Reinforcement Learning-Based Software Metric Selection for Defect Prediction

Dipender Singh[a], Abhinav Jamwal[b], Manish Agrawal[c] and Sandeep Kumar[d]
Department of Computer Science and Engineering, Indian Institute of Technology Roorkee, Roorkee, India

Keywords: Software Defect Prediction, Metric Selection, Reinforcement Learning, Deep Q-Network, Within-Project Defect Prediction.

Abstract: Software defect prediction aims to identify defect-prone modules so that testing resources can be allocated effectively. Modern software projects often contain a large number of software metrics, many of which may be irrelevant or redundant for prediction. Using all available metrics can increase model complexity and reduce prediction effectiveness. To address this issue, this paper proposes a reinforcement-learning-based metric selection framework called Reinforced Metric Selection for Software Defect Prediction (RMS-DP). The proposed approach formulates metric selection as a sequential decision-making problem and employs a Deep Q-Network (DQN) to learn an adaptive metric selection policy. The selected metric subset is then used to train a Logistic Regression classifier for defect prediction. The framework is evaluated on seven projects from the JIRA dataset containing 65 software metrics. Experimental results show that RMS-DP improves MCC and Recall compared with a non-metric-selection baseline while reducing the number of metrics from 65 to approximately 4–10 for most projects. Additional experiments further demonstrate the robustness of the selected metrics across multiple classifiers and highlight the importance of class-imbalance handling in defect prediction.

1 INTRODUCTION

Software defects significantly influence software reliability, system stability, and long-term maintenance costs (Chen and Huang, 2009). Detecting defect-prone components during the early stages of development enables development teams to prioritize testing activities and allocate quality assurance resources more efficiently. Consequently, software defect prediction has become an important research topic in software engineering (Thota et al., 2020). These approaches analyze historical project information together with machine learning models to estimate the likelihood that a software module contains defects. Among the different prediction settings, Within-Project Defect Prediction (WPDP) is widely adopted, where prediction models are constructed using historical data from the same software project (Hosseini et al., 2017). Software defect prediction models generally make use of software metrics derived from source code and software development activities. Modern software repositories often contain a large number of metrics describing different aspects of software modules. These include code complexity, size, change history, and developer activity. However, not all metrics contribute equally to defect prediction performance (Jureczko, 2011). Some metrics may be irrelevant, redundant, or noisy, which can increase model complexity and reduce prediction accuracy. Therefore, identifying a compact and informative subset of metrics is an important step in improving defect prediction models. Several feature selection techniques have been explored in defect prediction studies. These include filter-based methods, wrapper-based methods, and embedded approaches (Bai et al., 2022; Bal et al., 2025). Although these methods can reduce feature dimensionality, most existing feature selection approaches rely on static ranking or greedy search strategies, which evaluate metrics independently and often fail to capture complex interactions among metrics. As a result, useful combinations of metrics may be overlooked, and the resulting subsets may not general-

[a] https://orcid.org/0009-0009-5352-5453
[b] https://orcid.org/0000-0002-0213-3590
[c] https://orcid.org/0009-0009-0216-2592
[d] https://orcid.org/0000-0002-3250-4866

Singh, D., Jamwal, A., Agrawal, M. and Kumar, S.
Reinforcement Learning-Based Software Metric Selection for Defect Prediction.
DOI: 10.5220/0015039800004015

In *Proceedings of the 21st International Conference on Evaluation of Novel Approaches to Software Engineering (ENASE 2026) - Volume 2*, pages 1105-1112
ISBN: 978-989-758-828-0; ISSN: 2184-4895

ize well across different projects (Madeyski and Jureczko, 2015). In addition, the search space of possible metric subsets grows exponentially with the number of available metrics. This makes exhaustive exploration impractical for many software projects.

