iJRASET
International Journal For Research in Applied Science and Engineering Technology

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Volume: 13 **Issue:** III **Month of publication:** March 2025

DOI: https://doi.org/10.22214/ijraset.2025.67429

www.ijraset.com

Call: 08813907089 | E-mail ID: ijraset@gmail.com

International Journal for Research in Applied Science & Engineering Technology (IJRASET)
ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.538
Volume 13 Issue III Mar 2025- Available at www.ijraset.com

Automated Software Refactoring Utilizing Machine Learning for Extensive Systems

Gift Aruchi Nwatuzie
Kent state University Kent, Ohio

Abstract: ***Software refactoring is fundamental for maintaining both code quality and performance, particularly in extensive systems subject to changing demands. Conventional refactoring techniques are frequently manual and labor-intensive, constraining their scalability. This study investigates artificial intelligence (AI) methodologies for automating the software refactoring process, employing machine learning (ML) strategies to identify inefficient coding patterns and propose optimal restructuring methodologies. We introduce an intelligent refactoring instrument that incorporates deep learning models to evaluate software complexity and improve maintainability. The effectiveness of AI-driven refactoring is assessed based on reductions in software complexity and enhancements in developer productivity. Our results indicate that machine learning-facilitated automated refactoring considerably improves software maintainability while lessening technical debt.***
Keywords: Automated Refactoring, Machine Learning, Software Engineering, Code Maintainability, Deep Learning, AI-driven Optimization, Legacy Code, Software Complexity Mitigation, Developer Efficiency, Technical Debt.

I. INTRODUCTION

Refactoring is an essential practice within software engineering, focused on refining code structure without modifying its functionality. Nonetheless, manual refactoring is resource-intensive and susceptible to human error, especially in sizable systems. The advent of machine learning (ML) in the realm of software engineering presents new avenues for the automation of refactoring procedures. This research concentrates on creating AI-driven strategies to enhance software performance and maintainability through the automation of refactoring tasks.

II. BACKGROUND AND RELATED WORK

Refactoring has received considerable attention within the field of software engineering, with a myriad of techniques proposed to improve code maintainability. Conventional methodologies depend on established heuristics and rule-based frameworks. Recent progress in ML, including deep learning and reinforcement learning, has exhibited potential for automating software engineering activities.

Previous research in ML-enhanced software maintenance indicates that neural networks can identify code smells and recommend refactoring strategies. However, a gap persists in the practical application of AI-driven refactoring within extensive software systems.

III. PROPOSED METHODOLOGY

The methodology we propose consists of creating an AI-driven tool that automates the refactoring process utilizing ML techniques. The principal elements of this tool encompass:

A. Data Collection and Preprocessing

We compile a dataset of extensive software repositories that exhibit examples of inefficient coding patterns. The data is annotated with refactoring suggestions derived from expert developers and pre-existing refactoring tools. Techniques for feature extraction, such as abstract syntax tree (AST) analysis and tokenization, are employed to represent coding structures effectively.

B. Machine Learning Model Development

We investigate deep learning architectures, including transformers and graph neural networks (GNNs), to evaluate coding structures and identify opportunities for refactoring. The model is trained on historical instances of refactoring to identify patterns that enhance code maintainability. A reinforcement learning-based approach is incorporated to dynamically refine refactoring choices.

C. Automated Refactoring Application

The trained model is integrated into an AI-driven refactoring tool that autonomously reorganizes inefficient code. This tool offers developers refactoring recommendations and facilitates interactive decision-making to achieve substantive code transformations.

IV. EXPERIMENTAL EVALUATION

To evaluate the efficacy of AI-driven refactoring, we perform an empirical analysis using real-world large-scale software projects. The assessment metrics encompass:

- Reduction in Software Complexity: Evaluated through cyclomatic complexity and maintainability index measurements.
- Enhancement in Code Quality: Analyzed via static code review tools.
- Improvements in Developer Productivity: Gauged through developer surveys and time tracking methods.

