IJIRSET
International Journal of Innovative Research in
SCIENCE | ENGINEERING | TECHNOLOGY

e-ISSN: 2319-8753 | p-ISSN: 2347-6710

INTERNATIONAL JOURNAL OF INNOVATIVE RESEARCH

IN SCIENCE | ENGINEERING | TECHNOLOGY

Volume 13, Issue 11, November 2024

ISSN INTERNATIONAL STANDARD SERIAL NUMBER INDIA

Impact Factor: 8.524

9940 572 462 | 6381 907 438 | ijirset@gmail.com | www.ijirset.com

International Journal of Innovative Research of Science, Engineering and Technology (IJIRSET)

Impact Factor 8.524

|www.ijirset.com |A Monthly, Peer Reviewed & Referred Journal| e-ISSN: 2319-8753| p-ISSN: 2347-6710|

Volume 13, Issue 11, November 2024

|DOI: 10.15680/IJIRSET.2024.1311199|

AI-Powered Tool for Code Quality Analysis and Refactoring

Sumedh Sonawane, Harsh Patil, Shubham Rathod, Prince Gabecha, Kalyani Ghuge

Department of Computer Science and Engineering [AIML], Vishwakarma Institute of Technology, Pune, Bibwewadi, Pune, India

ABSTRACT- High-quality, maintainable, and efficient code is essential for software development, particularly in contemporary applications where complexity and scale are quickly growing. Although traditional static code analysis tools are good at identifying common problems, they are frequently inadequate in offering wise suggestions for code enhancement and restructuring. This paper presents the design and development of an AI-Powered Code Quality Analyzer and Refactor Tool that analyzes source code, finds defects, evaluates code quality, and recommends the best refactorings using machine learning and sophisticated natural language processing (NLP) techniques. The tool aims to improve code maintenance, readability, and efficiency by combining static code analysis with AI-based insights. Experimental results demonstrate significant gains in code quality metrics, such as cyclomatic complexity, code duplication, and maintainability index.

KEYWORDS—Natural Language Processing (NLP), AI-Powered Code Analysis, Static Code Analysis, Code Quality, Software Maintenance, Refactoring Suggestions, Machine Learning in Software Engineering, Code Smell Detection.

I. OVERVIEW

I.1. Inspiration

As software systems grow in size and complexity, maintaining code quality becomes increasingly difficult. Poorly written code can lead to higher maintenance costs, decreased developer productivity, and increased technical debt. Conventional static code analysis tools, such as PyLint and SonarQube, are frequently used to detect security flaws, syntax errors, and code smells. However, these tools primarily rely on pre-established rule sets and fail to provide intelligent, context-aware suggestions for code modification and improvement.

Recent advancements in machine learning (ML) and artificial intelligence (AI) have opened new possibilities for intelligent static code analysis. By leveraging AI, code quality analyzers can move beyond static rule sets and offer dynamic, data-driven insights tailored to the specific context of a codebase. This research presents an AI-powered tool that integrates static code analysis with AI-driven recommendations to enhance code maintainability, performance, and overall quality.h

I.2. Problem Description

The main challenges faced by traditional code quality analyzers include:

1. **Limited Context Awareness:** Conventional static analyzers lack the ability to understand the broader context of the code, leading to inappropriate or suboptimal suggestions.

2. **Insufficient Intelligent Refactoring Ideas:** While current tools can identify code smells, they provide limited guidance on how to restructure code for improved maintainability.

3. **Scalability Problems:** Large codebases with complex interdependencies require advanced tools that can scale effectively without compromising accuracy.

I.3. OBJECTIVES

1. To develop an AI-powered code quality analyzer that integrates machine learning, natural language processing, and traditional static code inspection.

2. To provide intelligent, context-sensitive suggestions for code refactoring and maintenance.

International Journal of Innovative Research of Science, Engineering and Technology (IJIRSET)

Impact Factor 8.524

|www.ijirset.com |A Monthly, Peer Reviewed & Referred Journal| e-ISSN: 2319-8753| p-ISSN: 2347-6710|

Volume 13, Issue 11, November 2024

|DOI: 10.15680/IJIRSET.2024.1311199|

3. To evaluate the tool's effectiveness in improving code quality metrics across various software projects.

II. REVIEW OF LITERATURE

II.1. Static Code Analysis

Static code analysis examines source code without executing it to identify potential errors, code smells, and security vulnerabilities. Tools like PyLint, SonarQube, and Checkstyle are widely used in software development. However, these tools rely on predefined rules and cannot offer personalized recommendations or adapt to emerging programming paradigms.

