International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 14, Issue 04, ICTEM 2.0 (2026)

AI Code Mentor: A Multi-Level Intelligent Framework for Automated Code Analysis and Learning Support

Mr. Puneet Kumar
Department of Computer Science & Engineering
Moradabad Institute of Technology
Moradabad, India
puneetkchahal@gmail.com

Yash Singh
Department of Computer Science & Engineering
Moradabad Institute of Technology
Moradabad, India
yashsingh3k@gmail.com

Trapti Chauhan
Department of Computer Science & Engineering
Moradabad Institute of Technology
Moradabad, India
traptichauhan9027@gmail.com

Shrishti Yadav
Department of Computer Science & Engineering
Moradabad Institute of Technology
Moradabad, India
shrishtiyadavy@gmail.com

Vinamre Pandey
Department of Computer Science & Engineering
Moradabad Institute of Technology
Moradabad, India
vinamrepandey9966@gmail.com

Abstract –

The AI Code Mentor Project is an innovative approach to combine Artificial Intelligence with software development tools, to help inexperienced programmers learn how to create, debug and improve their own code through tailored, intelligent assistance. This innovative approach effectively replaces traditional software development environments with a way for students to engage with their code as a learning opportunity to aid their own learning with various techniques, including Syntax Analysis, Logic Testing and Refactoring, as well as leveraging the capabilities of Large Language Models for increased usefulness and efficiency.

The primary function of AI Code Mentor is to help those who are new to programming tackle one of the most significant challenges they face as they learn to program: not knowing how to read and interpret compiler-generated error messages, as well as how to resolve issues that may arise. While numerous online resources exist to provide some level of assistance for beginner programmers by providing basic help, none of the current online resources focus on personalized mentoring on how to interpret compiler-generated error messages, nor do any of these resources provide examples of what constitutes best practices or logical reasoning when developing solutions for compiler-generated error messages. The AI Code Mentor uses a Complete Diagnostic Process that attempts to mimic an experienced mentor's experience. The diagnostic process begins with checking the code's syntax to ensure that it is syntactically correct using Python's built-in compiler. After determining that the syntax is correct, the AI Code Mentor will check the code for any logical errors by executing it in a secure, isolated testing environment. The AI Code Mentor then restructures the user's solution into a visual representation (abstract) by using a Structured Representation Tree (SRT). The AI Code Mentor then provides current artificial intelligence technology powered by GPT models to assist in guiding users in how to apply the recommendations to future code development.

Keywords- Programming Education, Code Analysis, Syntax Error Detection, Logic Testing, Refactoring, Learning Support System.

I. INTRODUCTION

Programming is an essential skill, and there is a high demand in almost every industry today. As a result, so many learners face problems at the initial level of programming. One of the most common barriers to success for many learners is the difficulty of interpreting error messages that are generated by programmers when they try to write their own code. Error messages are typically very technical in nature due to even programmers are not able to understand the error and the way how to fix it. Therefore, when the programmer receives error messages, they often end up improving their code without fully understanding the reason for the errors. This can significantly obstruct the long-term growth of learning.

This problem becomes even more challenging in an education system such as online classrooms or classrooms that contain many learners at once, as instructors may be unable to provide each student with the type of individualised feedback that they require in order to be successful in programming. Unfortunately, there is often a gap between what students

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 14, Issue 04, ICTEM 2.0 (2026)

require in order to be successful and the resources that traditional methods or tools for teaching programming can provide. New advancements in artificial intelligence (AI), such as Natural Language Processing (NLP) and Code Analysis, are providing excellent solutions for better Learning. The advantages of AI include the ability to read and understand code, find and describe errors in simple English, as well as help new programmers with support in the form of feedback that is relevant and easy to understand at their level of experience.

The AI Code Mentor is designed in a way that helps new programmers or learners to analyse their code step-by-step. It analyses the code not only for the correctness of syntax, but also the logic (what it is supposed to do), structure (how the overall structure of the program is), and style (the actual code itself). The AI Code Mentor also provides background information to the beginners about the reason for the feedback they are receiving (Amiri et al., 2025).

The AI Code Mentor not only provides a way for beginners to fix any mistakes they have made. the AI Code Mentor allows the new programmer to develop his or her problem-solving skills by giving him or her regular feedback with a human-readable format in case he or she is not able to understand the error or how to fix it. The AI Code Mentor helps to create a link between automated tools and human teachers, which makes learning programming easier, accessible, productive, and ethical (Zhang et al., 2024).

