Automated Code Review and Feedback Generation using Machine Learning

Abishek.S[1], Mr.Arokiaraj Christian St Hubert[2], Dhanvarshan.S[3], Dinesh kumar.S[4]

[1,3,4]UG Student, Department of Computer Science Engineering, Sri Manakula Vinayagar Engineering College, Madagadipet Puducherry 605107, India.

[2]Assistant Professor, Department of Computer Science Engineering, Sri Manakula Vinayagar Engineering College, Madagadipet Puducherry 605107, India

***Abstract*—An advanced tool designed to streamline and enhance the code review process through automated analysis and feedback. This system leverages Abstract Syntax Trees (AST) to parse and extract key structural features from code, enabling comprehensive syntactic and semantic analysis. By incorporating machine learning methods, such as the Random Forest algorithm, the tool evaluates code quality, identifies potential issues, and provides targeted feedback to developers. The feedback includes suggestions for improving code structure, readability, and adherence to best practices, helping developers to refine their coding technique In addition to quality assessment, the tool integrates a security feature check to detect vulnerabilities, such as SQL injection risks and other forms of malicious code.**

Keywords - Automated Code Review, Static Code Analysis, Rapid Code Feedback, Bug Detection, Security Vulnerability Detection, Code Improvement Suggestions, Predictive Code Review Tools…

I. INTRODUCTION

Code quality is a cornerstone of modern software engineering, directly impacting system reliability, maintainability, and overall performance. Traditionally, code reviews rely on manual assessment, where reviewers analyze code for errors, adherence to best practices, and potential optimizations. While effective in certain contexts, manual reviews are inherently limited by human subjectivity, scalability challenges, and the potential for oversight. As software development processes accelerate to meet market demands, there is a critical need for automated systems capable of delivering fast, accurate, and consistent code evaluations.

This study addresses these challenges by presenting an intelligent system that leverages machine learning models to automate the code review process. Unlike static analysis tools, which often provide generic feedback, the proposed system offers a more nuanced evaluation by analyzing submissions across multiple dimensions and delivering targeted feedback tailored to individual coding submissions. This approach not only reduces the reliance on human reviewers but also enhances the scalability and consistency of the evaluation process.

The system incorporates advanced machine learning algorithms, including Support Vector Regressor, Multi-Layer Perceptron Regressor, and Random Forest Regressor, to identify coding issues, assess structural and logical integrity, and provide actionable feedback. A unique contribution of this work is the introduction of the "Golden Feature Vector" framework, which serves as an ideal benchmark for evaluating code quality. Real-time feedback mechanisms enable developers to iteratively refine their submissions, reducing errors and enhancing efficiency.

Beyond technical evaluations, the system integrates robust security features to identify and mitigate malicious code, ensuring a secure coding environment. The platform is implemented as a web-based application using Streamlit, offering an intuitive interface for users to submit code and receive instant feedback. Due to its user-friendly and accessible architecture, the system is especially well-suited for educational environments, where substantial amounts of Programming tasks need regular and prompt feedback.

By automating the code review process, This work adds to the expanding corpus of research on using intelligent systems to improve software quality. The proposed solution addresses key limitations of traditional review methods and provides a scalable, efficient, and unbiased approach to evaluating programming tasks. The findings have implications for both academic and professional domains, highlighting the potential for machine learning to improve developer education and training while revolutionizing code assessment procedures.

This study presents a machine learning-driven system for automated code reviews, providing scalable, consistent, and real-time feedback to improve code quality.

II. RELATED WORK

[1] A Systematic Review of the Effects of Automatic Scoring and Automatic Feedback in Educational Settings

According to a systematic analysis of the literature, there is a noticeable rise in published research from 2016 to early 2020, indicating a growing interest in autonomous scoring and feedback in educational contexts. The review analyzed 125 studies, primarily focusing on higher education (92% of papers), while only a small fraction addressed early (2%) and secondary education (6%). The majority of research concentrated on the sciences (47%), Applications across disciplines came next (32%), then the arts and humanities (21%).Research shows that structured question formats, like multiple-choice and fill-in-the-blank, are where automatic feedback is most commonly used to help with clearly defined responses.Positive outcomes associated with automatic feedback include enhanced student engagement, improved learning through feedback utilization, and reduced instructor bias. But there are still issues, especially with regard to how these technologies affect education and the need for more research on how they affect student experiences and educational quality. Overall, the research indicates that even while autonomous scoring and feedback technologies are developing and finding more uses, there is still a great deal of space for advancement and in-depth research in their application.

