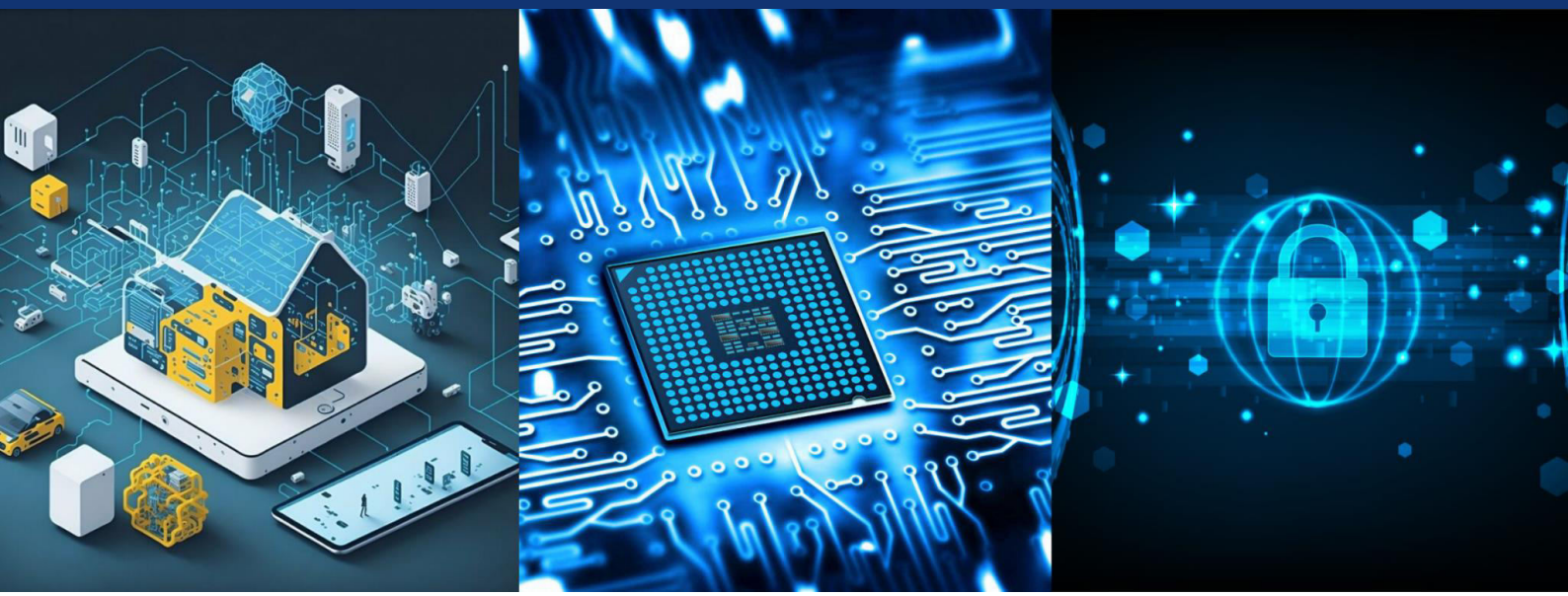
IJIRCCE

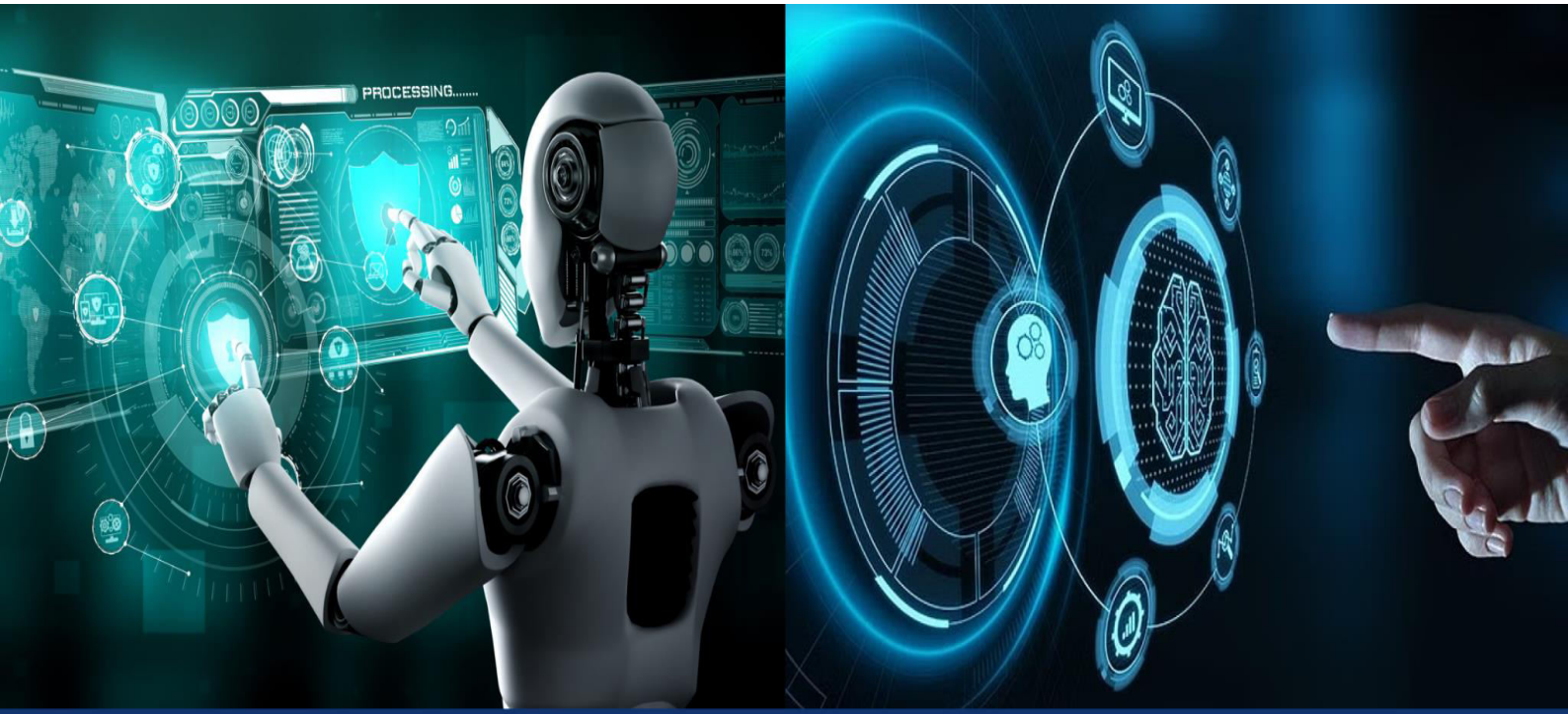
ISSN(O): 2320-9801

ISSN(P): 2320-9798



International Journal of Innovative Research in Computer and Communication Engineering

(A Monthly, Peer Reviewed, Refereed, Scholarly Indexed, Open Access Journal)



Impact Factor: 9.927

Volume 14, Issue 5, May 2026

© 2026 IJIRCCE | Volume 14, Issue 5, May 2026 DOI: 10.15680/IJIRCCE.2026.1405077

www.ijircce.com | e-ISSN: 2320-9801, p-ISSN: 2320-9798| Impact Factor: 9.927| ESTD Year: 2013|

International Journal of Innovative Research in Computer and Communication Engineering (IJIRCCE)

(A Monthly, Peer Reviewed, Refereed, Scholarly Indexed, Open Access Journal)

AI-Based Detection of Low-Quality Code Using Machine Learning Beyond Syntax-Level Analysis

Prof. Manjula P[1], Syeda Ayesha[2], Khushnida Hasansab Sayed[2]

Assistant Professor, Dept. of CS&E., Jain Institute of Technology, Davangere, Karnataka, India[1]

UG Student, Dept. of CS&E, Jain Institute of Technology, Davangere, Karnataka, India[2]