II. RELATED WORK

For many years, several different tools have been created to allow students to program more easily. However, most of the tools that have been made available to students focus primarily on only a few key areas of learning how to program. For example, Integrated Development Environment (IDE) tools, such as Visual Studio Code or PyCharm, can provide beginner programmers with visual indicators (syntax highlighting) and basic warnings for errors in their code. Nevertheless, they do not provide any advice or feedback for the beginner programmer to better understand their mistake(s). Another example is that of Geeks for Geeks, who offer large collections of coding examples and discussion forums in order to help students; this type of help, however, is only general and does not pertain directly to the student's code. Online judging systems (OJS) like HackerRank or CodeChef can help students in determining the output of their code; however, these types of systems do not assess the structure of the program or provide any suggestions on the style or design of the code. Consequently, it is common for many students to not be able to locate logical errors in their code, or to develop better techniques for writing clean and modular code (Das et al., 2016; Amiri et al., 2025).

The emergence of AI in the field of programming education has been documented in studies related to automated syntax analysis. In particular, structured parses and tracebacks were found to enhance the clarity of error messages while also alleviating cognitive overload for novice students. Regarding work on dynamic testing and sandbox execution, there is considerable potential to identify logical errors through analysis of expected output vs actual output during control execution (Schez et al., 2020). In addition to the above-mentioned topics, there are currently active research projects being conducted in higher education related to the use of Abstract Syntax Trees (ASTs). Among these projects is the investigation of using ASTs to find structural antipatterns and to provide direct recommendations on ways to refactor code, which is likely to be a significant support for students developing good programming practices. Furthermore, some emerging research verifies that employing large language models allows for the generation of natural-sounding explanations similar to those provided by human mentors, which aids in customising the student's learning experience (Gupta et al., 2017).

Most current offerings, however, continue to work in isolation from one another. For instance, while syntax checkers can identify mistakes in syntax, there is often no further elaboration on what went wrong. Testing frameworks can verify correctness, but don't provide any guidance on why something was checked or what it meant. Similarly, although style linters can assess the quality of code, they do not typically inform new coders of the rationale behind their recommendations and ultimately do not provide any assistance in learning how to code properly.

Furthermore, large language model systems may produce clear, articulate responses to questions asking for explanations related to errors made in code; however, they cannot often analyse execution traces, apply real-world code diagnostics or perform structural analyses, thus limiting their effectiveness as reliable sources of instructional support. Therefore, no single unified software-enabled platform offers a combined solution for performing syntax analysis, logic testing, structural evaluation, and an individualised tutorial feedback mechanism. Moment software attempts to address this void by uniting traditional analysis methodologies with contemporary AI mentoring; it provides a systematic and coordinated framework for learning how to programme effectively – designed specifically for novice programmers (Gupta et al., 2017; Phung et al., 2024).

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 14, Issue 04, ICTEM 2.0 (2026)

III. PROBLEM STATEMENT

3.1 Architecture

A recommended architecture consists of several interconnected components, with each part being responsible for one aspect of the functionality of all submitted programming by the student. The code entry layer is the first entry point for the submission of Python programs and optional test cases, while source code submissions through this layer will be accepted in their raw state, but are formatted before subsequent processing through diagnostic layers. Once the formatting of the submitted Python code is verified, the next step in the workflow is to compile the program in the first level of diagnostic processing, with errors being recorded in the traceback output generated through local compilation. By submitting a program for compilation, the system will identify all syntax errors that exist in the submitted code (such as missing or incorrectly formatted characters or incorrect indentation levels) and provide clearer messages to aid beginning programmers in understanding the source of the compilation error. Once the syntax stage is completed, and if the program can be executed, the system advances to the next phase program (Alshaikh, Tamang, and Rus, 2020).

