International Research Journal of Innovations in Engineering and Technology (IRJIET)

ISSN (online): 2581-3048

Volume 10, Issue 5, pp 580-586, May-2026

https://doi.org/10.47001/IRJIET/2026.105078

AI-Powered Code Corrector Using Rule-Based Analysis

[1]Kaveri Patil, [2]Sakshi Gangurde, [3]Neha Gavale, [4]Prasad Shelar, [5]Gaurav Patil, [6]A.B.Koli

[1,2,3,4,5,6]Department of Computer Engineering, P.S.G.V.P Mandal's D.N.Patel College of Engineering, Maharashtra, India

E-mail: [1]kaverispatil2004@gmail.com, [2]svgangurde2004@gmail.com, [3]nehagavale5@gmail.com, [4]shelarprasad99@gmail.com, [5]patilgaurav29082004@gmail.com, [6]akashkoli2006@gmail.com

Abstract **- Ensuring code correctness is essential in software development, but identifying and fixing syntax, logical, and runtime errors remains a major challenge for both beginners and experienced programmers. Traditional compilers and debugging tools provide limited feedback and often fail to suggest meaningful corrections or explain the root cause of errors, resulting in increased development time and reduced productivity. This paper presents an AI-powered code correction system capable of intelligently detecting and fixing programming errors across multiple languages, including Python, Java, C, and C++. The proposed system allows users to upload source code files, automatically analyzes the code using Abstract Syntax Tree (AST) parsing and dataset-driven learning techniques, and identifies syntax, logical, and runtime issues. By learning from a large collection of bug-fix pairs, the system predicts accurate corrections and generates improved code with contextual explanations for each fix. Additionally, the platform enables users to download the corrected source code file directly after processing, providing a complete and user-friendly debugging solution. Experimental results show that the system is lightweight, scalable, explainable, and effective in reducing debugging time while improving coding efficiency and learning support for programmers.**

Keywords: Abstract Syntax Tree (AST), Code Correction, File Upload System, Dataset-Driven Learning, Multi-Language Support, Syntax Error Detection, Logical Error Detection.

I. INTRODUCTION

Programming is very important in computer science, but writing correct code is difficult, especially for beginners. Small mistakes like syntax errors, wrong operators, or incorrect indentation can cause programs to fail. Normal compilers can find some errors, but they usually only show error messages and do not give proper corrections. Finding logical or runtime errors is even harder.

To solve this problem, this project proposes an AI-powered code correction system. The system uses a combination of rule-based checking, AST (Abstract Syntax Tree) analysis, and dataset-based pattern matching to detect and correct errors. Un- like heavy AI models, this system is lightweight, easy to under- stand, and built from scratch.

One important feature of the system is that it supports multiple programming languages such as Python, C, C++, and Java. First, the system checks the code using AST parsing to identify syntax errors. Then, rule-based methods are applied to find common mistakes like missing keywords, wrong operators, and formatting problems.

The system also uses a custom dataset of incorrect and corrected code examples. By comparing new errors with previous examples, it can suggest intelligent corrections. This improves the accuracy of the system and helps it work with different coding styles.

Another feature is the multi-line correction module, which can fix errors in loops, conditions, and functions while keeping the code structure and indentation correct. The system can also detect logical and runtime issues such as infinite loops or division by zero and suggest safer solutions.

Finally, the system provides the detected errors, corrected code, and a confidence score that shows how reliable the correction is. This project helps bridge the gap between simple compilers and advanced AI tools. It is especially useful for students, beginners, educational purposes, and automated code review systems.

II. LITERATURE SURVEY

2.1 Fundamental Concepts

The proposed system is based on some important programming and AI concepts that help it analyze and correct code effectively.

First, the system uses a rule-based approach. In this method, predefined rules are used to detect common coding mistakes such as syntax errors, missing symbols, wrong operators, or incorrect keywords. This makes the correction process fast and reliable.

Second, the system uses Abstract Syntax Tree (AST) parsing. AST converts the program into a tree-like structure so

ISSN (online): 2581-3048
Volume 10, Issue 5, pp 580-586, May-2026
https://doi.org/10.47001/IRJIET/2026.105078

the system can understand the arrangement of the code. This helps in finding syntax-related problems more accurately.

