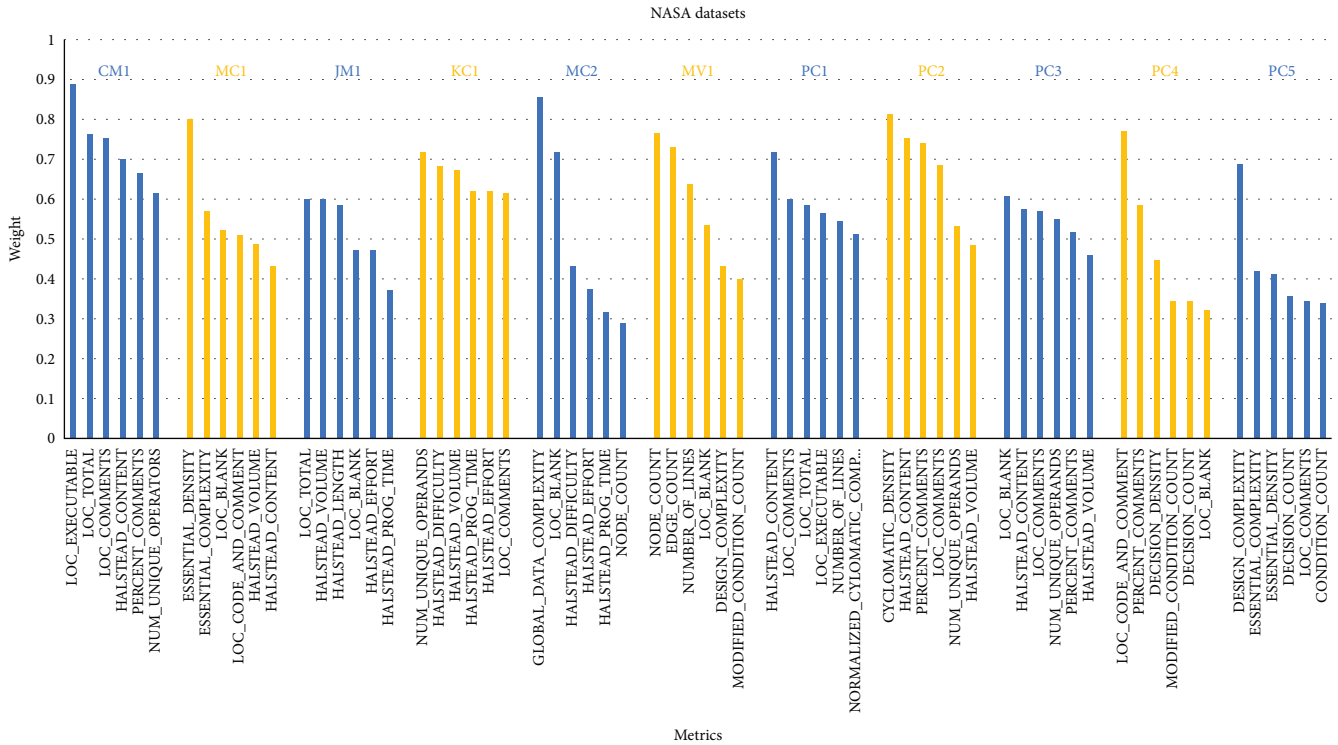

FIGURE 5: The weight of metric elements NASA datasets.

and the most important of the different datasets are LOC EXECUTABLE (CM1), ESSENTIAL DENSITY (MC1), LOC TOTAL (JM1), NUM UNIQUE OPERANDS (KC1), GLOBAL DATA COMPLEXITY (MC2), NODE COUNT (MW1), HALSTEAD CONTENT (PC1), CYCLOMATIC DENSITY (PC2), LOC BLANK (PC3), LOC CODE AND COMMENT (PC4), and DESIGN COMPLEXITY (PC5). In most datasets of NASA, the LOC type of metric elements have higher weight than others. Table 6 lists the irrelevant, redundant, and noisy features of the NASA dataset.

Table 6: The irrelevant, redundant, and noisy features of the NASA dataset.

Datasets	Metric
CM1	LOC_CODE_AND_COMMEN
	HALSTEAD_LEVEL
	DECISION_DENSITY
	MODIFIED_CONDITION_COUNT
	MAINTENANCE_SEVERITY
	MAINTENANCE_SEVERITY
	ESSENTIAL_DENSITY
	ESSENTIAL_COMPLEXITY
	EDGE_COUNT
	DESIGN_DENSITY
	HALSTEAD_DIFFICULTY
JM1	ESSENTIAL_COMPLEXITY
	LOC_CODE_AND_COMMENT
KC1	LOC_CODE_AND_COMMENT
MC1	BRANCH_COUNT
	MODIFIED_CONDITION_COUNT
	CYCLOMATIC_COMPLEXITY
	ESSENTIAL_DENSITY
	DESIGN_DENSITY
	DESIGN_COMPLEXITY
	GLOBAL_DATA_DENSITY
	GLOBAL_DATA_COMPLEXITY
	MAINTENANCE_SEVERITY
	PARAMETER_COUNT
MC2	DECISION_DENSITY
	LOC_CODE_AND_COMMENT
	DESIGN_DENSITY
	NUM_OPERANDS
	HALSTEAD_VOLUME
	PARAMETER_COUNT
	PERCENT_COMMENTS
	HALSTEAD_ERROR_EST
	HALSTEAD_LENGTH
	HALSTEAD_CONTENT
PC1	CONDITION_COUNT
	MULTIPLE_CONDITION_COUNT
	DESIGN_DENSITY
	DECISION_COUNT
	DECISION_DENSITY
	MAINTENANCE_SEVERITY
	ESSENTIAL_DENSITY
	ESSENTIAL_COMPLEXITY
	HALSTEAD_DIFFICULTY
PC2	PARAMETER_COUNT
	ESSENTIAL_COMPLEXITY
	ESSENTIAL_DENSITY
	DESIGN_COMPLEXITY
	MAINTENANCE_SEVERITY
	LOC_EXECUTABLE
	DECISION_DENSITY
PC3	DECISION_DENSITY
	PARAMETER_COUNT
	ESSENTIAL_COMPLEXITY
	DESIGN_DENSITY
	ESSENTIAL_DENSITY
PC4	PARAMETER_COUNT
	DESIGN_DENSITY
	CALL_PAIRS
	ESSENTIAL_COMPLEXITY
	ESSENTIAL_DENSITY
PC5	ESSENTIAL_COMPLEXITY
	ESSENTIAL_DENSITY
	PARAMETER_COUNT

6.3. RQ-2: How Can We Find the Interrelated Features and Leverage the Interrelation between Features to Generate the Sequence of Metric Pairs?

6.3.1. Experiment on Feature Interrelation. As we have discussed in Section 4.4, we use RMT to obtain the relation degree between two metric elements in 22 datasets. Since it is not the research focus of this paper, we only represent the results of the various metric elements relationship for the ant-1.7 dataset in PROMISE and KC1 dataset in NASA in Tables 7 and 8. Table 7 lists the results of the relationship of various metric elements for the dataset ant-1.7. The numbers in the first row and the first column indicate the order number of the measure element (see Table 3). The relation degree value between two metric elements ranges from −1 to 1. Table 8 lists the results of the relationship of various metric elements for the dataset KC1. The numbers in the first row and the first column indicate the order number of the measure element (see Table 2).

As we can see from Tables 7 and 8, the stronger the relation degree of metric elements, the larger the value is. These results provide an important data foundation for the selection of the sequence of metric pairs.

6.3.2. Experiment on the Sequence of Metric Pairs. Following the algorithm described in Section 4.5, the sequences of metric pairs generated from all 22 datasets of NASA and PROMISE repositories are shown in Table 9. The numbers in Table 9 are the order numbers of metric elements (see Tables 2 and 3). In Table 9, the first column represents the datasets from NASA and PROMISE, and the number in the second column (where the first 11 rows of metric elements are from Table 2, and the last 11 rows of metric elements are from Table 3) represents the sequences of metric pairs of the corresponding dataset. These sequences of metric pairs will be used as new metrics for SDP. We used three typical binary machine learning algorithms (SVM, KNN, decision tree) to verify our feature selection results. The

Table 7: The relationship of various metrics for the dataset ant-1.7.

	5	7	12	18	19	6	1	13	17	8
5	1	−0.25	0.72	0.66	−0.25	0.12	0.33	0.13	−0.62	0.18
7	−0.25	1	−0.38	0.78	0.72	−0.24	0.77	0.12	0.45	−0.43
12	0.72	−0.38	1	0.55	0.12	0.25	−0.24	0.28	0.72	0.12
18	0.66	0.78	0.55	1	0.13	−0.19	0.72	0.16	0.33	0.12
19	−0.25	0.72	0.12	0.13	1	0.12	0.35	0.18	0.22	−0.25
6	0.12	−0.24	0.25	−0.19	0.12	1	0.15	0.28	0.14	0.33
1	0.33	0.77	−0.24	0.72	0.35	0.15	1	0.11	−0.42	0.12
13	0.13	0.12	0.28	0.16	0.18	0.28	0.11	1	0.13	0.72
17	−0.62	0.45	0.72	0.33	0.22	0.14	−0.42	0.13	1	0.16
8	0.18	−0.43	0.12	0.12	−0.25	0.33	012	0.72	0.16	1

Table 8: The relationship of various metrics for the dataset KC1.

	1	4	10	13	22	25	11	17	40	3
1	1	0.73	0.28	0.6	0.73	0.69	0.62	0.77	0.63	0.73
4	0.73	1	0.33	0.52	1	0.96	0.82	0.92	0.67	0.87
10	0.28	0.33	1	0.29	0.33	0.31	0.32	0.34	0.26	0.3
13	0.6	0.52	0.29	1	0.52	0.52	0.41	0.61	0.46	0.47
22	0.73	1	0.33	0.52	1	0.97	0.82	0.92	0.67	0.87
25	0.69	0.96	0.31	0.52	0.97	1	0.78	0.92	0.68	0.82
11	0.62	0.82	0.32	0.41	0.82	0.78	1	0.72	0.51	0.7
17	0.77	0.92	0.34	0.61	0.92	0.92	0.72	1	0.81	0.86
40	0.63	0.67	0.26	0.46	0.67	0.68	0.51	0.81	1	0.68
3	0.73	0.87	0.3	0.47	0.87	0.82	0.7	0.86	0.68	1

experimental results show that (see Section 6.4) our feature selection method is superior to the baseline method (such as based on correlation or forward/backward selection) and advanced software defect methods (such as K-PCA-ELM).

As shown in Table 9, when the version of the software is different (such as the cross-version datasets xalan-2.4, xalan-2.5, xalan-2.6), the sequences of metric pairs generated by DQN are also different, and the experimental results show that the sequences of metric pairs from different versions [64] have better performance in SDP (see in Section 6.4).

6.4. RQ-3: Does the Binary Classification Algorithm-Based SDP Using DQN Have Better Performance Comparing with Those without Using DQN? In this section, we use three classification algorithms to verify the performance improvement of SDP using DQN against those without using it in various measures (precision, *F*-measure, AUC, and MCC). This paper adopts a 10-fold cross [65] validation method in order to evaluate the performance of the defect prediction model. Specifically, this method divides the data set into 10 parts, taking 8 of them as training data in turn and the remaining 2 as test data. The above process is repeated 10 times to ensure that each instance has been predicted once, and finally, the average of the results of these 10 runs is taken as the prediction performance of the model.

In Table 10, the first column lists all 22 datasets. The second column, fourth column, and sixth column list the precision values of three typical SDP models based on binary classification algorithms, SVM, KNN, and decision tree without using DQN-based feature extraction [41, 66, 67]. The third column, fifth column, and seventh column list precision values of corresponding three SDP models based on binary classification algorithms using DQN-based feature extraction (SVM-DQN, KNN-DQN, and decision tree-DQN). As we can see in Table 10, KNN-DQN approach improved by 5.6% compared with KNN method in the precision on average, and the decision tree-DQN classification approach improved by 11.1% compared with the decision tree method on average. As far as SVM and SVM-DQN are concerned, SVM-DQN outperforms SVM on 12 of total of 22 datasets and underperforms SVM on the other 10 datasets. The bold values in Table 10 indicate higher precision values between typical SDP and corresponding SDP with DQN.

To provide a visual comparison of the difference in prediction performance between the DQN and the three baseline defect predictors, we present the boxplots using the Scott-Knott ESD test according to the precision evaluation metrics. Figure 6 visualizes the results of six Scott-Knott ESD tests for defect predictors across 22 software projects. The horizontal bars in each box indicate the median indicator value for each defect predictor. The *x*-axis represents the

Table 9: The results of the sequence of metric pairs extraction.

