iJRASET
International Journal For Research in Applied Science and Engineering Technology

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Volume: 14 **Issue:** V **Month of publication:** May 2026

DOI: https://doi.org/10.22214/ijraset.2026.82571

www.ijraset.com

Call: 08813907089 | E-mail ID: ijraset@gmail.com

International Journal for Research in Applied Science & Engineering Technology (IJRASET)
ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.538
Volume 14 Issue V May 2026- Available at www.ijraset.com

An AI-Driven System for Automatic Code Explanation and Program Visualization

Meghana V. Bandiwadekar[1], Shravani Patil[2], Gayatri Mungarwadi[3], Afiyah Modak[4], Junedali Patel[5], Sarvesh Halade[6]
Dept of CSE, D. Y. Patil College of Engg & Tech, Kolhapur, India

Abstract: ***Beginners frequently find learning programming difficult because of its intricate syntax and abstract logical structures. Current AI-based coding tools prioritize code generation and completion over conceptual comprehension. This paper introduces CodeMindAI, a clever AI-powered platform that uses visual representation of program logic and automatic natural language explanation to enhance code comprehension. The suggested system uses large language models to analyze user-written code and produces flow-based visualizations and simplified explanations. Learners can investigate real-world projects with AI-assisted guidance thanks to integration with software repositories. Comparing the system to conventional development environments, experimental evaluation shows that it improves comprehension of control flow and functional behavior. The findings show that AI-powered multimodal learning platforms can efficiently enhance programming instruction and lessen the cognitive load on inexperienced programmers.***

Keywords: Artificial Intelligence, Code Understanding, Program Visualization, Natural Language Processing, Programming Education, Generative AI.

I. INTRODUCTION

The ability to program has become essential in today's digital and computing industries. However, because of complicated syntax, abstract logical structures, and challenges comprehending program behavior, learning to program is still a difficult task for novices. Conventional development environments provide little assistance for elucidating the underlying logic of programs, instead concentrating primarily on writing and running code. Because of this, inexperienced programmers frequently find it difficult to reconcile syntactic accuracy with conceptual comprehension [1].

Intelligent coding assistants that can automatically generate and complete code have been developed as a result of recent developments in artificial intelligence (AI). Although these tools greatly increase developer productivity, they focus more on code generation than code comprehension [2]. Students who only use these tools might create programs that are syntactically correct without fully comprehending how they work. This drawback emphasizes the necessity of AI-based solutions that support program comprehension as opposed to just writing code.

It has long been known that visualization techniques like flowcharts and structural diagrams are useful tools for comprehending program logic [3]. However, the majority of visualization tools necessitate the manual creation of diagrams or pre-established syntactic representations, which can be challenging for novices. In a similar vein, while online learning platforms offer structured tutorials, they do not provide personalized, real-time explanations for user-written code. Furthermore, it is uncommon for current tools to integrate visualization and explanation into a single learning environment.

This paper introduces CodeMindAI, an AI-assisted coding and learning platform that aims to improve code comprehension by automatically explaining code in natural language and visualizing program logic. Simplified textual explanations and corresponding visual representations of control flow are produced by the suggested system after it analyzes source code using large language models. Furthermore, learners can investigate real-world codebases with AI-assisted guidance thanks to integration with software repositories, which links abstract ideas with real-world application.

The work's primary contributions are (i) an AI-driven framework for automatically explaining code that is adapted to learner comprehension, (ii) a visualization mechanism that provides an intuitive representation of program logic, and (iii) an integrated learning environment that facilitates the exploration of real-world code repositories. Comparing the suggested method to traditional development tools, experimental findings show that it lowers cognitive load and enhances learner comprehension of program behavior.

This is how the rest of the paper is structured. Related research in AI-based code assistance and program visualization is reviewed in Section II. The suggested system architecture and methodology are explained in Section III. Experimental results and implementation details are presented in Section IV. The paper's conclusion and future research directions are covered in Section V.

©IJRASET: All Rights are Reserved

II. RELATED WORK

Many tools and systems that help programmers write and understand code have been developed as a result of recent advancements in artificial intelligence. Large language models are used by AI-driven code assistants such as GitHub Copilot, Tabnine, and Amazon CodeWhisperer to provide real-time code generation and completion. By reducing the amount of work required to write syntactically correct code, these tools significantly increase developer efficiency. However, they place more of an emphasis on writing code than on comprehending ideas. They don't specifically address the learning objectives of beginning programmers and only offer brief explanations of program logic.