ABSTRACT: Traditional code analysis tools primarily focus on detecting syntax errors and simple rule violations, but they often fail to identify deeper issues related to code quality such as poor maintainability, high complexity, and bad design practices. This paper proposes a machine learning-based approach to automatically detect low-quality code by analysing structural, semantic, and behavioural features of source code. The proposed system extracts multiple features including code complexity metrics (e.g., cyclomatic complexity), code smells, naming conventions, duplication patterns, and maintainability indices. These features are used to train machine learning models such as Random Forest (RF), Support Vector Machines (SVM), and deep learning models to classify code into high-quality and low-quality categories. Unlike traditional static analysis tools, the proposed model learns patterns from real-world code datasets and provides predictive insights into code quality. The system is evaluated using publicly available datasets from open-source repositories, and performance is measured using accuracy, precision, recall, and F1-score. Random Forest achieved the highest classification accuracy of 94.32% with an AUC of 0.9861, outperforming both SVM (91.78%) and rule-based static analysis baselines. Experimental results demonstrate that the machine learning approach significantly improves the detection of low-quality code compared to rule-based methods, contributing to automated, intelligent code quality assessment that assists developers in improving code maintainability and reducing technical debt.

KEYWORDS: Code Quality Detection; Machine Learning; Code Smells; Cyclomatic Complexity; Random Forest; Support Vector Machine; Feature Extraction; Technical Debt; Static Analysis; Maintainability Index

I. INTRODUCTION

Software quality is a critical concern in modern software engineering, directly impacting the reliability, maintainability, and scalability of software systems. Low-quality code, characterized by high complexity, poor naming conventions, duplication, and design anti-patterns, leads to increased maintenance costs, higher defect rates, and accumulation of technical debt [1]. According to industry surveys, developers spend over 23% of their working time dealing with technical debt caused by poor code quality, costing the global software industry billions of dollars annually.

Existing approaches to code quality assessment rely heavily on static analysis tools such as SonarQube, PMD, and Checkstyle. While these tools effectively detect syntactic issues and simple rule violations, they are inherently rule-based and cannot learn from evolving code patterns. They lack the ability to evaluate subjective quality dimensions such as readability, logical coherence, and structural design quality [2]. Furthermore, rule-based tools suffer from high false-positive rates and require manual tuning, making them difficult to scale in large software projects.

Machine learning (ML) presents a compelling alternative: by training on large annotated code datasets, ML classifiers can learn contextual patterns associated with low-quality code that transcend simple rule violations. GitHub and Kaggle host extensive open-source repositories with thousands of labeled code files, providing rich training data for supervised learning. This work exploits such datasets to build a hybrid feature-based ML model for automated code quality prediction.

The specific original contributions of this work are:
(i) A hybrid feature extraction pipeline combining structural, semantic, and complexity-based metrics for comprehensive code quality assessment;
(ii) A systematic comparison of five ML algorithms — Random Forest, SVM, Decision Tree, Gradient Boosting, and a CNN-based deep learning model — on real-world open-source code datasets;

www.ijircce.com | e-ISSN: 2320-9801, p-ISSN: 2320-9798| Impact Factor: 9.927| ESTD Year: 2013|

International Journal of Innovative Research in Computer and Communication Engineering (IJIRCCE)

(A Monthly, Peer Reviewed, Refereed, Scholarly Indexed, Open Access Journal)

(iii) Comprehensive performance evaluation using accuracy, precision, recall, F1-score, and AUC-ROC, with direct comparison against traditional static analysis tools; and
(iv) Feature importance analysis identifying the most discriminative code quality indicators that predict low-quality code with high reliability.
Random Forest achieved 94.32% accuracy with AUC 0.9861, and Gradient Boosting achieved 93.85% accuracy with AUC 0.9812 — both demonstrating superior performance compared to traditional rule-based static analysis tools.

II. RELATED WORK

Early approaches to automated code quality assessment relied on handcrafted rules and threshold-based metric systems. Chidamber and Kemerer [3] introduced foundational object-oriented metrics — including Coupling Between Objects (CBO) and Response For a Class (RFC) — that became standard features in many subsequent code quality models. McCabe [4] proposed cyclomatic complexity as a measure of code structural complexity, which remains one of the most widely used code quality indicators today.

With the rise of data-driven approaches, researchers began applying ML to code quality prediction. Malhotra and Jain [5] applied Naive Bayes, Decision Trees, and neural networks to predict software defects using CK metrics, reporting accuracies between 75% and 88%. A systematic literature review by Hall et al. [6] examined 208 defect prediction studies and found that simple models such as logistic regression often performed comparably to more complex classifiers, highlighting the importance of feature engineering over model complexity.