Datasets	The sequences of metric pairs
CM1	(17, 28), (36, 17), (10, 19), (17, 9), (9, 33), (22, 7), (5, 32), (6, 25), (9, 6), (26, 9), (12, 36), (22, 19), (25, 33)
JM1	(36, 40), (37, 1), (13, 29), (11, 31), (9, 40), (19, 36), (40, 19)
KC1	(24, 31), (9, 19), (29, 31), (40, 1), (38, 1), (38, 24), (9, 38), (25, 7), (9, 18)
MC1	(11, 28), (14, 1), (9, 8), (19, 9), (31, 5), (40, 5), (7, 19), (19, 35), (9, 36), (26, 5), (17, 29), (36, 35), (40, 36), (7, 25), (33, 29)
MC2	(32, 17), (9, 18), (19, 1), (31, 7), (40, 8), (6, 20), (27, 33), (22, 35), (14, 8), (23, 8), (26, 8), (40, 37), (17, 37), (38, 25), (1, 5), (4, 18), (1, 18)
MW1	(32, 27), (30, 18), (1, 9), (2, 9), (19, 8), (29, 2), (40, 7), (6, 5), (12, 36), (20, 25), (32, 10), (20, 39), (14, 3), (4, 5), (7, 9), (19, 5)
PC1	(30, 22), (38, 17), (1, 28), (19, 25), (9, 3), (28, 9), (32, 7), (40, 6), (27, 3), (29, 18), (30, 25), (27, 20), (34, 40), (8, 12)
PC2	(33, 17), (9, 18), (19, 8), (29, 39), (31, 5), (40, 5), (13, 12), (7, 35), (30, 16), (20, 2), (25, 7), (37, 18), (13, 32), (27, 14)
PC3	(30, 22), (1, 23), (38, 16), (3, 9), (19, 8), (22, 30), (40, 18), (31, 33), (26, 17), (33, 21), (17, 11), (2, 16), (12, 8), (24, 11), (36, 5)
PC4	(13, 35), (33, 6), (39, 7), (11, 9), (19, 25), (11, 37), (35, 11), (20, 29), (36, 17), (30, 20), (2, 31), (38, 40), (14, 17), (13, 29), (21, 28)
PC5	(39, 27), (9, 16), (1, 2), (19, 3), (31, 5), (40, 37), (22, 7), (21, 16), (8, 16), (32, 19), (29, 18), (28, 33), (20, 11), (30, 29), (35, 37)
ant-1.7	(19, 5), (11, 7), (19, 8), (11, 3), (14, 5), (7, 1), (12, 5)
tomcat	(7, 4), (12, 7), (17, 7), (7, 15), (12, 4), (12, 3), (15, 12), (12, 1)
xalan-2.5	(13, 6), (13, 11), (1, 8), (13, 1), (12, 11), (1, 7), (11, 7)
xalan-2.4	(11, 7), (13, 11), (13, 8), (11, 9), (9, 5), (11, 5), (5, 12), (7, 5), (9, 6)
xalan-2.6	(15, 13), (15, 9), (9, 19), (13, 7), (15, 1), (9, 1)
camel	(2, 7), (5, 7), (7, 13), (2, 13), (13, 6), (13, 1), (5, 8)
synampe	(12, 3), (15, 6), (12, 11), (14, 3), (12, 6), (2, 12), (12, 15), (7, 1), (7, 6), (12, 3)
velocity-1.6	(20, 11), (20, 18), (11, 8), (11, 4), (4, 20), (18, 7), (4, 18), (6, 5), (4, 7)
xerces-1.4	(5, 17), (6, 9), (9, 5), (19, 9), (17, 5), (12, 5), (8, 12), (6, 5), (7, 6)
prop-6	(13, 8), (18, 1), (11, 7), (7, 9), (13, 5), (11, 5), (7, 8), (12, 6), (11, 9)
poi-3.0	(5, 7), (2, 1), (15, 8), (15, 2), (7, 1), (4, 5), (7, 16), (7, 13)

Table 10: Comparison of precision values.

Datasets	SVM	SVM-DQN	KNN	KNN-DQN	Decision tree	Decision tree-DQN
CM1	0.8	**0.87**	0.82	**0.86**	**0.86**	0.85
JM1	0.71	**0.73**	0.7	**0.84**	0.42	**0.87**
KC1	0.74	**0.83**	0.72	**0.79**	0.47	**0.85**
MC1	**0.96**	0.82	**0.95**	0.84	**0.93**	0.79
MC2	0.76	**0.79**	0.69	**0.79**	0.73	**0.88**
MW1	0.72	**0.84**	**0.78**	**0.78**	0.73	**0.79**
PC1	**0.86**	0.78	**0.89**	0.86	0.76	**0.86**
PC2	0.82	**0.85**	0.84	**0.87**	0.79	**0.88**
PC3	**0.82**	0.79	0.69	**0.86**	0.73	**0.87**
PC4	0.76	**0.81**	0.71	**0.82**	0.74	**0.85**
PC5	0.81	**0.82**	0.82	**0.84**	0.84	**0.87**
ant-1.7	0.69	**0.73**	0.52	**0.71**	0.72	**0.81**
tomcat	**0.79**	0.74	0.74	**0.76**	0.75	**0.79**
xalan-2.5	**0.99**	0.76	**0.83**	0.74	**0.87**	0.78
xalan-2.4	**0.78**	0.71	**0.75**	0.72	**0.75**	**0.75**
xalan-2.6	**0.85**	0.77	0.75	**0.78**	**0.79**	0.78
camel	0.72	**0.75**	0.68	**0.79**	—	**0.76**
synampe	0.72	**0.73**	—	**0.73**	0.74	**0.78**
velocity-1.6	0.71	**0.75**	—	**0.78**	0.68	**0.72**
xerces-1.4	**0.85**	0.79	0.72	**0.75**	0.69	**0.75**
prop-6	**0.78**	0.71	0.66	**0.74**	0.59	**0.77**
poi-3.0	**0.81**	0.76	0.69	**0.72**	0.72	**0.76**

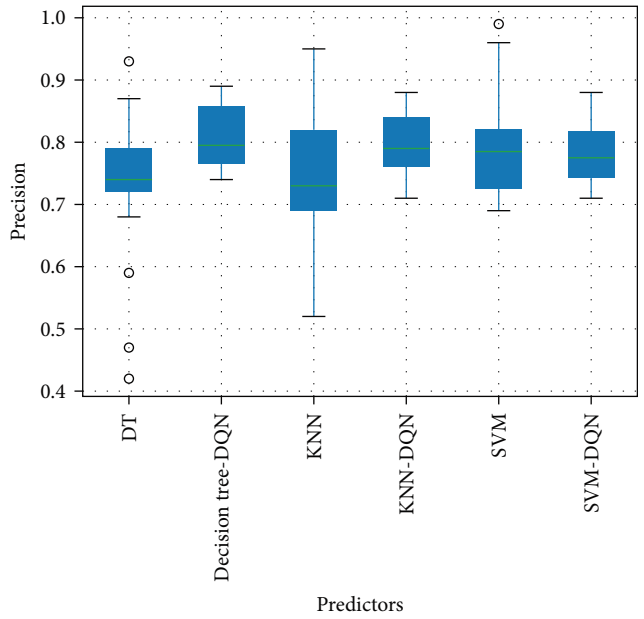

Figure 6: The Scott-Knott ESD ranking for DQN compared with three classic defect predictors in terms of precision evaluation indicator.

six defect predictors, and the y-axis represents the precision evaluation index. From Figure 6, we can observe that the DQN-based predictor ranked the highest, indicating that the DQN-based predictor had better prediction performance than the three classical defect methods.

In Table 11, the first column lists all the 22 datasets. The second column, fourth column, and sixth column list the F-measure values of three typical SDP models based on binary classification algorithms, SVM, KNN, and decision tree, without using DQN-based feature extraction. The third column, fifth column, and seventh column list F-measure values of three corresponding SDP models based on binary classification algorithms using DQN-based feature extraction (SVM-DQN, KNN-DQN, and decision tree-DQN). As we can see in Table 11, SVM-DQN-based SDP improved by 8.2%, KNN-DQN-based SDP improved by 12.9%, and decision tree-DQN-based SDP improved by 12.8% in the F-measure values on average compared with three baseline methods. The bold values in Table 11 indicate higher F-measure values between typical SDP and corresponding SDP with DQN.

We use boxplots and the Scott-Knott ESD test to compare the prediction performance of the DQN-based defect predictor with three classical baseline predictors in terms of F_1 evaluation metrics. Figure 7 displays the results of six Scott-Knott ESD tests across 22 software projects. The median indicator value of each defect predictor is shown as a horizontal bar in each box, while the six defect predictors are represented on the x-axis, and the F_1 evaluation index is represented on the y-axis. As depicted in Figure 7, the DQN-based predictor achieved the highest rank, indicating that it outperformed the three classical defect methods in terms of prediction accuracy.

In Table 12, the first column lists all the 22 datasets. The second column, fourth column, and sixth column list the AUC values of three typical SDP models based on binary classification algorithms, SVM, KNN, and decision tree, without using DQN-based feature extraction. The third column, fifth column, and seventh column list AUC values of corresponding three SDP models based on binary classification algorithms using DQN-based feature extraction (SVM-DQN, KNN-DQN, and decision tree-DQN). As we can see in Table 12, SVM-DQN-based SDP improved by 17.2%, KNN-DQN-based SDP improved by 20.1%, and decision tree-DQN-based SDP improved by 26.1% in the AUC values on average compared with three corresponding baseline methods. The bold values in Table 12 indicate higher AUC values between typical SDP and corresponding SDP with DQN.

From the results shown in Figure 8, it is evident that the DQN-based predictor achieved the highest rank, indicating superior prediction accuracy compared to the three classical baseline methods.

In Table 13, the first column lists all the 22 datasets. The second column, fourth column, and sixth column list the MCC values of three typical SDP models based on binary classification algorithms, SVM, KNN, and decision tree, without using DQN-based feature extraction. The third column, fifth column, and seventh column list MCC values of corresponding three SDP models based on binary classification algorithms using DQN-based feature extraction (SVM-DQN, KNN-DQN, and decision tree-DQN). As shown in Table 13, SVM-DQN-based SDP improved by 84.0%, KNN-DQN-based SDP improved by 69.6%, and decision tree-DQN-based SDP improved by 51.5% in the MCC values on average compared with three corresponding baseline methods. The bold values in Table 13 indicate higher MCC values between typical SDP and corresponding SDP with DQN.

From the results shown in Figure 9, it is evident that the DQN-based predictor achieved the highest rank, indicating superior prediction accuracy compared to the three classical baseline methods.

6.5. RQ4: Does DQN-FE-SDP Approach Have Better Performance Comparing with Other State-of-the-Art SDP Approaches? In order to further verify the effectiveness of our approach, we compare decision tree-DQN with the state-of-the-art SDP approaches of K-PCA-ELM [41], CFIW-TNB [60], MDA-O [61], and WEM [62] on four performance measures (precision, F-measure, AUC, or MCC) in the common datasets, and the results show that our SDP approach outperforms the state-of-the-art SDP approaches.

The main objective of K-PCA-ELM is to address two significant challenges for SDP models: class imbalance and overfitting. They investigated various kernel functions of ELM along with K-PCA and found better results compared with other classical SDP models. We compare our proposed SPD using DQN-based feature extraction (decision tree-DQN-based SPD) with it in four performance measures (precision, F-measure, AUC, and MCC) on the common datasets of CM1, JM1, KC1, MC1, MC2, PC1, ant-1.7, tomcat, xalan-2.5, camel, synampe, velocity-1.6, xerces-1.4, prop-6, and poi-3.0.

Table 14 shows the precision and F-measure values of K-PCA-ELM and decision tree-DQN SDP on common

Table 11: Comparison of *F*-measure values.