[2] Uncovering Determinants of Code Quality In Education Via Static Code Analysis

A review of the literature on code quality in education reveals several key insights. Studies highlight a strong connection between code quality metrics and students' academic outcomes, showing that factors such as code duplications, security vulnerabilities, and code smells can adversely affect grades and project timelines. Static code analysis tools, notably SonarQube, are identified as essential for assessing code quality, providing valuable feedback that supports both teaching and learning processes. Additionally, research indicates that instructional methods—whether traditional or online—and collaborative approaches significantly influence code quality, suggesting opportunities to optimize educational strategies for better results in software development. Project-specific factors, including size and the use of version control systems, also play a critical role in shaping code quality, pointing to the importance of a comprehensive perspective on these elements.

[3] Code Smell Detection Research Based on Pre-training and Stacking Models

But there are still issues, especially with regard to how these technologies affect education and the need for more research on how they affect student experiences and educational quality. Overall, the research indicates that even while autonomous scoring and feedback technologies are developing and finding more uses, there is still a great deal of space for advancement and in-depth research in their application. To enhance feature selection, the REFCV method is used, and the Borderline-SMOTE technique is applied to balance the dataset by generating positive samples. The technique is tested on 44 extensive real-world projects and uses a three-layer stacking model to identify code smells.According to the experimental findings, SCSmell maintains good precision, recall, and F1 scores while increasing detection accuracy by 10.38% when compared to current techniques. The study also highlights important gaps in current detection techniques, such as the sparse application of machine learning models, the poor use of textual features, and the dearth of publicly accessible code smell datasets. This method effectively combines machine learning techniques with deep learning to enhance code smell detection.

[4] Improving Automated Code Reviews: Learning From Experience

By producing feedback automatically, automated code review solutions seek to expedite the review process. Despite the success of existing systems like Code Reviewer, these models often treat all review examples equally, overlooking the variability in feedback quality stemming from differences in reviewer expertise. In order to improve the caliber of automated code reviews, this study investigates an experience-aware oversampling technique that gives expert reviewers' input top priority during training. The model is more able to handle important problems and offer perceptive, useful input by concentrating on examples from seasoned reviewers.The proposed methodology involves fine-tuning the Code Reviewer model using experience-aware oversampling and evaluating its performance across three dimensions: correctness, informativeness, and meaningfulness of generated comments. The results show that the updated model performs better than its predecessor, providing better explanations, useful suggestions, and feedback that is more semantically accurate.

[5] Detection of code smells using machine learning techniques combined with data-balancing methods

The study finds code smells—problems in software design that deviate from accepted practices—by combining data-balancing techniques with machine learning (ML) approaches. Early detection of code smells can direct refactoring efforts and have an impact on program quality and maintenance. While ML models can improve code smell detection, imbalanced data (where non-smelly instances outnumber smelly ones) can hinder their performance. In order to address this imbalance, the study proposes a method that combines five machine learning algorithms (decision tree, k-nearest neighbors, support vector machine, XGboost, and multi-layer perceptron) with random oversampling. The study's datasets, which came from open-source systems, featured code smells like "god class," "data class," "long method," and "feature envy." Dataset balance improves ML model accuracy, according to the study. On the original datasets for data class and long method smells, XGboost produced the best accuracy (100%) while random oversampling improved performance for all models.The results highlight how well ML algorithms and data-balancing strategies work together to enhance code smell detection, increasing the process' precision and dependability.

[6] An Analytical Study of Code Smells

The idea of "code smells," which are signs of subpar code that over time may cause software deterioration and maintenance problems, is examined in the paper "An Analytical Study of Code Smells".While code smells do not immediately affect functionality, they signal the need for refactoring to improve long-term maintainability. Software measures such as Lines of Code (LOC), Weighted Methods per Class (WMC), and Coupling Between Objects (CBO) are important for detecting code smells, according to the study.Problems like "God Class" and "Brain Class," which have a significant effect on code quality, can be found while using these metrics. The paper discusses various refactoring actions that address these smells, with the "Move Method" action being particularly effective in resolving multiple issues simultaneously. Additionally, the paper highlights the creation of a code smell repository that organizes knowledge on code smells, their detection tools, related metrics, and refactoring methods. For developers, this repository is a great resource because it offers advice on how to write better code.Analytical techniques are used to identify relationships between code smells and metrics.