To address these challenges, this study proposes a Reinforced Metric Selection for Software Defect Prediction (RMS-DP). The framework models metric selection as a sequential decision-making problem and employs a Deep Q-Network (DQN) to learn an adaptive selection policy. The agent explores different metric combinations and receives rewards based on prediction performance and subset compactness. Through repeated interactions, it learns to identify informative subsets that improve prediction quality while reducing redundancy. After selection, the chosen metrics are used to train a defect prediction model. Logistic Regression (LR) is adopted as the final classifier due to its simplicity, interpretability, and strong performance in metric-based studies. The main contributions of this work are summarized as follows:

- We propose RMS-DP, a reinforcement-learning-based framework that automatically learns an adaptive metric selection policy for software defect prediction.
- The proposed approach identifies compact and informative metric subsets while maintaining or improving prediction performance.
- An empirical evaluation is conducted on seven JIRA projects. This analyzes the effectiveness and robustness of the proposed framework.
- The study further investigates the impact of classifier choice and class imbalance handling on defect prediction performance.

To systematically evaluate the effectiveness of RMS-DP, the following research questions are investigated:

- **RQ1:** How does RMS-DP perform compared to a non-metric-selection method?
- **RQ2:** How does RMS-DP perform when used with different machine learning classifiers?
- **RQ3:** How important is class-imbalance handling for software defect prediction?

The remainder of this paper is organized as follows. Section 2 presents a review of related studies on defect prediction. Section 3 describes the proposed RMS-DP framework. Section 4 explains the experimental design and dataset. Section 5 reports and analyzes the experimental findings. Section 6 discusses threats to validity, and Section 7 concludes the paper while outlining directions for future work.

2 RELATED WORK

Within-Project Defect Prediction (WPDP) builds models using historical data from the same project. Research has improved feature representation, addressed class imbalance, and enhanced prediction reliability. (Pan et al., 2019) propose a CNN-based framework that captures semantic information from source code by converting Abstract Syntax Trees (ASTs) into embedded token sequences, extracting features via multi-layer CNNs, and applying logistic regression for prediction, emphasizing semantics over traditional metrics. (Gong et al., 2019) introduce STr-NN to handle class imbalance using pseudo-labeling and nearest neighbors, and integrate Transfer Component Analysis (TCA) for cross-project settings. (Khatri and Singh, 2022) survey CPDP methods, categorizing them into data standardization, training data selection, transfer learning, and ensemble learning, while noting challenges like imbalance and distribution mismatch. (Bai et al., 2022) propose 3SW-MSTL, a multi-source transfer learning framework with source selection, KMM-based alignment, and conditional weighting. (Bhutamapuram, 2023) present HIEL, an ensemble combining heterogeneous classifiers with probabilistic voting and semi-supervised defect severity prediction. (Liu et al., 2024) show that removing duplicated unchanged modules improves evaluation validity. (Bal et al., 2025) introduce DT-WPDP, which augments training data with selected cross-project instances and uses both deep and traditional models. Despite these advances, adaptive metric selection remains limited. Most methods rely on all metrics or static selection, ignoring prediction feedback. In contrast, RMS-DP formulates metric selection as a sequential decision problem using reinforcement learning, enabling adaptive construction of compact, informative subsets while maintaining reliable prediction performance.

3 METHODOLOGY

This section presents the proposed framework, named Reinforced Metric Selection for Software Defect Prediction (RMS-DP). The goal of RMS-DP is to automatically identify the most informative subset of software metrics for defect prediction. Software projects often contain many metrics, and only a subset contributes significantly to predicting defect-prone modules. Instead of directly using all available metrics, RMS-DP learns an adaptive metric selection policy through reinforcement learning. After preprocessing the project data, a Deep Q-Network (DQN)-

based agent explores different metric combinations. It selects those that maximize prediction performance while maintaining a compact metric subset. Finally, the selected subset is used to train a defect prediction model for classifying defective and non-defective modules. The overall workflow of the proposed RMS-DP framework is illustrated in Figure 1.