The second stage focuses on logical analysis and is based on sandboxed execution. In a controlled setting, the software is being run, and the subsequent results from the software are compared against the expected results (these may be supplied by the student via their own pre-constructed test cases or through the built-in test cases). As such, the comparison between the two sets of outputs will allow the software to indicate the areas where the student's program logic did not agree with what was anticipated. The second layer of this architecture looks at the source code that has been written by the student in regard to its style and structure, rather than just the execution of the program (Alshaikh, Tamang, and Rus, 2020). Rather than checking if the student's program runs or provides output, the second layer uses abstract syntax trees to create a visual representation of the code's structure and its patterns. Once the structure of the code has been identified by the AI, the architecture will provide students with sample solutions or ideas that they could use to improve their code. These sample solutions could include ideas regarding increasing the modularity and readability of their code and providing the same level of consistency in naming conventions as outlined in Python documentation (Schez et al., 2020).

Finally, the third layer will leverage AI to enhance this entire process. Once the system has collected information about syntax, logic and structural issues, it prepares a summary that is passed to a large language model. The model is not used to run the code but to provide guidance, explanations and educational support. As of now, the type of feedback being delivered resembles a mentor's direction to help them achieve success by showing what has been done incorrectly and explaining how that mistake occurred, as well as providing suggestions on what can be done to improve. Learners may review their results through the dashboard and document their progress on an ongoing basis by viewing the previous attempts. The architecture, therefore, creates a smooth and integrated workflow, where technical analysis and human-oriented explanation work together.

3.2 Methodology

In following a stepwise approach that reflects the analytical phases of the architecture, the system accepts code submissions from students and will check for the validity of both formatting and overall structure prior to proceeding further with verification processes. The system will then execute a static syntax checking process (i.e., compiling code) and generate a traceback of the errors, providing a simple explanation for these errors and suggestions for improvement or correction. The execution phase of the project occurs after a student compiles their code. During this phase, the code executes in a "sandbox," which limits the code's activity to areas that do not cause damage or pose a threat to the user. This allows the project to continue safely while also monitoring and evaluating the way the code runs while in the sandbox. The system will track all input and output during the execution phase and will record any behaviours or performance that differ from the user's specified parameters.

After the structural analysis of an application is completed, the next step is to develop a "structured analysis" of the application based on the executed portion of the code's structural components. In addition to developing a structured analysis, the application's abstract syntax tree will be examined for various patterns, including the following: nests of logic that are excessively deep; methods that are not included; duplicate logic; and inconsistent naming conventions (Taulli, 2024).

The system not only identifies and highlights these issues but also makes recommendations to improve the student's ability to produce code that is easy to read and has a well-structured or modular design. Once a sufficient amount of data has been collected, a prompt is created and submitted to the AI model. The AI model has one primary purpose: to convert technical-oriented diagnostics into easy-to-read educational content. The system does a careful job of limiting the amount of information sent to the model so it can reduce the amount of processing required and thus reduce the process latency. Thus, the methodologies combine automated diagnostic capability with expressiveness and the ability to mentor students (Das et al., 2016).

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 14, Issue 04, ICTEM 2.0 (2026)

3.3 Mitigating Considerations

To mitigate the previously identified practical issues involved in collaborative software development, the design choices made by the authors have incorporated several features and functionalities (Amiri et al., 2025). Given that LLM may produce erroneous, ambiguous, or contradictory output, the software application itself will always run separate technical checks. The LLM will provide guidance, and guidance will not include verification functionality. The software's LLM-generated advice will always require verification through empirical testing, structural validation before being presented to users as a final recommendation (Pankiewicz and Baker, 2023; Karsa and Goldschmidt, 2025).

Performance limits have been addressed through the modular nature of execution methods, caching of previously executed code and limiting the number of code lines sent to the LLM. Security concerns were also addressed by executing students' programs in a secure sandbox, which isolates students from file access, disallows students from networking with the outside world, and allows the software to time out execution. If students do not have adequate test case coverage for their programs, the software might prompt the students to generate more extensive test cases, or it may provide the capability to create automatic test cases to find errors that have not been tested for in their programs. Thus, the overall design of the software allows for the creation of both practical applications and user-safe applications while still providing students with individualised learning assistance (Karsa and Goldschmidt, 2025; Amiri et al., 2025).

