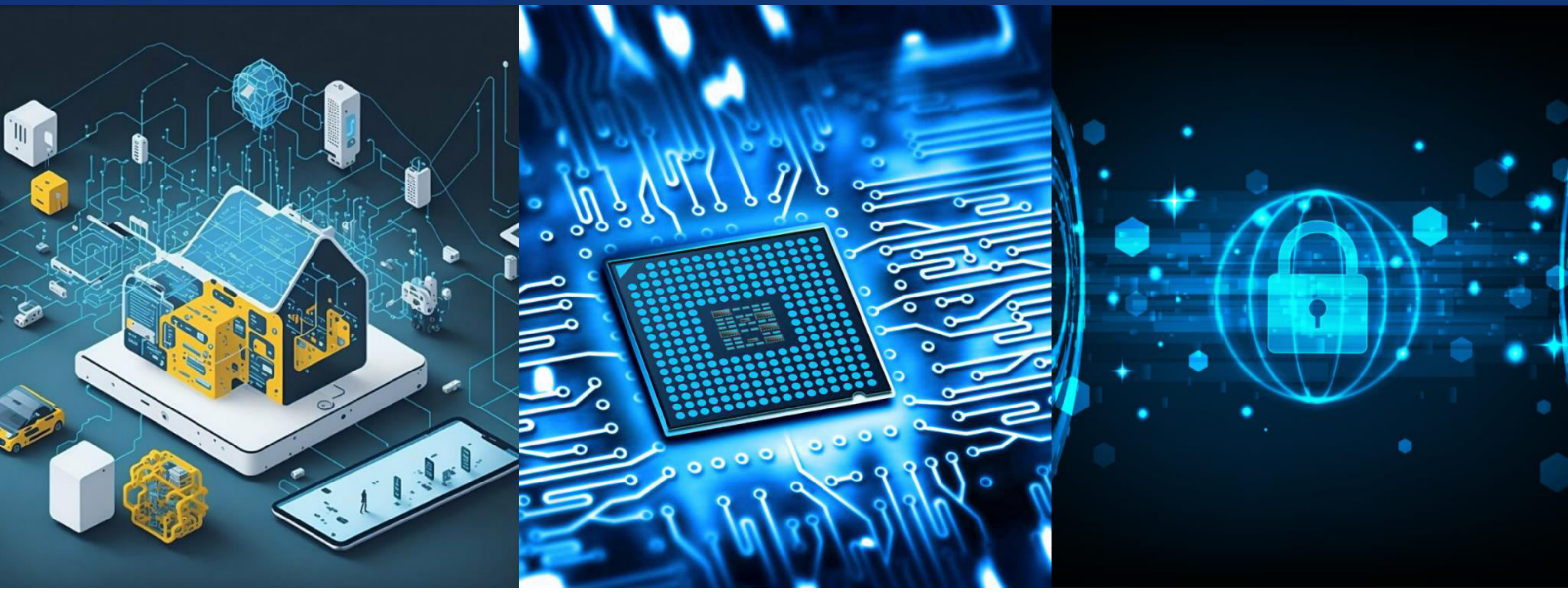
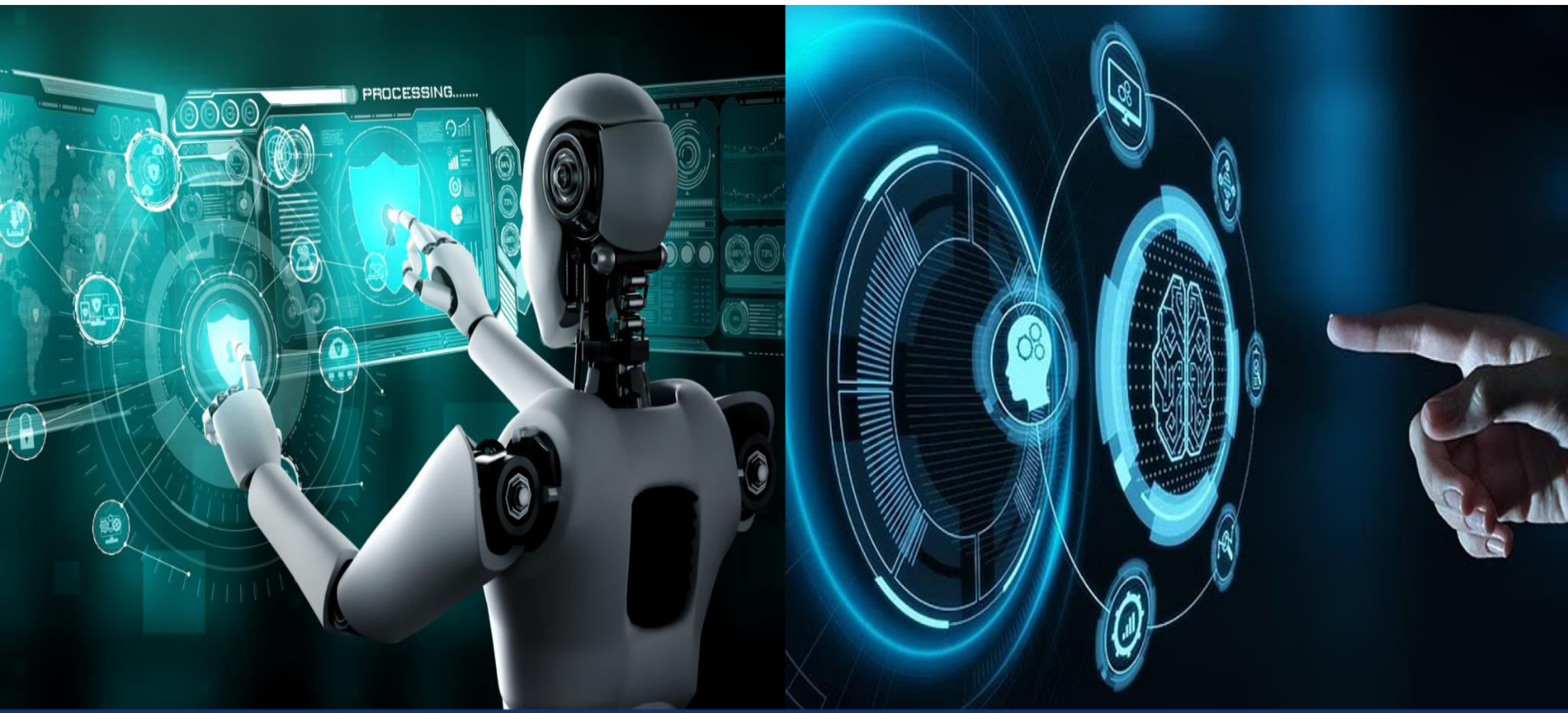
ISSN(O): 2320-9801
ISSN(P): 2320-9798