Datasets	SVM	SVM-DQN	KNN	KNN-DQN	Decision tree	Decision tree-DQN
CM1	0.8	**0.88**	0.82	**0.86**	**0.86**	0.86
JM1	0.71	**0.74**	0.7	**0.84**	0.42	**0.87**
KC1	0.74	**0.82**	0.72	**0.79**	0.47	**0.84**
MC1	**0.96**	0.82	0.95	0.84	**0.93**	0.80
MC2	0.76	**0.78**	0.69	**0.79**	0.73	**0.87**
MW1	0.72	**0.84**	0.78	**0.78**	0.73	**0.79**
PC1	**0.86**	0.78	0.89	0.86	0.76	**0.85**
PC2	0.82	**0.85**	0.84	**0.88**	0.79	**0.88**
PC3	**0.82**	0.79	0.69	**0.86**	0.73	**0.87**
PC4	0.76	**0.81**	0.71	**0.82**	0.74	**0.83**
PC5	0.81	**0.82**	0.82	**0.85**	0.84	**0.89**
ant-1.7	0.69	**0.73**	0.52	**0.71**	0.72	**0.81**
tomcat	**0.79**	0.74	0.74	**0.76**	0.75	**0.79**
xalan-2.5	**0.99**	0.76	**0.83**	0.74	**0.87**	0.79
xalan-2.4	**0.78**	0.71	**0.75**	0.72	**0.75**	0.74
xalan-2.6	**0.85**	0.77	0.75	**0.79**	**0.79**	0.79
camel	0.72	**0.75**	0.68	**0.79**	—	**0.76**
synampe	0.72	**0.73**	—	**0.77**	0.74	**0.78**
velocity-1.6	0.71	**0.75**	—	**0.78**	0.68	**0.75**
xerces-1.4	**0.85**	0.81	0.72	**0.75**	0.69	**0.74**
prop-6	**0.78**	0.71	0.66	**0.76**	0.59	**0.76**
poi-3.0	**0.81**	0.76	0.69	**0.72**	**0.72**	**0.74**

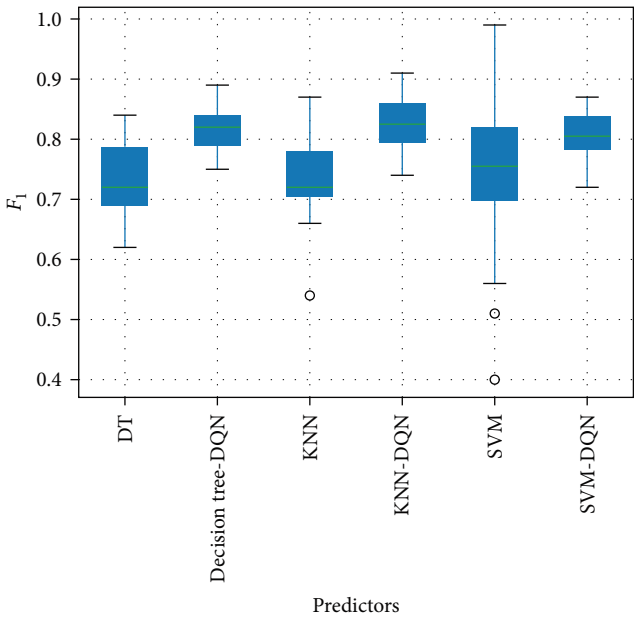

Figure 7: The Scott-Knott ESD ranking for DQN compared with three classic defect predictors in terms of F_1 evaluation indicator.

datasets. The bold values in Table 14 indicate the higher precision and *F*-measure value between the two SDP approaches across all 15 common datasets. In both precision and *F*-measure values, our approach achieves higher results on 11 of 15 datasets. From Table 14, we can see the decision tree-DQN-based SDP improves the precision and *F*-measure values by 3.5% and 3.7% comparing with K-PCA-ELM on average.

Table 15 shows the AUC and MCC values of K-PCA-ELM and decision tree-DQN-based SDP approach on common datasets. The bold values in Table 15 indicate the higher AUC or MCC values between the two SDP approaches across all 15 common datasets. It is shown in Table 15 that our approach outperforms K-PCA-ELM in MCC values on 11 of all 15 datasets and in AUC on 13 of all 15 datasets. From Table 15, our approach improves the AUC and MCC values by 4.1% and 5.7% comparing with K-PCA-ELM on average.

Figure 10 shows the results of precision, *F*-measure, AUC, and MCC values of two SDP approaches across 15 common datasets. On the *x*-axis, we represent NASA and PROMISE datasets, whereas, on the *y*-axis, we present precision, *F*-measure, AUC, or MCC values of the corresponding datasets.

A state-of-the-art SDP approach of CFIW-TNB proposed a cutting-edge SDP approach, CFIW-TNB, has introduced a dual weighting mechanism to enhance the learning process, taking into account both feature transfer and instance transfer. In their approach, they determine instance weight based on the local data interaction between source and target domains. Additionally, they assign higher feature weight to features that exhibit a strong correlation with the learning task, are uncorrelated with other features, and minimize the domain differences.

The results demonstrate that CFIW-TNB, equipped with this dual-weighting mechanism, outperforms scenarios with

Table 12: Comparison of AUC values.

Datasets	SVM	SVM-DQN	KNN	KNN-DQN	Decision tree	Decision tree-DQN
CM1	0.51	**0.74**	0.51	**0.82**	0.58	**0.79**
JM1	0.55	**0.73**	0.49	**0.79**	0.48	**0.75**
KC1	0.54	**0.74**	0.69	**0.81**	0.40	**0.82**
MC1	0.53	**0.78**	0.66	**0.82**	0.54	**0.85**
MC2	0.52	**0.71**	0.69	**0.78**	0.72	**0.82**
MW1	0.62	**0.78**	0.77	**0.81**	0.58	**0.8**
PC1	0.57	**0.78**	0.66	**0.81**	0.60	**0.83**
PC2	0.71	**0.79**	0.66	**0.85**	0.54	**0.87**
PC3	0.78	**0.81**	0.58	**0.82**	0.72	**0.84**
PC4	**0.82**	0.78	0.64	**0.81**	0.72	**0.8**
PC5	0.84	**0.86**	0.82	**0.87**	0.76	**0.82**
ant-1.7	0.51	**0.72**	0.82	**0.87**	0.81	**0.86**
tomcat	0.5	**0.77**	0.83	**0.85**	0.72	**0.85**
xalan-2.5	**0.8**	0.78	—	**0.86**	—	**0.86**
xalan-2.4	0.74	**0.79**	—	**0.84**	—	**0.85**
xalan-2.6	**0.81**	**0.81**	—	**0.84**	—	**0.85**
camel	0.83	**0.85**	0.81	**0.86**	0.79	**0.81**
synampe	0.64	**0.75**	—	**0.84**	0.81	**0.83**
velocity-1.6	0.52	**0.78**	0.72	**0.79**	**0.78**	0.76
xerces-1.4	**0.81**	0.76	0.66	**0.76**	0.81	**0.83**
prop-6	0.56	**0.72**	—	**0.81**	0.54	**0.82**
poi-3.0	**0.78**	**0.78**	0.72	**0.82**	0.69	**0.85**

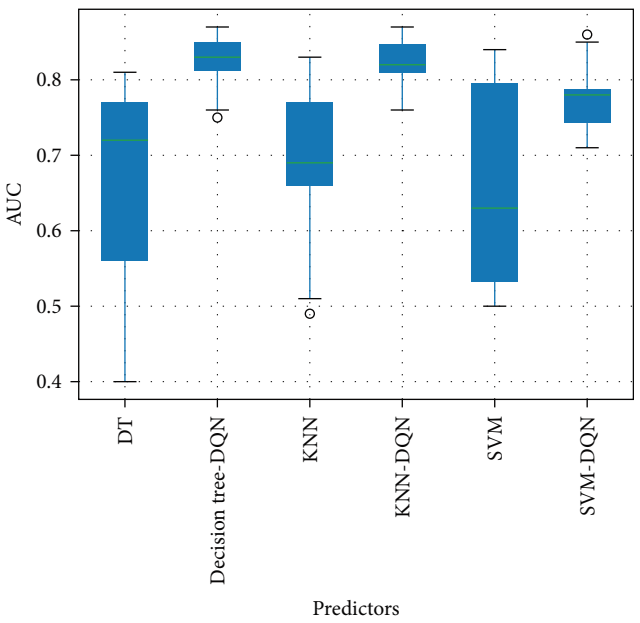

Figure 8: The Scott-Knott ESD ranking for DQN compared with three classic defect predictors in terms of the AUC evaluation indicator.

a single weight (either instance or feature weight) or no weighting at all.

We also compared our approach with CFIW-TNB in AUC and *F*-measure values on the common datasets of ant-1.7, xalan-2.4, xalan-2.5, xalan-2.6, poi-3.0, and xerces-1.4. Table 16 shows the AUC and *F*-measure values of our approach and CFIW-TNB on common datasets. The bold values in Table 16 indicate the higher AUC and *F*-measure values between the two SDP approaches across all six datasets. In both AUC and F-measure values, our approach outperforms CFIW-TNB on 5 of 6 datasets. From Table 16, we can observe the decision tree-DQN-based SDP improves the AUC and *F*-measure values by 11.4% and 31.8% comparing with CFIW-TNB on average.

Figure 11 shows the results of AUC and *F*-measure values of two SDP approaches across six common datasets. On the *x*-axis, we represent PROMISE datasets, whereas we present AUC or *F*-measure values of corresponding datasets in the *y*-axis.

The advanced SDP method, MDA-O, introduced a novel approach known as manifold-embedded distribution adaptation (MDA). This approach aims to reduce the distribution gap within the feature subspace of the manifold. To address the challenge of differing data distributions across various datasets, MDA-O involves the mapping of source and target project data into a manifold subspace. Subsequently, it conducts joint distribution adaptation for both conditional and marginal distributions within this manifold subspace, as detailed in [61].

We compare decision tree-DQN with MDA-O in AUC and *F*-measure values on the common datasets CM1, MW1, PC1, PC3, PC4, ant-1.7, poi-3.0, velocity-1.6, and xalan-2.6. Table 17 shows the AUC and *F*-measure values of MDA-O and decision tree + DQN SDP approach on common datasets, and the bold values in Table 17 indicate the higher AUC and F-measure values between the two SDP approaches across

Table 13: Comparison of MCC values.

Datasets	SVM	SVM-DQN	KNN	KNN-DQN	Decision tree	Decision tree-DQN
CM1	0.09	**0.54**	0.22	**0.68**	0.29	**0.69**
JM1	0.21	**0.56**	0.43	**0.68**	0.32	**0.55**
KC1	0.18	**0.61**	0.54	**0.61**	0.42	**0.62**
MC1	0.23	**0.68**	**0.47**	0.32	0.29	**0.64**
MC2	0.16	**0.56**	0.22	**0.71**	0.45	**0.63**
MW1	0.12	**0.50**	0.52	**0.67**	0.33	**0.61**
PC1	0.29	**0.55**	0.38	**0.58**	0.64	**0.71**
PC2	0.21	**0.66**	0.42	**0.55**	0.27	**0.49**
PC3	0.24	**0.62**	0.38	**0.71**	0.22	**0.53**
PC4	0.21	**0.69**	0.35	**0.44**	0.29	**0.49**
PC5	0.29	**0.53**	0.26	**0.51**	0.25	**0.60**
ant-1.7	0.09	**0.61**	0.28	**0.43**	0.37	**0.53**
tomcat	0.25	**0.59**	0.54	**0.63**	0.42	**0.69**
xalan-2.5	0.65	**0.71**	—	**0.69**	—	**0.56**
xalan-2.4	**0.77**	0.66	0.44	**0.53**	**0.72**	0.68
xalan-2.6	0.65	**0.72**	—	**0.62**	—	**0.68**
camel	0.29	**0.64**	0.13	**0.55**	0.37	**0.61**
synampe	0.64	**0.75**	—	**0.44**	0.48	**0.63**
velocity-1.6	0.15	**0.62**	0.36	**0.66**	0.4	**0.51**
xerces-1.4	**0.68**	0.59	0.25	**0.69**	—	**0.57**
prop-6	0.21	**0.42**	0.35	**0.55**	—	**0.61**
poi-3.0	0.57	**0.63**	0.29	**0.58**	—	**0.65**

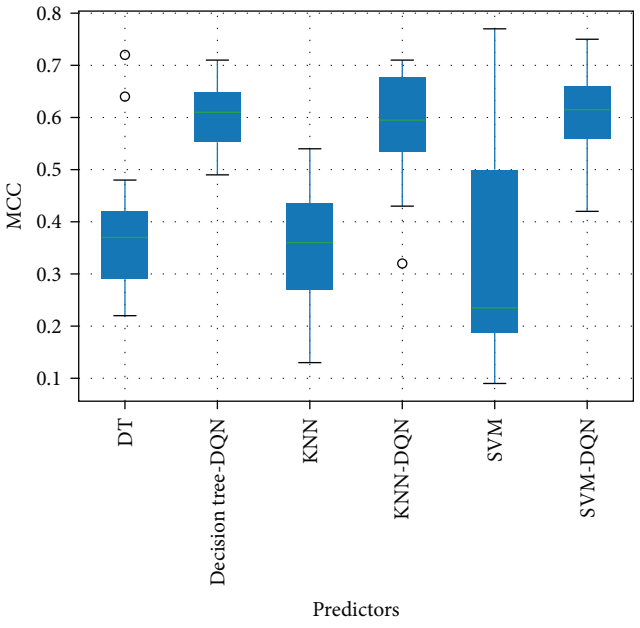

Figure 9: The Scott-Knott ESD ranking for DQN compared with three classic defect predictors in terms of MCC evaluation indicator.

all nine datasets. No matter in AUC values or *F*-measure values, our approach completely outperforms MDA-O on all nine datasets. From Table 17, our approach improves AUC and *F*-measure values by 9.3% and 95.5% comparing with MDA-O on average.

