IJCRT.ORG ISSN : 2320-2882

INTERNATIONAL JOURNAL OF CREATIVE RESEARCH THOUGHTS (IJCRT)

An International Open Access, Peer-reviewed, Refereed Journal

CODE MENTOR AI: A GPT-BASED INTELLIGENT CODE REVIEW SYSTEM FOR PROGRAMMING EDUCATION

Mr. S. Kesav Rao, S. Keerthi, T. Nikhitha, Y. Rajasekhar, M. Yogeswara Rao

[1]Assistant Professor (Guide), U.G. Student

Department of Computer Science and Engineering

Avanthi Institute of Engineering and Technology, Visakhapatnam, India

Abstract: This project focuses on building a GPT-based intelligent code review system to support programming education by providing clear, accurate, and meaningful feedback to students. Instead of checking only whether the program output is correct, the system understands the logic, structure, and intent of the code, similar to how an experienced mentor would review it. It analyzes aspects such as correctness, readability, efficiency, and common logical mistakes, and explains errors in a way that helps students learn rather than just fix the code. The feedback is designed to guide learners step by step, without directly revealing final solutions, encouraging independent thinking and problem-solving. By adapting its explanations based on the student's skill level, the system offers a more personalized learning experience. This approach reduces dependency on manual evaluation and enables consistent, scalable, and high-quality code reviews. Overall, the system aims to bridge the gap between automated assessment tools and human teaching, making programming learning more effective and learner-friendly.

Index Terms: *GPT-based code review, intelligent code review system, programming education, automated feedback, code analysis, machine learning, natural language processing, AI in education, personalized learning, code quality assessment, error detection, logical error analysis, readability and efficiency, adaptive learning system.*

I. INTRODUCTION

The fast evolution of information and communication technologies has redefined the sphere of software development and learning programming to a large extent, allowing the development of intelligent systems that assist in real-time coding, debugging, and learning. Traditional code compilers and programming environments are more oriented towards code execution and the simplest form of error detection with no detailed feedback, performance analysis, or customised learning support. This tends to result in inefficient debugging, inability to comprehend errors, and minimal progress in coding abilities, particularly for beginners.

AI and data-driven methods have become viable solutions that can be used to improve programming platforms by offering automated error detection, code quality evaluation, and actionable insights. Intelligent systems can detect patterns of errors through the analysis of user-written code, evaluate code across multiple dimensions such as syntax, logic, and efficiency, and deliver explanations in a human-friendly manner. These capabilities help fill the knowledge gap between writing code and comprehending it, making the process of learning more engaging and successful.

This project introduces a Python Smart Compiler System — an AI-based web platform that offers secure code execution and advanced analysis capabilities. The system combines safe execution mechanisms, intelligent error detection, code quality scoring, complexity analysis, AI-based

explanations, and a personalised mentor feedback system. Together, these components create an efficient, scalable, and accessible coding environment for programmers of all levels.

II. OBJECTIVES

The main objective of the project is to design and develop a Python-based Smart Compiler System that is an AI-driven, secure, interactive, and intelligent platform for writing, executing, and analysing code. The system is oriented towards enhancement of classical programming environments by offering real-time error detection, explanations, code quality assessment, and an enhanced user learning experience. It focuses on reducing debugging time, deepening code comprehension, and making programming more accessible and efficient for both learners and developers.

The technical objective of the project is to develop a secure and scalable web-based compiler founded on safe code execution methods, intelligent error analysis, and automatic grading of code quality. The application will provide actionable insights, track user progress, and facilitate adaptive learning through an AI-based explanation system, complexity analysis, and a personalised mentor system. This combination of modern technologies will ensure coding efficiency, improved understanding of programming principles, and an enriched overall development experience.

III. LITERATURE SURVEY

The concept of automated code compilation systems has been extensively studied to ease the programming task and minimise the need for manual debugging. Early compilers were primarily created to translate source code into runnable code and extract syntax errors. As integrated development environments (IDEs) evolved, features such as syntax highlighting, primitive debugging, and code suggestions were introduced. Nevertheless, the absence of intelligent feedback systems and detailed explanations has remained a limitation, particularly for learners [1][2].

