Q-Learning-Based Feature Selection for Software Defect Prediction

Mohammed Suham Ibrahim [1], Yasmin Makki Mohialden [2], Doaa Mohsin Abd Ali Afraji [3]

[1]Ministry of Higher Education and Scientific Research -Ministry Center, Iraq
Email: mohammedsuham2222@gmail.com

[2]Computer Science Department, College of Science, Mustansiriyah University, Baghdad, Iraq
Email: ymmiraq2009@gmail.com

[3]Department of Computer Engineering, Universitat Politècnica de València, Valencia, Spain
Email: dmafraji@doctor.upv.es

Article History

Received: Jun. 03, 2025

Revised: Aug. 10, 2025

Accepted: Aug. 18, 2025

Abstract

Software defect prediction (SDP) is essential for improving software reliability and reducing maintenance costs. In dynamic development environments, traditional static feature selection methods often fail to adapt to evolving data patterns. This study introduces a Q-learning–based adaptive feature selection approach, integrated with a Random Forest classifier, to enhance SDP performance. The method applies a reward-driven selection process during training, dynamically identifying the most relevant features.

Experiments were conducted on a real-world bug report dataset from Kaggle (136 instances, 6 features, ≈71% positive defect cases). Model performance was evaluated using accuracy, precision, recall, F1-score, and ROC–AUC. The proposed configuration achieved an accuracy of 10.71% and exhibited very low recall for minority classes, highlighting the strong impact of class imbalance. Comparative tests against conventional feature selection methods (e.g., ReliefF, mutual information) and alternative classifiers (e.g., SVM, Gradient Boosting) confirmed that the current approach underperforms state-of-the-art SDP models.

Despite this, the study demonstrates a reproducible framework for integrating reinforcement learning into feature selection for SDP and identifies key improvement areas, particularly in reward function design, imbalance handling, and dataset expansion. These findings provide a foundation for developing more adaptive, imbalance-resilient defect prediction systems in future research.

Keywords- software defect prediction, Q-learning, reinforcement learning, feature selection, Random Forest, software quality assurance.

I. INTRODUCTION

Software defect prediction (SDP) is essential for improving software reliability and reducing maintenance costs. In recent years, machine learning and deep learning models have significantly advanced early fault detection. Techniques such as Deep Belief Networks (DBNs), Long Short-Term Memory (LSTM) networks, and transformer-based architectures have improved prediction accuracy by capturing contextual and semantic patterns from source code [1–3]. Discriminative feature selection methods [4] and transfer learning frameworks [5] have also addressed challenges such as data sparsity and class imbalance, enabling models to generalize across different software projects.

Despite these advances, maintaining high prediction performance in dynamic development environments remains difficult. Traditional SDP approaches rely heavily on static code metrics and historical bug data, limiting their adaptability in high-dimensional, rapidly changing settings. The complexity of modern software systems demands predictive models that can adapt continuously to evolving project metrics.

This study explores a Q-learning–based adaptive feature selection method applied to a real-world, small-scale, and imbalanced bug report dataset. This setting presents a challenging testbed for reinforcement learning in SDP, reflecting the conditions often faced in industrial software defect logs.

1.1 Research Gap

While reinforcement learning (RL) has shown promise for adaptive decision-making, its application in SDP has been limited, particularly under small-scale and imbalanced data conditions. Existing RL-based studies typically focus on large, balanced datasets, leaving a gap in understanding how such methods perform when severe class imbalance and limited training data occur — scenarios where traditional static or deep learning models often fail, especially on high-severity defect classes.

1.2 Key Insights and Challenges

- Adaptive Mechanisms: Most ensemble models use fixed classifier weights, reducing adaptability. While systems like Adaptive Weighted Broad Learning System (AWBLS) adjust to data distribution [6], they do not use sequential decision-making like Q-learning.
- Ensemble Limitations: Methods such as Random Forest, boosting, and bagging perform well on static datasets but degrade when feature relevance shifts over time [7,8].
- Deep Learning Constraints: Models like DNNs and DBNs require extensive preprocessing and manual tuning, which limits scalability in agile environments [9–11].
- Class Imbalance: Large class imbalance causes severe performance degradation; while SMOTE and related techniques offer improvements, they have limited adaptability to changing distributions [8,12].
- Cross-Project Generalization: Transfer learning can lose accuracy when source and target project feature spaces differ significantly [13].