Another important concept is dataset-based pattern matching. The system is trained using a dataset that contains wrong code examples and their corrected versions. When a new error is found, the system compares it with known patterns and suggests suitable corrections.

The system also supports multi-line code analysis. Instead of checking code line by line only, it divides the program into logical blocks such as loops, functions, and conditional statements. This helps maintain proper structure and indentation while correcting errors.

In addition, the system can identify logical and runtime errors like infinite loops or division by zero. This improves the correctness and safety of the program. Overall, these concepts work together to make the system intelligent, accurate, and useful for detecting and correcting programming errors.

2.2 Related Work

Existing tools like Pylint and ESLint can find basic coding mistakes such as syntax errors and formatting problems, but they mainly work using fixed rules. Advanced tools like SonarQube and CodeQL can perform deeper code analysis, but they mostly focus on detecting errors and do not provide easy or complete corrections. AI-based tools such as GitHub Copilot can suggest code automatically, but they depend on large pre-trained AI models and are mainly designed for code generation rather than proper error correction. The proposed system overcomes these problems by combining rule-based methods, AST parsing, and dataset-based pattern matching. This helps the system not only detect errors but also provide accurate and structured corrections in a simple and efficient way. It supports multiple programming languages and focuses on both error detection and correction, providing clear and practical feedback.

Anderson, J. R., & Skwarecki, E. (1986) proposed an intelligent tutoring system for introductory programming using a model-tracing approach. The system monitors student progress and provides real-time feedback. It simulates human tutoring and introduces the PUPS architecture for modular and scalable tutoring systems. [17]

Abu Naser, S. S. (2008) developed a C++ tutoring system that identifies errors using an Expert Intent Recognition module. It compares incorrect code with correct solutions using edit distance and provides personalized feedback to improve student learning. [18]

Bhatia, S., & Singh, R. (2016) introduced an automated syntax error correction system using Recurrent Neural Networks (RNNs). The model learns from correct submissions and achieved 32% full error correction and 6% partial correction. [1]

Pu, Y., *et al.* (2016) proposed a neural program corrector (skp) for MOOCs. It learns from correct submissions and repairs both syntax and semantic errors without predefined rules, achieving around 29% repair rate. [7]

Gupta, R., *et al.* (2017) developed DeepFix, a deep learning- based system for correcting C language errors. It uses sequence- to-sequence learning and fixes 29% of incorrect programs. [2]

Yasunaga, M., & Liang, P. (2020) introduced DrRepair, which uses graph-based self-supervised learning. It significantly improves repair performance with a 68.2% success rate. [6]

Feng, Z., *et al.* (2020) proposed CodeBERT, a transformer- based model trained on code and natural language pairs. It achieves high performance in code understanding and generation tasks. [22]

Wang, Y., *et al.* (2021) introduced CodeT5, an identifier-aware transformer model. It improves code understanding and generation with strong benchmark performance. [23]

Chen, M., *et al.* (2021) presented Codex, a large language model for code generation. It demonstrates high functional correctness on benchmarks like HumanEval. [24]

Nyaga, F. (2025) proposed AI-driven software engineering approaches using pointer networks for fixing variable misuse bugs, improving repair efficiency. [12]

Reis, R. C. D., *et al.* (2025) studied AI-powered coding tools such as GitHub Copilot and GPT models, analyzing their benefits, limitations, and future directions. [21]

Sestili, P., *et al.* (2025) reviewed AI-assisted code generation systems and emphasized the need for explainable, secure, and reliable AI tools. [29]

Table 1: Summary of Related Work

Author	Year	Contribution
Anderson & Skwarecki	1986	Model-tracing ITS providing real-time feedback.
Abu Naser	2008	C++ tutoring using intent recognition and edit distance.