3.1 Problem Formulation

Let a software project be denoted as:

$$D = \{(x_i, y_i)\}_{i=1}^{N} \tag{1}$$

where N is the total number of software modules, $x_i \in \mathbb{R}^m$ is the metric vector of the i-th module, and $y_i \in \{0,1\}$ is the corresponding defect label, where 1 denotes a defective module and 0 denotes a non-defective module. Here, m represents the total number of software metrics extracted from the project. The task of defect prediction is to learn a classifier that can map the software metrics of a module to its defect label. However, not all metrics contribute equally to prediction performance. Some metrics are highly informative, whereas others may be irrelevant, redundant, or noisy. If all metrics are directly used for model construction, the prediction model may suffer from increased complexity, reduced interpretability, and degraded performance. Therefore, the problem addressed in this work is to identify an optimal subset of metrics from the full metric set. Let $M = \{f_1, f_2, \ldots, f_m\}$ denote the complete set of available metrics, and let $S \subseteq M$ denote a selected subset. The objective is to determine the best subset S^* such that the defect prediction performance is maximized while the subset remains compact. This can be written as:

$$S^* = \arg\max_{S \subseteq M} \text{Perf}(S) \tag{2}$$

where $\text{Perf}(S)$ denotes the prediction performance obtained using the subset S. A major challenge in this problem is that the total number of possible metric subsets is 2^m. This makes exhaustive search impractical even for a moderate number of metrics. To address this issue, the proposed RMS-DP framework formulates metric selection as a sequential decision-making problem and solves it using reinforcement learning. In this way, the model does not simply rank metrics once. Instead, it learns how to progressively build a useful subset based on predictive feedback.

3.2 Data Processing

Before metric selection and prediction, the data are preprocessed to improve quality and reliability, as defect datasets often contain missing values, duplicates, class imbalance, and differences in metric scales (Bhandari et al., 2023). First, instances with missing values are removed to avoid unstable or misleading patterns. Duplicate instances are then eliminated to prevent bias caused by repeated observations. The cleaned data are divided into training and testing sets: the training set is used for metric selection and model learning, while the test set is reserved for final evaluation. Since defect data are typically highly imbalanced, class imbalance handling is applied only to the training set (Singh and Kumar, 2026). SMOTE (Chawla et al., 2002) is used to generate synthetic minority samples, which helps improve defect pattern learning and reduces bias toward the majority class. The test set is kept unchanged. Finally, feature normalization is applied to bring all metrics to a similar scale, preventing large-valued features from dominating others and supporting stable reinforcement learning and classification.

3.3 Reinforcement-Learning-Based Metric Selection

The core component of RMS-DP is a reinforcement-learning-based metric selection strategy. The main motivation is that traditional feature selection methods usually rely on static ranking or greedy subset construction (Afsar et al., 2022). Such methods often fail to capture the dependency among metrics and may overlook useful combinations. In contrast, reinforcement learning treats metric selection as a sequential process, where each decision affects the next state and the final prediction performance. In the RMS-DP framework, metric selection is modeled as a Markov Decision Process (MDP) (Wang et al., 2025). At each step, an agent observes the current subset of selected metrics, performs an action to modify the subset, and then receives a reward based on the quality of that subset. Through repeated interaction, the agent gradually learns which metrics should be selected and which should be ignored.

3.3.1 State Representation

A state represents the current metric subset and is encoded as a binary vector:

$$s_t = [b_1, b_2, \ldots, b_m] \tag{3}$$

where

$$b_j = \begin{cases} 1, & \text{if the } j\text{-th metric is selected,} \\ 0, & \text{otherwise.} \end{cases} \tag{4}$$