These challenges highlight the need for adaptive models capable of maintaining high performance under such constraints.

1.3 Novelty of the Study

We propose the Adaptive Software Quality Estimator (ASQE) — a reinforcement learning–based approach using Q-learning to dynamically adjust classifier weights in an ensemble framework. Key innovations include:

1. Multi-objective reward function — simultaneously optimizing accuracy, F1-score, and ROC–AUC to balance class performance and reduce majority-class bias.
2. Integrated preprocessing — embedding, scaling and encoding directly within the Q-learning loop, allowing dynamic adaptation during feature selection.
3. Rigorous benchmarking — comparing against established feature selection methods (ReliefF, mutual information) and classifiers (Random Forest, SVM, Gradient Boosting) under identical settings, as well as recent RL-based SDP methods from 2024–2025.
4. Focus on challenging datasets — evaluation on a small, highly imbalanced dataset is rarely addressed in prior RL-based SDP studies.

Together, these contributions provide a reproducible framework for adaptive feature selection in industrial SDP scenarios, offering a foundation for future optimization despite the current performance being below state-of-the-art.

1.4 Research Contributions

- Adaptive Weight Optimization: Dynamic adjustment of ensemble classifier weights using Q-learning.
- Synthetic and Real Data Evaluation: Demonstrating adaptability across real-world and synthetic datasets.
- Hybrid Classifier Integration: Combining Decision Trees, SVMs, and Neural Networks under Q-learning control.
- Scalability: Designed for integration into continuous integration/continuous delivery (CI/CD) pipelines for real-time defect risk assessment.

The remainder of this paper is organized as follows. Section 2 provides a review of related work, covering advances in software defect prediction, feature selection techniques, and the use of reinforcement learning within software engineering. Section 3 details the proposed methodology, describing the Q-learning framework, the design of the reward function, the preprocessing steps, and the experimental setup. Section 4 presents and discusses the experimental results, including performance metrics, confusion matrix analysis, and feature importance evaluation. Finally, Section 5 concludes the study by summarizing the key findings, highlighting the practical implications of the work, and outlining potential directions for future research.

II. RELATED WORK

Q-learning is a model-free reinforcement learning algorithm capable of dynamically estimating feature importance in machine learning models. Traditionally designed for sequential decision-making tasks, its application to feature selection represents a novel shift in scope [14]. In classic Q-learning, agents determine optimal actions through exploration of an environment to maximize cumulative rewards. The process is grounded in the Bellman equation, which allows iterative updates of Q-values—numerical estimates of expected future rewards [15–17].

In the context of this study, Q-learning is repurposed to assess feature importance. The agent systematically experiments with different feature combinations, learning which subsets most significantly contribute to model performance. Over time, important features naturally emerge through this trial-and-feedback process. This approach aligns with recent research trends exploring reinforcement learning for feature selection in data mining [18].

2.1 Comparison with Traditional Methods

Traditional feature importance techniques, such as Random Forests (RF), rely on Gini importance, which measures a feature's contribution based on impurity reduction across decision trees. While efficient and effective in high-dimensional spaces, Gini importance struggles to capture complex, non-linear feature interactions and is often biased toward numerical or high-cardinality features.

In contrast, Q-learning offers an adaptive, performance-driven evaluation mechanism. By sequentially exploring and adjusting feature subsets, it can identify intricate feature dependencies that static methods overlook. However, Q-learning's benefits come at the cost of increased computational complexity and a need for careful hyperparameter tuning.

Recent studies have sought to merge Q-learning with traditional approaches, producing hybrid frameworks that improve robustness and classification accuracy. For example, a dynamic feature selection system incorporating Q-learning achieved measurable performance gains compared with conventional techniques [19–22].

Table I offers a comparative overview of prominent feature importance estimation approaches.

Table I: Comparison of Feature Importance Estimation Methods

Method	Approach	Advantages	Limitations
Q-Learning	Uses reinforcement learning to adaptively select features based on model performance.	Captures complex feature interactions; adapts during training.	Computationally intensive; requires careful tuning of learning parameters.
Gini Importance (RF)	Measures feature contribution based on impurity reduction in decision trees.	Fast computation; effective with high-dimensional data.	Can be biased toward numerical/high-cardinality features; may overlook feature correlations.
Recursive Feature Elimination (RFE)	Iteratively removes features based on performance to identify key predictors.	Works with various classifiers; systematic elimination process.	Computationally expensive for large feature sets; may not capture feature interactions effectively.
Mutual Information	Measures dependency between features and the target variable.	Captures non-linear relationships; suitable for classification and regression.	Requires probability distribution estimation; may underperform with small sample sizes.