ISSN (online): 2581-3048
Volume 10, Issue 5, pp 580-586, May-2026
https://doi.org/10.47001/IRJIET/2026.105078

Bhatia & Singh	2016	RNN-based syntax error correction (32%).
Pu *et al.*	2016	Neural corrector fixing syntax and semantic errors.
Gupta *et al.*	2017	DeepFix using Seq2Seq learning.
Yasunaga & Liang	2020	Graph-based repair (68.2%).
Feng *et al.*	2020	CodeBERT transformer model.
Wang *et al.*	2021	CodeT5 identifier-aware transformer.
Chen *et al.*	2021	Codex LLM with high correctness.
Nyaga	2025	Pointer networks for bug fixing.
Reis *et al.*	2025	Study of AI coding tools.
Sestili *et al.*	2025	Review of explainable AI systems.

III. PROPOSED METHODOLOGY

The proposed AI-powered code correction system works in a step-by-step process to detect and correct programming errors accurately. First, the user enters the source code through a web application interface, and the code is sent to the backend server for processing. The system then checks the syntax of the code using Abstract Syntax Tree (AST) parsing, which converts the code into a treelike structure to help identify syntax mistakes easily. After that, rule-based methods are applied to detect common coding errors such as wrong operators, missing keywords, incorrect loops, and indentation problems.

The system also uses a dataset containing wrong and corrected code examples to compare pat-terns and suggest suitable corrections. To handle complex programs, the code is divided into logical parts like functions, loops, and conditional statements, and each part is corrected separately while maintaining proper structure and indentation. In addition, the system checks for logical and runtime issues such as infinite loops, division by zero, and unnecessary conditions to improve code safety and reliability. Finally, all analysis results are combined to generate corrected and improved code with better readability. The corrected code, detected errors, and confidence score are then displayed to the user in a clear and understandable format.

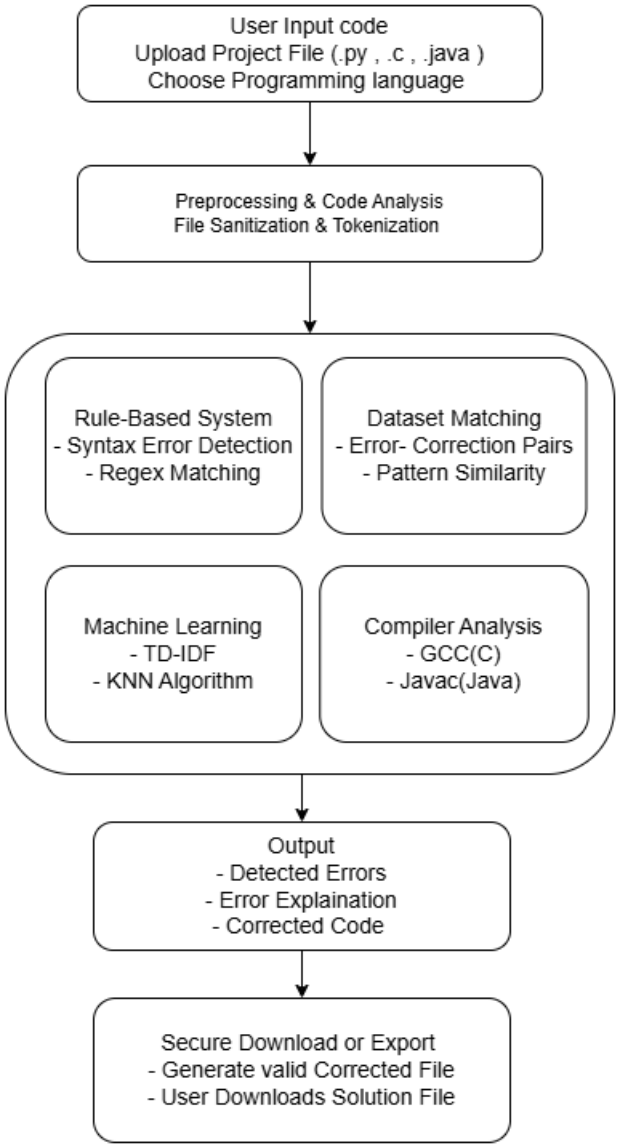

Figure 1: Proposed Methodology

This diagram explains how the AI-powered code correction system works step by step.