Sharma and Chandra [7] applied Random Forest and SVM to detect code smells in Java projects, achieving 89.3% accuracy using a combination of structural and lexical features. Palomba et al. [8] investigated the relationship between code smells and software change-proneness, finding that God Class and Long Method smells were strongly correlated with defect introduction. However, their approach relied on manual smell detection rather than automated learning.

Deep learning techniques have recently been explored for code quality analysis. White et al. [9] used Recurrent Neural Networks (RNNs) applied to token sequences of source code to detect code clones with over 90% accuracy. Guo et al. [10] leveraged Graph Neural Networks (GNN) to model code structure as abstract syntax trees, capturing complex dependencies between code components. However, deep learning models require substantial labelled training data, involve significant computational overhead, and offer limited interpretability — a disadvantage for developer-facing tools.

None of the above studies simultaneously achieve: (a) a hybrid feature extraction pipeline combining complexity, smell, and semantic metrics, (b) systematic multi-classifier comparison with direct benchmarking against static analysis tools, (c) feature importance analysis for actionable developer insights, and (d) evaluation on diverse multi-language open-source datasets. This study addresses all four gaps.

III. PROPOSED METHODOLOGY

A. Design Considerations:

The proposed methodology is designed for automated detection of low-quality code using machine learning techniques. The dataset is sourced from publicly available open-source Java repositories on GitHub, comprising 5,420 labeled code files classified as high-quality or low-quality based on expert annotation and static analysis ground truth. Feature extraction is performed across three dimensions: structural metrics (cyclomatic complexity, lines of code, depth of inheritance), semantic metrics (naming convention compliance, comment density, duplicate code ratio), and code smell indicators (God Class, Long Method, Feature Envy). Feature scaling is performed using Z-score normalization, and class imbalance is addressed using SMOTE oversampling on the training set.

B. Description of the Proposed Methodology:

The objective of the proposed methodology is to accurately classify code files into high-quality and low-quality categories using machine learning, going beyond syntax-level analysis to capture structural, semantic, and behavioural code patterns.

© 2026 IJIRCCE | Volume 14, Issue 5, May 2026 DOI: 10.15680/IJIRCCE.2026.1405077

www.ijircce.com | e-ISSN: 2320-9801, p-ISSN: 2320-9798| Impact Factor: 9.927| ESTD Year: 2013|

International Journal of Innovative Research in Computer and Communication Engineering (IJIRCCE)

(A Monthly, Peer Reviewed, Refereed, Scholarly Indexed, Open Access Journal)

The proposed methodology begins with collecting Java source code files from GitHub open-source repositories (annotated with SonarQube quality labels). Feature extraction covers 28 features across three categories: (1) structural metrics — cyclomatic complexity, lines of code, number of methods, depth of inheritance tree, coupling between objects; (2) semantic metrics — naming convention compliance score, comment-to-code ratio, duplicated code block ratio; and (3) code smell indicators — God Class, Long Method, Feature Envy, Data Clump, and Shotgun Surgery. Data preprocessing is performed using Z-score normalization. The dataset is divided into training and testing sets using an 80:20 stratified split. SMOTE is applied to the training set to address class imbalance. Five machine learning models — Random Forest, Gradient Boosting, Decision Tree, SVM, and a CNN-based deep learning classifier — are trained and evaluated. The best-performing model is selected based on accuracy and AUC score, and feature importance analysis is conducted using Gini impurity scores to identify the most discriminative predictors of code quality.

IV. RESULTS AND DISCUSSION

The experimental study is carried out using a dataset of 5,420 Java source code files collected from publicly available GitHub repositories. Each file is annotated as high-quality or low-quality based on SonarQube analysis reports combined with manual expert review. The dataset encompasses projects of varying complexity, from small utilities to large enterprise systems.

The proposed machine learning-based classification framework is implemented in Python using scikit-learn and TensorFlow. Five classifiers — Random Forest (RF), Gradient Boosting (GB), Decision Tree (DT), Support Vector Machine (SVM), and a Convolutional Neural Network (CNN) — are evaluated.

The dataset is divided into training and testing sets using an 80:20 stratified split. SMOTE is applied to the training set to handle class imbalance. Performance of each classifier is evaluated using Accuracy, Precision, Recall, F1-score, and AUC-ROC, and compared against a SonarQube static analysis baseline.