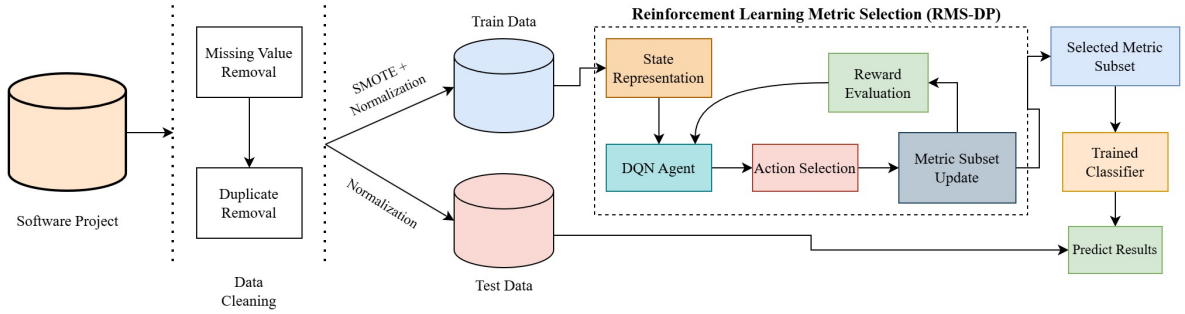

Figure 1: Overall workflow of the proposed RMS-DP framework for reinforcement-learning-based metric selection and software defect prediction.

3.3.2 Action Space

At each step, the agent updates the metric subset by adding an unselected metric or removing a selected one. This toggle-based mechanism enables exploration of different metric combinations and allows evaluation of metrics both individually and in relation to the current subset. An episode starts with an empty subset and proceeds through sequential actions that modify it. At each step, the agent selects an add or remove action. The episode ends when a maximum number of steps is reached or no further performance improvement is observed. The final subset is then used to compute the reward and update the learning policy.

3.3.3 Reward Design

The reward guides the agent toward better metric subsets. In RMS-DP, the reward is designed to favor subsets that achieve strong defect prediction performance while remaining compact. A general reward formulation is given by

$$R_t = \alpha \cdot \mathrm{Perf}(S_t) - \beta \cdot \frac{|S_t|}{m} \tag{5}$$

where $\mathrm{Perf}(S_t)$ denotes the predictive performance obtained using the selected subset S_t, $|S_t|$ is the number of selected metrics, and α and β are balancing factors. The first term encourages the agent to choose informative metrics, while the second term penalizes unnecessarily large subsets. Therefore, the learned policy aims to produce a small but effective feature subset.

3.3.4 Deep Q-Network Policy

To learn the metric selection policy, RMS-DP employs a Deep Q-Network (DQN) (Fan et al., 2020), which approximates the action-value function $Q(s,a)$ estimating the expected long-term benefit of taking action a in state s. Here, the state represents the current subset of selected metrics, and the action updates it by adding or removing a metric. Thus, metric selection is modeled as a sequential decision process where actions iteratively construct candidate subsets. At each step, the state vector is input to the Q-network, which outputs Q-values for available actions. The agent selects actions using an exploration–exploitation strategy, where exploration tests new combinations and exploitation favors previously rewarding actions. After each action, the subset is updated, evaluated on training data using a classifier, and assigned a reward based on predictive performance and subset compactness. Higher rewards are given to subsets achieving better performance with fewer metrics.

Algorithm 1: RMS-DP for Software Defect Prediction.

Input: Project $D = \{(x_i, y_i)\}_{i=1}^{N}$.
Output: Defect prediction output using selected metric subset S^*.

Remove missing and duplicate instances from D.
Split D into training set D_{tr} and test set D_{te}.
Apply SMOTE on D_{tr} and normalize D_{tr} and D_{te}.
Initialize metric subset state s_0 and DQN policy network.
for *each episode* **do**
    Observe current state s_t and select action a_t.
    Update subset S_t, compute reward R_t, and obtain new state s_{t+1}.
    Update DQN using (s_t, a_t, R_t, s_{t+1}).
Obtain final selected subset S^* from the learned policy.
Train LR on D_{tr} using S^* and evaluate on D_{te}.

Using the observed transition (s_t, a_t, r_t, s_{t+1}), the parameters of the Q-network are iteratively updated so that actions leading to more informative metric subsets receive higher preference over time. Through repeated interaction with the environment across multiple episodes, the DQN gradually learns a policy that favors compact and predictive metric subsets. The action-value update is expressed as

$$Q(s_t,a_t) \leftarrow Q(s_t,a_t) + \eta \Big[r_t + \gamma \max_{a'} Q(s_{t+1},a') - Q(s_t,a_t) \Big] \tag{6}$$

where η denotes the learning rate, r_t denotes the re-

Table 1: Overview of JIRA projects used for experimental evaluation.