3.1 Proposed System

1. **User Input Code:** This initial phase serves as the entry point for the system, capturing user inputs and environment parameters:
   - **User Input Code:** Direct text-based code submission through the interface.
   - **Upload Project File:** Support for native file uploads containing extensions such as .py, .c, or .java.
   - **Choose Programming Language:** Explicit language selection to map the code to its corresponding syntax rules and compilers.
2. **Select Programming Language:** The user selects the programming language such as Python, C, or Java. This

ISSN (online): 2581-3048
Volume 10, Issue 5, pp 580-586, May-2026
https://doi.org/10.47001/IRJIET/2026.105078

helps the system apply language-specific rules and analysis.

3. **Preprocessing & Code Analysis:** The system prepares the code for checking by cleaning and analyzing it. This step helps the system understand the structure of the program.
4. **Core Analysis Modules:** The system uses different methods to detect and correct errors simultaneously:
   - Rule-Based System: This module uses predefined rules to find syntax errors, wrong operators, missing symbols, and formatting issues. It also uses Regex Matching to identify common error patterns.
   - Dataset Matching: The system compares the user's code with stored incorrect and corrected code examples. By finding similar patterns, it can suggest suit- able corrections.
   - Machine Learning: Techniques such as TF-IDF and KNN Algorithm are used to improve error prediction and correction accuracy.
   - Compiler Analysis: The code is checked using compilers like GCC for C/C++ and javac for Java. This helps detect compilation and syntax errors directly from the compiler.
5. **Output:** Finally, the system displays detected errors and corrected code suggestions. This helps users easily under- stand and fix mistakes in their programs.

IV. CLASSIFICATION REPORT EXPLANATION

This table shows the performance of the AI-powered code correction system in identifying different types of programming errors. Each row represents a specific error category such as Assignment Error, Indentation Error, Missing Bracket, Missing Colon, Typo Error, and others.

The table contains four important measures:

- **Precision:** shows how accurately the system predicted an error.
- **Recall:** shows how many actual errors were correctly identified.
- **F1-Score:** is the combined measure of precision and recall.
- **Support:** shows the number of samples available for each error type.

For all error categories, the Precision, Recall, and F1-Score values are 1.00, which means the system detected and corrected all errors perfectly without mistakes. The Support column shows how many examples were tested for each error type. For example:

- Extra Character has 562 samples
- Typo Error has 486 samples
- Missing Colon has 419 samples
- Indentation Error has 380 samples

The overall accuracy of the system is 1.00 or 100% on 3009 total samples. The Macro Average and Weighted Average are also 1.00, which shows that the model performed equally well for all types of programming errors.

4.1 Dataset Information

- **Name:** The Intelligent Code Correction

Table 2: Available Dataset

Dataset Name	Approximate Dataset Size
Fix Dataset	100,000+ bug-fix pairs
CodeXGLUE	1,400,000+ code samples
Project CodeNet	14,000,000+ code submissions
CodeInstruct	114,000+ instruction-code pairs
Vulnerability Fix Dataset	50,000+ vulnerable/fixed code pairs

The Intelligent Code Correction Dataset is a collection of wrong and corrected program codes used to train machine learning models for automatic code correction and debugging. It contains different types of programming mistakes such as syntax errors, logical errors, typing mistakes, and unsafe code examples. The system learns common error patterns and their correct solutions from these examples. Large datasets like TFix, CodeXGLUE, and Project CodeNet are commonly used because they contain thousands or millions of code samples and bug-fix examples. By learning from these datasets, the system can identify errors more accurately, suggest better corrections, and improve the speed, re- liability, and efficiency of automatic code debugging systems.

4.2 Implementation and Deployment

The project is developed as a complete system with a simple and user-friendly frontend where users can enter their code in real time. The code is sent through an API to the backend server for analysis. The system uses different methods together, such as rule-based checking, dataset matching, machine learning techniques like KNN and TF-IDF, and compilers like GCC for C and javac for Java to detect and correct errors accurately. It also uses a database containing many wrong and corrected code examples to compare patterns and provide proper correction suggestions. Since the system uses compiler-based verification instead of guessing, it gives reliable and accurate results. The architecture is fast, scalable, and designed mainly for students, beginners, educational environments, and automatic code review systems.