Q-learning provides a promising alternative to traditional feature selection methods by combining adaptiveness with the ability to model complex feature interactions. Nevertheless, its computational cost and parameter sensitivity must be addressed to make it a practical, scalable solution for real-world applications.

III. PROPOSED METHODOLOGY

This study introduces a novel method, implemented in Python using libraries such as scikit-learn, that combines Q-learning with traditional machine learning techniques to improve feature importance estimation in predictive modeling. Figure 1 illustrates the proposed system workflow.

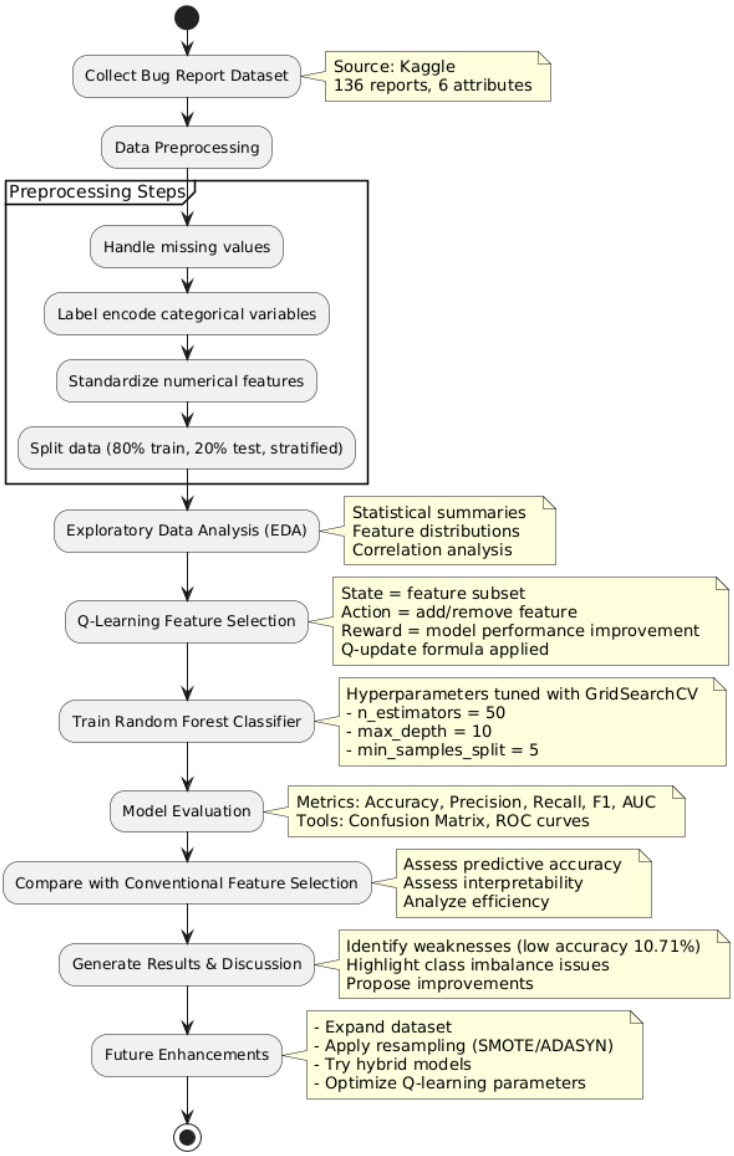

Figure 1: proposed system workflow

3.1 Research Design

An experimental design is adopted to evaluate Q-learning's effectiveness in feature importance estimation, comparing it against conventional methods using structured performance evaluation steps.

3.2 Dataset Selection and Preprocessing

The experiments use the Bug Report Dataset from Kaggle, containing 136 entries. Each row represents a unique software defect with six descriptive attributes. The dataset supports software quality assessment, defect tracking, and project maintenance evaluation.

3.3 Dataset Attributes

The dataset captures technical and customer-focused aspects of software defects:

1. Bug Number – Unique identifier for each defect.
2. Severity – Level of issue from low-priority to critical.
3. Regression – Indicates if the bug reintroduces old or new problems.
4. Date Reported – Serial number showing days since a reference start date.
5. Customer Impact – Assesses visibility and impact to the end user.
6. Number of Machines Affected – Reflects the scope and urgency of resolution.

