International Journal of Engineering Research and Science & Technology

www.ijerst.org

ISSN : 2319-5991

Vol. 22 No. 2 (2026)

ijerst.editor@gmail.com

editor@ijerst.com

International Journal of Engineering Research and Science & Technology

ISSN 2319-5991 www.ijerst.com
Vol. 22, No. 2, 2026
© 2026 IJERST. All Rights Reserved
Peer Reviewed, Referred & Indexed Journal

AI-Assisted Multi-Language Code Bug Detection and Auto-Fixing System

DASAM KODANDA RAMA SAI DURGAPRASAD

PG Scholar. Department of MCA, DNR College, Bhimavaram, Andhra Pradesh

V.SARALA

(Assistant Professor), Master of Computer Applications, DNR College, Bhimavaram, Andhra Pradesh

ABSTRACT

Software development is an inherently complex process where bugs and errors can significantly impact performance, security, and usability. Detecting and fixing these issues early in the development lifecycle is crucial to reducing development costs and improving software quality. This project presents an **AI-Assisted Multi-Language Code Bug Detection and Auto-Fixing System**, a desktop-based application developed using Python and Tkinter that provides intelligent debugging support for Python, C, and Java programs. The system integrates static code analysis, heuristic-based artificial intelligence techniques, and external linting tools to identify potential bugs, syntax errors, and bad programming practices. For Python programs, the application leverages the Abstract Syntax Tree (AST) module to perform syntax validation, ensuring that code structure adheres to language rules. Additionally, the system incorporates a heuristic-based AI bug prediction module that detects common programming mistakes such as infinite loops, improper exception handling, misuse of comparison operators, and outdated syntax usage.

To enhance the robustness of the analysis, the tool integrates with Pylint, a widely used static code analysis tool, to generate detailed error and warning reports. These reports provide insights into code quality, helping developers adhere to coding standards and best practices. For C and Java languages, the system performs basic structural checks to identify missing essential components such as print statements and class declarations. A key feature of the proposed system is its **Auto-Fix module**, which automatically corrects simple syntax issues such as missing colons and trailing whitespace. This feature reduces manual debugging effort and improves developer productivity. The application also includes a user-friendly graphical interface with code editing capabilities, syntax highlighting, and real-time error visualization, making it suitable for both beginners and experienced programmers. The system is designed to be extensible, allowing future integration of advanced machine learning models for more accurate bug prediction and automated code repair. By combining traditional static analysis with AI-driven

techniques, this project contributes to the development of intelligent programming tools that assist developers in writing clean, efficient, and error-free code.

Overall, this project demonstrates how intelligent debugging systems can enhance software development processes by providing automated assistance, reducing human errors, and improving code reliability. It serves as a practical solution for academic learning as well as real-world software engineering applications.

Keywords: Bug Detection, Static Code Analysis, Python AST, Pylint, Machine Learning, Code Editor, Auto Fixing, Software Debugging, Tkinter GUI, Heuristic Analysis

I. INTRODUCTION

In modern software development, writing error-free and efficient code is a challenging task due to increasing system complexity and tight development timelines. Bugs in software can lead to unexpected behavior, system crashes, and even security vulnerabilities. Therefore, effective debugging and error detection mechanisms are essential to ensure software reliability and maintainability.Traditional debugging methods often rely on manual code inspection and testing, which can be time-consuming and error-prone. Developers typically use compilers, interpreters, and debugging tools to identify issues in their code. However, these tools may not always detect logical errors or poor coding practices. As a result, there is a growing need for intelligent systems that can assist developers in identifying and fixing bugs automatically.This project introduces an **AI-Assisted Multi-Language Code Bug Detection and Auto-Fixing System**, designed to simplify the debugging process through automation and intelligent analysis. The system focuses on providing a unified platform where developers can write, analyze, and fix code in multiple programming languages, including Python, C, and Java.