The use of natural language processing techniques to convert source code into descriptions that are readable by humans has been studied in the context of automatic code summarization and explanation [4]. Neural models can produce succinct textual descriptions of functions and classes, according to studies on code summarization [5]. Although these approaches help comprehend software components, they frequently don't integrate with visualization techniques and don't provide interactive feedback in a learning environment. Furthermore, a lot of current systems generate generic summaries that aren't customized for particular code inputs or learners.

It has long been known that program visualization is a useful strategy for improving program comprehension [6]. Visual representations of program structure and execution flow are provided by tools such as UML diagramming systems and flowchart generators. However, the majority of visualization tools require predefined syntax or manual diagram specification, which can be difficult for novices [7].

For small programs, systems such as Python Tutor provide step-by-step execution traces; however, their scalability is limited, and they are not made to integrate with complex codebases or real-world software repositories.

Intelligent tutoring systems and online learning platforms provide structured programming exercises and courses. These platforms offer little flexibility for analyzing random user-written code and instead concentrate on predefined problem sets and tutorials. Furthermore, they usually don't integrate various learning modalities—like visual aids, textual explanations, and audio feedback—into a single framework. As a result, students frequently use different tools to write code, comprehend logic, and visualize execution behavior.

The use of AI in educational settings to promote adaptive learning in programming has been the subject of some recent research. Based on student performance, these systems seek to offer guidance and feedback. However, rather than providing a thorough explanation of program behavior, the majority of current approaches concentrate on error detection or code correction. Furthermore, most systems do not facilitate the exploration of real-world repositories, which restricts their applicability to real-world software development scenarios.

In conclusion, current research in code summarization, program visualization, and AI-assisted programming has significantly improved productivity and partial understanding. However, existing systems usually deal with these issues separately. Unified platforms that concurrently offer interactive learning support for arbitrary user-written code, visual representation of program logic, and automatic natural language explanation are lacking [8]. The proposed CodeMindAI system, which combines AI-driven explanation and visualization in a single learning-oriented environment, is motivated by these limitations.

III. PROBLEM STATEMENT AND MOTIVATION

Because program logic, control flow, and abstract concepts are hard to grasp, learning programming can be difficult for novices. The main goals of current development environments and AI-based coding tools are code generation and syntax correction; they provide little assistance in elucidating the inner workings of programs. Because of this, students frequently write correct code without fully understanding how it behaves.

Code understanding is only partially covered by the educational platforms and program visualization tools available today. While learning platforms rely on fixed tutorials that do not adapt to arbitrary user-written code, visualization tools require manual input or predefined syntax. Furthermore, because these tools are usually used separately, students are forced to switch between platforms in order to code, explain, and visualize.

The goal of this work is to provide an integrated AI-assisted learning environment in order to overcome these limitations. The goal of the suggested system is to visually depict program logic on a single platform and automatically explain user-written code in natural language. The system aims to facilitate efficient and learner-centered programming instruction by minimizing tool fragmentation and placing a strong emphasis on conceptual understanding.

©IJRASET: All Rights are Reserved | SJ Impact Factor 7.538 | ISRA Journal Impact Factor 7.894 |

ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.538
Volume 14 Issue V May 2026- Available at www.ijraset.com

IV. PROPOSED SYSTEM

By automatically explaining and visualizing program logic, the suggested system, CodeMindAI, is an AI-assisted platform intended to enhance code comprehension. The system uses an AI-based analysis engine to process source code that has been created by the user or downloaded from software repositories. The engine creates simplified natural language explanations by interpreting the code's structure and behavior.

The system transforms program logic into flow-oriented visual representations that show control structures like loops, function calls, and conditions in order to facilitate visual learning. Users can better grasp the relationships between program components and the execution flow with the aid of these visual outputs. Additionally, learners can investigate real-world codebases with AI-assisted guidance thanks to repository integration.

Code input, AI-based analysis, explanation and visualization creation, and result presentation comprise the overall workflow. A single environment for learner-centric programming support is provided by this integrated approach, which also lessens the need for multiple tools.

V. SYSTEM ARCHITECTURE

The suggested CodeMindAI system's architecture is based on a modular client-server architecture that facilitates scalability, adaptability, and effective user request processing. The user interface layer, application server layer, AI processing layer, and data and integration layer are the four main layers that make up the system. While interacting with other layers via clearly defined interfaces, each layer carries out a particular task.