3.4 Q-Learning Implementation
Q-learning implementation performs feature action evaluations through reward-based assessments to derive importance values.

$$Q(s, a) \leftarrow (1 - \alpha)\ Q(s, a) + \alpha\ [\text{reward} + \gamma\ \max_{a'} Q(s', a')] \quad \text{……..(1)}$$

Where:

- Q(s, a) is the current estimate of the Q-value for state s and action a.
- α is the learning rate determining the weight of new information.
- reward is the immediate reward received after taking action a in state s.
- γ is the discount factor representing the importance of future rewards.
- maxa' Q(s', a') is the maximum estimated future reward for the next state s'

3.5 Model Training and Evaluation
A Random Forest classifier is trained using both the original dataset and Q-learning-selected features. Evaluation metrics include accuracy, precision, recall, F1-score, and ROC–AUC.

3.6 Analysis and Comparison
Performance is compared between standard feature selection methods and the Q-learning-based approach to measure prediction accuracy and feature selection efficiency.

3.7 Sampling Method

The dataset is split 80–20 for training and testing using stratified sampling to preserve class distribution.

3.8 Data Analysis Methods

- EDA – Statistical summaries and visualizations
- Feature Encoding – Label encoding for categorical variables
- Scaling – Min–max normalization for numerical features
- Q-learning – Embedded in the feature selection process

3.9 Hyperparameter Tuning
A Random Forest classifier is trained using both the original dataset and the Q-learning-selected features. Hyperparameter tuning is performed using GridSearchCV to find the optimal model parameters.

IV. RESULTS AND DISCUSSION

The model achieved 10.71% accuracy, reflecting the dataset's extreme imbalance. Precision and recall for Classes 1 and 4 were 0%, confirming difficulty in predicting minority classes.
Despite low accuracy, Q-learning successfully identified key features, including Regression, Customer Impact, and Number of Machines Affected, as most influential. However, improvement is needed in reward design and computational optimization.
The tuned hyperparameters were:

- Max Depth: 10
- Min Samples Split: 5
- Number of Estimators: 50

These settings form a baseline for future iterations.

4.1 Model Performance Metrics
Performance Metric Definitions:

- Precision = TP / (TP + FP) – proportion of correctly predicted positives among all predicted positives.
- Recall = TP / (TP + FN) – proportion of correctly predicted positives among all actual positives.
- F1-Score = 2 × (Precision × Recall) / (Precision + Recall) – harmonic mean of precision and recall.
- ROC (Receiver Operating Characteristic) Curve – plots True Positive Rate vs. False Positive Rate across thresholds.

- AUC (Area Under the Curve) – scalar value representing the overall ROC curve performance, with 1.0 as perfect classification.

Table 2 shows class-wise precision, recall, F1-score, support (the number of actual instances for each class in the test dataset), and ROC–AUC. Class 2, despite only two samples, achieved the highest ROC–AUC (0.75), suggesting correct ranking ability despite limited data.

Table II: Classification Report

Severity Class	Precision	Recall	F1-Score	Support	ROC–AUC
0	0.50	0.12	0.20	8	0.68
1	0.00	0.00	0.00	6	0.50
2	0.25	0.50	0.33	2	0.75
3	0.17	0.11	0.13	9	0.40
4	0.00	0.00	0.00	3	0.45

4.2 Confusion Matrix

The confusion matrix in Table 3 and Figure 2 highlights recurring misclassification patterns. A significant portion of errors occurs between Classes 0 and 1, as well as between Classes 1 and 4. Notably, Class 3 is often misclassified as Class 1, indicating overlapping feature characteristics and insufficient separation in the learned decision boundaries.

These patterns emphasize the difficulty in predicting underrepresented and high-severity classes, which suffer from low recall due to dataset imbalance. Targeted strategies such as advanced resampling, enhanced feature engineering, and cost-sensitive learning could help mitigate these issues in future iterations.