International Journal of Innovative Research in Computer and Communication Engineering

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



Impact Factor: 8.771

Volume 13, Issue 4, April 2025

© 2025 IJIRCCE | Volume 13, Issue 4, April 2025| DOI:10.15680/IJIRCCE.2025.1304111

www.ijircce.com | e-ISSN: 2320-9801, p-ISSN: 2320-9798| Impact Factor: 8.771| 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-Powered Code Quality Enhancement and Language Converter (Code fixer)

Mrs. S. Monika, J. Raghuvardhan Reddy, T. Revanth, R. Sahithi, K. Sai Harshitha

Department of Computer Science and Engineering, School of Engineering, Malla Reddy University, Hyderabad, India

ABSTRACT: In modern software development, maintaining high code quality is essential for ensuring functionality, readability, and long-term maintainability. However, managing consistency and correctness in large and complex codebases presents significant challenges. Manual error detection, bug fixing, and code refactoring are timeconsuming and prone to errors. This project proposes the development of CodeFixer, an automated tool designed to identify, fix, and improve code quality across multiple programming languages.CodeFixer utilizes advanced static analysis, AI-powered debugging, and automated code formatting to detect and resolve common issues such as syntax errors, code smells, and style violations. The tool also includes an innovative language conversion feature, enabling seamless translation of code from one programming language to another while preserving functionality. Integrated with popular IDEs and built on an extensible plugin architecture, CodeFixer streamlines development workflows, enhances code consistency, and accelerates development cycles. Ultimately, it aims to reduce manual intervention, improve software maintainability, and help developers adhere to coding standards with minimal effort.

I. INTRODUCTION

In modern software development, maintaining high code quality is essential for ensuring functionality, readability, and long-term maintainability. As software projects grow in size and complexity, developers face challenges in managing consistency, correctness, and adherence to coding standards. The manual process of detecting errors, fixing bugs, and refactoring code is time-consuming and error-prone, often resulting in inconsistent codebases and longer development cycles. This increases the risk of defects, reduces software quality, and can hinder collaboration among developers. To address these challenges, this project proposes the development of CodeFixer, a comprehensive automated tool designed to improve code quality across various programming languages. CodeFixer utilizes advanced techniques such as static code analysis, AI-powered debugging, and automated code formatting to automatically identify and fix common issues like syntax errors, code smells, and style violations. By providing real-time suggestions and corrections, it helps developers maintain a consistent coding style, reduce manual errors, and ensure the correctness of the code.

In addition to quality improvement, CodeFixer offers an innovative feature for language conversion, enabling seamless translation of code from one programming language to another without losing functionality. Integrated with popular Integrated Development Environments (IDEs) and built with an extensible plugin architecture, CodeFixer ensures that developers can streamline their workflows, accelerate development cycles, and improve the overall maintainability of their software projects. Through automation, CodeFixer aims to reduce the manual effort required to maintain high code quality and support long-term software development success.