The application is built using Python's Tkinter library, which provides a graphical user interface (GUI) for easy interaction. Users can load source code files, analyze them for errors, and receive detailed feedback on potential issues. The system employs Python's Abstract Syntax Tree (AST) module for syntax validation, ensuring that Python code is structurally correct.In addition to syntax checking, the system uses heuristic-based artificial intelligence techniques to predict common programming mistakes. These heuristics are designed based on typical coding errors observed in real-world scenarios, such as infinite loops, improper exception handling, and incorrect syntax usage. This approach enables the system to provide proactive suggestions to developers.Another important component of the system is its integration with Pylint, which enhances the analysis by providing detailed reports on code quality, style violations, and potential errors. This combination of built-in analysis and external tools ensures comprehensive code evaluation.The Auto-Fix module further improves developer productivity by automatically correcting minor syntax issues, reducing the need for manual intervention. Additionally, the system includes features such as syntax highlighting and error marking, which help users quickly identify and understand issues in their code.Overall, this project aims to bridge the gap between traditional debugging tools and modern intelligent systems by providing an integrated solution for code analysis and bug fixing. It serves as an effective tool for both educational purposes and professional software development.

LITERATURE SURVEY (WITH EXISTING METHODS)

The field of automated bug detection and software analysis has been extensively studied, with various approaches proposed to improve code quality and reliability. Traditional static analysis tools such as Pylint, PMD, and FindBugs have been widely used to detect syntax errors, code smells, and violations of coding standards. These tools analyze source code without executing it, making them efficient for early-stage debugging.Research in static code analysis has shown that tools like Pylint can effectively identify common programming errors and enforce coding conventions. However, these tools often rely on predefined rules and may not detect logical errors or context-specific issues. To address this limitation, researchers have explored the use of machine learning techniques for bug prediction.Machine learning-based approaches analyze historical code data to identify patterns associated with bugs. Techniques such as decision trees, neural networks, and support vector machines have been used to predict defect-prone code segments. These models can learn from past errors and provide more accurate predictions compared to rule-based systems.Another significant advancement in this field is the use of Abstract Syntax Trees (ASTs) for code analysis. AST-based approaches allow for a deeper understanding of code structure, enabling more precise detection of syntax and semantic errors. Studies have demonstrated that AST analysis is particularly effective for languages like Python, where indentation and structure play a critical role.

Heuristic-based methods have also been proposed as a lightweight alternative to machine learning models. These methods use predefined patterns to detect common coding mistakes, such as infinite loops, improper exception handling, and misuse of operators. While less sophisticated than ML models, heuristics are computationally efficient and easy to implement.Recent research has focused on integrating multiple techniques to improve bug detection accuracy. Hybrid systems that combine static analysis, machine learning, and heuristic methods have shown promising results. These systems leverage the strengths of each approach, providing comprehensive code analysis and reducing false positives.Automated code fixing is another area of active research. Tools like AutoFix and DeepFix use machine learning models to generate code corrections automatically. These systems aim to reduce developer effort by providing ready-to-use fixes for common errors.Despite these advancements, there are still challenges in achieving fully automated and accurate bug detection. Issues such as scalability, language diversity, and context awareness remain open research problems. The proposed system addresses some of these challenges by combining static analysis, heuristic AI techniques, and user-friendly interfaces.

II. EXISTING SYSTEM

Existing debugging systems primarily rely on compilers, interpreters, and static analysis tools such as Pylint, PMD, and FindBugs. These tools are effective in identifying syntax errors, code style violations, and certain types of logical issues. However, they often operate independently and require developers to use multiple tools for comprehensive

analysis.One major limitation of existing systems is their lack of integration. Developers must switch between different tools for syntax checking, code quality analysis, and debugging, which can be time-consuming and inefficient. Additionally, most traditional tools do not provide automatic code fixing capabilities, requiring manual intervention to resolve issues.Another drawback is the limited use of artificial intelligence in conventional debugging tools. While some modern tools incorporate machine learning techniques, they are often complex and require large datasets for training. As a result, these tools may not be accessible to beginners or small-scale developers.Furthermore, existing systems typically focus on a single programming language, limiting their applicability in multi-language development environments. This creates challenges for developers working on projects that involve multiple technologies.Overall, while existing tools provide valuable support for debugging, they lack the integration, automation, and intelligence required for modern software development.