With the introduction of Artificial Intelligence and machine learning technologies, research attention shifted to intelligent programming systems. AI-driven solutions have been demonstrated to deconstruct code, identify error patterns, and produce human-interpretable explanations that enhance comprehension. Studies have explored automated error analysis, code quality assessment, and complexity analysis using various mechanisms including pattern recognition, static analysis, and natural language processing [3][4][5]. Additionally, research on secure code execution environments has highlighted the use of sandboxing and subprocess isolation to safely execute user-generated code [6].

Recent papers have discussed the development of web-based smart compilers that integrate frontend interfaces with backend processing systems to provide real-time coding assistance. Such systems rely on RESTful APIs to organise code execution, error analysis, and feedback generation. Studies have also explored AI assistants and chatbots for interactive learning [7][8][9]. The Python Smart Compiler System proposed in this paper advances this work by combining secure execution, intelligent diagnostics, code quality scoring, and AI-assisted explanations into a single unified system — offering a more efficient, learner-centred, and scalable solution for modern programming education [10][11][12][13][14][15].

IV. SYSTEM ARCHITECTURE

The system architecture of Code Mentor AI follows a layered design with several interconnected modules that together enable intelligent code execution and analysis. The architecture is structured across seven distinct layers, each responsible for a specific set of functions:

4.1 User Layer

This is the layer where users interact with the system through web browsers on desktop or mobile devices, where they can write, execute, and analyse Python code in real time.

4.2 Frontend Layer

Built using HTML, CSS, and JavaScript, this layer incorporates the CodeMirror editor to provide an interactive and user-friendly programming interface with syntax highlighting and real-time responsiveness.

4.3 Backend Layer

The backend is implemented in Python using the FastAPI framework and handles core system functions including code execution, error analysis, code quality grading, complexity analysis, and API request processing.

4.4 Execution Layer

This layer provides safe code execution through subprocess isolation, configurable time limits to prevent infinite loops, and restricted access to unsafe modules, ensuring protection against malicious or dangerous code.

4.5 AI Integration Layer

Developed to integrate AI services, this layer provides users with detailed error descriptions, code insights, chatbot support, and personalised learning recommendations powered by GPT-based models.

4.6 Database Layer

User submissions, error history, performance statistics, and mentor insights are stored in a SQLite database. This enables progress tracking and delivery of user-specific, data-driven feedback.

4.7 Integration Layer

RESTful APIs connect the frontend and backend layers, ensuring seamless, real-time communication and interoperability across all system components.

V. METHODOLOGY

The Python Smart Compiler System follows a structured, modular methodology that combines frontend technologies, backend processing, AI-based services, and database systems into a unified architecture. The implementation is divided into four main phases:

5.1 User Interface and Input Processing

The web interface is designed using HTML, CSS, and JavaScript, configured with the CodeMirror editor for syntax highlighting and an improved coding environment. JavaScript handles client-side interactions including input validation, button functions, and API calls, ensuring that user inputs are captured accurately and forwarded to the backend with minimal latency.

5.2 Backend Analysis and Processing

The Python FastAPI framework serves as the backend and manages all fundamental system functions. It handles user requests, code submissions, error analysis, code quality scoring, and complexity analysis. Systematic communication between components is maintained through RESTful APIs, enabling accurate and timely result generation and feedback delivery.

5.3 AI Implementation and Integration

The system employs a secure execution engine that runs user code as an isolated subprocess with configurable time constraints to prevent infinite loops and unchecked operations. AI models are integrated to provide detailed mistake explanations, code insights, and chatbot-based guidance. This combination enhances both system security and the quality of learning support provided to users.

5.4 Data Storage and Learning Management

User submissions, error history, performance metrics, and mentor insights are stored in a SQLite database. This enables progress monitoring, identification of frequent error patterns, and generation of personalised feedback. The database schema is designed for consistency and efficient data retrieval.

5.5 System Design Principles

The system is built on a modular architecture in which the frontend, backend, execution engine, AI services, and database are autonomous components connected via APIs. This design enhances maintainability, scalability, and flexibility for future enhancements. A three-tier architecture separates the Presentation Layer (HTML, CSS, JavaScript), the Application Layer (FastAPI Python modules), and the Data Layer (SQLite), ensuring better performance, security, and simplified debugging.