Project	No. of Metrics	No. of Instances	Defect (%)
activemq-5.3.0	65	2367	10.90
camel-1.4.0	65	1515	18.81
derby-10.5.1.1	65	2705	14.16
groovy-1_6_BETA_1	65	821	8.53
hbase-0.95.2	65	1834	26.34
hive-0.12.0	65	3993	19.83
wicket-1.5.3	65	1566	31.93

ward obtained after taking action a_t in state s_t, and γ is the discount factor that determines the importance of future rewards. By iteratively updating the Q-network according to this rule, RMS-DP is able to explore the large metric search space adaptively and identify informative subsets that contribute to improved defect prediction performance.

3.4 Defect Prediction Classifier

After reinforcement-learning-based metric selection identifies the final metric subset, these metrics are used to train the defect prediction model. RMS-DP employs Logistic Regression (LR) as the final predictor due to its widespread use in defect prediction and suitability for tabular metric data (Jiang et al., 2024; Bal and Kumar, 2025). LR is simple, effective, and interpretable. Since RMS-DP already performs intelligent metric selection, a complex classifier is unnecessary. Using LR allows clearer assessment of the selected metrics' contribution, ensuring that performance primarily reflects the effectiveness of RMS-DP rather than classifier complexity. The overall steps of the proposed RMS-DP framework are presented in Algorithm 1. Lines 1–3 perform preprocessing, including data cleaning, train–test splitting, SMOTE-based imbalance handling, and normalization. Line 4 initializes the metric subset state and DQN policy network. Lines 5–8 describe the reinforcement learning process. Finally, Lines 9–10 obtain the selected subset and train and evaluate the Logistic Regression classifier.

4 EXPERIMENTAL SETUP

This section describes the datasets, evaluation metrics, and experimental protocol used to assess the effectiveness of the proposed RMS-DP framework.

4.1 Dataset Overview

To evaluate RMS-DP, experiments are conducted on the JIRA defect dataset (Yatish et al., 2019), collected from open-source repositories. It contains software metrics extracted from Java projects, where each instance represents a Java source file. The dataset includes 65 metrics: 54 static code metrics, 5 process metrics, and 6 ownership metrics, capturing structural and development characteristics of software modules. Following prior work, seven representative JIRA projects are selected. These projects vary in size and defect distribution, enabling evaluation of RMS-DP robustness across different settings. Table 1 summarizes the selected projects.

4.2 Evaluation Metrics

Software defect prediction is typically formulated as a binary classification task with strong class imbalance, where non-defective instances dominate. This makes accuracy misleading and overly optimistic. To address this, Matthews Correlation Coefficient (MCC) and Recall are used as primary evaluation metrics (Yao and Shepperd, 2021). MCC is well-suited for imbalanced data as it considers all components of the confusion matrix, providing a balanced performance measure. Recall evaluates the ability to correctly identify defective modules, which is critical since missed defects increase maintenance and debugging costs. Together, MCC and Recall offer a reliable assessment of overall performance and defect detection capability.

4.3 Evaluation Protocol

Experiments follow a structured protocol to ensure robust, unbiased results. For each project, data is split into training and testing sets: the training set is used for RMS-DP metric selection and model training, while the test set is reserved for evaluation. A five-fold cross-validation is applied, where data is divided into five folds, four used for training and one for testing, until each fold serves as the test set. To reduce randomness, the entire process is repeated ten times with different seeds, and results are averaged. In each fold, metric selection is performed using only training data, and the test fold is excluded from selection, training, and imbalance handling. SMOTE and feature normalization are applied only to training data, and the learned transformations are then applied to the test fold, preventing data leakage and ensuring reliable performance estimation.