Figure 12 shows the results of AUC and *F*-measure values of two SDP approaches on nine common datasets in bar and line graphs. On the *x*-axis, we represent NASA and PROMISE datasets, whereas, on the *y*-axis, we present AUC or *F*-measure values of corresponding datasets.

WEM is an advanced approach to address the challenge of single-rule limitations in problem-solving. Developed as a weighted ensemble model under the principles of ensemble learning, WEM aims to enhance the diversity of rules for improved predictive accuracy. In contrast to the constraints posed by individual rules, WEM effectively boosts predictions by integrating multiple rules, including FWCAR, BR, and DT. A detailed description of this method can be found in [62].

We compare decision tree-DQN with WEM in MCC and *F*-measure values on the common datasets ant-1.7, velocity-1.6, xalan-2.4, xalan-2.5, and xalan-2.6. Table 18 shows the MCC and *F*-measure values of WEM and decision tree + DQN SDP approach on common datasets, and the bold values in Table 18 indicate the higher MCC and *F*-measure values between the two SDP approaches across all five datasets. No matter in MCC values or *F*-measure values, our approach completely outperforms WEM on all five datasets. From Table 18, our approach improves MCC and *F*-measure values by 92.6% and 24.8% comparing with WEM on average.

Figure 13 shows the results of MCC and *F*-measure values of two SDP approaches on nine common datasets in bar and line graphs. On the *x*-axis, we represent NASA and PROMISE datasets, whereas, on the *y*-axis, we present MCC or *F*-measure values of corresponding datasets.

Table 14: Comparison of precision and *F*-measure with K-PCA-ELM.

Datasets	Precision		*F*-measure	
	K-PCA-ELM	Decision tree-DQN	K-PCA-ELM	Decision tree-DQN
CM1	0.81	**0.85**	0.81	**0.86**
JM1	0.76	**0.87**	0.8	**0.87**
KC1	0.78	**0.85**	0.76	**0.84**
MC1	0.73	**0.79**	0.75	**0.80**
MC2	0.76	**0.88**	0.8	**0.87**
PC1	0.78	**0.86**	0.79	**0.85**
ant-1.7	0.75	**0.81**	0.76	**0.81**
tomcat	0.77	**0.79**	**0.83**	0.79
xalan-2.5	**0.78**	**0.78**	**0.82**	0.79
camel	**0.8**	0.76	**0.79**	0.76
synampe	0.74	**0.78**	0.78	**0.78**
velocity-1.6	**0.79**	0.72	**0.81**	0.75
xerces-1.4	**0.8**	0.75	**0.82**	0.78
prop-6	0.75	**0.77**	**0.78**	0.76
poi-3.0	**0.81**	0.76	**0.79**	0.74

Table 15: Comparison of AUC and MCC with K-PCA-ELM.

Datasets	AUC		MCC	
	K-PCA-ELM	Decision tree-DQN	K-PCA-ELM	Decision tree-DQN
CM1	**0.8**	0.79	0.62	**0.69**
JM1	0.73	**0.75**	**0.59**	0.55
KC1	0.76	**0.82**	0.52	**0.62**
MC1	0.75	**0.84**	0.50	**0.64**
MC2	**0.81**	**0.81**	0.60	**0.63**
PC1	0.80	**0.82**	0.59	**0.72**
ant-1.7	0.76	**0.85**	0.51	**0.53**
tomcat	0.84	**0.85**	**0.68**	**0.68**
xalan-2.5	0.81	**0.84**	**0.63**	0.56
camel	0.78	**0.81**	0.55	**0.59**
synampe	0.75	**0.82**	0.51	**0.61**
velocity-1.6	**0.79**	0.76	**0.57**	0.48
xerces-1.4	**0.82**	**0.82**	**0.65**	0.55
prop-6	0.78	**0.82**	0.56	**0.64**
poi-3.0	0.79	**0.84**	0.59	**0.68**

6.6. RQ-5: Does the Feature Extraction Algorithm Based on DQN Outperform Four State-of-the-Art Feature Selection Approaches in SDP? The aim of this research is to validate the performance of a feature extraction algorithm based on DQN in the tasks of feature selection and feature extraction, as well as its effect on improving the performance of ECE. For this purpose, we compared it with four other popular feature selection or feature extraction algorithms, including CS test, PCA, and IG, where PCA is a feature extraction algorithm. Before applying these feature selection algorithms, we also employed corresponding data preprocessing techniques to better reflect the performance of the proposed feature extraction algorithm in this study. In the binary classification machine learning task, we used the decision tree classifier, as introduced in this paper. Through the comparison with these four commonly used feature selection or feature extraction algorithms, our objective is to further validate the superiority of the DQN-based feature extraction algorithm proposed in this research in terms of performance and investigate its effect on improving the performance of ECE.

Table 19 depicts the precision value of all four feature selection or extraction approaches across a total of 22 software projects from three datasets, including 12 projects from PROMISE and 12 projects from NASA. In Table 19, we recorded the precision performance indicators of five feature selection or feature extraction techniques on each dataset. Please note that the maximum value of each row is marked

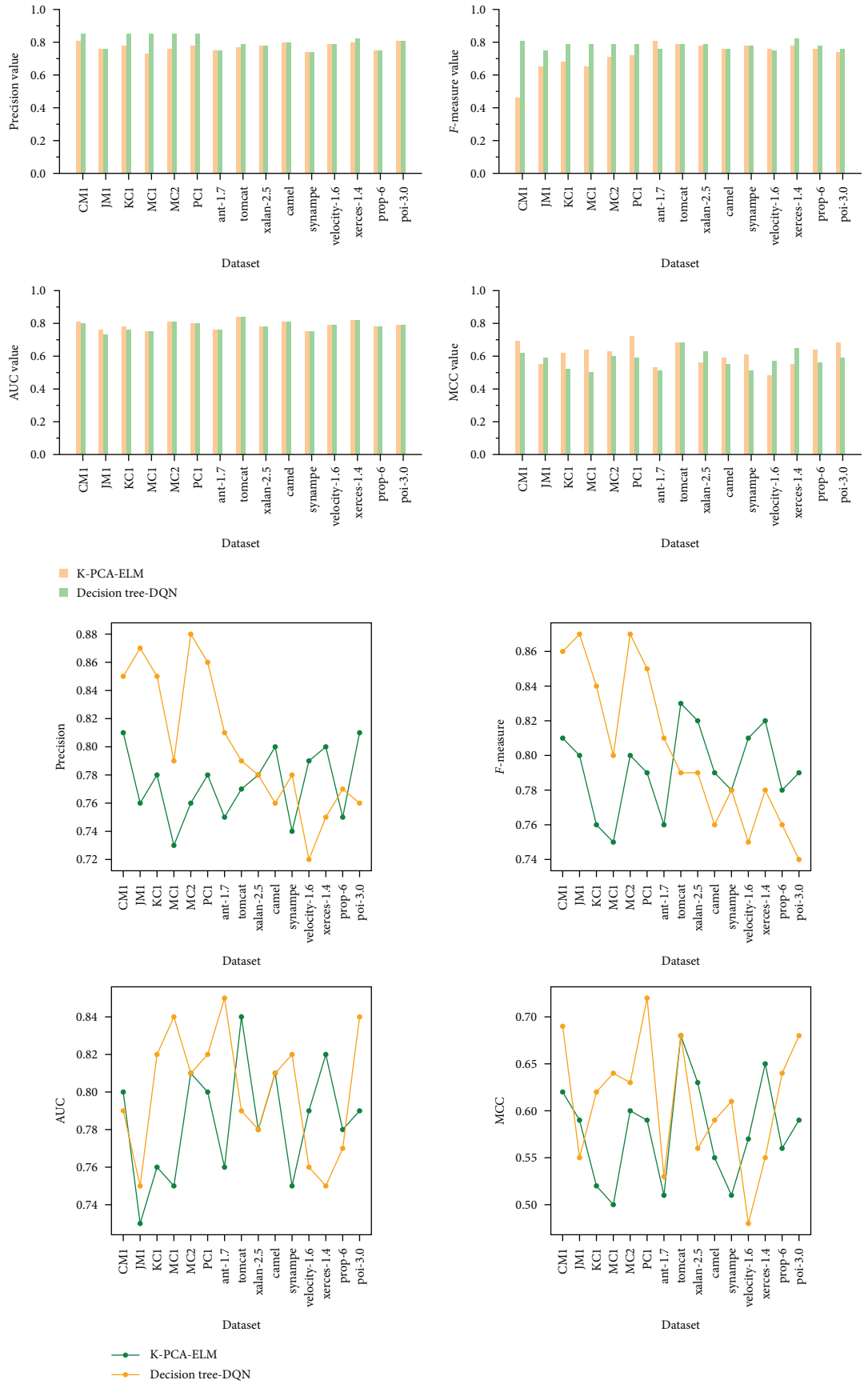

FIGURE 10: Bar and line graphs of precision, *F*-measure, AUC, and MCC values comparison.

Table 16: Comparison of AUC and *F*-measure with CFIW-TNB.

Datasets	AUC		*F*-measure	
	Decision tree-DQN	CFIW-TNB	Decision tree-DQN	CFIW-TNB
ant-1.7	**0.85**	0.78	**0.81**	0.758
xalan-2.4	**0.85**	0.67	**0.74**	0.32
xalan-2.5	**0.84**	0.68	**0.79**	0.57
xalan-2.6	**0.85**	0.69	**0.79**	0.63
poi-3.0	**0.82**	0.8	0.74	**0.78**
xerces-1.4	0.82	**0.85**	**0.74**	0.704

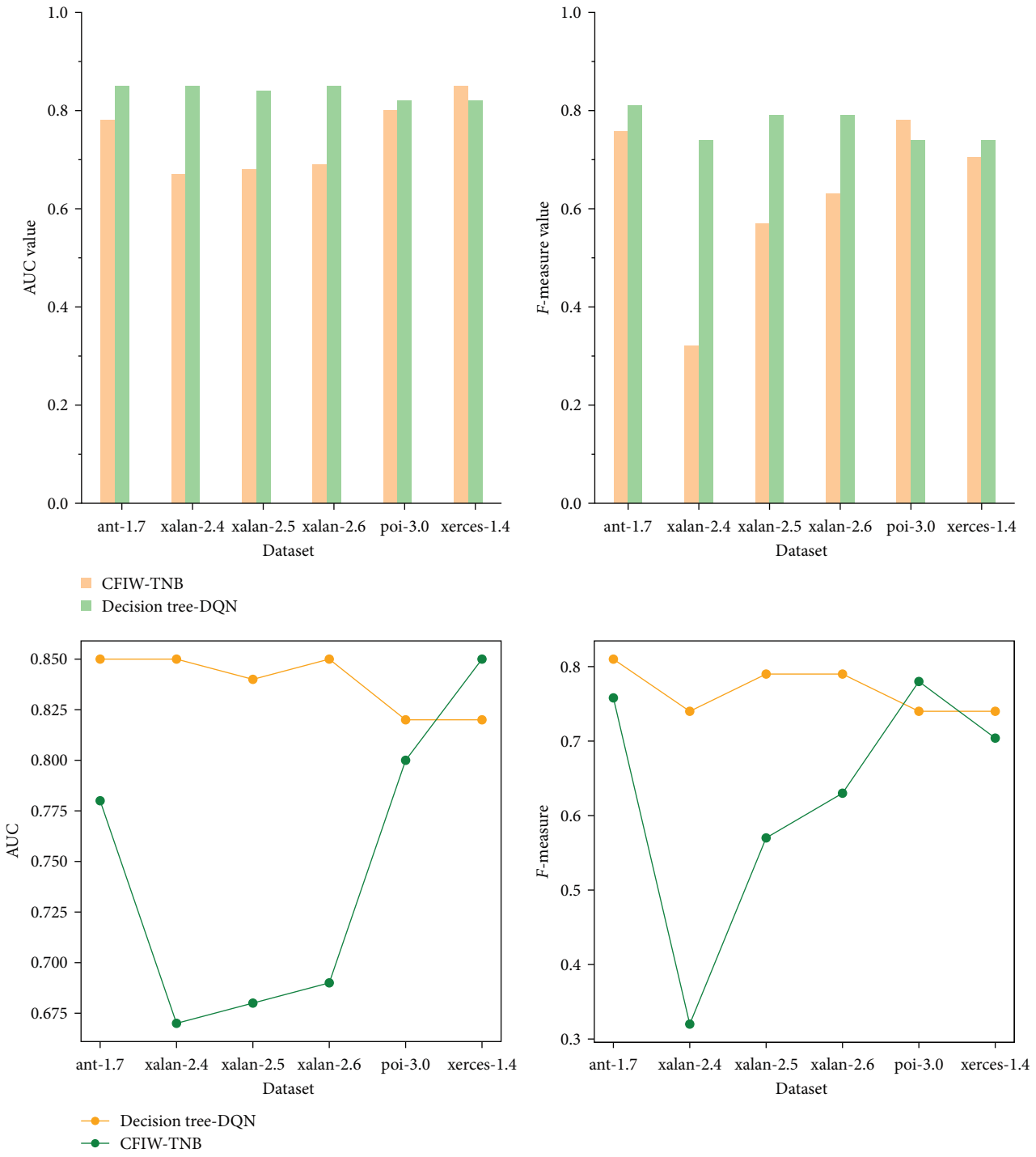

Figure 11: Comparison of AUC and *F*-measure values for CFIW-TNB and decision tree-DQN.