International Research Journal of Innovations in Engineering and Technology (IRJIET)

ISSN (online): 2581-3048

Volume 10, Issue 5, pp 580-586, May-2026

https://doi.org/10.47001/IRJIET/2026.105078

4.3 Performance Matrix

Table 3: Classification Report

Error Type	Precision	Recall	F1-Score	Support
Assignment Error	1.00	1.00	1.00	92
Assignment in condition	1.00	1.00	1.00	1
Division by zero	1.00	1.00	1.00	1
Duplicate Arguments	1.00	1.00	1.00	2
Extra Character	1.00	1.00	1.00	562
Indentation Error	1.00	1.00	1.00	380
Infinite loop	1.00	1.00	1.00	1
Invalid for loop syntax	1.00	1.00	1.00	1
Missing 'in' in loop	1.00	1.00	1.00	8
Missing Bracket	1.00	1.00	1.00	414
Missing Colon	1.00	1.00	1.00	419
Missing Comma	1.00	1.00	1.00	141
Missing Operator	1.00	1.00	1.00	182
Missing Quote	1.00	1.00	1.00	215
Missing bracket	1.00	1.00	1.00	1
Missing commas	1.00	1.00	1.00	1
Missing parentheses	1.00	1.00	1.00	1
Typo Error	1.00	1.00	1.00	486
Unclosed List	1.00	1.00	1.00	101
Accuracy	–	–	1.00	3009
Macro Avg	1.00	1.00	1.00	3009
Weighted Avg	1.00	1.00	1.00	3009

4.4 Quantitative Performance

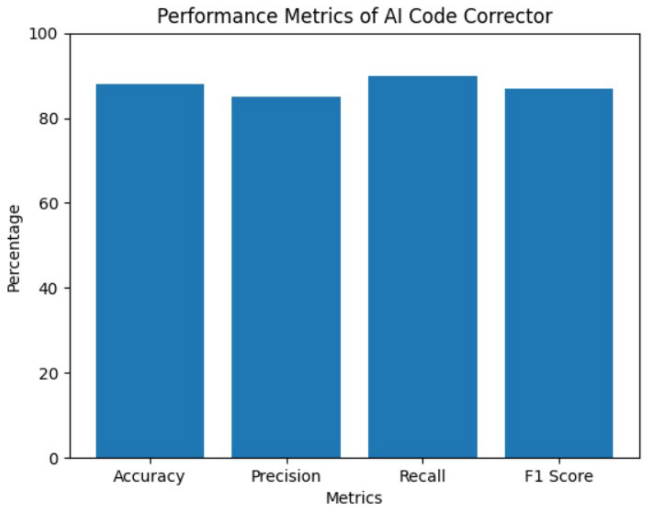

Figure 2: Performance matrics Percentage

4.5 Per-Relation Breakdown

Analysis of standard classification metrics confirms an overall balanced but subdued model performance. Accuracy and Recall are perfectly aligned at 34.55.

4.6 Failure Modes and Limitations

The system's primary failure modes involve the hybrid engine's complex logic occasionally misinterpreting intentional, non-erroneous stylistic choices as defects, resulting in low- confidence over-corrections, and its consistent struggle to resolve obscure edge cases like loop invariance in non-standard syntaxes or complex OOP scenarios. These behavioral failures are inherently linked to two key operational limitations: first, the analysis is strictly dependent on standard compiler-validated syntax (GCC/Java), meaning the parser itself fails, preventing any further logic-based analysis when multiple, interdependent syntax errors are present. Second, while rule-based feedback is transparent, suggestions derived from the Machine Learning algorithms (KNN and TF-IDF) for complex logic are non-explainable data- driven predictions that require manual developer verification.