IV. PROPOSED SYSTEM

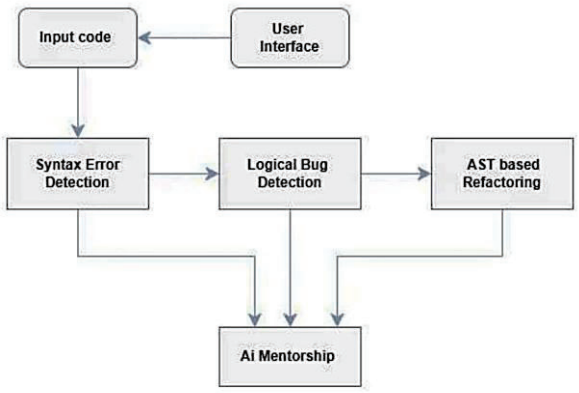
The objective of the **AI Code Mentor Project** is to help novice programmers through an intelligent, modular, and expandable learning system that enables users to identify errors in their coding and enhance the quality of their work by acquiring effective problem-solving skills. The system is entirely software-driven and does not depend on specialised hardware or IoT devices. Using dynamic and static program analysis techniques, sandbox execution, and large language models (LLMs), this system will offer developers personalised insights and recommendations regarding their code (Schez et al., 2020; Yang et al., 2024; Karsa and Goldschmidt, 2025; Ali et al., 2025).

A. System Architecture

The layered architecture of this system will enable an easily understandable, flexible model for long-term maintenance. the top layer of the system is the user interface, where the student interacts with the system, and below that is the application layer. This layer is responsible for managing the orchestration of the diagnostic workflow. The bottom layer of the system is for storing data so that every attempt and feedback submitted by a learner can be preserved for future use.

The frontend is the means by which the user types or uploads their code, configures the input settings, and views the results from the analysis report. The front end consists of an editor that is web-based so that students can type and correct their programs with the benefit of syntax highlighting. The dashboards allow learners to see their progress on previous submissions and make it much easier for them to see how their understanding develops from one submission to the next.

The system's back end is the facility that is at the core of the system. All requests that are performed through the user interface are sent to the server. At the server, the different types of engines are initiated to process the request for analysis. This includes inspecting the syntax of the program, running it in a sandbox, and collecting the results at runtime. The back end also handles the user session, performs authentication for users who need it, and coordinates all the steps that are part of the diagnostic process. Specifically, the backend's design allows even the most complex interactions to happen smoothly and efficiently (Amiri et al., 2025). Logic analysis refers to the process of executing a user's program (as submitted) in a controlled and predictable manner, gathering the results generated from that execution, and comparing those results against what we predicted would happen. The process of structural evaluation involves the use of an 'abstract syntax tree' for the purpose of finding places in the code where we can organise it into more meaningful ways (Modules), and to simplify the way in which we build our loops. Following the completion of the Technical Phases of Logic Analysis and Structural Evaluation, the final mentorship component of the Integrated Language Model generates a type of personalised mentorship that occurs in a format similar to the way a human instructor would. In addition to simply identifying errors, the mentor will provide a detailed explanation of the reasoning behind each identified error and will also provide suggestions regarding possible Best Practices that the user may apply to their next submission of the Code.



International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 14, Issue 04, ICTEM 2.0 (2026)

Fig. 1: System Architecture

B. Functional Modules

The AI Code Mentor consists of four separate modules that operate independently to support learners through the various stages of programming analysis. Each module supports the different stages of a learner's programming journey by helping them to understand their submission, identifying and fixing errors in submitted code, improve their submission, and giving helpful feedback on their submissions.

Module 1 focuses on identifying syntax and coding errors through diagnosis. By analysing a learner's submission for structural or syntax errors, this module shows the point at which an error occurred in the submitted code. The module detects a variety of errors, such as missing or misplaced symbols, inconsistent indentation, incorrect operators, and invalid programming statements that prevent the code from executing correctly. Instead of presenting learners with a short compiler message that is difficult to comprehend, this module provides explanations that are geared toward beginners, and the goal is to provide an understanding of the cause of the error, as well as the steps required to rectify it (Ali et al., 2025).

Module two measures how well the logic in the student's code works. The student's submitted piece of code runs inside a 'sandbox', allowing safe execution while maintaining isolation from other areas of the student's computer. Once the program has completed its execution, the module will automatically compare the results obtained from the executed program against the results that are expected and will generate a report to notify the user of any discrepancies between the two results, along with identifying runtime problems and exceptions that occur. Furthermore, it provides the user with information that describes why and how the program produces different behaviour from what was originally intended, and what events or situations resulted in those variations in behaviour. By providing users with all of this relevant information, it encourages critical thinking about how a program is supposed to behave and encourages them to think in terms of input, output, and execution flow instead of just copying and pasting answers.