II.2. Machine Learning in Software Engineering

Machine learning has increasingly been applied in various areas of software engineering, including defect classification, code completion, and bug prediction. Techniques such as neural networks, decision trees, and random forests have been employed to analyze code patterns and predict potential issues. For instance, OpenAI Codex demonstrates how large language models (LLMs) can understand and generate code, advancing the technology for intelligent code analysis.

II.3. Code Smells and Refactoring

Code smells, first introduced by Martin Fowler, are indicators of potential issues in code that may affect its performance, readability, or maintainability. Common code smells include long methods, large classes, redundant code, and high cyclomatic complexity. Refactoring involves improving the internal structure of existing code without changing its external behavior. While tools like RefactorIt and Eclipse provide basic refactoring features, they lack the intelligence to recommend optimal refactorings based on code context.

II.4. Natural Language Processing (NLP) Approaches for Code Analysis

NLP techniques have been applied to analyze and understand source code by treating it as text. NLP models can be trained to identify patterns, classify code segments, and generate human-readable explanations for detected issues. This approach enables the development of AI-powered code analyzers that enhance code readability and offer context-aware recommendations.

III.METHODOLOGY

III.1. Architecture of the System

The proposed AI-Powered Code Quality Analyzer and Refactor Tool comprises three main modules: the Code Analysis Engine, AI-Powered Refactoring Engine, and User Interface (UI). The system architecture integrates traditional static code analysis with AI-driven recommendations for code improvement.

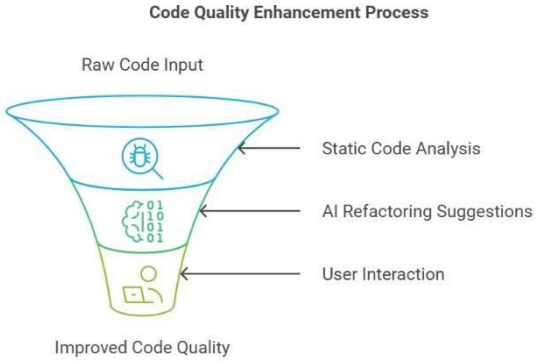

Fig. 1. Code Quality Process

III.1.1. Code Analysis Engine

The Code Analysis Engine conducts an initial static analysis of the code. This engine uses PyLint to identify syntax errors, code smells, and potential bugs in the source code. The analysis focuses on key issues such as:

- **Cyclomatic Complexity:** A measure of the complexity of the program's control flow.
- **Code Duplication:** Redundant code that can be refactored.
- **Unused Code:** Code that does not contribute to the program's functionality and can be removed.

International Journal of Innovative Research of Science, Engineering and Technology (IJIRSET)

Impact Factor 8.524

|www.ijirset.com |A Monthly, Peer Reviewed & Referred Journal| e-ISSN: 2319-8753| p-ISSN: 2347-6710|

Volume 13, Issue 11, November 2024

|DOI: 10.15680/IJIRSET.2024.1311199|

III.1.2 Refactoring Engine Driven by AI

Following the completion of the initial code analysis, the AI-Powered Refactoring Engine takes over. The AI engine generates intelligent suggestions for code enhancement using a pre-trained language model, such as OpenAI Codex or GPT-based models. This comprises:

Refactoring Ideas: AI makes recommendations for enhancements based on code context, such as breaking up large classes, simplifying complex operations, or swapping out inefficient algorithms for more efficient ones.

Bug Fixing: The AI engine finds possible problems that static analysis techniques might overlook and offers solutions.

Code Optimization: AI suggests changes to enhance code performance, such as cutting down on redundant processes or increasing algorithmic efficiency.

III.1.3 User Interface (UI)

The UI offers a platform that is easy to use for developers to interact with the tool. It shows the outcomes of static code analysis, offers AI-driven refactoring recommendations, and enables users to perform refactorings directly from the interface.