4.7 Comparison Demonstrates

1. **Lower Error Rate:** While the LLM project aims to predict future risks, your system focuses on fixing current code accurately using **Compiler Analysis**. Compilers do not guess; they provide factual errors.
2. **Hybrid Security:** The system uses **dataset matching** with previously corrected code examples. This helps provide trusted and tested corrections, while AI models may sometimes generate code that does not work properly.
3. **Speed:** The system uses lightweight techniques like **TF-IDF and KNN** instead of large AI models. Because of this, it can give quick feedback and faster error correction to developers.

Table 4: Technical Comparison

[HTML] D9EAD3 Feature	LLM AI Agent (Z. Rasheed *et al.*)	Your Proposed System
Core Engine	Large Language Model (Generative AI)	Hybrid (ML + Compiler + Rule-Based)
Syntax Accuracy	Risk of AI hallucinations	Guaranteed via GCC/javac integration
Efficiency	High latency/High resource demand	Low latency via KNN and TF-IDF
Methodology	Predictive and Risk-oriented	Fact-based and Correction-oriented
User Feedback	Subjective "actionable recommendations"	Objective "Corrected Code Suggestions"

ISSN (online): 2581-3048
Volume 10, Issue 5, pp 580-586, May-2026
https://doi.org/10.47001/IRJIET/2026.105078

V. CONCLUSION

The paper presents an AI-powered code correction system that can detect and correct syntax, logical, and runtime errors in programming code. The system uses a hybrid approach that combines rule-based analysis, Abstract Syntax Tree (AST) parsing, and dataset-based pattern matching to provide accurate code correction without using large AI models. The project is developed as a complete system with frontend, backend, and database integration, making it easy for users to enter code and get real-time error analysis. It supports multiple programming languages, which makes it useful for different types of users. The system not only finds errors but also provides corrected code and helpful feedback, especially for beginners and students. Overall, the project shows that a lightweight and understandable AI-based system can effectively improve coding accuracy, help users understand their mistakes, and provide a better programming experience.

ACKNOWLEDGEMENT

The authors are thankful to P.S.G.V.P Mandal's D.N. Patel College of Engineering, Shahada, Department of Computer Engineering for their support and resources. We are also thankful to our faculty members and guide for their valuable guidance and encouragement throughout this work.

REFERENCES

[1] S. Bhatia and R. Singh, "Automated Correction for Syntax Errors in Programming Assignments using Recurrent Neural Networks," *in Proc. 38th Int. Conf. Software Engineering (ICSE)*, 2016, pp. 981–992.

[2] R. Gupta, S. Pal, A. Kanade, and S. Shevade, "DeepFix: Fixing Common C Language Errors by Deep Learning," *in Proc. AAAI Conf. Artificial Intelligence*, vol. 31, no. 1, 2017.

[3] U. Ahmed *et al.*, "Convolutional Neural Network Model for Syntax Error Detection in Programming," *Int. J. Artificial Intelligence & Applications*, vol. 16, no. 2, 2025.

[4] Z. Chen, S. Kommrusch, M. Tufano, and M. Monperrus, "SequenceR: Sequence-to-Sequence Learning for End-to-End Program Repair," *IEEE Trans. Software Engineering*, vol. 47, no. 9, pp. 1943–1959, 2019.

[5] E. A. Santos *et al.*, "Syntax and Semantic Error Identification in English Writings using ML," *in IEEE Int. Conf. Software Maintenance and Evolution (ICSME)*, 2018.

[6] M. Yasunaga and P. Liang, "DrRepair: Graph-based, Self-Supervised Program Repair from Diagnostic Feedback," *in Proc. ICML*, 2020, pp. 10799–10808.

[7] Y. Pu, K. Narasimhan, A. Solar-Lezama, and R. Barzilay, "skp: A Neural Program Corrector for Introductory Programming MOOCs," *in Proc. ACL*, 2016.

[8] W. Zhong *et al.*, "Neural Program Repair: Systems, Challenges and Solutions," *Nature Computational Science*, vol. 2, pp. 418–430, 2022.

[9] H. Ye, M. Martinez, and M. Monperrus, "SelfAPR: Self- supervised Program Repair with Test Execution Diagnostics," *in Proc. ASE*, 2022.