5 EXPERIMENTAL RESULTS AND ANALYSIS

5.1 RQ1: How Does RMS-DP Perform Compared to a Non-Metric-Selection Method?

This research question evaluates the effectiveness of the proposed RMS-DP framework in comparison with a non-metric-selection baseline. The objective is to analyze whether reinforcement-learning-based metric selection can improve defect prediction performance. It also analyzes whether it can reduce the number of software metrics used. To answer this question, RMS-DP is compared with a baseline method. The baseline directly uses all available metrics without performing metric selection. For each project, the defect prediction model is trained using the selected metric subset generated by RMS-DP. It is compared with the model trained using the complete set of metrics. The comparison focuses on MCC and Recall. These are suitable evaluation metrics for imbalanced defect prediction tasks. Table 2 presents the results of RMS-DP and the non-metric-selection baseline across the seven projects. From Table 2, it can be observed that RMS-DP improves the overall prediction performance across most projects. The average MCC increases from 0.31 to 0.35. The average Recall improves from 0.58 to 0.63. This indicates that the reinforcement-learning-based metric selection strategy helps the model identify defect-prone modules more effectively. Another important observation is that RMS-DP significantly reduces the number of metrics used for prediction. The baseline method uses all 65 metrics. RMS-DP typically selects only 4 to 10 metrics for most projects. This shows that RMS-DP can identify compact and informative metric subsets while maintaining or improving prediction performance. Overall, these results demonstrate that reinforcement-learning-based metric selection is effective for defect prediction, and it improves prediction quality while simultaneously reducing feature dimensionality.

Table 2: Performance comparison of RMS-DP with and without metric selection.

Project	With RMS-DP			Without RL Metric Selection		
	MCC	Recall	Metrics	MCC	Recall	Metrics
activemq-5.3.0	0.41	0.73	10	0.34	0.59	65
camel-1.4.0	0.43	0.60	5	0.36	0.55	65
derby-10.5.1.1	0.38	0.51	4	0.33	0.57	65
groovy-1_6_BETA_1	0.36	0.64	7	0.34	0.70	65
hbase-0.95.2	0.34	0.60	4	0.29	0.51	65
hive-0.12.0	0.30	0.73	6	0.26	0.63	65
wicket-1.5.3	0.23	0.58	7	0.22	0.53	65
Average	**0.35**	**0.63**	–	0.31	0.58	–

5.2 RQ2: How Does RMS-DP Perform When Used with Different Machine Learning Classifiers?

In the proposed RMS-DP framework, Logistic Regression (LR) is used as the primary defect prediction model due to its effectiveness and widespread use. To evaluate the robustness of the selected metric subsets, this research question examines their performance with additional classifiers. The subsets selected by RMS-DP are tested using k-Nearest Neighbors (KNN), Naive Bayes (NB), Random Forest (RF), and Support Vector Machine (SVM). The comparison focuses on MCC and Recall, enabling analysis of both overall classification quality and defect detection ability. Table 3 summarizes classifier-wise performance across seven projects. The results show that RMS-DP achieves competitive performance across all classifiers, indicating that the selected metric subsets are informative and robust. Among them, LR achieves the best average MCC (0.35) and Recall (0.63), with SVM as the second-best. KNN, NB, and RF produce slightly lower but comparable results. Overall, the findings confirm that RMS-DP generates effective metric subsets across models, while LR provides the most consistent and balanced performance, supporting its use as the final classifier.

5.3 RQ3: How Important Is Class-Imbalance Handling for Software Defect Prediction?

This research question examines the impact of class-imbalance handling on RMS-DP performance. Software defect datasets are typically highly imbalanced, with far fewer defective modules. Without proper handling, models tend to favor the majority class. To evaluate this, RMS-DP with SMOTE is compared to RMS-DP without SMOTE using MCC and Recall, as shown in Table 4 across seven projects. Results show a strong impact of imbalance handling. With SMOTE, average MCC increases from 0.22 to 0.35, and Recall from 0.30 to 0.63. This indicates a substantial improvement in detecting defective modules. The effect is particularly evident in Recall: for example, in hive-0.12.0, Recall rises from 0.12 to 0.73, and in wicket-1.5.3, from 0.08 to 0.58. Without SMOTE, many defects are missed. Overall, class-imbalance handling is essential, and integrating SMOTE in RMS-DP significantly enhances prediction reliability and defect detection.