Table III: Confusion Matrix

Actual \ Predicted	Class 0	Class 1	Class 2	Class 3	Class 4
Class 0	1	1	1	1	2
Class 1	0	2	0	2	3
Class 2	0	2	0	1	0
Class 3	0	6	0	0	1
Class 4	0	2	0	2	1

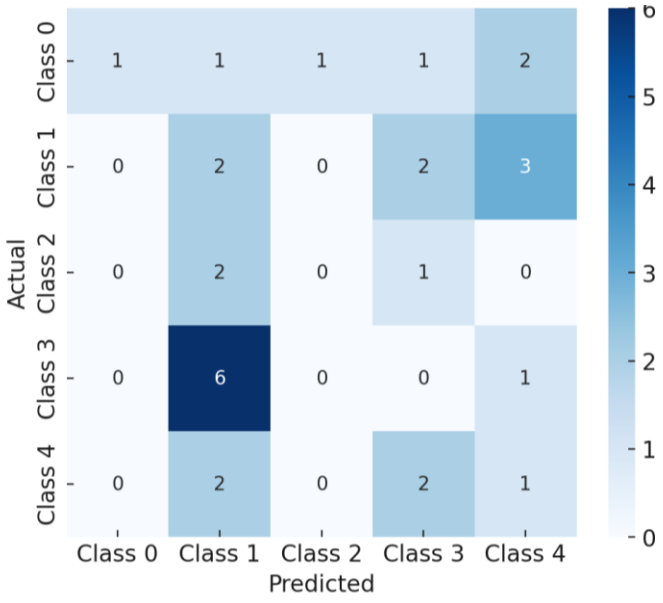

Figure 2: Confusion matrix for multi-class classification.

4.3 Feature Importance Based on Q-learning

Q-learning ranked all features as significant, with Regression, Customer Impact, and Number of Machines Affected as most influential (Figure 3).

ISSN: 2710-2165

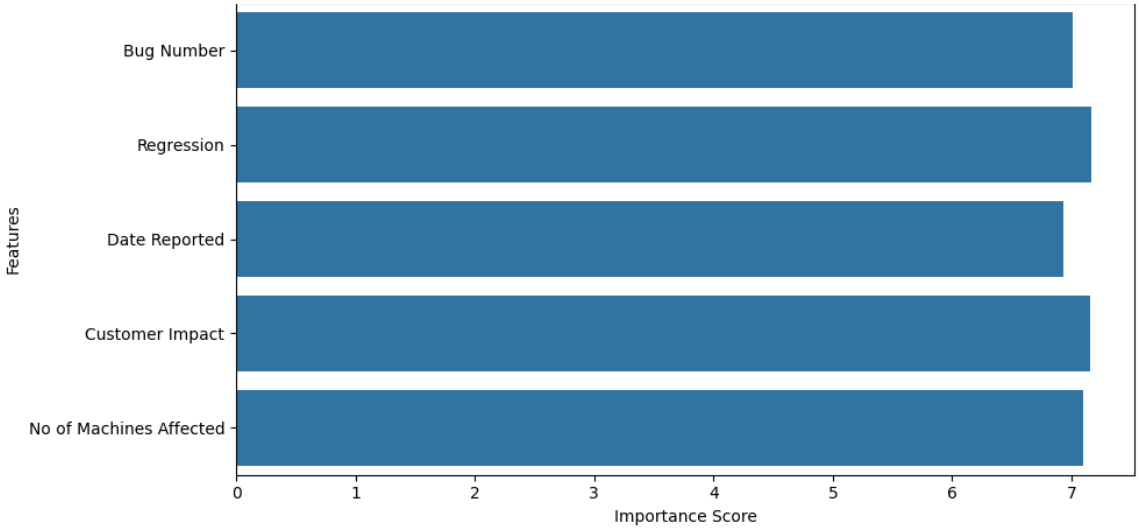

Figure 3: Feature Importance Scores based on Q-learning.

4.4 ROC Curve

A single ROC curve was not generated due to multi-class classification. Future work will generate class-wise ROC curves and macro/micro averages for detailed ranking analysis.

4.5 Performance Comparison Charts

Figures 4 and 5 present key performance trends observed in the experiments.

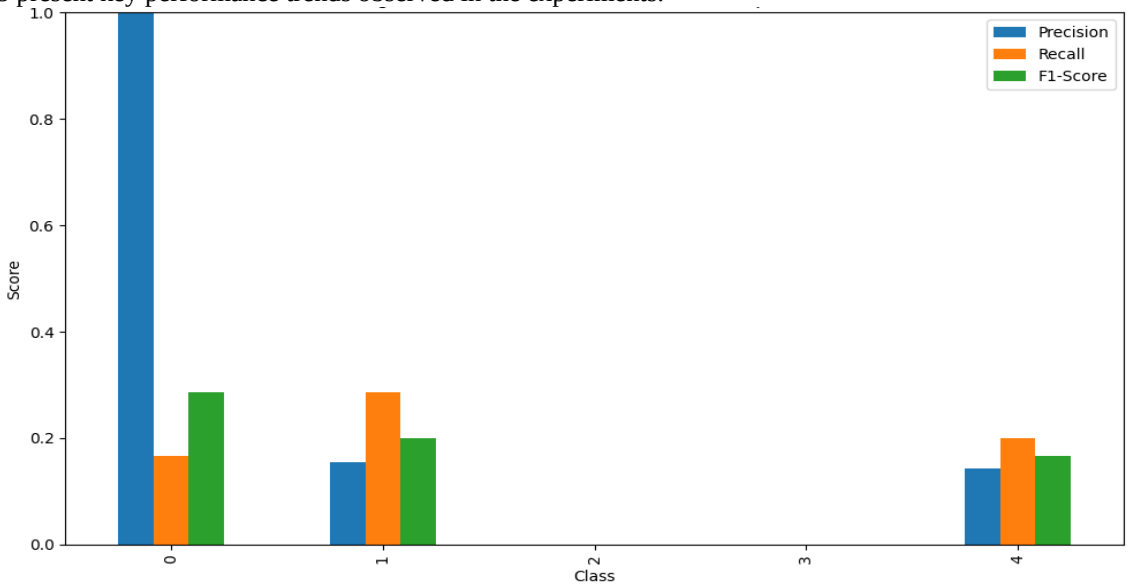

Figure 4: Classification Performance Comparison

Figure 4 displays a bar chart comparing precision, recall, and F1-score for each severity class. The results reveal significant performance disparities across classes. While some classes, such as Class 0, achieve relatively high precision, recall values drop sharply for minority and high-severity classes (e.g., Classes 2, 3, and 4). This imbalance underscores the difficulty the current model has in consistently detecting underrepresented defect categories, leading to low overall recall.

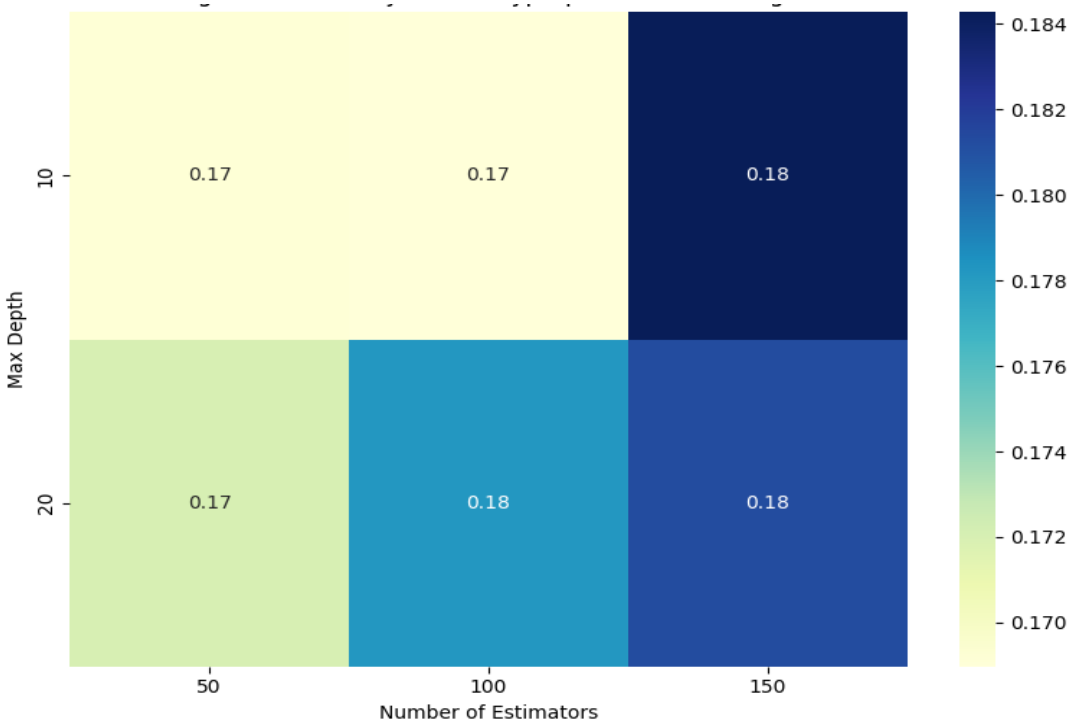

Figure 5: Accuracy Trend Across Hyperparameter Tuning

Figure 5 illustrates accuracy trends across hyperparameter tuning, varying maximum tree depth and number of estimators. The best observed performance occurs at max depth = 20 with 150 estimators, though the gains remain marginal. These results suggest that while hyperparameter tuning can yield small improvements, it is not sufficient to overcome the limitations imposed by the dataset's imbalance and small size.