API-based communication via RESTful interfaces ensures structured JSON data exchange, enhanced system interoperability, and the potential for future integration with mobile or external applications. Real-time processing mechanisms enable live program output, instantaneous error detection and description, and immediate code quality scores. Security is maintained through subprocess isolation, blocking of

dangerous modules and functions, timeout controls, input verification, and a restricted execution environment.

VI. EXPERIMENTAL RESULTS

The Python Smart Compiler System was developed as a web-based application and tested for functionality, speed, and performance under various conditions. Users were able to write and execute Python code through the interface, with client-side validation ensuring appropriate input handling before requests were sent to the backend. The FastAPI-based execution engine managed code processing in real time, while the secure execution environment prevented unsafe operations and infinite loops through timeout mechanisms.

The system generated code quality scores, complexity analysis results, and error explanations with suggested fixes immediately upon execution. User submission records and error profiles were stored in the SQLite database, enabling retrieval of personalised mentor insights. The results confirm that the system is reliable and effective in maintaining safe code execution, delivering real-time feedback, and supporting an improved learning experience.

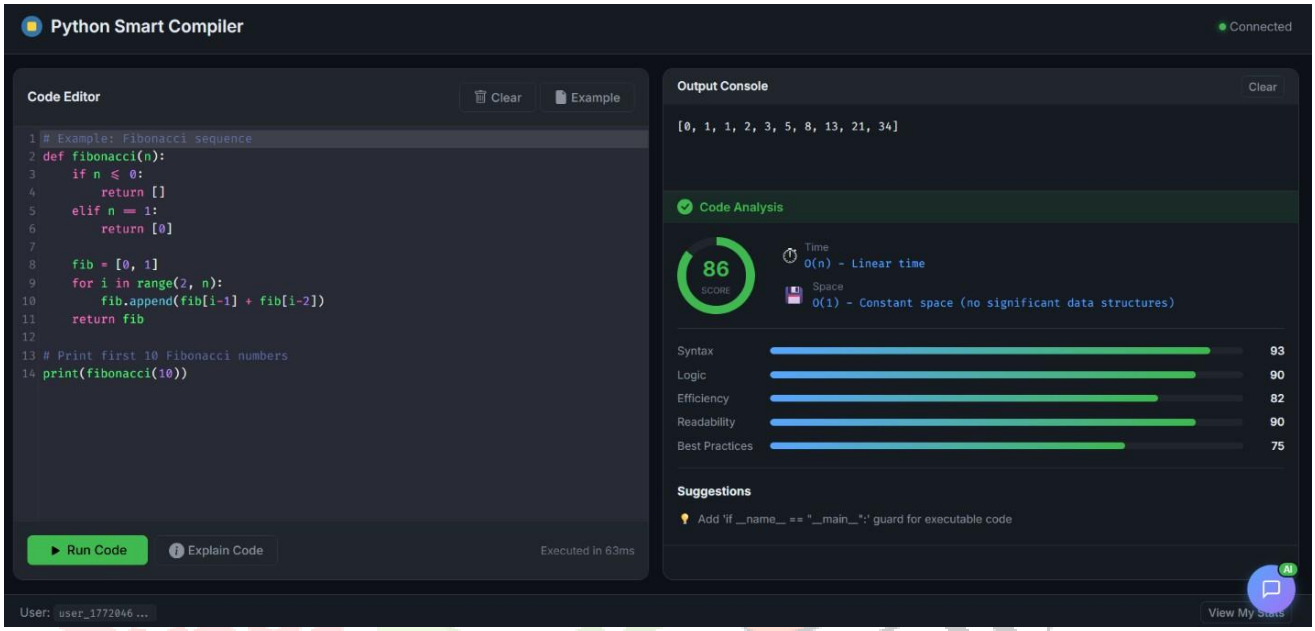

Fig. 1: Main interface of the Code Mentor AI system.