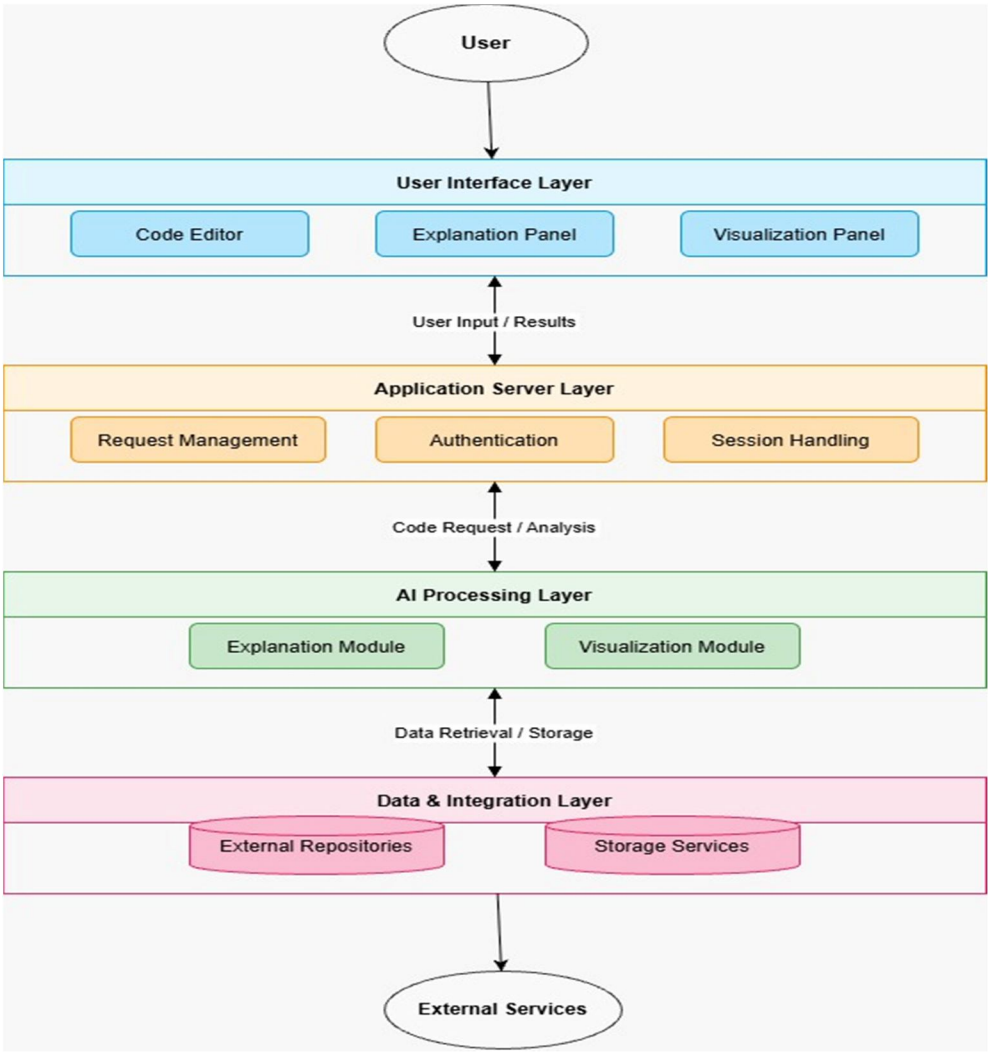


Fig. 1 Layered architecture of CodeMindAI system.

©IJRASET: All Rights are Reserved

A. User Interface Layer

As shown in Fig. 1, an interactive web-based environment for code writing and analysis is provided by the user interface layer. In the given figure, this layer consists of a code editor, an explanation panel, and a visualization panel. Users can interact with real-time suggestions and input or load source code using the code editor, as illustrated in the figure. While the visualization panel shows graphical depictions of program flow, the explanation panel shows descriptions of the code logic in natural language. As depicted in the given figure, this layer is responsible for rendering the output from the backend services and recording user input.

B. Application Server Layer

As illustrated in Fig. 1, the application server layer serves as a bridge connecting the AI processing components and the user interface.

In the given figure, it coordinates data exchange between modules, manages user requests, and deals with authentication and session handling. The server sends the request to the AI processing layer when a user submits code for analysis, gathering the visual data and explanations that are produced, as shown in the figure. Additionally, it maintains session data and guarantees safe access to outside services. This layer, as represented in the given figure, makes it possible to separate computational processing from presentation logic.