In addition to providing feedback on whether or not the program is correct, module three also evaluates the overall structure of the program. The evaluation uses the abstract syntax tree technique to analyse how well the student has structured their statements, loops and functions. Based on the evaluation of the program's structure, module three will provide recommendations to the student that include: increasing the modularity of the program, using more meaningful variable names, reducing the amount of redundant code, or breaking up large sections of code into less complex and more reusable components. Students will learn through this third module that simply writing a correct program is only the first step in creating a well-written program and that they also need to write code that is clear, readable, and efficient, according to established professional standards.

This module of the program includes Mentoring and Learning Support services to help students with their development at an intermediate level through the usage of Human-Like Explanations, Step-By-Step Instructions, and Conceptual Reinforcement to help students better understand Cause and Effect, how to learn from their errors, and ultimately build their Confidence progressively. In the end, this comprehensive module enhances a student's overall knowledge rather than simply offering surface-level corrections.

C. ER Diagram and Database Design

A logical and coherent Link between the Entities that comprise a database will provide for the ability to maintain Clear, Logical Relationships between the data stored within each entity that comprises that database

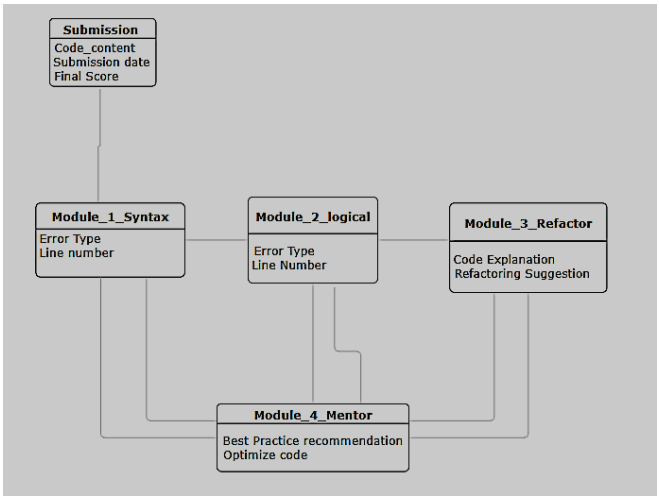


Fig. 2: ER Diagram

Each entity can be thought of as a key to the learning process, which provides a systematic way to Store and Retrieve Data Efficiently. Each entity has associated attributes that assist in defining how those entities interact with one another. The User Entity includes both Students and Instructors and captures a Student's Programming Attempt in the Submission Entity. the results of the diagnostic processes are documented independently for better understanding. Syntax problems are identified as Syntax Errors, and logic problems (the wrong function or an object that failed to execute) are identified as Logic Errors. Recommendations for the clarity and organisation of code are captured in Style Suggestions. The aggregation of these records into the Feedback Report provides a summary of the performance of the learner. Collectively, these records are the foundation for retaining

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 14, Issue 04, ICTEM 2.0 (2026)

previous programming attempts, tracking the progression of each programmer's development, and motivating them to keep on learning over the long term (Bobadilla et al., 2023).

V. METHODOLOGY

A. System Architecture

The layers of the AI Code Mentor serve unique functions but all interact in ways that provide a unified experience. The first layer encompasses a "presentation," which is comprised of all components required to allow the user to enter code, receive software feedback, and keep tabs on how well they are progressing through their coding education. the presentation layer was designed to be easy to use, because many of the users of this platform are novice developers (Amiri et al., 2025).

The second layer, known as the application layer, represents the true processing power behind the application. This is where the majority of the logic takes place: the application will check the code for syntax errors, safely interpret the code within a controlled environment and evaluate any errors that occur. In addition, the second layer employs AI models to generate the explanations. Once the analyses have been performed by the application layer, the results will then be sent back to the presentation layer for user display.

At the bottom of the system is the data layer. This holds all the "stuff" your students submit to the system, their diagnostic results, options made available to students based on performance, and their previous attempts at exercises. This data is kept in an orderly fashion so that your students can easily look back at their previous attempts and see how much they have progressed.