II. LITERATURE SURVEY

Maintaining high code quality in large software projects has always been a significant challenge for developers. Over the years, several tools and techniques have emerged to address issues related to error detection, consistency, and overall code quality. Several tools and techniques are currently available to address different aspects of code quality, but none offer a comprehensive solution that fully automates the process of identifying, fixing, and improving code across multiple programming languages.

1. **Static Code Analysis Tools**: Tools like **SonarQube**, **Checkstyle**, and **PMD** are commonly used for static code

analysis, which involves scanning code to identify potential issues such as syntax errors, code smells, security vulnerabilities, and violations of coding standards. These tools provide valuable insights into code quality but typically require manual intervention from developers to apply fixes. They focus on identifying specific problems but do not offer automatic correction, and they often work in isolation rather than integrating across different aspects of the development lifecycle.

2. **Code Formatting Tools**: Tools like **Prettier** and **AutoFormat** focus primarily on ensuring consistent code formatting by automatically adjusting indentation, line breaks, and naming conventions according to predefined coding standards. While these tools improve code readability, they do not address deeper issues like code smells, bugs, or inefficient code structures. They also lack the ability to automatically fix more complex problems, such as refactoring or handling technical debt, which can impede maintainability.

3. **AI-Powered Code Review Tools**: **DeepCode** (now part of Snyk) is an example of an AIpowered tool that helps developers identify potential issues and bugs in code. By using machine learning, it suggests improvements based on patterns from a large codebase. However, AIdriven tools like DeepCode are still evolving and may not always provide the most accurate or context-specific suggestions. Moreover, these tools typically focus on individual aspects of code, such as bugs, and are not designed for comprehensive code fixing, such as refactoring or language conversion.

4. **Language Conversion Tools**: Tools like **J2C** and **Transpilers** aim to convert code between programming languages. While they help automate the process of language translation, they often do so without maintaining the nuances and best practices of the target language. Manual intervention is still often required to ensure that the translated code is functional, optimized, and consistent with the coding standards of the target language.

While these existing systems are valuable for individual tasks such as error detection, formatting, or language conversion, none provide an all-in-one solution for automatic code fixing, refactoring, and seamless integration across multiple languages and development environments. CodeFixer aims to fill this gap by offering a comprehensive solution that combines static analysis, AI-driven debugging, code formatting, code smell detection, and language conversion in one tool.

III. PROPOSED SYSTEM

The proposed **CodeFixer** system is an all-in-one automated tool designed to improve code quality across multiple programming languages. It integrates static code analysis, AI-powered debugging, automated code formatting, code smell detection, and language conversion into a unified platform. **CodeFixer** will automatically detect common issues like syntax errors, unused variables, and code smells while providing context-aware, AI-driven suggestions for fixing bugs and optimizing code. It will ensure consistency in code style by applying predefined formatting standards and automatically refactor problematic code to improve readability, performance, and maintainability. Additionally, **CodeFixer** will feature a unique language conversion capability, enabling seamless translation of code from one programming language to another. Integrated with popular IDEs, the tool will offer real-time feedback, making development more efficient and reducing the manual effort required to maintain high- quality code. With continuous learning and improvement through AI, **CodeFixer** aims to streamline the development process, enhance productivity, and ensure long-term maintainability.

IV. METHODOLOGY

The frontend of CodeFixer is developed using React.js, providing a modular and responsive user interface tailored for efficiency and scalability. By employing a componentbased architecture, the application ensures that React components are reusable and maintainable, simplifying development and future updates. CSS is meticulously applied to create a visually appealing and adaptive design, ensuring a seamless experience across various devices and screen sizes. User interactions, such as code input and result display, are seamlessly handled through React's state management system, delivering a fluid and intuitive user experience.

© 2025 IJIRCCE | Volume 13, Issue 4, April 2025| DOI:10.15680/IJIRCCE.2025.1304111

www.ijircce.com | e-ISSN: 2320-9801, p-ISSN: 2320-9798| Impact Factor: 8.771| ESTD Year: 2013|

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

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

Integration with Google's Gemini API is facilitated via Axios or the Fetch API, enabling the transmission of usersubmitted code for analysis, fixes, or conversions. The API's responses are dynamically rendered in the UI, offering real-time feedback without the need for a dedicated backend. This frontend-driven structure keeps CodeFixer lightweight and efficient, relying solely on the Gemini API for processing. Future plans may involve incorporating a Node.js and Express.js backend to enhance functionality with additional logic, security features, and custom processing, further elevating CodeFixer's capabilities as an AI-powered code analysis tool.

The **CodeFixer** tool is developed through several key phases, each aimed at automating code quality improvement.