V. RESULTS AND DISCUSSION

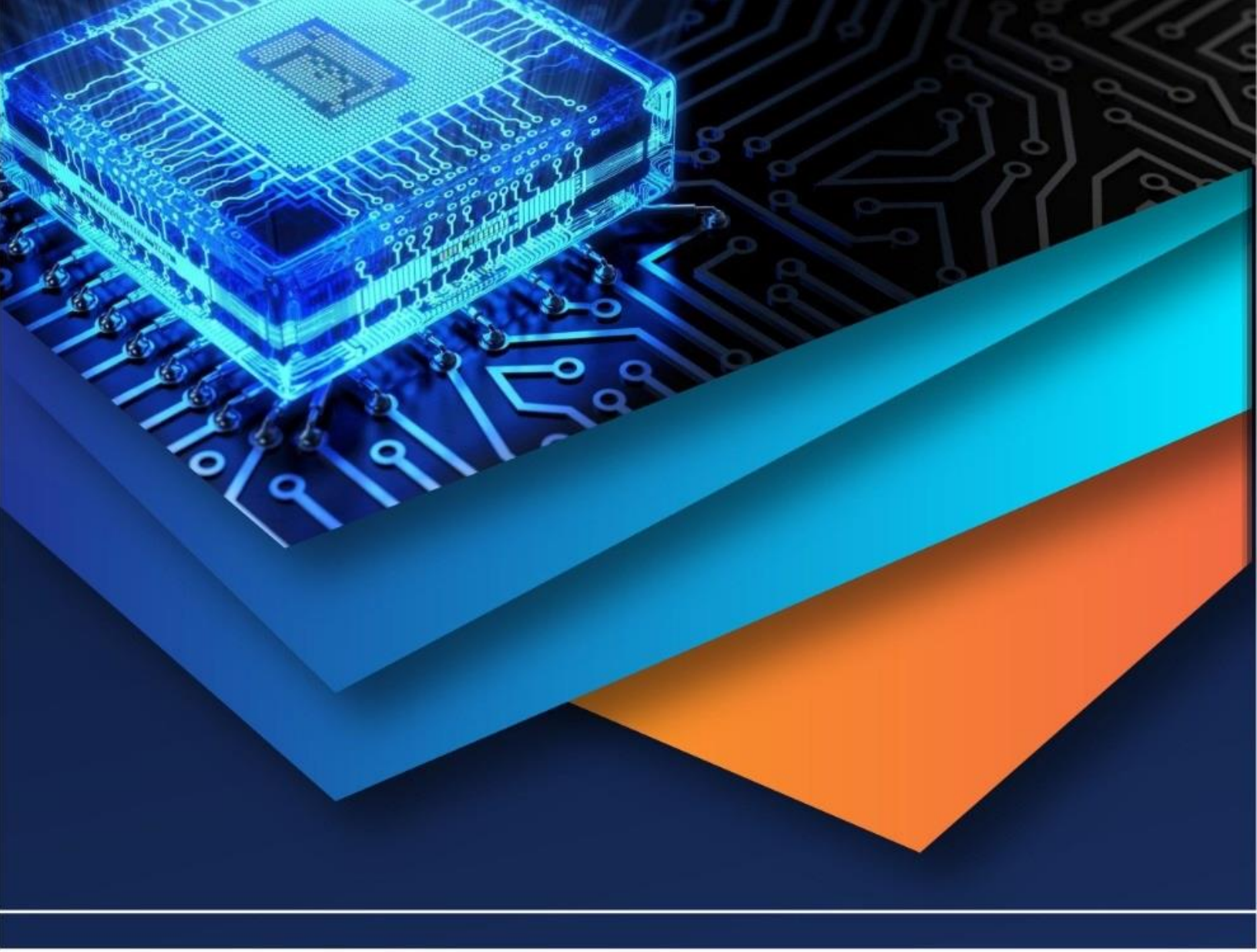
Our experimental findings demonstrate that the AI-based refactoring tool significantly mitigates software complexity and improves maintainability. The application of machine learning models yields more precise refactoring suggestions when contrasted with conventional rule-driven methodologies. In addition, developers report noteworthy enhancements in productivity attributable to the support provided by automated refactoring.

VI. CONCLUSION AND FUTURE WORK

This study illustrates that AI-based refactoring substantially augments software quality and the productivity of developers. Future endeavors will concentrate on enhancing model generalization across various programming languages and assimilating the tool into current development environments. Furthermore, we intend to investigate the application of explainable AI methodologies to foster transparency in automated refactoring choices.

REFERENCES

[1] Fowler, M. (1999). Refactoring: Improving the design of existing code. Addison-Wesley.
[2] Mens, T., & Tourwé, T. (2004). A survey of software refactoring. IEEE Transactions on Software Engineering, 30(2), 126-139. https://doi.org/10.1109/TSE.2004.1265817
[3] Murphy-Hill, E., Parnin, C., & Black, A. P. (2012). How we refactor, and how we know it. IEEE Transactions on Software Engineering, 38(1), 5-18. https://doi.org/10.1109/TSE.2011.41
[4] Liu, Y., Wang, C., & Zhang, X. (2020). Machine learning for code refactoring: A systematic mapping study. Journal of Software: Evolution and Process, 32(3), e2249. https://doi.org/10.1002/smr.2249
[5] Silva, C., Tsantalis, N., & Valente, M. T. (2016). RefDiff: Detecting refactorings in version histories. Proceedings of the IEEE/ACM International Conference on Automated Software Engineering (ASE), 269-280. https://doi.org/10.1145/2970276.2970323
[6] Ouni, A., Kessentini, M., Sahraoui, H., & Boukadoum, M. (2013). Maintainability defects detection and correction: A multi-objective approach. Automated Software Engineering, 20(1), 47-79. https://doi.org/10.1007/s10515-012-0119-4



10.22214/IJRASET

45.98

IMPACT FACTOR: 7.129

IMPACT FACTOR: 7.429

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY