International Journal of Software Science and Computational Intelligence
Volume 17 • Issue 1 • January-December 2025

Predicting Software Refactoring With a Reinforcement Learning Framework Using Deep Q Network

Archana Patnaik
Gandhi Institute of Engineering and Technology University, Gunupur, India

Sanjay Misra
https://orcid.org/0000-0002-3556-9331
Institute for Energy Technology, Halden, Norway

Neelamadhab Padhy
https://orcid.org/0000-0002-8512-3469
Gandhi Institute of Engineering and Technology University, Gunupur, India

Lov Kumar
National Institute of Technology, Kurukhetra, India

Rasmita Panigrahi
Gandhi Institute of Engineering and Technology University, India
Received: August 2nd, 2025 | **Accepted:** October 31st, 2025

ABSTRACT

The primary objective of this work is to identify the scope of requirements for existing software projects by considering the source code metrics. In this work, the authors used Deep Q Network (DQN) for refactoring prediction, for which three open-source real-time projects, Antlr4, Junit, and Oryx, are taken into consideration. Different code metrics like Cyclomatic Complexity, Lines of Code, Coupling between Objects Lack of Cohesion are taken as input parameters. Based on the experimental analysis, five different types of refactoring predictions are performed, and the requirement of Extract Method and Move Class is higher as compared to other techniques. JUnit framework identifies a high scope of refactoring for which the performance evaluation metrics values are with an accuracy 97%, recall of .96, f-measure of 0.96, precison of 0.96. Q-Learning can be considered as one of the best techniques for identifying the need for code alteration by considering the probable values that resemble the Q-Value used for analysing the maintainability index.

KEYWORDS

Refactoring, Source Code Metrics, Machine Learning, Reinforcement Learning

INTRODUCTION

Refactoring mechanism is a part of software development activities that promotes source code quality enhancement. Most commonly used code alteration techniques include: (a)Move class is used to handle complex code structure, which is segregated into small modules by using the function concept. (b) Rename variable is used to change the name of the variable, which is further updated in the code structure. (c)The inline method contributes toward providing a comprehensive and concise version of code by removing the original method declaration. (d)The move method creates a new method

DOI: 10.4018/IJSSCI.393452

by considering the properties of the existing class. (e) The extract method breaks down the set of codes into small subsections, assigns a separate method name and body. In order to enable long-term maintainability (Palit et al., 2024; Samsonov et al., 2022; Ksontini et al., 2025) and high code quality, it is mandatory for the software developer to alter the code multiple times by considering the test cases. It is observed that by merging reinforcement learning with supervised fine-tuning, quality evaluation can be performed more flexibly. Sometimes Dockerfiles are also considered as an input parameter for implementing context learning, which suggests automated practice for developers for evaluating the complex process. Metrics (Nikolaidis et al., 2024) acts as a guiding parameter for analyzing the negative and positive effects of refactoring in projects that as high coupling, suggests more need of code restructuring. Quality attributes are tested for MapDB, Antlr4, Junit, and McMMO open-source project using Support Vector Machine (SVM). In order to perform automatic (Akour et al., 2022; Prasad et al., 2025) code refactoring, a hybrid model is developed by using a combinational neural network for which the deep learning autoencoder concept is implemented. The Python-based (Noei et al., 2024) project also consists of complexity issues for which the Python-Ref tool is developed to identify the scope of improvement. Code quality (Martins et al., 2021) is directly affected by the complexity that occurs due to the presence of code smells, which can be detected by using different hybrid machine learning approaches.

Researchers proposed a RefactorBERT that acts as an evaluation metric for ensuring the performance of the model that refines the parameters by applying reinforcement learning. Automated code transformation (Jesse et al., 2023; Pinheiro et al., 2024) is performed on the Abstract Syntax Tree (AST) by using agent-based reinforcement learning. To improve the model by promoting trivial and non-trivial refactoring ensemble classifiers, Gradient Boosting and Random Forest are implemented. Software metrics act as an input parameter for code smell detection, which further performs refactoring (Sagar et al., 2021) prediction at class, package, and method levels. Most commonly found code smells are data class, feature envy, and long method, for which extract class, extract method, move method, and inline method restructuring can be implemented. Supervised (Agnihotri et al., 2020) machine learning approaches like Naïve Bayes Classification, Random Forest, Support Vector Machines, Logistic Regression and Decision Tree classifiers are used. For enhancing the effectiveness of the hybrid model, deep learning (Aniche et al., 2020; Alnezi et al., 2020) is harnessed by considering seven different open-source projects. Data sampling techniques like RUSBoost and SMOTE are implemented to deal with the imbalanced dataset. Code smell aims to identify the negative impact of complexity on source code, which is further divided into different categories like refusal bequest, large class, long method, god class, and duplicate code. By using (Patnaik et al., 2021) anomaly detection techniques, we can identify the presence of outliers that act as a key parameter for bad smell detection. The clustering approach provides better results for unlabeled data, which can be predicted by using K-Means algorithms.

Many hybrid models are designed based on the metrics that act as a base indicator for understanding the modern-driven approach used in quality assurance purposes. Ensemble machine learning algorithms are used for smell detection by considering object-oriented quality attributes. Manual (Fraihat et al., 2024; Dallal et al., 2024; Armijo et al., 2022) refactoring possesses the ability to deal with time complexity issues and error-prone activities, which can be identified by using different machine learning classifiers. To work (Pandiyavathi et al., 2024) on a deep learning refactoring prediction model, data is processed, followed by feature extraction for which Bi-LSTM and Deep Context Temporal Network (DCTN) are used. Technical debt is the prime cause of code complexity, which further leads to the presence of code smells (Tang et al., 2021; Khaleel et al., 2024; Sheneamer et al., 2020), which can be further removed by using automated software refactoring. For clone-based prediction, AST and PGD act as the main parameters improving the results of classifiers. To decrease risk of failure and increase (Mhawish et al., 2020) maintainability LIME algorithm is used for code smell detection. Some research proves that refactoring (Nyamawe et al., 2019) is based on a feature request that can be performed by using some automated tools like RefFinder, RefactoringCrawler, and