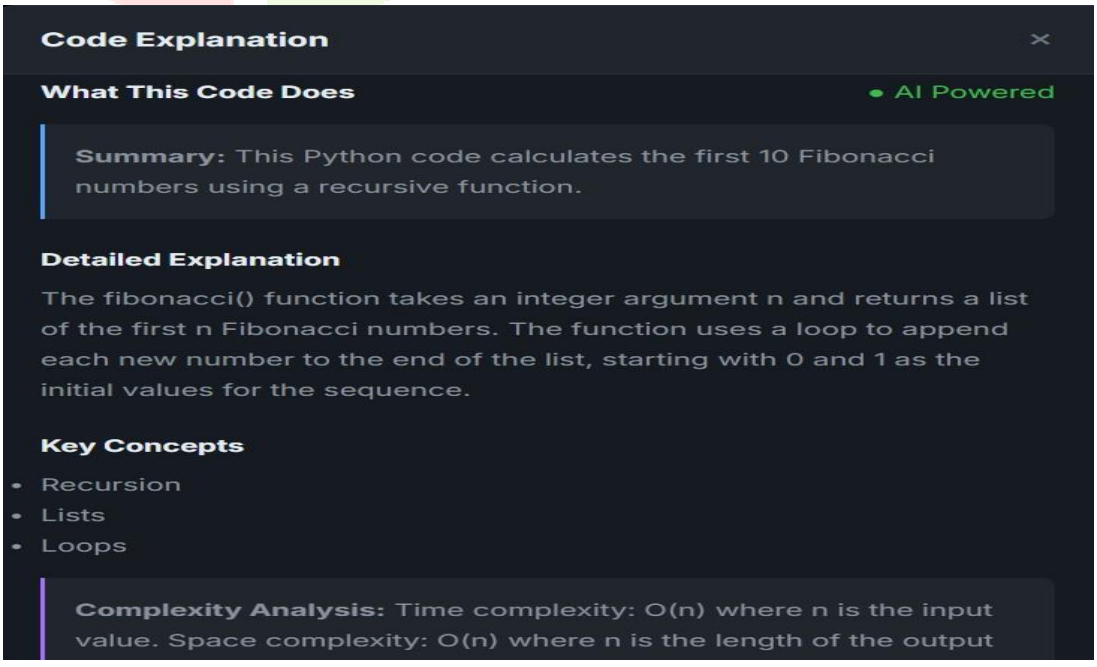

Fig. 2: Code explanation panel showing AI-generated feedback.