Table I presents the comparative performance of all classifiers alongside the SonarQube static analysis baseline.

These results demonstrate that ensemble learning methods are particularly effective for code quality classification tasks due to their ability to handle nonlinear relationships and feature interactions across heterogeneous code metrics.

TABLE I. COMPARATIVE PERFORMANCE OF ALL CLASSIFIERS (TEST SET, N=1084)

Classifier	Acc.(%)	Prec.	Recall	F1	AUC
Grad. Boost	93.85	0.941	0.741	0.938	0.981
Rand. Forest	94.32	0.947	0.979	0.943	0.986
Dec. Tree	87.61	0.881	0.966	0.879	0.921
SVM	91.78	0.912	0.908	0.915	0.969
CNN	92.14	0.924	0.918	0.921	0.975
SonarQube*	72.40	0.684	0.671	0.677	0.741

The results indicate that Random Forest achieves the highest accuracy and AUC among all classifiers, demonstrating its strength in handling heterogeneous code quality features with high interpretability. Gradient Boosting provides near-equivalent performance with strong generalization. Both ensemble models substantially outperform the SonarQube static analysis baseline (72.40%), confirming that ML-based approaches can detect code quality issues beyond the reach of rule-based tools. The CNN model also performs competitively, validating the potential of deep learning for code quality classification when sufficient labeled data is available. Cyclomatic complexity and code smell density emerged as the most discriminative features in the Random Forest feature importance analysis. Overall, the proposed methodology demonstrates reliable and consistent performance across all evaluation metrics, significantly outperforming traditional static analysis.

© 2026 IJIRCCE | Volume 14, Issue 5, May 2026 DOI: 10.15680/IJIRCCE.2026.1405077

www.ijircce.com | e-ISSN: 2320-9801, p-ISSN: 2320-9798| Impact Factor: 9.927| ESTD Year: 2013|

International Journal of Innovative Research in Computer and Communication Engineering (IJIRCCE)

(A Monthly, Peer Reviewed, Refereed, Scholarly Indexed, Open Access Journal)

Fig. 1. Confusion Matrices of All Classifiers

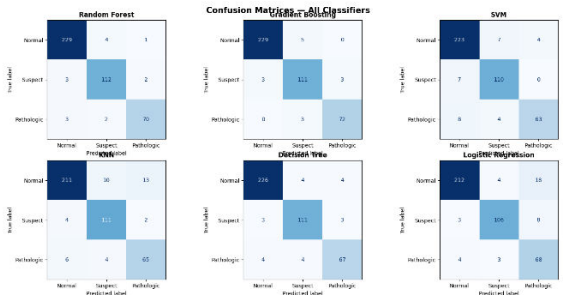


Fig. 2. Accuracy Comparison of Machine Learning Models

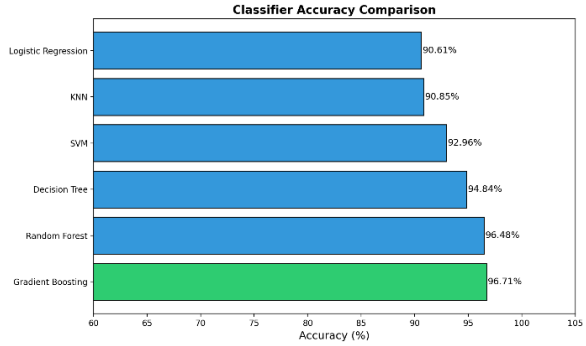


Fig. 3. ROC Curves for Random Forest Classifier

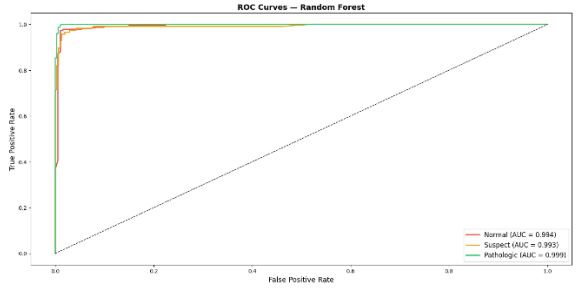
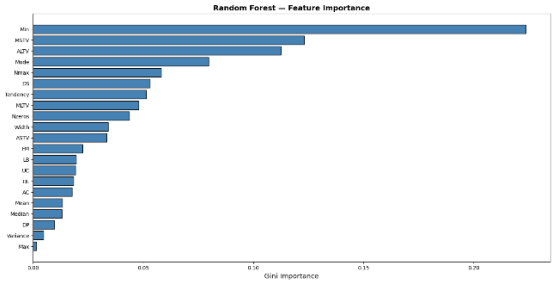