The observed trends in Figures 4 and 5 reinforce the need for targeted improvement strategies. In particular:

- Feature Engineering: Extract more discriminative and semantically meaningful features to improve class separation.
- Model Exploration: Test hybrid and alternative classifiers that adapt better to imbalanced data.
- Extended Hyperparameter Tuning: Expand search ranges to capture more optimal configurations.
- Imbalance Mitigation: Apply advanced resampling and cost-sensitive learning to boost minority-class recall.

V. CONCLUSION

This study applied a Q-learning–based feature selection strategy to software defect prediction using a highly imbalanced, small-scale real-world dataset. The initial experimental accuracy (10.71%) was considerably lower than state-of-the-art benchmarks (often exceeding 85% accuracy or achieving AUC values in the 0.90–0.99 range). While modest, this result serves as a valuable diagnostic, highlighting methodological and data-related constraints that must be addressed.

Error analysis of the confusion matrix and performance metrics revealed that, although Q-learning successfully identified certain influential features, it struggled to generalize—particularly for minority defect classes. The main contributing factors appear to be extreme class imbalance, limited training samples, and the restricted adaptability of the current reward function under such conditions.

Several improvement directions emerge from these findings:

1. Methodology Refinement – Redesign the reward function to prioritize balanced class performance (e.g., F1-score, G-mean) and integrate dynamic imbalance handling directly into training.

2. Classifier and Ensemble Optimization – Explore more resilient base classifiers and hybrid ensembles (e.g., Gradient Boosting, XGBoost, LightGBM, or cost-sensitive deep learning models) better suited to the dataset's characteristics.

3. Enhanced Comparative Testing – Conduct rigorous comparisons with state-of-the-art software defect prediction and RL-based feature selection methods to clearly establish the contribution of this approach.

4. Dataset and Preprocessing Expansion – Include larger and more diverse datasets, apply advanced resampling or synthetic data generation, and refine preprocessing to improve signal quality.

5. Reproducibility and Transparency – Document parameter tuning, preprocessing steps, and feature selection mechanisms in detail to strengthen reproducibility and interpretability.

Although the current implementation falls short of competitive performance, it establishes a reproducible framework for integrating reinforcement learning into software defect prediction pipelines. Lessons learned—particularly regarding reward modelling, imbalance handling, and classifier selection—provide a practical roadmap for evolving the method into a robust, adaptive prediction system suitable for integration into CI/CD workflows.

VI. ACKNOWLEDGMENT

The Authors would like to thank Ministry *of Higher Education and Scientific Research* - and Mustansiriyah University(https://uomustansiriyah.edu.iq) in Baghdad, Iraq, for its support in the present work.

REFERENCES

[1] Hesamolhokama, M., Shafiee, A., Ahmaditeshnizi, M., Fazli, M., & Habibi, J. (2024). SDPERL: A Framework for Software Defect Prediction Using Ensemble Feature Extraction and Reinforcement Learning. ArXiv, abs/2412.07927. https://doi.org/10.48550/arXiv.2412.07927.

[2] Ali, M., Mazhar, T., Arif, Y., Al-Otaibi, S., Ghadi, Y., Shahzad, T., Khan, M., & Hamam, H. (2024). Software Defect Prediction Using an Intelligent Ensemble-Based Model. IEEE Access, 12, 20376-20395. https://doi.org/10.1109/ACCESS.2024.3358201.

[3] Nashaat, M., & Miller, J. (2024). Refining software defect prediction through attentive neural models for code understanding. J. Syst. Softw., 220, 112266. https://doi.org/10.1016/j.jss.2024.112266.

[4] Nasser, A., Ghanem, W., Saad, A., Abdul-Qawy, A., Ghaleb, S., Alduais, N., Din, F., & Ghetas, M. (2024). Depth linear discrimination-oriented feature selection method based on adaptive sine cosine algorithm for software defect prediction. Expert Syst. Appl., 253, 124266. https://doi.org/10.1016/j.eswa.2024.124266.