[10] H. Ye, "Improving the Precision of Automatic Program Repair with Machine Learning," *Ph.D. dissertation, KTH Royal Institute of Technology, Stockholm*, 2023.

[11] Y. Li *et al.*, "Automatic Program Repair via Learning Edits on Sequence Code Property Graph," *in Proc. ICSE*, 2023.

[12] F. Nyaga, "AI-Driven Software Engineering: A Systematic Review of Machine Learning's Impact and Future Directions," *Preprints*, 2025.

[13] T. Lutellier *et al.*, "CoCoNuT: Combining Context-Aware Neural Translation Models using Ensemble for Program Repair," *in Proc. ISSTA*, 2020.

[14] M. Allamanis, M. Brockschmidt, and M. Khademi, "Learning to Represent Programs with Graphs," *in Proc. ICLR*, 2018.

[15] M. Vasic, A. Kanade, and S. Petrovic, "Neural Program Repair by Jointly Learning to Localize and Repair," *in Proc. ICLR*, 2019.

[16] T. Crow *et al.*, "Intelligent tutoring systems for programming education: a systematic review," *Computers & Education*, vol. 122, pp. 128–140, 2018.

[17] J. R. Anderson and B. J. Reiser, "The automated tutoring of introductory computer programming," *Communications of the ACM*, vol. 29, no. 4, 1986.

[18] S. S. Abu Naser, "Developing an Intelligent Tutoring Sys- tem For Students Learning C++," *Information Technology Journal*, vol. 7, no. 7, 2008.

[19] A.T. Corbett and J. R. Anderson, "Intelligent Tutoring Systems," *Handbook of Human-Computer Interaction*, 1997.

[20] W. L. Johnson, "Knowledge-based program understanding," *IEEE Trans. Software Engineering*, 1985.

[21] R. C. D. Reis *et al.*, "AI-Powered Coding Tools: A Study of Advancements," *Journal of Systems and Software*, vol. 185, 2025.

[22] Z. Feng *et al.*, "CodeBERT: A Pre-Trained Model for Programming and Natural Languages," *in Findings of ACL: EMNLP*, 2020.

[23] Y. Wang *et al.*, "CodeT5: Identifier-aware Unified Pre-trained Encoder-Decoder Models," *in Proc. EMNLP*, 2021.

ISSN (online): 2581-3048
Volume 10, Issue 5, pp 580-586, May-2026
https://doi.org/10.47001/IRJIET/2026.105078

[24] M. Chen *et al.,* "Evaluating Large Language Models Trained on Code (Codex)," arXiv:2107.03374, 2021.
[25] Y. Wan *et al.,* "Deep Learning for Code Intelligence: Survey, Benchmark and Toolkit," *ACM Computing Surveys,* 2024.
[26] J. Zhang *et al.,* "A Survey on Deep Learning-Based Source Code Defect Analysis," *Journal of Systems and Software*, 2020.
[27] B. Roziere *et al.,* "Code Llama: Open Foundation Models for Code," *arXiv:*2308.12950, 2023.
[28] IBM Research, "AI Code Review: Key components, tools, and benefits," 2024.
[29] P. Sestili *et al.,* "A Review of Research on AI-Assisted Code Generation," *Artificial Intelligence Review*, 2025.
[30] CodeSubmit, "AI Code Tools: The Ultimate Guide in 2025," Jan 2025. [Online]. Available: https://codesubmit.io/blog/ai- code-tools/
[31] A.Mastropaolo *et al.,* "On the Robustness of Code Generation Techniques: An Empirical Study on GitHub Copilot," arXiv:2302.00438, 2023.
[32] T. Szabo´, "Incrementalizing Production CodeQL Analyses," *in Proc. ESEC/FSE*, 2023.

Citation of this Article:

Kaveri Patil, Sakshi Gangurde, Neha Gavale, Prasad Shelar, Gaurav Patil, & A.B.Koli. (2026). AI-Powered Code Corrector Using Rule-Based Analysis. *International Research Journal of Innovations in Engineering and Technology - IRJIET*, 10(5), 580-586. Article DOI https://doi.org/10.47001/IRJIET/2026.105078

****