[7] Rethinking AI code generation: a one-shot correction approach basedonuserfeedback

This paper introduces a novel methodology, **One-shot Correction**, to address these challenges in natural language-to-code translation without requiring additional re-training. The approach employs decomposition techniques to divide code translation into smaller sub-problems, enabling the construction of the final code from individual query chunks. These chunks are refined using user feedback or selectively generated by the model. Our evaluation demonstrates that this method achieves performance comparable to or exceeding existing models, while offering a clear and interpretable process. This interpretability allows for the detailed examination of unexpected outcomes and provides actionable insights for further improvements.
Additionally, we show that user feedback can significantly enhance code translation models without necessitating re-training. To validate the practical utility of our methodology, we developed a prototype graphical user interface (GUI) application. This tool simplifies the customization and evaluation of suggested code, providing users with an intuitive platform to leverage the benefits of One-shot Correction effectively.

[8] Analysis of Learning Behavior in an Automated Programming Assessment Environment: A Code Quality Perspective

The study offers a thorough summary of educational technology research, with a focus on automated assessment systems and programming education. Specifically in a Java programming classroom, it highlights the use of machine learning techniques like Support Vector Machines and Decision Trees to forecast student success based on programming activity data. Key findings reveal that timely homework submissions and effective code quality are strong indicators of academic success, with a prediction model achieving 87% accuracy in identifying at-risk students. The study categorizes learners into distinct groups based on their behaviors, emphasizing the correlation between submission patterns and final grades. Additionally, it underscores the significance of using Educational Data Mining (EDM) techniques to enhance learning outcomes and facilitate timely interventions for students who may struggle. Overall, the research advocates for further exploration of assignment design and its impact on student engagement and performance.

III. LITERATURE REVIEW OF RELATED WORKS

SL.NO	TITLE	AUTHOR	YEAR	ADVANTAGE	DISADVANTAGE
1	A Systematic Review of the Effects of Automatic Scoring and Automatic Feedback in Educational Settings	Marcelo Guerra Hahn, SilviaMargarita Baldiris Navarro,Luis de la Fuente Valentín, Daniel Burgos	2021	The study provides a comprehensive review of automatic scoringand feedback, highlighting their growing importance in education.	The study overlooks key issues such as content quality, student privacy, and tool specific requirements.
2	Uncovering Determinants of Code Quality In Education Via Static Code Analysis	Danilo Nikolic,Darko Stefanovic,Miroslav Nikolic, DuSanka Dakic,Miroslav Stefanovic, Sara Koprivica	2024	The study highlights the importance of static code analysis tools in enhancing code quality and provides practical guidelines for educators.	The study did not find significant impacts from collaboration mode or teaching method on code quality, limiting insights on these factors.
3	Code Smell Detection Research Based on Pre-training and Stacking Models	DongwenZhang, Shuai Song, YangZhang*, Haiyang Liu, Gaojie Shen	2024	When compared to existing methods, SCSmell improves average accuracy by 10.38% while maintaining acceptable precision, recall, and F1 scores.	Overfitting and underfitting in deep learning models can result in subpar detection outcomes.
4	Improving Automated Code Reviews: Learning From Experience	Hong Yi Lin, Patanamon Thongtanunam, Christoph Treude, Wachiraphan Charoenwet	2024	Experience-aware oversampling improves the quality and meaningfulness of automated code reviews without introducing new data	Potential underestimation of experience metrics due to deleted reviews or multiple accounts
5	Detection of code smells using machine learning techniques combined with data-balancing methods	NasraldeenAlnor Adam Khleel, Károly Nehéz	2023	Code smell detection accuracy is improved using machine learning approaches.	Data imbalance can reduce the effectiveness of machine learning algorithms and provide biased outcomes.
6	An Analytical Study of Code Smells	LidaBamizadeh, Binod Kumar, AjayKumar, Shailaja Shirwaikar	2021	Refactoring actions can optimize software quality and eliminate code smells	Code smells may not interrupt functionality but can lead to long-term decay and reduced software quality
7	Rethinking AI code generation: a one-shot correction approach basedonuserfeedback	KimTuyen Le, Artur Andrzejak	2024	Seamlessly integrates user feedback without retraining, improving model efficiency	Feedback dependency may limit generalizability across diverse user populations
8	Analysis of Learning Behavior in an Automated Programming Assessment Environment: A Code Quality Perspective	Hsi-Min Chen, Bao-An Nguyen, Yi-Xiang Yan, Chyi-Ren Dow	2021	The iterative learning approach promotes adherence to code quality and improves learning results.	The degree of difficulty of the tasks may have an impact on how well at-risk pupils are identified early on, which could restrict how broadly the findings can be applied.