III. PROPOSED METHOD

The proposed system is an **AI-Assisted Multi-Language Code Bug Detection and Auto-Fixing System** that addresses the limitations of existing debugging tools. It provides a unified platform for analyzing and fixing code in multiple programming languages, including Python, C, and Java.The system integrates static analysis, heuristic-based AI techniques, and external linting tools to provide comprehensive code evaluation. For Python programs, it uses the AST module for syntax validation and Pylint for detailed analysis. The heuristic AI module predicts common bugs based on predefined patterns, offering proactive suggestions to developers. A key feature of the proposed system is its Auto-Fix module, which automatically corrects simple syntax errors such as missing colons and trailing spaces. This reduces manual debugging effort and improves productivity.

The system also includes a user-friendly GUI with features such as syntax highlighting, error marking, and real-time feedback. This makes it accessible to both beginners and experienced developers. By combining multiple analysis techniques into a single platform, the proposed system provides a more efficient and intelligent solution for code debugging.

IV. IMPLEMENTATION

The system is implemented using Python with the Tkinter library for the graphical user interface. The application is structured into multiple modules, each responsible for a specific functionality. The main class, **AdvancedBugFinder**, initializes the GUI and manages user interactions. The interface includes a code editor, output panel, and control buttons for file operations, analysis, and auto-fixing. The file handling module allows users to open and edit source code files. The editor supports syntax highlighting for better readability. Keywords are highlighted using predefined tags, improving the user experience. The analysis module is the core component of the system. For Python code, it uses the AST module to check for syntax errors. If an error is detected, the system highlights the corresponding line and displays a detailed message in the output panel. The

AI bug prediction module uses regular expressions to identify common coding mistakes. This module is designed to be extensible, allowing future integration of machine learning models.

The system also integrates Pylint באמצעות subprocess calls. Temporary files are created to run Pylint, and the results are displayed in the output panel. This ensures comprehensive analysis of code quality. For C and Java languages, a generic analysis module performs basic checks to identify missing components. While less advanced than the Python analysis, it provides useful feedback for beginners. The Auto-Fix module processes the code line by line, applying simple corrections such as adding missing colons and removing trailing spaces. The updated code is displayed in the editor, and a summary of applied fixes is shown in the output panel. Overall, the implementation focuses on modularity, extensibility, and user-friendliness.

V. ALGORITHMS

The system uses several algorithms for code analysis and bug detection:

1. **Syntax Analysis Algorithm (AST आधारित)**
   Parses Python code using AST to detect syntax errors.
2. **Heuristic Bug Detection Algorithm**
   Uses pattern matching (regex) to identify common bugs:
   - Infinite loops
   - Bare except blocks
   - Invalid print syntax
3. **Static Analysis Algorithm (Pylint Integration)**
   Executes Pylint using subprocess to analyze code quality.
4. **Auto-Fix Algorithm**
   - Read code line by line
   - Remove trailing spaces
   - Add missing colons
   - Update editor
5. **Keyword Highlighting Algorithm**
   Searches for keywords and applies text tags for visualization.

These algorithms work together to provide comprehensive debugging support

VI. SYSTEM DESIGN

The system follows a modular architecture consisting of the following components:

1. **User Interface Module**
   Developed using Tkinter, this module provides an interactive environment for code editing and analysis.
2. **File Management Module**
   Handles file opening, reading, and displaying content in the editor.
3. **Analysis Module**
   - Python Analysis (AST + AI + Pylint)

- Generic Analysis (C/Java)