Together, these three layers provide you with a platform that is easily scalable, secure, and can be extended to accommodate new enhancement functionality without significant modification to the three-layer architecture (Ahmed et al., 2022).

B. Model Training

A system's ability to offer helpful advice stems from its ability to differentiate between correct and incorrect computer programming. Therefore, an artificial intelligence (AI) model receives numerous examples of actual code in order to train itself. Some of these example codes function correctly, whereas others contain erroneous coding practices (for instance: incorrect syntax, logical errors, etc.). By using both types of examples, the AI model trains itself in the means of recognising accurate programming from flawed programming (Ali et al., 2025).

To train the AI model to recognise programming errors, two different methods are used: 1) Deterministic analysis technique: This technique evaluates how code was created and verifies whether or not the logic in the code appears to be consistent with the intended functionality. 2) Language model technique (LLMs): This technique creates responses that come across as "natural" and "supportive". Although the deterministic method provides accurate results to the AI model, the LLM allows the AI to convey to students errors in simple and straightforward terms, which does not leave any room for confusion or discouragement (Karsa and Goldschmidt, 2025).

Using a supervised training approach, the AI model will only improve itself. After each experiment, the AI model will compare its prediction against the correct answer, utilising this knowledge to correct itself. Over time, the AI will learn to identify a plethora of programming errors and to propose solutions in a clear and upbeat manner similar to that of an excellent teacher.

C. Error Detection and Classification

The main role of this application is to identify errors and appropriately categorise them based on specific criteria. When students submit their assignment materials, this application will keep track of what the student has done while completing the assignment submission process. If the submission of the assignment was unsuccessful, the application would identify these types of errors using a combination of different identifiers.

After the platform identifies an error, it categorises it to allow for the submission of a targeted explanation. Syntax errors are the easiest to find (e.g., missing parentheses, incorrectly indented lines, or misspelt words). These errors include, but are not limited to, those caused by programming language conventions. Logical errors are more difficult to locate (e.g., a program may produce the expected result, but the underlying logic may produce an unexpected answer). Runtime errors occur while a program is executing, and usually result in generating exceptions.

By categorising errors into the above-mentioned categories, the platform can provide students with relevant explanations of their specific error type. Therefore, students can spend less time guessing about what the potential error might be while debugging their code. Furthermore, the platform encourages students to differentiate between syntax, logical errors and runtime errors to better prepare them to be competent programmers in the future (Das et al., 2016).

D. Data Preprocessing

The first step to conducting an analysis on a program is to ensure that the input data has been cleaned and formatted

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 14, Issue 04, ICTEM 2.0 (2026)

for analysis. The data pre-processing process accomplishes this through removing extraneous characters, ordering of data, and formatting of the data so that it can be interpreted by the analysis tools properly.

Many students will submit code that contains additional white space, odd formatting, or comments that are not pertinent to the error identification process. These errors are generally removed or normalised so that the data used by the model can be concentrated on what is actually important. The improved data will subsequently allow for the identification of significant trends and relationships much more easily than without going through the data cleaning process, as otherwise the system may have made erroneous assumptions regarding the error or omitted significant information from consideration.

E. Performance Evaluation

Testing and evaluating systems once they've been developed helps to confirm that the system's operating characteristics are in line with expected performance. To evaluate the performance of the system, the results generated by the system will be compared to what the expected results should have been. By analysing the methodologies of the tests performed, one can better understand how well the platform can identify errors and whether the information provided by the platform to learners will aid in improving their learning.

To help with this performance assessment process, various metrics will be collected during testing. Examples of these metrics are the number of errors detected, the percentage of times the system provided correct and useful feedback, and how quickly the system was able to respond to the learner. The use of performance metrics such as precision, recall, and overall accuracy assists in determining whether the system is functioning at an acceptable level or if changes or additional development efforts are required.

Performance testing should be conducted multiple times to ensure that the system will produce the same or similar output in similar situations. If the results of the performance assessments demonstrate that the platform can be relied on, it will provide an excellent tool in providing instructional support to students and will assist in providing opportunities for students to improve and eventually develop the independence and competence to program on their own.

VI. RESULTS AND DISCUSSION

The AI Code Mentor has shown its potential in aiding students who are learning programming for the first time through its use and testing on various Python Applications. Its four-tier diagnostic analysis framework provides an outstanding level of insight for both Syntax Error Detection and Logical Bug Identification, as well as AST-based Refactoring and Personalized Mentor Feedback. Across all tiers, there was coincidental agreement of usability for programming students, providing an understanding of their coding mistakes as well as incentives to improve coding practices.

Syntax Errors identified in all Tiers helped Programming Learners understand Syntax Errors and develop a Logical Solution to resolve them. Logical Bugs were identified by the execution of a Controlled 'Test Suite', wherein the actual results from the execution of the program were compared to the expected results. The AST-Refactoring (AST-based Refactoring Component) Tier assisted and inspired Programming Learners to adopt more Efficient, modularised coding Techniques by encouraging the application of Function Modularisation and Optimized Loop Structures, two characteristics of "Pythonic Coding". Finally, the Solutions provided by Mentors were encouraged to use Large Language Models, providing Programming Learners with Contextual Solutions for making an Initial Sense of Programming Concepts and Resolving Errors in Programming Code.

There was a substantial positive impact on student learning outcomes as a result of the user trial analysis.

Most students reported more confidence in their coding abilities after using the system, and the majority of students have improved their learning speed due to immediate feedback regarding their mistakes, as well as what to do to correct them.

In addition, the majority of teachers stated that the system’s dashboard enables them to view students' overall difficulties with different aspects of coding, long-term learning patterns, and where students are in their progress through all of the coding modules. The dashboard provides instructors with a way to easily identify the common issues students encounter when coding and to adapt their teaching style to accommodate these needs.

Deterministic analysis and AI work very well together, because while the deterministic modules enable a high degree of accuracy in identifying and classifying the errors in students' coding, the AI mentoring function made the findings from the technical output of the learning module easier for students to digest in a way that supported their learning about the programming concepts associated with their code. The AI mentoring feature also allowed students to not only fix their coding errors but also to develop an understanding of programming concepts and principles, thereby encouraging them to be better programmers in the long term.

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 14, Issue 04, ICTEM 2.0 (2026)

AI Code Mentor is an excellent resource and enhances the ability to learn programming using traditional methods. The automated diagnostics offered by AI Code Mentor, combined with the simulated human-to-human interaction, provide a superior learning opportunity by creating an immediate skill acquisition with error-correction, while also providing a deeper understanding of the concepts of coding, which together create a deeper learning experience than would be possible using either one of these approaches alone. Therefore, the findings suggest that AI Code Mentor is highly likely to enhance coding abilities, increase effective coding habits and develop uniquely catered learning environments in an efficient and scalable manner through software.

VII. CONCLUSION

The AI Code Mentor employs artificial intelligence to furnish an all-encompassing educational atmosphere encompassing the advancement of student learning, enhancing student self-assurance and supplying educators with meaningful commentary on their pupils' accomplishments. The AI Code Mentor utilises a synergistic blend of automated code analysis, execution in sanctioned environments (sandboxing), evaluation of Abstract Syntax Trees, as well as AI tutorials to furnish students with customised justifications of concepts necessary for students to attain their objectives in programming.

Moving Forward: Future improvements to the AI Code. The features of the Mentor System will continue to improve. Multiple coding examples will be included in the dataset, including edge cases and advanced patterns, which will enable improved accuracy and better relevance of the error detection of the AI. Another area for improvement is an updated method for adapting the explanations given to the learner so that they cater to the strengths of that particular learner and provide a more personalised approach, thereby providing clear, easy-to-follow feedback to the learner. The last area for improvement could be the ability to include multiple programming languages on the platform, which would greatly improve the versatility of the solution, allowing all students, irrespective of ability, to take advantage of the benefits of the AI Code Mentor. Likewise, incorporates both automated test cases.

AI Code Mentor has the potential to help students evaluate the quality and efficiency of the code they produce and highlight areas for potential performance improvement. Through the incorporation of collaborative features such as peer reviews of code and community discussion forums, AI Code Mentor enables students to learn collaboratively and socially by sharing experiences and ideas to help one another.

AI Code Mentor is an advanced, intelligent and scalable tool that places the learner as the focal point of the learning process. With future enhancements, AI Code Mentor will enable students to rapidly identify coding errors, develop a deeper understanding of programming and create effective coding skills, thereby building problem-solving capacity over time.