IV. PROPOSED SYSTEM

The proposed Automated Code Review Feedback Generator addresses the limitations of existing systems by combining machine learning, static code analysis, and an interactive web-based interface. This system is designed to evaluate code submissions comprehensively, providing scores and constructive feedback based on multiple facets of code quality, including correctness, readability, efficiency, and adherence to best practices. By automating the grading and feedback process, the proposed system ensures speed, consistency, and depth in evaluations.

One of the key innovations in this system is the Golden Feature Vector, an ideal feature representation derived from high-quality code submissions. When a code is evaluated, its feature vector is compared with the Golden Feature Vector to identify discrepancies and suggest improvements. Additionally, the system has strong security features to safeguard user data, such as code sanitization, HTTPS encryption, session management, and safe password storage.

Deployed as a user-friendly web application using Streamlit, the system enables users to upload code and instantly receive feedback. Features like progress tracking allow users to monitor improvements and set goals, enhancing their coding skills over time.

A. DATA COLLECTION

The training dataset for the system consists of annotated code submissions with scores reflecting multiple quality dimensions, such as correctness, readability, and efficiency. The machine learning models are trained on these annotated submissions to identify patterns in both high- and low-quality code. The data includes a diverse set of programming problems and solutions, ensuring the model's robustness and generalization across different coding scenarios.

B. PRE-PROCESSING:

Pre-processing plays a critical role in ensuring accurate analysis of code submissions. Steps include:

Input Validation: Ensuring uploaded code adheres to the expected format and syntax.

Normalization: Standardizing code structure for consistency in analysis.

Feature Extraction Preparation: Transforming raw code into a structured representation, such as Abstract Syntax Trees (ASTs), to facilitate feature extraction.

C. FEATURE EXTRACTION:

Feature extraction identifies key attributes of the code relevant for evaluation. This process includes:

Direct Metrics: Loop and conditional counts, function definitions, and variable usage.

AST-Based Metrics: Structural features such as nesting depth, modularity, and data structure usage.

Derived Features: Efficiency indicators like computational complexity and memory usage.

These features are critical for the machine learning model to differentiate between high-quality and suboptimal code submissions.

D. MODEL TRAINING

The system employs a combination of MLP and Random Forest regression models that were trained using the features that were extracted.These models predict a score for the submission by assessing multiple quality dimensions, including correctness, readability, and efficiency.

E. SECURITY FEATURES

Robust security measures are embedded within the system to ensure safe and reliable code evaluation:

Input Validation and Sanitization: Preventing malicious scripts and injections.

Database Security: To reduce the danger of SQL injection, use parameterized queries and ORM frameworks.

Secure Communication: Encrypting data transmission via HTTPS.

User Authentication: Employing hashing algorithms for password storage and session expiration for unauthorized access prevention.

F. RESULT AND DISCUSSION

The proposed Automated Code Review Feedback Generator demonstrates significant improvements in accuracy and feedback quality compared to traditional systems. By leveraging the hybrid approach of feature extraction and machine learning, it provides targeted, actionable feedback that enhances the coding skills of users. The system's architecture ensures efficient analysis of submissions while maintaining data security and user privacy. The integration of the Golden Feature Vector adds a layer of personalization, making feedback both specific and constructive.

Validation metrics, including accuracy and loss graphs, illustrate the system's learning progress and predictive performance. The results indicate consistent improvements in feedback quality over time, making the platform an invaluable tool for coding education and practice.