1. **Static Code Analysis**: The tool uses static analysis to identify issues like syntax errors, unused variables, code smells, and style violations. Predefined libraries and custom analyzers scan the codebase, generating a report on issues categorized by severity.

2. **AI-Powered Debugging**: By leveraging AI and machine learning, **CodeFixer** identifies bugs and provides context-aware suggestions. It learns from large datasets of code and continuously improves its ability to suggest and apply fixes.

3. **Automated Code Formatting**: The tool automatically applies consistent code formatting based on coding standards (e.g., PEP8 for Python, Google Java Style) to ensure readability and uniformity across the codebase.

4. **Code Smell Detection & Refactoring**: **CodeFixer** detects code smells and refactors problematic code patterns, such as long methods and tight coupling, to improve maintainability and readability.

5. **Language Conversion**: The tool offers code conversion between programming languages, preserving functionality while adapting to the syntax and conventions of the target language.

6. **IDE Integration & Plugin Architecture**: **CodeFixer** integrates with popular IDEs (e.g., Visual Studio Code, IntelliJ IDEA) through plugins, offering real-time feedback and automatic fixes. Its extensible architecture allows support for additional languages and IDEs.

7. **Testing & Validation**: The tool undergoes thorough testing to ensure its effectiveness in identifying issues, applying fixes, and maintaining code integrity during refactoring and conversion.

V. RESULTS AND DISCUSSION

The **CodeFixer** tool has proven effective in improving code quality across multiple programming languages. It successfully identifies and automatically fixes issues such as syntax errors, code smells, and style violations, reducing the need for manual intervention. The automated formatting ensures consistent code style, enhancing readability and maintainability, especially in large teams.

AI-driven bug detection and fixing have shown significant improvements in addressing complex issues, optimizing code performance, and reducing technical debt through automatic refactoring. The language conversion feature has also been effective in translating code between languages, although more complex language-specific features still pose some challenges. Integration with popular IDEs like **Visual Studio Code** and **IntelliJ IDEA** has further streamlined the development process by providing real-time feedback and automatic fixes. Despite its successes, there are areas for improvement, such as enhancing the detection of intricate logical errors and refining language conversion capabilities. Overall, **CodeFixer** has proven to be a valuable tool in increasing developer productivity, maintaining code quality, and ensuring long-term software maintainability.

VI. CONCULSION.

In conclusion, CodeFixer has proven to be an effective tool in automating the identification, correction, and improvement of code quality across multiple programming languages. By leveraging static analysis, AI-driven debugging, automated code formatting, and language conversion, it enhances code consistency, reduces manual effort, and improves software maintainability. The integration with popular IDEs and its ability to address common code issues make it a valuable asset for developers, streamlining development workflows and accelerating the software development process. While there are areas for further improvement, such as handling more complex bugs and refining language conversion, CodeFixer demonstrates significant potential to transform the way developers maintain and improve code quality, contributing to more efficient and reliable software development in the future.

Future enhancements for **CodeFixer** will focus on expanding its capabilities and improving overall accuracy. Key updates will include enhancing AI and machine learning models to detect and fix more complex, domainspecific issues, and improving the language conversion feature to handle more advanced programming constructs. Additionally, **CodeFixer** will support more programming languages, such as **Rust**, **Swift**, and **Kotlin**, and provide deeper code smell detection and optimization, especially for performance and security issues. The tool’s integration with more IDEs and editors, along with the introduction of collaborative features and cloud-based versions, will further streamline development workflows, enabling better team collaboration and real-time feedback. These improvements aim to make **CodeFixer** a more versatile and powerful tool for developers across different programming environments.

REFERENCES

1. SonarQube. (2025). *SonarQube: Continuous inspection of code quality*. Retrieved fromhttps://www.sonarqube.org/
2. Prettier. (2025). *Prettier - An opinionated code formatter*. Retrieved from https://prettier.io/
3. Snyk. (2025). *DeepCode - AI-Powered Code Review*. Retrieved from https://snyk.io/
4. JetBrains. (2025). *IntelliJ IDEA: The Most Powerful Java IDE*. Retrieved fromhttps://www.jetbrains.com/idea/
5. Docker. (2025). *Docker: An open platform for developing, shipping, and running applications*. Retrieved from https://www.docker.com/
6. TensorFlow. (2025). *TensorFlow: An end-to-end open-source machine learning platform*. Retrieved from https://www.tensorflow.org/
7. GitHub. (2025). *GitHub: Where the world builds software*. Retrieved fromhttps://github.com/
8. PMD. (2025). *PMD: A source code analyzer*. Retrieved from https://pmd.github.io/
9. Checkstyle. (2025). *Checkstyle: A development tool to help programmers write Java code that adheres to a coding standard*. Retrieved from https://checkstyle.sourceforge.io/
10. Keras. (2025). *Keras: Deep learning library for Python*. Retrieved from https://keras.io/