Table 3: Performance of RMS-DP with different machine learning classifiers.

Project	RMS-DP-LR		RMS-DP-KNN		RMS-DP-NB		RMS-DP-RF		RMS-DP-SVM	
	MCC	Recall	MCC	Recall	MCC	Recall	MCC	Recall	MCC	Recall
activemq-5.3.0	0.41	0.73	0.33	0.73	0.38	0.58	0.42	0.63	0.36	0.64
camel-1.4.0	0.43	0.60	0.34	0.60	0.40	0.40	0.34	0.59	0.41	0.60
derby-10.5.1.1	0.38	0.51	0.34	0.58	0.37	0.46	0.29	0.54	0.36	0.54
groovy-1_6_BETA_1	0.36	0.64	0.32	0.70	0.31	0.58	0.36	0.63	0.36	0.66
hbase-0.95.2	0.34	0.60	0.28	0.48	0.37	0.47	0.31	0.64	0.33	0.56
hive-0.12.0	0.30	0.73	0.32	0.60	0.23	0.90	0.32	0.52	0.29	0.71
wicket-1.5.3	0.23	0.58	0.22	0.43	0.18	0.58	0.19	0.37	0.20	0.56
Average	**0.35**	**0.63**	0.31	0.59	0.32	0.57	0.32	0.56	0.33	0.61

Table 4: Performance comparison of RMS-DP with and without SMOTE.

Project	With SMOTE		Without SMOTE	
	MCC	Recall	MCC	Recall
activemq-5.3.0	0.41	0.73	0.29	0.39
camel-1.4.0	0.43	0.60	0.34	0.44
derby-10.5.1.1	0.38	0.51	0.29	0.38
groovy-1_6_BETA_1	0.36	0.64	0.19	0.21
hbase-0.95.2	0.34	0.60	0.28	0.45
hive-0.12.0	0.30	0.73	0.11	0.12
wicket-1.5.3	0.23	0.58	0.03	0.08
Average	**0.35**	**0.63**	0.22	0.30

6 THREATS TO VALIDITY

Internal Validity: Internal validity addresses biases in training and evaluation. Randomness from reinforcement learning and data partitioning may influence results. To reduce this, five-fold cross-validation is repeated ten times with different seeds. Metric selection, model training, and SMOTE-based imbalance handling are applied only to training data.

External Validity: External validity concerns generalizability. The study evaluates seven JIRA projects in a within-project setting. While widely used, validation on more diverse projects would further strengthen the general applicability of RMS-DP.

Construct Validity: Construct validity examines whether evaluation metrics reflect defect prediction performance. MCC and Recall are used as they suit imbalanced classification, capturing both balanced prediction quality and defect detection ability.

7 CONCLUSION

This paper presents Reinforced Metric Selection for Software Defect Prediction (RMS-DP), a reinforcement-learning framework that models metric selection as a sequential decision process. It uses a Deep Q-Network (DQN) to learn an adaptive policy for selecting compact, informative metric subsets, improving prediction performance while reducing dimensionality. Evaluation on seven JIRA projects shows that RMS-DP outperforms a no-selection baseline while using significantly fewer metrics. The selected subsets remain effective across classifiers, with Logistic Regression providing the most stable results. The results also highlight the importance of handling class imbalance, as SMOTE preprocessing improves defect detection. Overall, reinforcement-learning-based metric selection effectively identifies useful metrics and enhances prediction reliability. Future work will explore advanced reinforcement learning methods to improve selection efficiency. The framework can be extended to cross-project and heterogeneous defect prediction, and incorporating semantic code representations and process metrics may further improve effectiveness.

REFERENCES

Afsar, M. M., Crump, T., and Far, B. (2022). Reinforcement learning based recommender systems: A survey. *ACM Computing Surveys*, 55(7):1–38.

Bai, J., Jia, J., and Capretz, L. F. (2022). A three-stage transfer learning framework for multi-source cross-project software defect prediction. *Information and Software Technology*, 150:106985.