ACKNOWLEDGMENTS

The authors wish to express their sincere appreciation for the opportunity given to them by **Moradabad Institute of Technology (MIT), Moradabad**, to conduct this research. The technical resources and the positive atmosphere created by MIT provided the environment for the authors to learn and advance their skills while developing the *AI Code Mentor - Intelligent Python Programming Assistant.*

The authors are also very grateful to their project guide, **Mr Puneet Kumar**, for providing them with continuous guidance, constructive feedback, and dedicated supervision throughout the project. With his expertise and constructive suggestions, **Mr Puneet Kumar** shaped the direction and quality of this research project. The authors also thank **Prof. (Dr.) Rohit Garg, Prof. (Dr.) Manish Gupta** and the faculty members of the Department of Computer Science and Engineering at MIT Moradabad for providing them with support, motivation, and valuable advice.

At last, the authors want to thank their families and friends for their unwavering support, encouragement, and patience during the entire project, and for believing in the authors and motivating them to overcome challenges and ultimately achieve the goals of the project.

REFERENCES

[1] Amiri, S. M. H., & Islam, M. M. (2025). Enhancing Python Programming Education with an AI-Powered Code Helper: Design, Implementation, and Impact. *Software Engineering*, *11*(1), 1-17.

[2] Bobadilla, S., Glassey, R., Bergel, A., & Monperrus, M. (2023). Sobo: A feedback bot to nudge code quality in programming courses. *IEEE Software*, *41*(2), 68-76.

[3] Ahmed, U. Z., Fan, Z., Yi, J., Al-Bataineh, O. I., & Roychoudhury, A. (2022). Verifix: Verified repair of programming assignments. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, *31*(4), 1-31.

[4] Das, R., Ahmed, U. Z., Karkare, A., & Gulwani, S. (2016). Prutor: A system for tutoring CS1 and collecting

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 14, Issue 04, ICTEM 2.0 (2026)

student programs for analysis. *arXiv preprint arXiv:1608.03828*.

[5] Schez-Sobrino, S., Gmez-Portes, C., Vallejo, D., Glez-Morcillo, C., & Redondo, M. A. (2020). An intelligent tutoring system to facilitate the learning of programming through the usage of dynamic graphic visualizations. *Applied sciences*, *10*(4), 1518.

[6] Alshaikh, Z., Tamang, L., & Rus, V. (2020, June). A socratic tutor for source code comprehension. In *International conference on artificial intelligence in education* (pp. 15-19). Cham: Springer International Publishing.

[7] Phung, T., Pădurean, V. A., Singh, A., Brooks, C., Cambronero, J., Gulwani, S., ... & Soares, G. (2024, March). Automating human tutor-style programming feedback: Leveraging gpt-4 tutor model for hint generation and gpt-3.5 student model for hint validation. In *Proceedings of the 14th learning analytics and knowledge conference* (pp. 12-23).

[8] Yang, A. C., Lin, J. Y., Lin, C. Y., & Ogata, H. (2024). Enhancing python learning with PyTutor: Efficacy of a ChatGPT-Based intelligent tutoring system in programming education. *Computers and Education: Artificial Intelligence*, *7*, 100309.

[9] Ali, F., Ahmed, A., Alipour, M. A., & Terashima-Marin, H. (2025). Adoption of AI-coding assistants in programming education: exploring trust and learning motivation through an extended technology acceptance model. *Journal of Computers in Education*, 1-39.

[10] Taulli, T. (2024). *AI-assisted programming: Better planning, coding, testing, and deployment*. " O'Reilly Media, Inc.".

[11] Pankiewicz, M., & Baker, R. S. (2023). Large Language Models (GPT) for automating feedback on programming assignments. *arXiv preprint arXiv:2307.00150*.

[12] Zhang, X., Zhu, F., Wang, K., Cao, G., Xue, Y., & Liu, M. (2024). Bring the intelligent tutoring robots to education: a systematic literature review. *IEEE Transactions on Learning Technologies*.

[13] Karsa, Z. I., & Goldschmidt, B. (2025). Automatic evaluation of programming tasks supported by language models. *IEEE Access*.

[14] Gupta, R., Pal, S., Kanade, A., & Shevade, S. (2017, February). Deepfix: Fixing common c language errors by deep learning. In *Proceedings of the aaai conference on artificial intelligence* (Vol. 31, No. 1).