CONCLUSION

In conclusion, the Automated Code Review Feedback Generator offers a comprehensive and innovative solution for evaluating code submissions. By combining machine learning, static code analysis, and a secure, interactive interface, the system addresses key challenges in code evaluation. The inclusion of advanced feedback mechanisms and robust security features ensures a seamless and safe user experience, empowering learners to refine their coding skills effectively.

REFERENCE

[1] M. Y. Mhawish and M. Gupta, "Predicting Code Smells and Analysis of Predictions: Using Machine Learning Techniques and Software Metrics," J. Comput. Sci. Technol., vol. 35, no. 6, pp. 1428–1445, Nov. 2020, doi: 10.1007/S11390-020-0323-7.

[2] F. Pecorelli, D. Di Nucci, C. De Roover, and A. De Lucia, "On the role of data balancing for machine learning-based code smell detection," MaLTeSQuE 2019 - Proc. 3rd ACM SIGSOFT Int. Work. Mach. Learn. Tech. Softw. Qual. Eval. co-located with ESEC/FSE 2019, pp. 19–24, Aug. 2019, doi: 10.1145/3340482.3342744.

[3] N. A. A. Khleel and K. Nehéz, "Deep convolutional neural network model for bad code smells detection based on oversampling method," Indones. J. Electr. Eng. Comput. Sci., vol. 26, no. 3, pp. 1725–1735, Jun. 2022, doi: 10.11591/IJEECS.V26.I3.PP1725-1735.

[4] J. A. Harer, L. Y. Kim, R. L. Russell, O. Ozdemir, L. R. Kosta, A. Rangamani, et al., "Automated software vulnerability detection with machine learning", *CoRR*, vol. abs/1803.04497, 2018.

[5] Z. Li, D. Zou, S. Xu, X. Ou, H. Jin, S. Wang, et al., "Vuldeepecker: A deep learning-based system for vulnerability detection", *CoRR*, vol. abs/1801.01681, 2018.

[6] S. Jain and A. Saha, "Rank-based univariate feature selection methods on machine learning classifiers for code smell detection," Evol. Intell., vol. 15, no. 1, pp. 609–638, Mar. 2022, doi: 10.1007/S12065-020-00536- Z.

[7] F. Pecorelli, D. Di Nucci, C. De Roover, and A. De Lucia, "A large empirical assessment of the role of data balancing in machine-learning-based code smell detection," J. Syst. Softw., vol. 169, p. 110693, Nov. 2020, doi: 10.1016/J.JSS.2020.110693.

[8] E. Tempero et al., "The Qualitas Corpus: A curated collection of Java code for empirical studies," Proc. - Asia-Pacific Softw. Eng. Conf. APSEC, pp. 336–345, 2010, doi: 10.1109/APSEC.2010.46.

[9] G. Saranya, H. Khanna Nehemiah, A. Kannan, and V. Nithya, "Model level code smell detection using EGAPSO based on similarity measures," Alexandria Eng. J., vol. 57, no. 3, pp. 1631–1642, Sep. 2018, doi: 10.1016/J.AEJ.2017.07.006.

[10] W. Xu and X. Zhang, "Multi-granularity code smell detection using deep learning method based on abstract syntax tree," Proc. Int. Conf. Softw. Eng. Knowl. Eng. SEKE, vol. 2021-July, pp. 503–509, 2021, doi: 10.18293/SEKE2021-014.

[11] F. Pecorelli, F. Palomba, D. Di Nucci, and A. De Lucia, "Comparing heuristic and machine learning approaches for metric-based code smell detection," IEEE Int. Conf. Progr. Compr., vol. 2019-May, pp. 93–104, May 2019, doi: 10.1109/ICPC.2019.00023.

[12] M. Hadj-Kacem and N. Bouassida, "A hybrid approach to detect code smells using deep learning," ENASE 2018 - Proc. 13th Int. Conf. Eval. Nov. Approaches to Softw. Eng., vol. 2018-March, pp. 137–146, 2018, doi: 10.5220/0006709801370146.

[13] N. A. A. Khleel and K. Nehez, "Comprehensive Study on Machine Learning Techniques for Software Bug Prediction," Int. J. Adv. Comput. Sci. Appl., vol. 12, no. 8, pp. 726–735, 2021, doi: 10.14569/IJACSA.2021.0120884. [29] S. Jain and A. Saha, "Improving perf