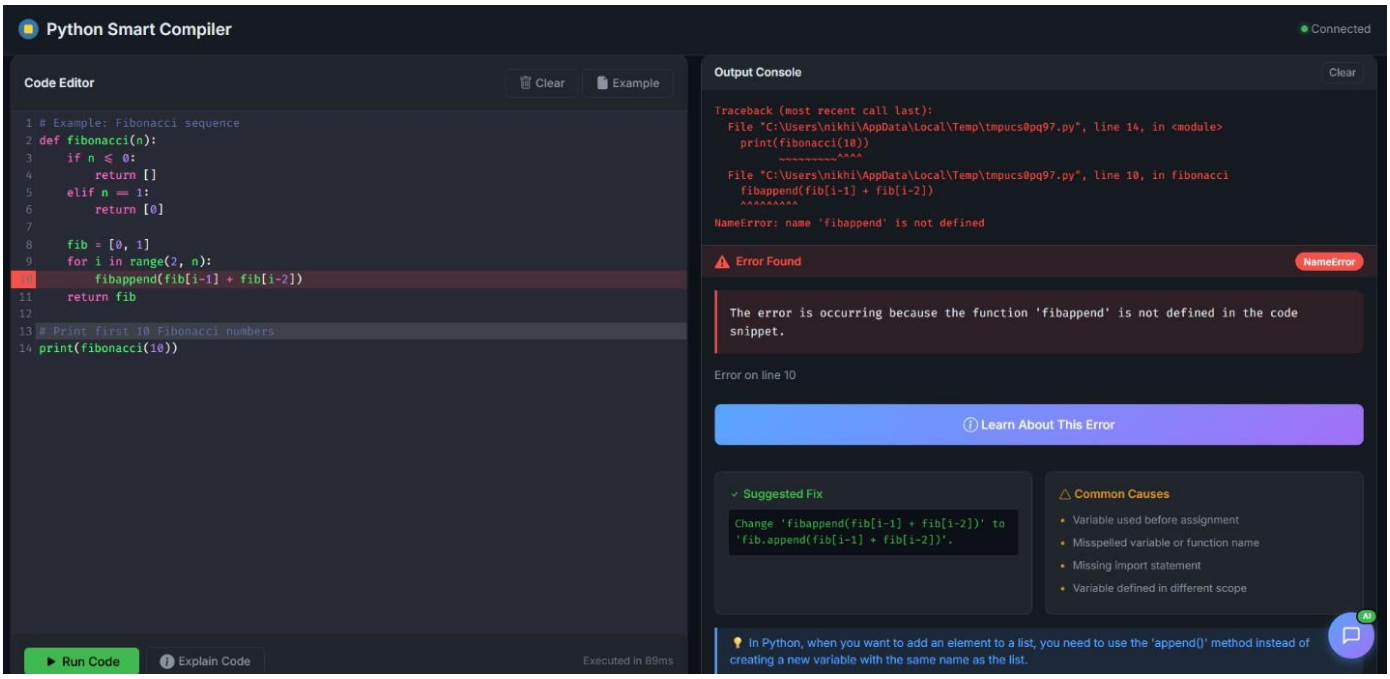

Fig. 3: Real-time error detection output.

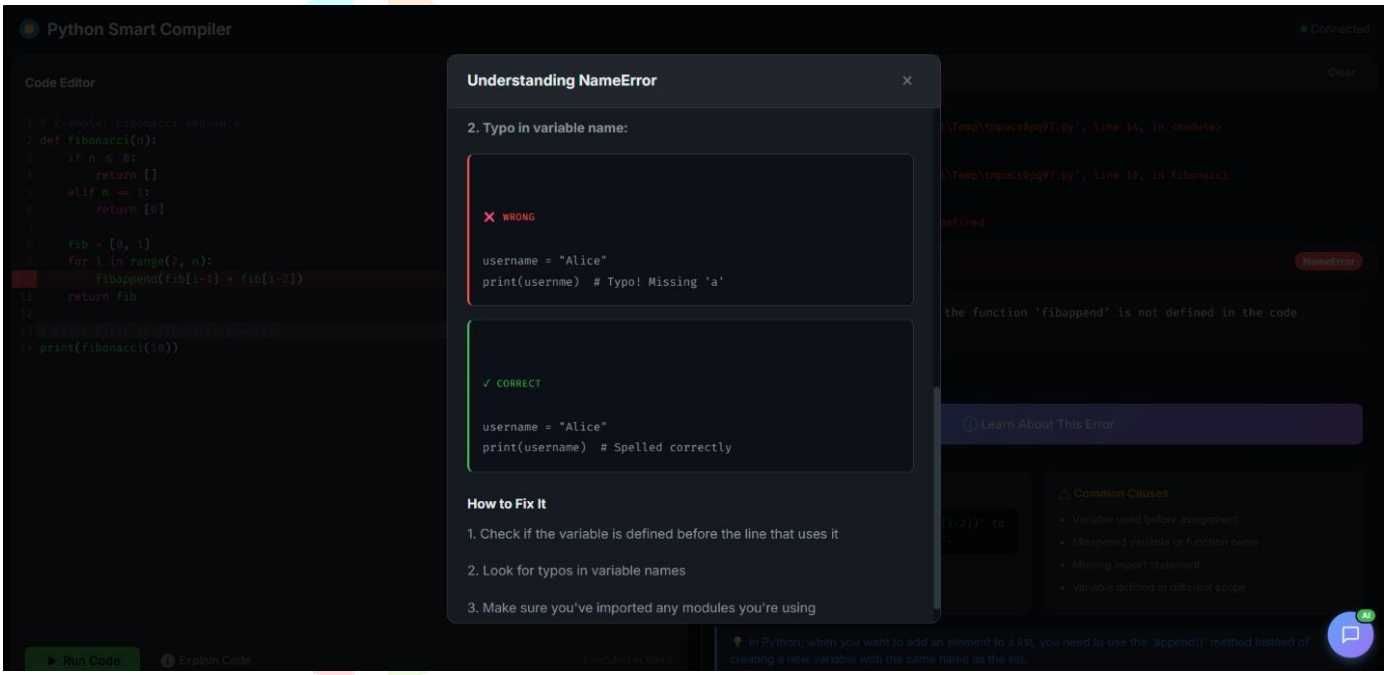

Fig. 4: Detailed error explanation with suggested corrections.

Fig. 5: Learning statistics dashboard displaying user progress.