Table 17: Comparison of AUC and *F*-measure with MDA-O.

Datasets	AUC		*F*-measure	
	Decision tree-DQN	MDA-O	Decision tree-DQN	MDA-O
CM1	0.79	**0.78**	**0.86**	0.34
MW1	0.84	**0.77**	**0.79**	0.23
PC1	**0.82**	0.74	**0.85**	0.22
PC3	**0.84**	0.78	**0.87**	0.33
PC4	**0.80**	0.77	**0.83**	0.35
ant-1.7	**0.85**	0.76	**0.81**	0.46
poi-3.0	**0.84**	0.72	**0.74**	**0.74**
velocity-1.6	**0.76**	0.69	**0.75**	0.55
xalan-2.6	**0.85**	0.75	**0.79**	0.66

in bold. From Table 19, it can be observed that the proposed feature extraction algorithm based on DQN outperforms the other four feature selection or feature extraction techniques in 14 out of 22 datasets. Furthermore, we calculated that the DQN-based feature extraction technique has higher average values in terms of precision compared to PCA, ESE, IG, and CS. Specifically, the DQN-based feature extraction technique improved the average values by 5.75%, 12.8%, 11.6%, and 8.45%, respectively, compared to PCA, ESE, IG, and CS across all datasets.

To visually compare the predictive performance differences between DQN and the four baseline feature selection or feature extraction methods, we presented boxplots of Scott-Knott ESD tests for evaluation metrics. Figure 14 displays the Scott-Knott ESD test results for all five feature selection or feature extraction methods across a total of 22 software projects. The horizontal lines within each box represent the median values for each feature selection or feature extraction method. The *x*-axis represents the five feature selection or feature extraction methods, while the *y*-axis represents the evaluation metrics. From Figure 14, it can be observed that the DQN algorithm is positioned at the highest level, indicating that DQN-FE-SDP achieves the best predictive performance in terms of precision evaluation metric compared to the four baseline feature selection or feature extraction methods. We also observed that the medians obtained by DQN-FE-SDP for the precision evaluation metric are higher than those obtained by the four baseline feature selection or feature extraction methods, providing strong evidence for the superiority of DQN-FE-SDP.

In Table 20, the first column lists all the 22 datasets, the other columns list five F_1 values for feature selection or feature extraction. As shown in Table 20, in 22 datasets, the DQN-based feature extraction method proposed in this paper showed good performance in 15 datasets. In addition, this paper proposes that the feature extraction method has a higher average value on F_1 measures in 22 datasets. Specifically, DQN-FE-SDP is improved by 10.5%, 6.04%, 6.58%, and 4.82% over PAC, ECE, IG, and CS in F_1 measures, respectively.

As can be seen from Figure 15, the DQN-FE-SDP algorithm performs the best in the precision evaluation metrics, at the highest level, with better performance compared to the other four baseline feature selection or extraction methods. Moreover, for the F_1 evaluation index, the median DQN-FE-SDP algorithm was also significantly higher than the median of the other four baseline feature selection or extraction methods. This further validates the superiority of the DQN-FE-SDP algorithm in software project prediction.

Table 21 presents 22 datasets in the first column, with the other columns listing the AUC values for five feature selection or extraction methods. The results indicate that the DQN-based feature extraction method proposed in this paper demonstrated strong performance in 21 out of 22 datasets. Moreover, the paper demonstrated that the proposed feature extraction method had a higher average AUC value across all 22 datasets. Specifically, the DQN-FE-SDP method showed improvements of 13.3%, 11.03%, 11.3%, and 8.55% over PAC, ECE, IG, and CS, respectively, in terms of AUC measure.

Figure 16 presents a boxplot where the *x*-axis represents five feature selection or extraction methods, and the *y*-axis represents the AUC evaluation metric. The results indicate that the DQN-FE-SDP algorithm outperformed the other four baseline feature selection or extraction methods in terms of precision evaluation metrics, with the highest level of performance. Additionally, for the F_1 evaluation metric, the median value of the DQN-FE-SDP algorithm was significantly higher than the median values of the other four baseline feature selection or extraction methods. These findings further affirm the superiority of the DQN-FE-SDP algorithm in software project prediction.

Table 22 consists of 22 datasets, with the first column listing them and the other columns presenting MCC values for five different feature selection or extraction methods. According to Table 22, the DQN-based feature extraction method proposed in this paper demonstrated strong performance in 19 out of the 22 datasets. Furthermore, the paper suggests that the proposed feature extraction method has a higher average MCC value across all 22 datasets. More specifically, compared to PAC, ECE, IG, and CS, the DQN-FE-SDP algorithm improved MCC measures by 14.96%, 15.5%, 10.96%, and 11.17%, respectively.

Figure 17 displays a boxplot where the *x*-axis represents five different feature selection or extraction methods, and the *y*-axis represents the MCC evaluation metric. The results

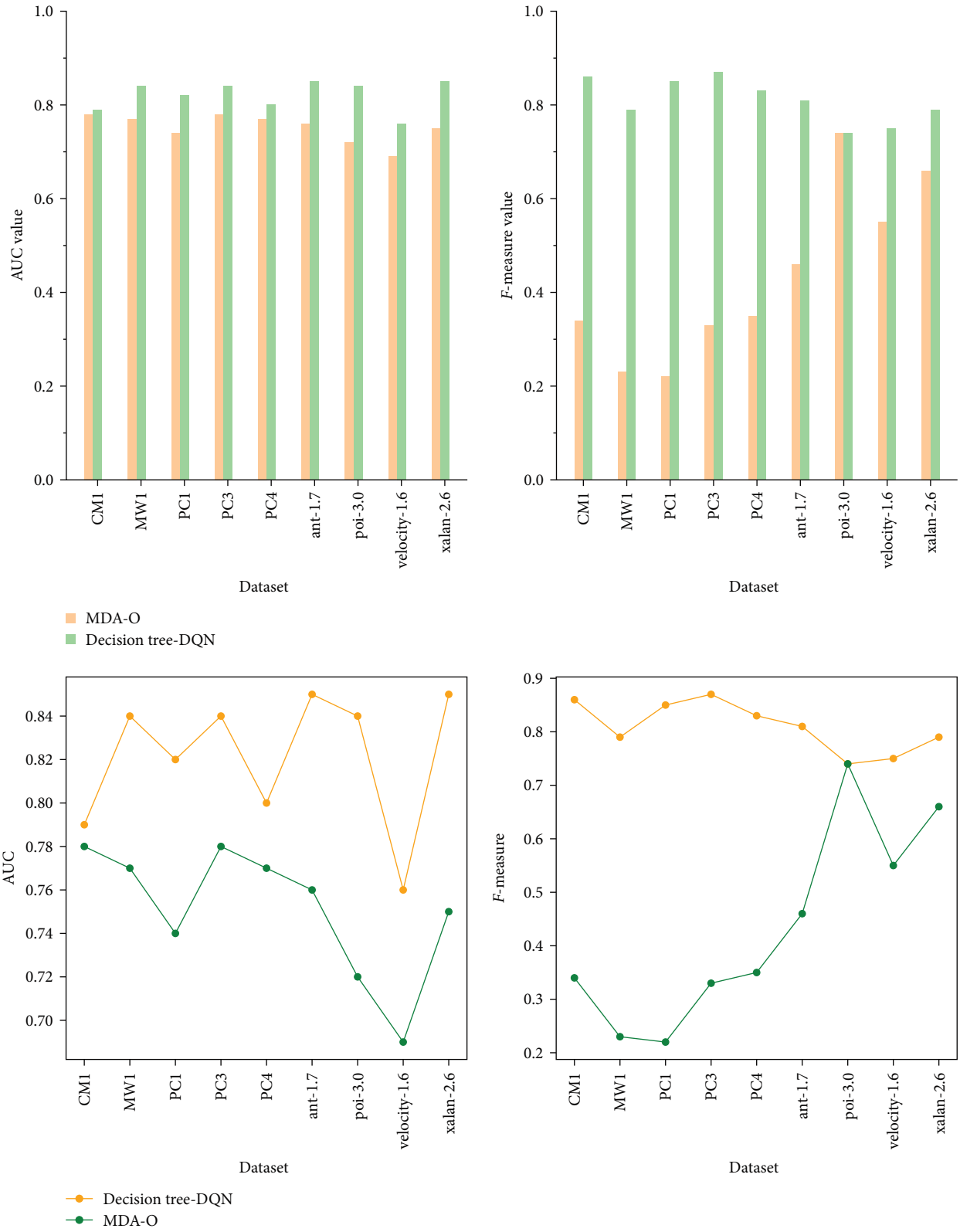

Figure 12: Comparison of AUC and *F*-measure values for MDA-O and decision tree-DQN.

Table 18: Comparison of MCC and *F*-measure with WEM.

Datasets	MCC		*F*-measure	
	Decision tree-DQN	WEM	Decision tree-DQN	WEM
ant-1.7	**0.85**	0.51	**0.81**	0.46
velocity-1.6	**0.76**	0.45	**0.75**	0.65
xalan-2.4	**0.85**	0.38	**0.79**	0.68
xalan-2.5	**0.85**	0.28	**0.79**	0.65
xalan-2.6	**0.85**	0.54	**0.79**	0.71

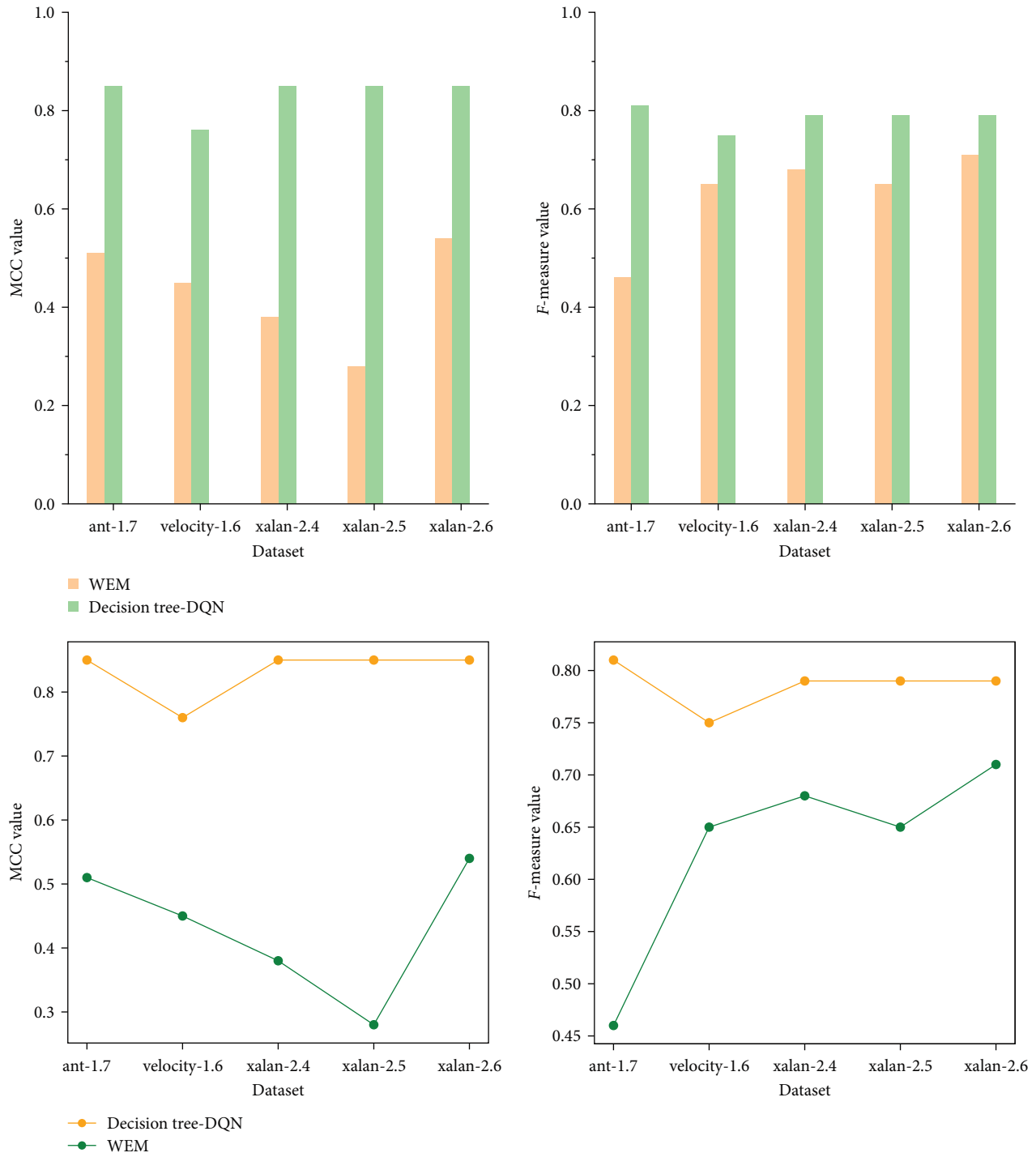

Figure 13: Comparison of MCC and *F*-measure values for WEM and decision tree-DQN.