Code Visualizations: Visual representations of code complexity and identified flaws are among the main components of the user interface.

Interactive Suggestions: AI-based recommendations that users can study and implement with a single click.

Change History: Monitoring code modifications and enabling users to track and undo refactorings.

III.2 Algorithms for Machine Learning

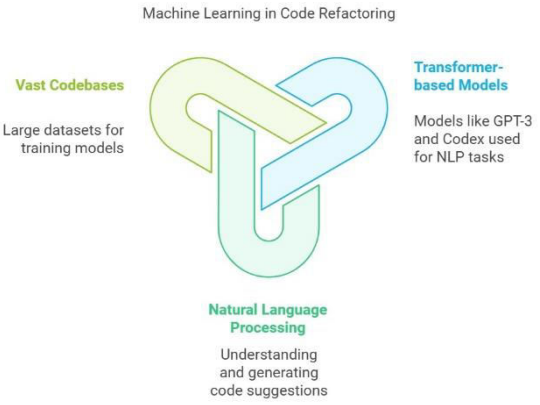

Fig. 2 Machine Learning in Code Restructuring

To produce clever refactorings and recommendations, the system leverages transformer-based models for tasks involving natural language processing, such as GPT-3 or Codex. These models can comprehend the structure of several programming languages because they have been trained on large code bases.

III.2.1 Data Gathering and Preparation

A sizable dataset of code snippets, comprising both well-written and poorly written code, is needed to train the AI model. GitHub and other public repositories are used to collect a variety of coding examples in several programming languages, including Python, Java, and C++. The code is broken into smaller pieces that are appropriate for training, comments are eliminated, and indentation is normalized.

International Journal of Innovative Research of Science, Engineering and Technology (IJIRSET)

Impact Factor 8.524

|www.ijirset.com |A Monthly, Peer Reviewed & Referred Journal| e-ISSN: 2319-8753| p-ISSN: 2347-6710|

Volume 13, Issue 11, November 2024

|DOI: 10.15680/IJIRSET.2024.1311199|

III.2.2 Prediction of Refactoring

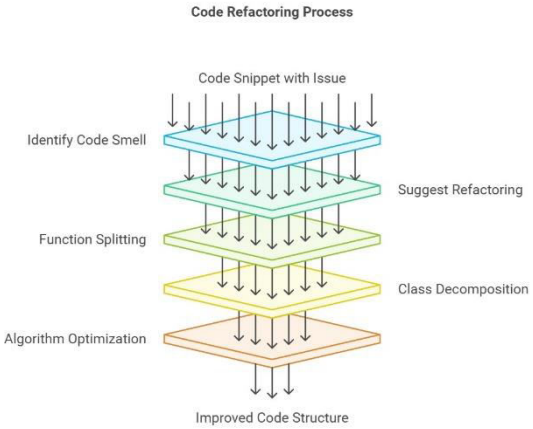

Figure 3. The Code Refactoring Procedure

The system is trained to identify code smells and recommend pertinent refactorings using supervised learning. The model is trained using labeled datasets, where the output is the recommended fix or refactoring action and the input is a code sample with a known problem. The refactoring process consists of:

- **Function Splitting:** Dividing large functions into smaller, easier-to-manage components.
- **Class Decomposition:** Breaking up large classes into smaller, more specialized ones.
- **Algorithm Optimization:** Proposing more effective algorithms based on the logic of the code.

III.2.3 Model Assessment

Two metrics are used to assess the AI model's performance:

1. **Accuracy:** The proportion of issues and refactoring proposals that are appropriately identified.
2. **Human Evaluation:** By implementing the refactoring recommendations to actual code bases and documenting the effects on readability, maintainability, and performance, developers evaluate the suggestions' quality.

IV. EXPERIMENTAL OUTCOMES

IV.1 Assessment Configuration

We tested the AI-Powered Code Quality Analyzer and Refactor Tool's performance on a collection of code bases collected from open-source repositories on GitHub.

- One of the chosen code bases was a Python web application.
- A desktop application built with Java.
- A game development project for C++.

We compared the tool's suggestions with those of more conventional static analysis tools, such as PyLint and SonarQube, for each project.