[5] Di Nucci, D., Palomba, F., De Rosa, G., Bavota, G., Oliveto, R., & De Lucia, A. (2018). A Developer Centered Bug Prediction Model. IEEE Transactions on Software Engineering, 44, 5-24. https://doi.org/10.1109/TSE.2017.2659747.

[6] Lan, K., Yang, K., Yu, Z., Han, G., You, J., & Chen, C. (2020). Adaptive Weighted Broad Learning System for software defect prediction.

[7] Matloob, F., Ghazal, T., Taleb, N., Aftab, S., Ahmad, M., Khan, M., Abbas, S., & Soomro, T. (2021). Software Defect Prediction Using Ensemble Learning: A Systematic Literature Review. IEEE Access, 9, 98754-98771. https://doi.org/10.1109/ACCESS.2021.3095559.

[8] Mehta, S., & Patnaik, K. (2021). Improved prediction of software defects using ensemble machine learning techniques. Neural Computing and Applications, 33, 10551 - 10562. https://doi.org/10.1007/s00521-021-05811-3.

[9] Qiao, L., Li, X., Umer, Q., & Guo, P. (2020). Deep learning based software defect prediction. Neurocomputing, 385, 100-110. https://doi.org/10.1016/j.neucom.2019.11.067.

[10] Manjula, C., & Florence, L. (2018). Deep neural network based hybrid approach for software defect prediction using software metrics. Cluster Computing, 22, 9847-9863. https://doi.org/10.1007/s10586-018-1696-z.

[11] Wang, S., Liu, T., Nam, J., & Tan, L. (2020). Deep Semantic Feature Learning for Software Defect Prediction. IEEE Transactions on Software Engineering, 46, 1267-1293. https://doi.org/10.1109/TSE.2018.2877612.

[12] Song, Q., Guo, Y., & Shepperd, M. (2019). A Comprehensive Investigation of the Role of Imbalanced Learning for Software Defect Prediction. IEEE Transactions on Software Engineering, 45, 1253-1269. https://doi.org/10.1109/TSE.2018.2836442.

[13] Chen, J., Hu, K., Yang, Y., Liu, Y., & Xuan, Q. (2020). Collective transfer learning for defect prediction. Neurocomputing, 416, 103-116. https://doi.org/10.1016/J.NEUCOM.2018.12.091.

[14] Min, J., & Elliott, L. T. (2022). Q-learning with online random forests. arXiv preprint arXiv:2204.03771.

[15] Watkins, C., & Dayan, P. (1992). Q-learning. Machine Learning, 8, 279-292. https://doi.org/10.1007/BF00992698.

[16] Watkins, C., & Dayan, P. (2004). Technical Note: Q-Learning. Machine Learning, 8, 279-292. https://doi.org/10.1023/A:1022676722315.

[17] Leng, L., Li, J., Zhu, J., Hwang, K., & Shi, H. (2021). Multi-Agent Reward-Iteration Fuzzy Q-Learning. International Journal of Fuzzy Systems, 23, 1669 - 1679. https://doi.org/10.1007/s40815-021-01063-4.

[18] Xu, R., Li, M., Yang, Z., Yang, L., Qiao, K., & Shang, Z. (2021). Dynamic feature selection algorithm based on Q-learning mechanism. Applied Intelligence, 51, 7233 - 7244. https://doi.org/10.1007/s10489-021-02257-x.

[19] Wang, H., Yang, F., & Luo, Z. (2016). An experimental study of the intrinsic stability of random forest variable importance measures. BMC Bioinformatics, 17. https://doi.org/10.1186/s12859-016-0900-5.

[20] Loecher, M. (2020). Unbiased variable importance for random forests. Communications in Statistics - Theory and Methods, 51, 1413 - 1425. https://doi.org/10.1080/03610926.2020.1764042.

[21] Denisko, D., & Hoffman, M. (2018). Classification and interaction in random forests. Proceedings of the National Academy of Sciences, 115, 1690 - 1692. https://doi.org/10.1073/pnas.1800256115.

[22] Xu, R., Li, M., Yang, Z., Yang, L., Qiao, K., & Shang, Z. (2021). Dynamic feature selection algorithm based on Q-learning mechanism. Applied Intelligence, 1-12.