Table 19: Comparison of precision with four state-of-the-art feature selection or feature extraction algorithms.

Datasets	Decision tree-DQN	PCA	ECE	IG	CS
CM1	**0.85**	0.73	0.78	0.78	0.81
JM1	**0.87**	0.72	0.78	0.76	0.82
KC1	**0.85**	0.59	0.68	0.72	0.75
MC1	**0.79**	0.79	0.85	0.76	0.87
MC2	**0.88**	0.78	0.72	0.72	0.76
MW1	**0.79**	0.72	0.69	0.77	0.68
PC1	**0.86**	0.69	0.78	0.82	0.72
PC2	**0.88**	0.72	0.78	0.75	0.78
PC3	**0.87**	0.82	0.79	0.78	0.75
PC4	**0.85**	0.69	0.78	0.75	0.78
PC5	**0.87**	0.72	0.77	0.79	0.81
ant-1.7	**0.81**	0.78	0.79	0.79	0.79
tomcat	0.79	0.68	**0.82**	0.71	0.72
xalan-2.5	0.78	0.73	0.88	0.82	**0.89**
xalan-2.4	**0.75**	0.77	0.78	0.79	0.77
xalan-2.6	0.78	0.79	0.81	0.81	**0.82**
camel	0.76	0.78	0.83	0.83	**0.86**
synampe	0.78	0.79	0.79	0.79	**0.81**
velocity-1.6	0.72	0.77	0.77	0.79	**0.8**
xerces-1.4	0.75	0.79	0.76	0.79	**0.8**
prop-6	0.77	**0.81**	0.77	0.76	0.78
poi-3.0	**0.76**	**0.76**	0.72	**0.76**	0.75

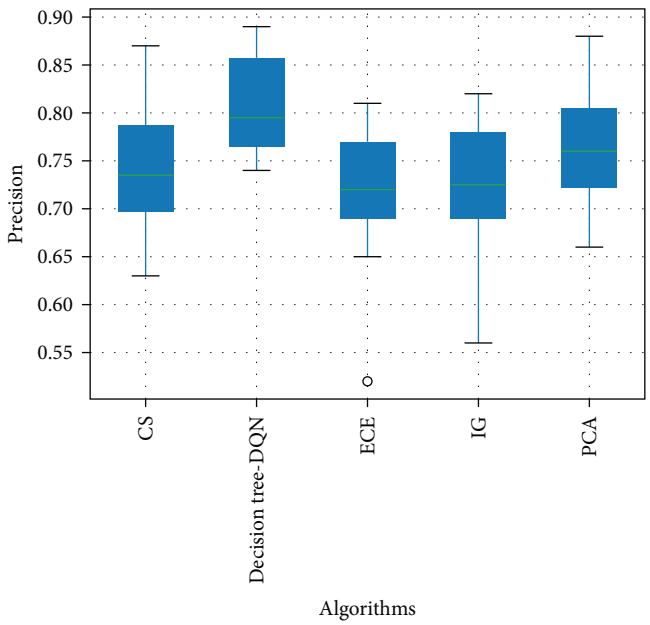

Figure 14: The Scott-Knott ESD ranking for DQN compared with four classic feature selection algorithms in terms of precision evaluation indicators.

indicated that the DQN-FE-SDP algorithm outperformed the other four baseline feature selection or extraction methods in terms of precision evaluation metrics, exhibiting superior performance at the highest level.

Table 20: Comparison of F_1 value with four state-of-the-art feature selection or feature extraction algorithms.

Datasets	Decision tree-DQN	PCA	ECE	IG	CS
CM1	**0.86**	0.84	0.75	0.72	0.79
JM1	**0.87**	0.82	0.79	0.75	0.69
KC1	**0.84**	0.72	0.78	0.76	0.78
MC1	0.80	0.78	0.81	0.65	**0.87**
MC2	**0.87**	0.73	0.72	0.65	0.86
MW1	0.79	0.81	0.69	**0.81**	0.72
PC1	**0.85**	0.76	0.77	0.69	0.63
PC2	**0.88**	0.77	0.66	0.79	0.78
PC3	0.87	**0.88**	0.72	0.69	0.81
PC4	0.83	**0.86**	0.79	0.78	0.79
PC5	**0.89**	0.79	0.76	0.79	0.82
ant-1.7	**0.81**	0.76	0.77	0.68	0.72
tomcat	**0.79**	0.69	0.66	0.72	0.77
xalan-2.5	**0.79**	0.72	0.75	0.78	0.69
xalan-2.4	0.74	**0.81**	0.65	0.66	0.72
xalan-2.6	**0.79**	0.75	0.77	0.69	0.77
camel	**0.76**	0.75	0.69	0.72	0.69
synampe	**0.78**	0.66	0.52	**0.78**	0.65
velocity-1.6	0.75	0.69	0.65	**0.82**	0.72
xerces-1.4	**0.74**	0.71	0.71	0.73	0.73
prop-6	**0.76**	**0.76**	0.69	0.56	0.69
poi-3.0	0.74	**0.79**	0.69	0.74	0.74

The bold values indicate the best-performing value among these five methods on different datasets.

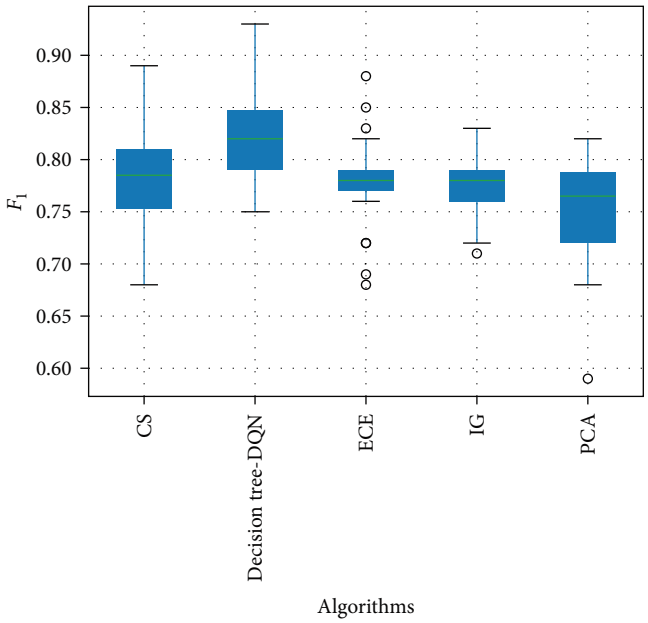

Figure 15: The Scott-Knott ESD ranking for DQN compared with four classic feature selection algorithms in terms of F_1 evaluation indicators.

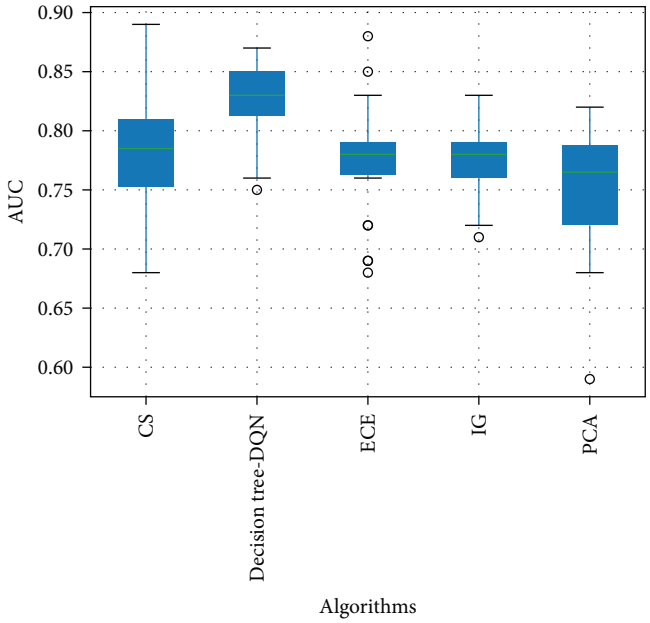

Figure 16: The Scott-Knott ESD ranking for DQN compared with four classic feature selection algorithms in terms of AUC evaluation indicators.

Table 21: Comparison of AUC value with four state-of-the-art feature selection or feature extraction algorithms.

Datasets	Decision tree-DQN	PCA	ECE	IG	CS
CM1	**0.79**	0.74	0.72	0.72	0.72
JM1	**0.75**	0.71	0.71	0.74	0.73
KC1	**0.82**	0.61	0.7	0.71	0.76
MC1	**0.85**	0.69	0.77	0.74	0.79
MC2	**0.82**	0.72	0.68	0.72	0.75
MW1	**0.8**	0.78	0.69	0.67	0.69
PC1	**0.83**	0.72	0.72	0.72	0.76
PC2	**0.87**	0.71	0.73	0.77	0.74
PC3	**0.84**	0.79	0.71	0.79	0.78
PC4	**0.8**	0.72	0.72	0.74	0.74
PC5	**0.82**	0.78	0.75	0.71	0.79
ant-1.7	**0.86**	0.67	0.76	0.74	0.75
tomcat	**0.85**	0.69	0.79	0.75	0.77
xalan-2.5	**0.86**	0.71	0.81	0.79	0.81
xalan-2.4	**0.85**	0.73	0.76	0.81	0.79
xalan-2.6	**0.85**	0.76	0.79	0.76	0.81
camel	**0.81**	0.77	0.79	0.73	0.79
synampe	**0.83**	0.72	0.77	0.72	0.77
velocity-1.6	0.76	0.74	0.73	**0.77**	0.72
xerces-1.4	**0.83**	0.77	0.75	0.76	0.79
prop-6	**0.82**	0.79	0.77	0.72	0.78
poi-3.0	**0.85**	0.71	0.73	0.74	0.76

The bold values indicate the best-performing value among these five methods on different datasets.

Table 22: Comparison of MCC value with four state-of-the-art feature selection or feature extraction algorithms.

Datasets	Decision tree-DQN	PCA	ECE	IG	CS
CM1	**0.69**	0.61	0.58	0.58	0.61
JM1	**0.55**	0.51	0.48	0.46	0.52
KC1	**0.62**	0.55	0.52	0.52	0.55
MC1	**0.64**	0.55	0.55	0.52	0.57
MC2	**0.63**	0.53	0.52	0.59	0.56
MW1	0.61	**0.69**	0.59	0.55	0.59
PC1	**0.71**	0.65	0.61	0.62	0.68
PC2	0.49	0.43	**0.51**	0.43	0.44
PC3	**0.53**	0.42	0.47	0.51	0.51
PC4	**0.49**	0.41	0.45	0.45	0.44
PC5	**0.6**	0.58	0.53	0.59	0.59
ant-1.7	**0.53**	0.52	0.51	0.49	0.51
tomcat	**0.69**	0.57	0.42	0.61	0.62
xalan-2.5	0.56	0.43	**0.58**	0.52	0.52
xalan-2.4	**0.68**	0.59	0.58	0.59	0.57
xalan-2.6	**0.68**	0.58	0.51	0.61	0.62
camel	**0.61**	0.54	0.53	0.53	0.56
synampe	**0.63**	0.55	0.59	0.59	0.51
velocity-1.6	**0.51**	0.47	0.47	0.45	0.41
xerces-1.4	**0.57**	0.45	0.46	0.52	0.49
prop-6	**0.61**	0.42	0.57	0.56	0.45
poi-3.0	**0.65**	0.51	0.52	0.56	0.59

The bold values indicate the best-performing value among these four methods on different datasets.