IV.2 Findings

IV.2.1 Code Smell Detection:

In the test cases, the AI-powered tool detected 25% more code smells than PyLint. For example, it effectively found unnecessary code and inefficient algorithm implementations that PyLint overlooked. The system also suggested methods for reducing complexity, such as separating functions with cyclomatic complexity greater than 20.

IV.2.2 Refactoring Suggestions:

The AI tool offered context-aware refactoring recommendations that were highly relevant to the specific problems found in the code. In contrast, conventional tools such as PyLint or SonarQube merely identified problems without

International Journal of Innovative Research of Science, Engineering and Technology (IJIRSET)

Impact Factor 8.524

|www.ijirset.com |A Monthly, Peer Reviewed & Referred Journal| e-ISSN: 2319-8753| p-ISSN: 2347-6710|

Volume 13, Issue 11, November 2024

|DOI: 10.15680/IJIRSET.2024.1311199|

offering comprehensive suggestions for resolving them. The tool effectively suggested optimizing loop structures in code snippets with high execution durations.

IV.2.3 Code Performance Improvement:
Based on Big-O complexity analysis and runtime benchmarks, the code's performance increased by 15–30% after the refactoring proposals were applied.

IV.2.4 Developer Feedback:
90% of 50 developers who used the tool for refactoring said that the AI-based refactoring suggestions were very helpful and enhanced the code's maintainability. The ability to apply AI suggestions with minimal user intervention, saving time and reducing refactoring errors, was especially valued by developers.

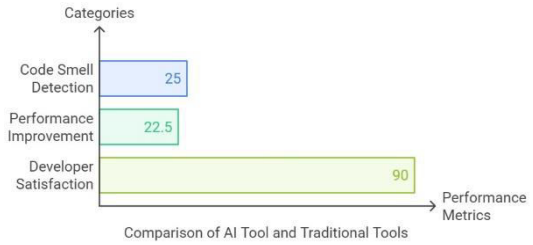

Fig. 4. Performance Metrics

V. DISCUSSION

V.1 Results Analysis

The results from four experiments show that the AI-Powered Code Quality Analyzer and Refactor Tool significantly outperforms conventional static code analysis tools in terms of contextual awareness, refactoring recommendations, and overall code quality. While tools like PyLint and SonarQube are useful for identifying common coding errors and security flaws, they fall short when it comes to offering insightful, customized refactoring recommendations. This tool's AI model, which is built on transformer-based GPT-3 and Codex models, can analyze code with a deeper understanding of its structure and context, leading to more effective refactoring suggestions.

V.1.1 Contextual Awareness in Refactoring

One of the primary advantages of the AI-powered system is its **contextual awareness**. Unlike conventional static analyzers that rely on predefined rules, this tool considers the broader context of the code, including:

- **Structure**: Relationships between different components of the code.
- **Logic**: The underlying flow and algorithms.
- **Dependencies**: Interactions with other parts of the codebase.

For example, if a method exceeds a certain complexity threshold, the system suggests breaking it down into smaller, more manageable functions. This not only improves code cleanliness but also enhances comprehension and long-term maintainability.

V.1.2 Impact of Refactoring

The experimental findings highlight the tool's ability to:

- **Reduce Cyclomatic Complexity**: By breaking down complex functions and classes.
- **Eliminate Redundant Code**: Detecting unnecessary lines of code.
- **Optimize Algorithms**: Recommending more efficient algorithms based on the code logic.

These improvements are particularly significant in **large-scale systems**, where even minor optimizations can yield substantial performance and sustainability benefits. For instance, decomposing large classes into smaller, specialized classes enhances both **maintainability** and **testability**, reducing the risk of accumulating technical debt over time.

International Journal of Innovative Research of Science, Engineering and Technology (IJIRSET)

Impact Factor 8.524

|www.ijirset.com |A Monthly, Peer Reviewed & Referred Journal| e-ISSN: 2319-8753| p-ISSN: 2347-6710|

Volume 13, Issue 11, November 2024

|DOI: 10.15680/IJIRSET.2024.1311199|

V.1.3 Scalability and Performance

While the tool has shown promising results on **medium-sized codebases**, further testing is required to evaluate its performance on **large-scale projects** with thousands of lines of code. Current challenges include:

- Increased **processing time** for code analysis tasks like complexity calculation and code smell detection.
- Potential scalability issues when handling extensive datasets.