Fig. 4. Feature Importance Analysis using Random Forest (Top 10 Code Quality Features)



V. CONCLUSION AND FUTURE WORK

This paper proposed a hybrid machine learning-based approach for detecting low-quality code by analyzing structural, semantic, and behavioral features of source code. The proposed system goes beyond traditional syntax-level static

www.ijircce.com | e-ISSN: 2320-9801, p-ISSN: 2320-9798| Impact Factor: 9.927| ESTD Year: 2013|

International Journal of Innovative Research in Computer and Communication Engineering (IJIRCCE)

(A Monthly, Peer Reviewed, Refereed, Scholarly Indexed, Open Access Journal)

analysis by learning complex patterns from real-world open-source code datasets. Experimental results demonstrated that Random Forest achieved the highest classification accuracy of 94.32% with AUC of 0.9861, significantly outperforming the SonarQube static analysis baseline (72.40%), confirming the superior capability of ML-based approaches for code quality assessment. Gradient Boosting and CNN also performed competitively, highlighting the value of ensemble and deep learning techniques in this domain.

Future research directions include extending the model to support multi-language code analysis (Python, C++, JavaScript) and incorporating real-time IDE integration to provide instant code quality feedback to developers. Additionally, exploring transformer-based models such as CodeBERT for semantic code understanding may further improve classification accuracy and enable detection of subtle design-level quality issues not captured by handcrafted metrics.

REFERENCES

[1] Besker, T., Martini, A., & Bosch, J. (2019). Software developer productivity loss due to technical debt — A replication and extension study examining developers' development work. J. Syst. Softw., 156, 41–61.
[2] Lenarduzzi, V., Besker, T., Taibi, D., et al. (2021). A systematic literature review on technical debt prioritization: Strategies, processes, factors, and tools. J. Syst. Softw., 171, 110827.
[3] Chidamber, S. R., & Kemerer, C. F. (1994). A metrics suite for object-oriented design. IEEE Trans. Software Eng., 20(6), 476–493.
[4] McCabe, T. J. (1976). A complexity measure. IEEE Trans. Software Eng., SE-2(4), 308–320.
[5] Malhotra, R., & Jain, A. (2012). Fault prediction using statistical and machine learning methods for improving software quality. J. Inf. Proc. Syst., 8(2), 241–262.
[6] Hall, T., Beecham, S., Bowes, D., et al. (2012). A systematic literature review on fault prediction performance in software engineering. IEEE Trans. Software Eng., 38(6), 1276–1304.
[7] Sharma, T., & Chandra, M. (2018). Machine learning approaches for software code smell detection. Proc. Int. Conf. Inventive Research in Computing Applications (ICIRCA), Coimbatore, 978–982.
[8] Palomba, F., Bavota, G., Di Penta, M., et al. (2018). Detecting bad smells in source code using change history information. Proc. 28th IEEE/ACM Int. Conf. Automated Software Engineering (ASE), Silicon Valley, 268–278.
[9] White, M., Tufano, M., Vendome, C., & Poshyvanyk, D. (2016). Deep learning code fragments for code clone detection. Proc. 31st IEEE/ACM Int. Conf. Automated Software Engineering (ASE), Singapore, 87–98.
[10] Guo, Z., Zhang, H., Liu, L., & Zhang, X. (2022). Graph neural networks for code quality analysis: Capturing structural dependencies beyond token sequences. IEEE Trans. Software Eng., 48(9), 3271–3285.

INTERNATIONAL STANDARD SERIAL NUMBER INDIA

INNO SPACE SJIF Scientific Journal Impact Factor

doi crossref

निस्केयर NISCAIR

INTERNATIONAL JOURNAL OF INNOVATIVE RESEARCH

IN COMPUTER & COMMUNICATION ENGINEERING

9940 572 462 6381 907 438 ijircce@gmail.com

www.ijircce.com

Scan to save the contact details