Bal, P. R. and Kumar, S. (2025). Cross project defect prediction using dropout regularized deep learning and

unique matched metrics. *ACM Transactions on Management Information Systems*, 16(3):1–32.

Bal, P. R., Shukla, S., and Kumar, S. (2025). An approach to software defect prediction for small-sized datasets: Pr bal et al. *Applied Intelligence*, 55(7):560.

Bhandari, K., Kumar, K., and Sangal, A. L. (2023). Data quality issues in software fault prediction: a systematic literature review. *Artificial Intelligence Review*, 56(8):7839–7908.

Bhutamapuram, U. S. (2023). Some investigations of machine learning models for software defects. In *2023 IEEE/ACM 45th International Conference on Software Engineering: Companion Proceedings (ICSE-Companion)*, pages 259–263. IEEE.

Chawla, N. V., Bowyer, K. W., Hall, L. O., and Kegelmeyer, W. P. (2002). Smote: synthetic minority oversampling technique. *Journal of artificial intelligence research*, 16:321–357.

Chen, J.-C. and Huang, S.-J. (2009). An empirical analysis of the impact of software development problem factors on software maintainability. *Journal of Systems and Software*, 82(6):981–992.

Fan, J., Wang, Z., Xie, Y., and Yang, Z. (2020). A theoretical analysis of deep q-learning. In *Learning for dynamics and control*, pages 486–489. PMLR.

Gong, L., Jiang, S., Bo, L., Jiang, L., and Qian, J. (2019). A novel class-imbalance learning approach for both within-project and cross-project defect prediction. *IEEE Transactions on Reliability*, 69(1):40–54.

Hosseini, S., Turhan, B., and Gunarathna, D. (2017). A systematic literature review and meta-analysis on cross project defect prediction. *IEEE Transactions on Software Engineering*, 45(2):111–147.

Jiang, S., Chen, Y., He, Z., Shang, Y., and Ma, L. (2024). Cross-project defect prediction via semantic and syntactic encoding. *Empirical Software Engineering*, 29(4):80.

Jureczko, M. (2011). Significance of different software metrics in defect prediction. *Software Engineering: An International Journal*, 1(1):86–95.

Khatri, Y. and Singh, S. K. (2022). Cross project defect prediction: a comprehensive survey with its swot analysis. *Innovations in Systems and Software Engineering*, 18(2):263–281.

Liu, X., Zhou, Y., Lu, Z., Mei, Y., Yang, Y., Qian, J., and Zhou, Y. (2024). Unveiling the impact of unchanged modules across versions on the evaluation of within-project defect prediction models. *Journal of Software: Evolution and Process*, 36(12):e2715.

Madeyski, L. and Jureczko, M. (2015). Which process metrics can significantly improve defect prediction models? an empirical study. *Software Quality Journal*, 23(3):393–422.

Pan, C., Lu, M., Xu, B., and Gao, H. (2019). An improved cnn model for within-project software defect prediction. *Applied Sciences*, 9(10):2138.

Singh, D. and Kumar, S. (2026). An approach for cross-project defect prediction using composite distribution alignment and domain-adversarial neural networks. *IEEE Transactions on Reliability*, 75:804–818.

Thota, M. K., Shajin, F. H., Rajesh, P., et al. (2020). Survey on software defect prediction techniques. *International Journal of Applied Science and Engineering*, 17(4):331–344.

Wang, Y., Wang, R., Sun, J., Deng, F., Wang, G., and Chen, J. (2025). Attention enhanced reinforcement learning for flexible job shop scheduling with transportation constraints. *Expert Systems with Applications*, 282:127671.

Yao, J. and Shepperd, M. (2021). The impact of using biased performance metrics on software defect prediction research. *Information and Software Technology*, 139:106664.

Yatish, S., Jiarpakdee, J., Thongtanunam, P., and Tantithamthavorn, C. (2019). Mining software defects: Should we consider affected releases? In *2019 IEEE/ACM 41st international conference on software engineering (ICSE)*, pages 654–665. IEEE.