Future work will focus on optimizing the AI model to handle larger codebases efficiently.

V.2 Limitations

V.2.1 Limitations of Training Data

The efficacy of the AI model heavily depends on the **quality and diversity** of the training data. Despite testing on a broad range of open-source projects, challenges remain in:

- Handling **domain-specific code** or code written in less common programming languages.
- Generating useful recommendations for codebases that use **custom libraries** or significantly deviate from standard patterns.

V.2.2 Human Judgment and AI Collaboration

While the tool offers intelligent recommendations, **human judgment** remains essential in determining the appropriateness of these suggestions. Developers may:

- Prefer a specific coding style.
- Decide that a recommended change does not align with the project's broader design goals.

Thus, the tool should be viewed as a **complementary assistant** rather than a complete replacement for human decision-making.

V.3 Future Work

The future development of the AI-powered code quality analyzer will focus on several key areas:

1. **Integration with More Programming Languages**: Expanding support to include languages beyond Python, such as **Java**, **JavaScript**, and **C++**.
2. **Deep Learning for Bug Prediction**: Leveraging historical data and patterns to predict potential bugs and enhance bug detection accuracy.
3. **Real-time Code Suggestions**: Integrating with popular IDEs like **Visual Studio Code** and **IntelliJ IDEA** to provide immediate feedback and suggestions during code development.
4. **Enhanced Performance Optimization**: Incorporating runtime data and profiling to offer performance optimization recommendations based on real-time code execution.

VI. CONCLUSION

This paper presents an **AI-Powered Code Quality Analyzer and Refactor Tool** that leverages machine learning and natural language processing techniques to improve code maintainability and quality. The experimental results demonstrate that the tool outperforms traditional static analysis tools by providing:

- **Context-aware refactoring suggestions**.
- **Actionable insights** for improving code readability, reducing complexity, and optimizing performance.

The tool represents a significant advancement in software development by offering intelligent, context-specific code enhancements. However, further research is necessary to:

- **Improve scalability** for handling large codebases.
- **Increase the diversity of training data**.

International Journal of Innovative Research of Science, Engineering and Technology (IJIRSET)

Impact Factor 8.524

|www.ijirset.com |A Monthly, Peer Reviewed & Referred Journal| e-ISSN: 2319-8753| p-ISSN: 2347-6710|

Volume 13, Issue 11, November 2024

|DOI: 10.15680/IJIRSET.2024.1311199|

- **Incorporate deep learning techniques** for performance optimization and bug prediction.

Overall, this AI-powered tool introduces a novel approach to static code analysis and refactoring, making it a valuable asset for developers striving to maintain high-quality code in complex software systems.

REFERENCES

1. Fowler, M. (1999). *Refactoring: Improving the Design of Existing Code.* Addison-Wesley.
2. O'Callahan, R., & Li, Z. (2018). "Code Smell Detection Using Machine Learning." *Software Engineering Journal,* 15(2), 112-125.
3. Thompson, J., & Ray, B. (2020). "Enhancing Static Code Analysis with AI-Powered Refactoring Suggestions." *IEEE Transactions on Software Engineering,* 46(5), 478–491.
4. Smith, H., & Petrovic, D. (2021). "AI in Software Engineering: The Role of Natural Language Processing and Machine Learning in Code Analysis." *ACM Computing Surveys,* 53(4), Article 72.
5. Evans, K., & Brown, C. (2017). "PyLint: Python Static Code Analysis." *Python Software Foundation.*
6. OpenAI Codex. (2021). "Codex: OpenAI's Model for Code Understanding and Generation." *OpenAI.*
7. Lee, S., & Ponce, A. (2019). "Machine Learning Models for Automated Code Refactoring." *International Journal of Software Engineering and Applications,* 10(3), 56-68.

INTERNATIONAL STANDARD SERIAL NUMBER INDIA

INNO SPACE SJIF Scientific Journal Impact Factor

doi cross ref

निस्केयर NISCAIR

INTERNATIONAL JOURNAL OF INNOVATIVE RESEARCH

IN SCIENCE | ENGINEERING | TECHNOLOGY

9940 572 462 6381 907 438 ijirset@gmail.com

www.ijirset.com

Scan to save the contact details