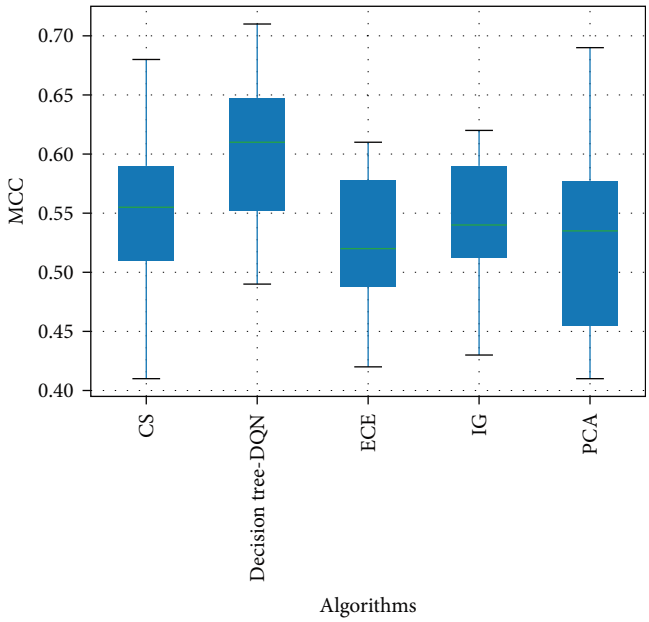

Figure 17: The Scott-Knott ESD ranking for DQN compared with four classic feature selection algorithms in terms of MCC evaluation indicators.

7. Threats to Validity

In this section, we examine three potential sources of validity issues that could impact the results of our experiments.

7.1. Implementation of Compared Models. We compared the various feature selection models with our proposed DQN-FE-SDP model. Since these feature selection algorithms use different datasets from those used in this paper, we implemented these models using Python. Therefore, the outcomes of these models might capture all nuances in the comparison method, and it is possible that the parameters of these models were not meticulously tuned. To ensure a fair comparison, we applied uniform decision tree classification techniques and oversampling methods for SDP. It is worth noting that our feature selection approach may vary from that presented in the original paper.

7.2. Integrity and Representativeness of the Software Measures. Feature extraction is a critical process in SDP. However, when using predetermined measurement elements, there may be limitations in fully capturing all the relevant features of a project. As such, it is essential to leverage DL techniques to extract additional features from the source code for more comprehensive and accurate feature extraction. By doing so, various indicators can be obtained for SDP, leading to better results in identifying potential defects in software projects. Overall, integrating DL-based feature extraction methods has the potential to enhance the accuracy and effectiveness of SDP models, providing developers with more reliable tools for improving software quality.

7.3. Measurement Validity. Measurement validity is a critical aspect in evaluating the effectiveness of any research study.

Table 23: Performance metrics for different configurations—the number of convolution layers.

The number of convolution layers	Metrics			
	F-score	Precision	AUC	MCC
One layer	0.51	0.53	0.57	0.50
Three layers	0.53	0.55	0.59	0.52
Our approach (two layers)	**0.78**	**0.79**	**0.83**	**0.62**

The bold values indicate the best-performing value among these four methods on different datasets.

In this paper, we utilize four commonly used evaluation indicators—F_1, MCC, precision, and AUC—to assess the performance of our proposed methodology for SDP. As these indicators have been widely applied and accepted as reliable measures in previous studies, we believe that our construct validity is satisfactory.

7.4. Hyperparameter Validity. However, another critical factor that could impact the accuracy of our experimental results is the selection of appropriate parameter settings. Several studies have shown that different parameter configurations can significantly influence the outcomes of defect prediction models. To address this issue, we plan to incorporate advanced automated parameter optimization techniques in our future experiments. By leveraging these techniques, our goal is to identify the optimal parameter settings for our proposed methodology, thereby reducing potential threats to construct validity and enhancing the overall reliability of our research.

The choice of hyperparameters plays a vital role in the performance of machine learning models. Overtuning or improper selection of hyperparameters may lead to overfitting on the validation set or failure to converge. Hence, during hyperparameter tuning, careful consideration of suitable parameter combinations is crucial to ensure that the resulting model exhibits good generalization and practicality. Through proper hyperparameter selection, we can gain better insights and evaluations of the effectiveness of our proposed approach, laying a strong foundation for further research and practical applications in the field of SDP.

7.5. Ablation Experiment. In our ablation experiments, we focused on validating the impact of two key design choices on the performance of our proposed DQN model. First, we compared the performance across different layers, including one layer, two layers (our selected configuration), and three layers, aiming to determine the most effective layer configuration. Second, we examined the effect of various padding strategies, specifically comparing zero-padding (our selected configuration) with other methods, such as average, maximum, and minimum value padding, to identify the superior strategy in enhancing performance.

Table 23 presents the performance metrics for different the number of convolution layers. Table 24 presents the performance metrics for different padding processes. Table 25 presents the performance metrics for different combination configurations. Specifically, our chosen of configuration, two

Table 24: Performance metrics for different configurations—various of padding processes.

Padding data	Metrics			
	F-score	Precision	AUC	MCC
Average	0.58	0.60	0.64	0.57
Maximum	0.59	0.61	0.65	0.58
Minimum	0.55	0.57	0.61	0.54
Our approach (zero)	**0.78**	**0.79**	**0.83**	**0.62**

The bold values indicate the best-performing value among these four methods on different datasets.

Table 25: Performance metrics for different combination configurations.

Configurations	Metrics			
	F-score	Precision	AUC	MCC
One layer, average	0.59	0.61	0.65	0.57
One layer, maximum	0.57	0.59	0.63	0.56
One layer, minimum	0.54	0.56	0.60	0.53
One layer, zero	0.51	0.53	0.57	0.50
Two layer, average	0.58	0.60	0.64	0.57
Two layer, maximum	0.59	0.61	0.65	0.58
Two layer, minimum	0.55	0.57	0.61	0.54
Three layer, average	0.57	0.59	0.63	0.56
Three layer, maximum	0.60	0.62	0.66	0.59
Three layer, minimum	0.56	0.58	0.62	0.55
Three layer, zero	0.53	0.55	0.59	0.52
Our approach (two layer, zero)	**0.78**	**0.79**	**0.83**	**0.62**

The bold values indicate the best-performing value among these four methods on different datasets.

convolution layers, and zero-padding demonstrated significant superiority across all performance metrics, including *F*-score, precision, AUC, and MCC. These findings provide direct rational for our selection.

8. Conclusion and Future Work

The increasing demand for accurate SDP techniques across various testing and operational phases has driven the exploration of machine learning-based approaches. This paper focuses on enhancing the precision, *F*-measure, AUC, and MCC values in SDP by introducing a novel deep *Q*-network-based feature extraction method. The objective is to eliminate irrelevant, redundant, and noisy features, thereby improving the overall effectiveness of SDP.

Our approach begins with the optimization of datasets through preprocessing using the BalanceCascade algorithm. Following this, we employ the ECE to calculate the weight ranking of metric elements, constructing a relation matrix using RMT to evaluate the interdependence of features. The final step involves generating a revised feature set, utilizing a *Q*-learning algorithm in conjunction with a CNN model.

To validate the efficacy of the DQN-based feature extraction approach on binary classification algorithm-based SDPs, extensive experiments were conducted on 11 NASA MDP repository and 11 PROMISE repository datasets. Comparisons between traditional SVM, KNN, and decision tree-based SDPs and their DQN-enhanced counterparts revealed significant improvements: SVM-DQN demonstrated an average enhancement of 8.2%, 17.2%, and 84.0% in *F*-measure, AUC, and MCC, respectively; KNN-DQN exhibited improvements of 5.6%, 12.9%, 20.1%, and 69.6%; decision tree-DQN showcased improvements of 11.1%, 12.8%, 26.1%, and 51.5%.

Additionally, we compared our proposed DQN-FE-SDP with four state-of-the-art SPD methods using common datasets. The results underscored the superiority of our approach, with average improvements of 3.5%, 3.7%, 4.1%, and 5.7% in precision, *F*-measure, AUC, and MCC compared to K-PCA-ELM; 11.4% and 31.8% improvements in AUC and *F*-measure compared to CFIW-TNB; and 9.3% and 95.5% improvements in AUC and *F*-measure compared to MDA-O. Moreover, our approach showed an outstanding improvement of 92.6% in MCC and 24.8% in *F*-measure compared to WEM.

In our future research endeavors, we aim to delve deeper into the intricacies of the reward function within our proposed DQN-based feature extraction approach. Specifically, we will explore how variations in reward values impact the prediction performance of the model. This investigation will involve a meticulous analysis of different reward structures to identify optimal configurations that maximize the efficacy of the DQN-FE-SDP system.

Furthermore, our plan includes a substantial expansion of the research scope by incorporating a more diverse set of metrics. Currently, our focus has been on specific metrics relevant to the SDP context. However, we recognize the importance of considering a broader spectrum of metrics that may provide additional insights into the performance of the DQN-based approach. This expansion will involve the inclusion of various software metrics, potentially encompassing different programing languages and development paradigms.

In addition to diversifying the metrics, we intend to explore the applicability of our proposed DQN-based feature extraction approach across a wider range of classifiers. While our current experiments have demonstrated significant improvements with traditional classifiers such as SVM, KNN, and decision tree, we acknowledge that different classifiers may yield varied results. By systematically applying our approach to an extended set of classifiers, including ensemble methods and DL models, we aim to assess its versatility and robustness in diverse SDP scenarios.

To achieve these goals, we plan to leverage an expanded collection of datasets, covering a more comprehensive range of software development domains. This will not only enhance the generalizability of our findings but also allow us to identify specific nuances and challenges that may arise in different application contexts. By systematically evaluating the performance of our DQN-based approach across various datasets, metrics, and classifiers, we aspire to provide a nuanced understanding of its strengths and limitations in real-world SDP scenarios.

Looking ahead, our future research endeavors will focus on a more in-depth exploration of the reward function's impact on prediction performance. Additionally, we plan to expand the scope of our research by incorporating diverse

metrics and employing a broader range of classifiers across an extended set of datasets. This will contribute to a more comprehensive understanding of the proposed DQN-based feature extraction approach and its potential applicability in various SDP scenarios.

Data Availability

The datasets generated during the current study are available in the Github repository: https://github.com/asqwq/SDP-using-DQN-based-feature-extraction.

Conflicts of Interest

The authors declare that there are no conflicts of interest regarding the publication of this paper.

Acknowledgments

This work was supported by the National Key Research and Development Program of China (grant number 2018YFC0831706), the Science and Technology Development Plan Project of Jilin Province (grant number 20200201166JC), and the Central Government Funds for Guiding Local Scientific and Technological Development (grant number 2021Szvup047).

References

[1] A. G. Koru and H. Liu, "Building effective defect-prediction models in practice," *IEEE Software*, vol. 22, no. 6, pp. 23–29, 2005.

[2] M. Kakkar, S. Jain, A. Bansal, and P. S. Grover, "Combining data preprocessing methods with imputation techniques for software defect prediction," *International Journal of Open Source Software and Processes*, vol. 9.1, pp. 1–19, 2018.

[3] A. V. Phan and M. Le Nguyen, "Convolutional neural networks on assembly code for predicting software defects," in *2017 21st Asia Pacific Symposium on Intelligent and Evolutionary Systems (IES)*, pp. 37–42, IEEE, Hanoi, Vietnam, November 2017.

[4] A. Chug and S. Dhall, "Software defect prediction using supervised learning algorithm and unsupervised learning algorithm," in *Confluence 2013: The Next Generation Information Technology Summit (4th International Conference)*, pp. 173–179, IET, Noida, September 2013.

[5] S. Lessmann, B. Baesens, C. Mues, and S. Pietsch, "Benchmarking classification models for software defect prediction: a proposed framework and novel findings," *IEEE Transactions on Software Engineering*, vol. 34, no. 4, pp. 485–496, 2008.

[6] H. Ji, S. Huang, Y. Wu, Z. Hui, and C. Zheng, "A new weighted naive Bayes method based on information diffusion for software defect prediction," *Software Quality Journal*, vol. 27, no. 3, pp. 923–968, 2019.

[7] D. Gray, D. Bowes, N. Davey, Y. Sun, and B. Christianson, "Using the support vector machine as a classification method for software defect prediction with static code metrics," in *Engineering Applications of Neural Networks: 11th International Conference, EANN 2009*, pp. 223–234, Springer, Berlin, Heidelberg, London, UK, August 2009.

[8] B. Ghotra, S. McIntosh, and A. E. Hassan, "Revisiting the impact of classification techniques on the performance of defect prediction models," in *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*, pp. 789–800, IEEE, Florence, Italy, May 2015.

[9] T. M. Khoshgoftaar and N. Seliya, "Tree-based software quality estimation models for fault prediction," in *Proceedings Eighth IEEE Symposium on Software Metrics*, pp. 203–214, IEEE, Ottawa, ON, Canada, June 2002.

[10] H.-M. Wong, X. Chen, H.-H. Tam et al., "Feature selection and feature extraction: highlights," in *Proceedings of the 2021 5th International Conference on Intelligent Systems, Metaheuristics & Swarm Intelligence*, pp. 49–53, Association for Computing Machinery, Victoria, Seychelles, April 2021.