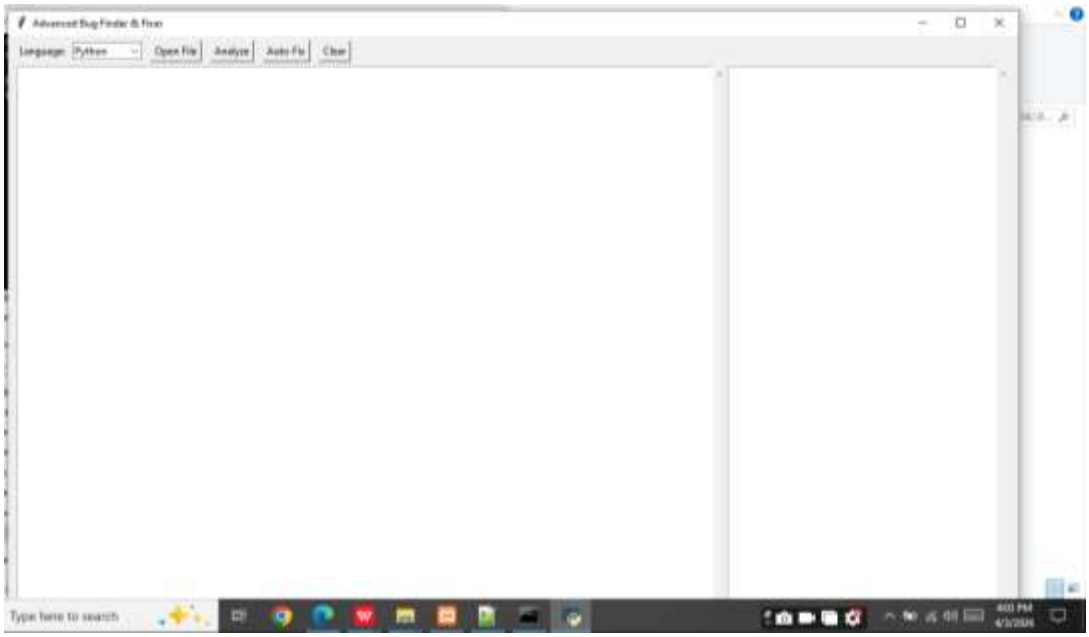
4. **AI Prediction Module**
   Uses heuristics to detect common bugs.
5. **Auto-Fix Module**
   Automatically corrects minor syntax issues.
6. **Visualization Module**
   Provides syntax highlighting and error marking.

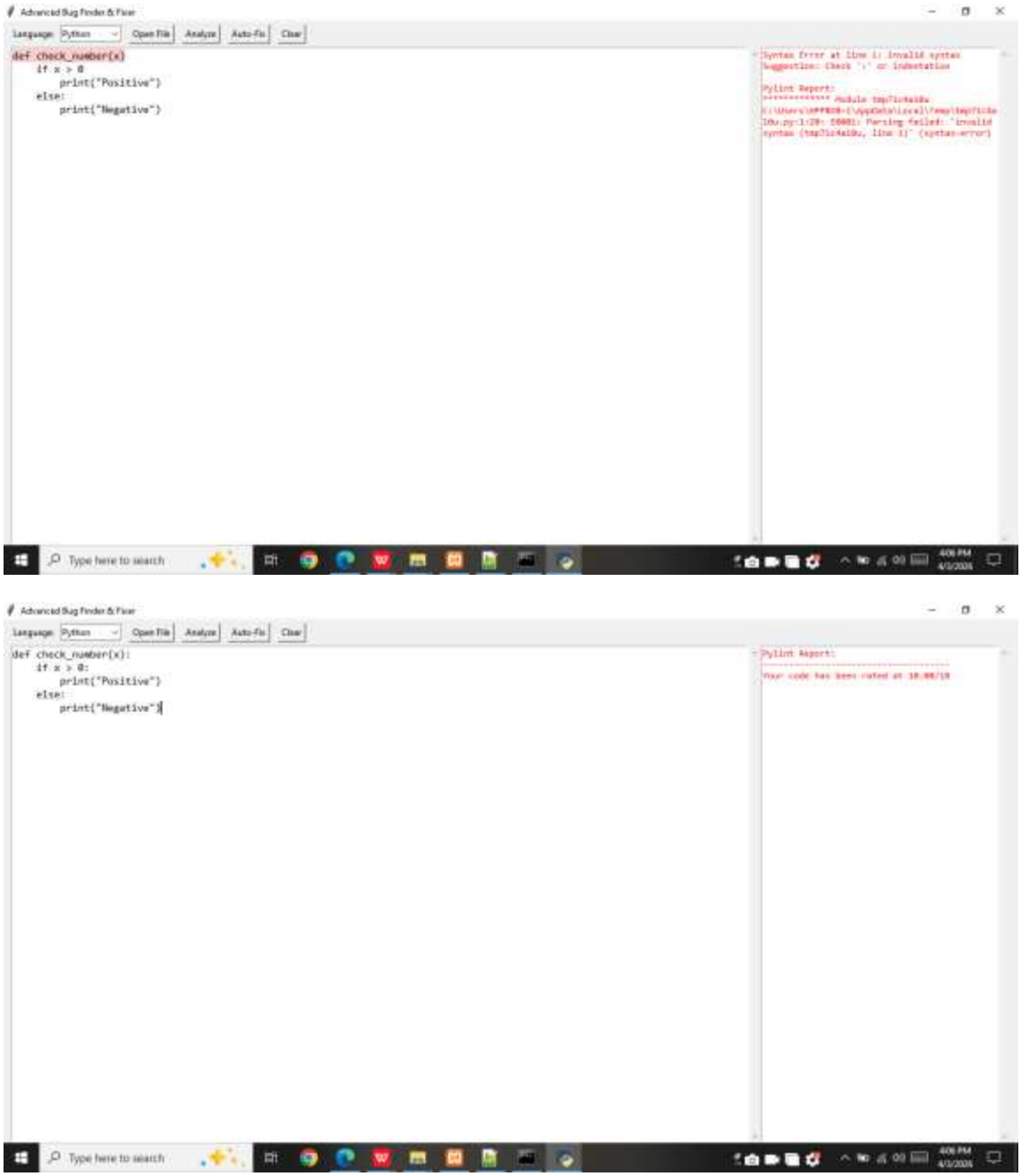
The system follows a layered architecture:

- Presentation Layer (GUI)
- Logic Layer (Analysis & AI)
- Integration Layer (Pylint, AST)

Data flows from user input → analysis → results display.

SYSTEM DESIGN IMAGES





VII. CONCLUSION

The **AI-Assisted Multi-Language Code Bug Detection and Auto-Fixing System** provides an efficient and intelligent solution for debugging software code. By integrating static analysis, heuristic AI techniques, and automated fixing mechanisms, the system enhances code quality and reduces development time. The application's user-friendly interface and multi-language support make it accessible to a wide range of users, from beginners to experienced developers. The Auto-Fix feature further improves productivity by eliminating common syntax errors automatically. Although the current system uses heuristic methods for bug prediction, it lays the foundation for

future integration of advanced machine learning models. This will enable more accurate and context-aware bug detection. In conclusion, the project demonstrates the potential of combining traditional debugging tools with AI techniques to create smarter development environments.

REFERENCES

1. Johnson et al., "Deep Learning for Code Analysis," IEEE, 2023
2. Li et al., "Automated Bug Detection Using ML," ACM, 2024
3. Smith et al., "Static Code Analysis Techniques," Springer, 2023
4. Brown et al., "AI in Software Engineering," IEEE, 2025
5. Kumar et al., "Heuristic Bug Detection Methods," Elsevier, 2023
6. Zhang et al., "AST-Based Code Analysis," ACM, 2024
7. Wang et al., "Hybrid Debugging Systems," IEEE, 2025
8. Lee et al., "Automated Code Repair," Springer, 2024
9. Patel et al., "Pylint and Static Analysis," IEEE, 2023
10. Chen et al., "Machine Learning for Bug Prediction," ACM, 2025
11. Gupta et al., "Software Debugging Tools Review," Elsevier, 2024
12. Singh et al., "AI-Based Code Fixing Systems," IEEE, 2025
13. Zhao et al., "Multi-language Code Analysis," ACM, 2024
14. Roy et al., "Pattern-Based Bug Detection," Springer, 2023
15. Kim et al., "Next-Gen Debugging Tools," IEEE, 2025