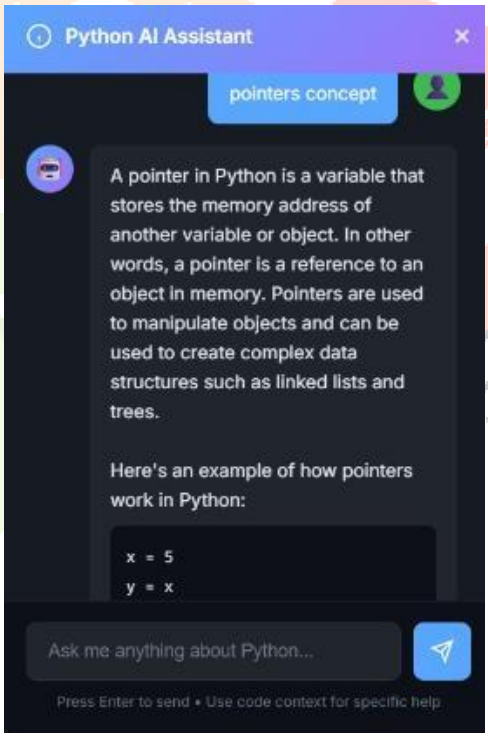

Fig. 6: Python AI assistant chat interface.

VII. CONCLUSION

The Python Smart Compiler System represents a practical demonstration of how intelligent technologies can be applied to advance programming education. By combining web technologies, a FastAPI server-side framework, and a SQLite database, the proposed system delivers a safe, fast, and live platform for writing, executing, and analysing Python code. Its core features — including secure code execution, detailed error analysis, code quality rating, and complexity assessment — collectively reduce the time spent on manual debugging, accelerate code comprehension, and enhance the overall learning experience.

The GPT-based code review component bridges the gap between automated assessment and genuine mentorship by adapting explanations to the learner's skill level and guiding students toward solutions

without revealing them directly. This modular and scalable architecture ensures the system can be extended with additional language support, predictive analytics, and mobile accessibility in future work.

ACKNOWLEDGEMENT

The authors express sincere gratitude to all those who supported the successful completion of this project. Special thanks are due to the project guide for their valuable guidance, continuous support, and constructive suggestions throughout the development process. The authors are also grateful to the faculty members of the Department of Computer Science and Engineering at Avanthi Institute of Engineering and Technology, Visakhapatnam, for providing the necessary resources and technical knowledge that shaped this work.

REFERENCES

[1] U. Cihan et al., "Automated Code Review in Practice," arXiv preprint, 2024.
[2] R. Tufano and G. Bavota, "Automating Code Review: A Systematic Literature Review," arXiv, 2025.
[3] R. Tufano et al., "Deep Learning-based Code Reviews: A Paradigm Shift or a Double-Edged Sword?" arXiv, 2024.
[4] Y. Denisov-Blanch et al., "Predicting Expert Evaluations in Software Code Reviews," arXiv, 2024.
[5] Y. Ramaswamy, "Automated Code Reviews and Static Analysis in DevOps Workflows: Tools, Challenges, and Trends," NeuroQuantology, 2024.
[6] D. Badampudi et al., "Modern Code Reviews: Survey of Literature and Practice," ACM Computing Surveys, 2024.
[7] M. Vijayvergiya et al., "AI-Assisted Assessment of Coding Practices in Modern Code Review," arXiv, 2024.
[8] "Structuring Meaningful Code Review Automation in Developer Community," Engineering Applications of Artificial Intelligence, 2024.
[9] "Automating Modern Code Review Processes with Code Similarity Measurement," Information and Software Technology, 2024.
[10] C. Negri-Ribalta et al., "A Systematic Literature Review on the Impact of AI Models on Code Generation Security," Frontiers in Big Data, 2024.
[11] F. F. Xu et al., "A Systematic Evaluation of Large Language Models of Code," ACM SIGPLAN Symposium, 2022.
[12] Y. Wu et al., "How Effective are Neural Networks for Fixing Security Vulnerabilities," ISSTA Conference, 2023.
[13] International Research Journal, "AI-Based Code Review System Using Deep Learning Techniques," IRJET, 2025.
[14] IJRASET, "AI Code Review Assistant: A Web-Based Solution for Automated Code Analysis," 2025.
[15] "Artificial Intelligence for Source Code Understanding Tasks: A Systematic Mapping Study," Information and Software Technology, 2025.