[11] V. Mnih, K. Kavukcuoglu, D. Silver et al., "Human-level control through deep reinforcement learning," *Nature*, vol. 518, pp. 529–533, 2015.

[12] T.-H. Chan, K. Jia, S. Gao, J. Lu, Z. Zeng, and Y. Ma, "PCANet: a simple deep learning baseline for image classification?" *IEEE Transactions on Image Processing*, vol. 24, no. 12, pp. 5017–5032, 2015.

[13] R. S. Sutton and A. G. Barto, "Reinforcement learning: an introduction," *IEEE Transactions on Neural Networks*, vol. 9, no. 5, pp. 1054–1054, 1998.

[14] Z. Qu, C. Hou, C. Hou, and W. Wang, "Radar signal intra-pulse modulation recognition based on convolutional neural network and deep Q-learning network," *IEEE Access*, vol. 8, pp. 49125–49136, 2020.

[15] M. Hessel, J. Modayil, H. Van Hasselt et al., "Rainbow: combining improvements in deep reinforcement learning," in *Thirty-Second AAAI Conference on Artificial Intelligence*, AAAI Press, Palo Alto, California USA, 2018.

[16] J. P. Keating and N. C. Snaith, "Random matrix theory and $\zeta(1/2+it)$," *Communications in Mathematical Physics*, vol. 214, pp. 57–89, 2000.

[17] N. Zheng, G. Loizou, X. Jiang, X. Lan, and X. Li, *MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications*, Computer Vision and Pattern Recognition, 2017.

[18] D. Rodriguez, I. Herraiz, R. Harrison, J. Dolado, and J. C. Riquelme, "Preliminary comparison of techniques for dealing with imbalance in software defect prediction," in *Proceedings of the 18th International Conference on Evaluation and Assessment in Software Engineering*, pp. 1–10, Association for Computing Machinery, London, England, United Kingdom, May 2014.

[19] J. Sayyad Shirabad and T. J. Menzies, "The PROMISE repository of software engineering databases," *School of Information Technology and Engineering*, 2005, https://www.mendeley.com/catalogue/899f795a-cfe3-3849-b202-02301a9c9a0c/.

[20] X.-Y. Liu, J. Wu, and Z.-H. Zhou, "Exploratory undersampling for class-imbalance learning," *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, vol. 39, no. 2, pp. 539–550, 2009.

[21] A. Balaram and S. Vasundra, "Prediction of software fault-prone classes using ensemble random forest with adaptive synthetic sampling algorithm," *Automated Software Engineering*, vol. 29, Article ID 6, 2022.

[22] N. Bouguila, J. H. Wang, and A. Ben Hamza, "A bayesian approach for software quality prediction," in *2008 4th International IEEE Conference Intelligent Systems*, pp. 11-49–11-54, IEEE, Varna, Bulgaria, September 2008.

[23] L. Kumar, S. K. Sripada, A. Sureka, and S. K. Rath, "Effective fault prediction model developed using least square support vector machine (LSSVM)," *Journal of Systems and Software*, vol. 137, pp. 686–712, 2018.

[24] T. Wang, W. Li, H. Shi, and Z. Liu, "Software defect prediction based on classifiers ensemble," *Journal of Information and Computational Science*, vol. 8, no. 16, pp. 4241–4254, 2011.

[25] J. Zheng, "Cost-sensitive boosting neural networks for software defect prediction," *Expert Systems with Applications*, vol. 37, no. 6, pp. 4537–4543, 2010.

[26] M. Owhadi-Kareshk, Y. Sedaghat, and M.-R. Akbarzadeh-T, "Pre-training of an artificial neural network for software fault prediction," in *2017 7th International Conference on Computer and Knowledge Engineering (ICCKE)*, pp. 223–228, IEEE, Mashhad, Iran, October 2017.

[27] A. Mahaweerawat, P. Sophasathit, and C. Lursinsap, "Software fault prediction using fuzzy clustering and radial-basis function network," 2002.

[28] A. Viet Phan, M. Le Nguyen, and L. Thu Bui, "Convolutional neural networks over control flow graphs for software defect prediction," in *2017 IEEE 29th International Conference on Tools with Artificial Intelligence (ICTAI)*, pp. 45–52, IEEE, Boston, MA, USA, November 2017.

[29] G. Lu, Z. Lie, and L. I. Hang, "Deep belief network software defect prediction model," *Computer Science*, vol. 44, no. 4, pp. 229–233, 2017.

[30] S. Albahli, "A deep ensemble learning method for effort-aware just-in-time defect prediction," *Future Internet*, vol. 11, no. 12, Article ID 246, 2019.

[31] G. Giray, K. E. Bennin, Ö. Köksal, Ö. Babur, and B. Tekinerdogan, "On the use of deep learning in software defect prediction," *Journal of Systems and Software*, vol. 195, Article ID 111537, 2023.

[32] S. U. Jin-Shu, Z. Bo-Feng, and X. U. Xin, "Advances in machine learning based text categorization," *Journal of Software*, vol. 17, no. 9, pp. 1848–1859, 2006.

[33] L. Haifeng, W. Yuanyuan, Y. Zeqing, and C. Qi, "A method of reducing features based on feature selection twice in text classification," *Journal of the China Society for Scientific and Technical Informatio*, vol. 28, pp. 23–27, 2009.

[34] P. A. Estévez, M. Tesmer, C. A. Perez, and J. M. Zurada, "Normalized mutual information feature selection," *IEEE Transaction on Neural Networks*, vol. 20, pp. 189–201, 2009.

[35] H. Li, C.-J. Li, X.-J. Wu, and J. Sun, "Statistics-based wrapper for feature selection: an implementation on financial distress identification with support vector machine," *Applied Soft Computing*, vol. 19, pp. 57–67, 2014.

[36] X. Zhang, K. Ben, and J. Zeng, "Cross-entropy: a new metric for software defect prediction," in *2018 IEEE International Conference on Software Quality, Reliability and Security (QRS)*, pp. 111–122, IEEE, Lisbon, Portugal, July 2018.

[37] L. Galavotti, F. Sebastiani, and M. Simi, *Experiments on the use of Feature Selection and Negative Evidence in Automated Text Categorization*, pp. 59–68, Springer-Verlag, Berlin, Heidelberg, 2000.

[38] A. Ali, N. Khan, M. Abu-Tair, J. Noppen, S. McClean, and I. McChesney, "Discriminating features-based cost-sensitive approach for software defect prediction," *Automated Software Engineering*, vol. 28, no. 2, pp. 1–18, 2021.

[39] J. Priyadarshini, M. Premalatha, R. Čep, M. Jayasudha, and K. Kalita, "Analyzing physics-inspired metaheuristic algorithms in feature selection with K-nearest-neighbor," *Applied Sciences*, vol. 13, no. 2, Article ID 906, 2023.

[40] J. Jiarpakdee, C. Tantithamthavorn, and C. Treude, "The impact of automated feature selection techniques on the interpretation of defect models," *Empirical Software Engineering*, vol. 25, no. 5, pp. 3590–3638, 2020.

[41] S. K. Pandey, D. Rathee, and A. K. Tripathi, "Software defect prediction using K-PCA and various kernel-based extreme learning machine: an empirical study," *IET Software*, vol. 14, no. 7, pp. 768–782, 2020.

[42] R. A. Fisher, "The use of multiple measurements in taxonomic problems," *Annals of Eugenics*, vol. 7, no. 2, pp. 179–188, 1936.

[43] R. Malhotra and K. Khan, "A study on software defect prediction using feature extraction techniques," in *2020 8th International Conference on Reliability, Infocom Technologies and Optimization (Trends and Future Directions) (ICRITO)*, pp. 1139–1144, IEEE, Noida, India, June 2020.

[44] S. Xiang, F. Nie, and C. Zhang, "Learning a Mahalanobis distance metric for data clustering and classification," *Pattern Recognition*, vol. 41, no. 12, pp. 3600–3612, 2008.

[45] M. M. NezhadShokouhi, M. A. Majidi, and A. Rasoolzadegan, "Software defect prediction using over-sampling and feature extraction based on Mahalanobis distance," *The Journal of Supercomputing*, vol. 76, no. 1, pp. 602–635, 2020.

[46] N. Zhang, S. Ying, K. Zhu, and D. Zhu, "Software defect prediction based on stacked sparse denoising autoencoders and enhanced extreme learning machine," *IET Software*, vol. 16, no. 1, pp. 29–47, 2022.

[47] M. Shepperd, Q. Song, Z. Sun, and C. Mair, "Data quality: some comments on the NASA software defect datasets," *IEEE Transactions on Software Engineering*, vol. 39, no. 9, pp. 1208–1215, 2013.

[48] Y. Wu, Y. Ke, Z. Chen, S. Liang, H. Zhao, and H. Hong, "Application of alternating decision tree with AdaBoost and bagging ensembles for landslide susceptibility mapping," *CATENA*, vol. 187, Article ID 104396, 2020.

[49] X. Wang, B. Yu, A. Ma, C. Chen, B. Liu, and Q. Ma, "Protein–protein interaction sites prediction by ensemble random forests with synthetic minority oversampling technique," *Bioinformatics*, vol. 35, no. 14, pp. 2395–2402, 2019.

[50] J. Y. Wu, "Study and analyze on feature selection in text categorization for engineering domain," in *Emerging Materials and Mechanics Applications*, vol. 487, pp. 383–386, Trans Tech Publications Ltd., Advanced Materials Research, 2012.

[51] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," 2014.

[52] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems*, vol. 25, Curran Associates, Inc., 2012.

[53] X. Glorot, A. Bordes, and Y. Bengio, "Deep sparse rectifier neural networks," in *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, pp. 315–323, PMLR, Fort Lauderdale, FL, USA, 2011.

[54] M. D. Zeiler and R. Fergus, "Visualizing and understanding convolutional networks," in *Computer Vision–ECCV 2014*, pp. 818–833, Springer, Cham, 2014.

[55] U. Upadhyay, B. Rudra, U. Upadhyay, U. Upadhyay, and B. Rudra, "Face mask recognition system using CNN model," 2021.

[56] J. A. Hanley and B. J. McNeil, "The meaning and use of the area under a receiver operating characteristic (ROC) curve," *Radiology*, vol. 143, no. 1, pp. 29–36, 1982.

[57] F. J. Linnig, J. Mandel, and J. M. Peterson, "Plan for studying accuracy and precision of analytical procedure," *Analytical Chemistry*, vol. 26, no. 7, pp. 1102–1110, 1954.

[58] M. Sokolova, N. Japkowicz, and S. Szpakowicz, *Beyond Accuracy, F-Score and ROC: A Family of Discriminant Measures for Performance Evaluation*, pp. 1015–1021, Springer, Berlin Heidelberg, 2006.

[59] B. W. Matthews, "Comparison of the predicted and observed secondary structure of t4 phage lysozyme," *Biochimica et Biophysica Acta (BBA) - Protein Structure*, vol. 405, no. 2, pp. 442–451, 1975.
[60] Q. Zou, L. Lu, S. Qiu, X. Gu, and Z. Cai, "Correlation feature and instance weights transfer learning for cross project software defect prediction," *IET Software*, vol. 15, no. 1, pp. 55–74, 2021.
[61] Y. Sun, X.-Y. Jing, F. Wu, and Y. Sun, "Manifold embedded distribution adaptation for cross-project defect prediction," *IET Software*, vol. 14, no. 7, pp. 825–838, 2020.
[62] F. Yang, G. Zeng, F. Zhong, W. Zheng, and P. Xiao, "Interpretable software defect prediction incorporating multiple rules," in *2023 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*, pp. 940–947, IEEE, Taipa, Macao, March 2023.
[63] K. Zhu, S. Ying, N. Zhang, and D. Zhu, "Software defect prediction based on enhanced metaheuristic feature selection optimization and a hybrid deep neural network," *Journal of Systems and Software*, vol. 180, Article ID 111026, 2021.
[64] M. Tsunoda, A. Monden, K. Toda et al., "Using bandit algorithms for selecting feature reduction techniques in software defect prediction," in *2022 IEEE/ACM 19th International Conference on Mining Software Repositories (MSR)*, pp. 670–681, IEEE, Pittsburgh, PA, USA, May 2022.
[65] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "A simple way to prevent neural networks from overfitting," *The Journal of Machine Learning Research*, vol. 15, pp. 1929–1958, 2014.
[66] S. Goyal and P. K. Bhatia, "Comparison of machine learning techniques for software quality prediction," *International Journal of Knowledge and Systems Science*, vol. 11, no. 2, pp. 20–40, 2020.
[67] I. H. Laradji, M. Alshayeb, and L. Ghouti, "Software defect prediction using ensemble learning on selected featues," *Information and Software Technology*, vol. 58, pp. 388–402, 2015.