C. AI Processing Layer

As shown in Fig. 1, the system's primary analytical component is the AI processing layer. In the given figure, this layer includes the explanation module and the visualization module. It uses language models and program analysis techniques to examine the input code's structure and semantics.

This layer generates two main outputs: a logical depiction of the program control flow and an explanation of the code in natural language, as illustrated in the figure.

While the visualization module converts control structures into flow-based representations, the explanation module concentrates on converting code behavior into descriptions that are understandable to humans. Both textual and visual comprehension of program behavior are supported by this design, as depicted in the given figure.

D. Data and Integration Layer

As illustrated in Fig. 1, interaction with external repositories and storage services is controlled by the data and integration layer. In the given figure, this layer manages external repositories and storage services. It keeps track of previously examined code and produced outputs and facilitates the retrieval of source code from software repositories. Additionally, this layer manages communication with outside services that are used to store and retrieve codes, as shown in the figure. The system retains extensibility and permits the addition of new services with little architectural alteration by separating integration tasks from the main logic, as represented in the given figure.

E. Data Flow and Interaction

As shown in Fig. 1, when the user chooses a repository file or enters source code, the system starts to operate. The application server receives this input from the user interface, verifies it, and then sends it to the AI processing layer, as illustrated in the given figure. After analyzing the code, the AI layer produces visual aids and explanations.

After that, the application server receives these results and sends them to the user interface for display. The generated outputs are saved for later access by optional storage operations, as depicted in the figure. Effective processing and reliable communication between system components are guaranteed by this pipeline.

F. Architectural Advantages

As depicted in Fig. 1, CodeMindAI's modular architecture offers a number of benefits. It first makes it possible for each layer to be developed and tested independently, as shown by the separation of layers in the given figure. Secondly, it facilitates scalability by separating the user interface from computationally demanding AI processing. Third, by clearly separating issues like data management, analysis, and user interaction, it encourages maintainability. All things considered, as illustrated in the given figure, the architecture is built to accommodate future improvements, such as more learning features and support for additional programming languages.

©IJRASET: All Rights are Reserved | SJ Impact Factor 7.538 | ISRA Journal Impact Factor 7.894 |

ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.538
Volume 14 Issue V May 2026- Available at www.ijraset.com

VI. IMPLEMENTATION DETAILS

A web-based client-server architecture was used to implement the suggested CodeMindAI system. The implementation's main goals are to offer a unified platform with an interactive coding interface, AI-based analysis, and visualization support, as represented in the given figure.

A. Frontend Implementation

As illustrated in Fig. 2, the frontend consists of a web-based code editor that contains an explanation panel and a visualization panel. In the given figure, the process begins when the user enters source code or selects a repository file through the user interface. The code editor has syntax highlighting and allows users to interact with their code through the code editor for several programming languages.

Users have the option of entering source code into the code editor directly or loading source code files from a software repository, as shown in the figure. The explanation panel presents program logic in natural language, while the visualization panel displays the flow-based representation of control structures.

As depicted in the given figure, asynchronous communication between the web-based user interface and backend services ensures real-time interaction and response delivery.

B. Backend Implementation

As shown in Fig. 2, the backend server handles user request processing, session management, and communication coordination with the AI-processing components. In the given figure, the backend receives the analysis request from the frontend and performs validation and authentication.

If the request is invalid, an error message is returned to the user interface, as illustrated in the figure. If the request is valid and authenticated, the backend forwards the code to the AI processing layer for further analysis.

The server then collects all generated explanation and visual data before sending them back to the frontend, as depicted in the given figure. Additionally, separation of the frontend and backend provides better maintainability and independent scaling of compute resources.

C. AI-Based Code Analysis

As illustrated in Fig. 2, the AI processing module analyzes the code after it is forwarded by the backend server. In the given figure, this step includes analyzing code structure and behavior using language models and static analysis techniques. The system generates user-friendly natural language descriptions explaining functions, loops, and conditional statements within the program. Prompt templates assist the AI in generating clear, concise, and student-oriented explanations.

As shown in the figure, the output includes both a textual explanation and an intermediate logical control-flow model representing program behavior.

D. Visualization Generation

As shown in Fig. 2, after code analysis, the visualization module generates visualization metadata and builds flow-based representations of the program logic. In the given figure, logical elements such as conditions, loops, and function calls are converted into structured visual components.

These components are packaged and prepared for display on the user interface. The generated visualization helps users understand program execution visually, as illustrated in the figure. This combination of textual explanation and graphical representation improves learning and comprehension of program logic.

©IJRASET: All Rights are Reserved

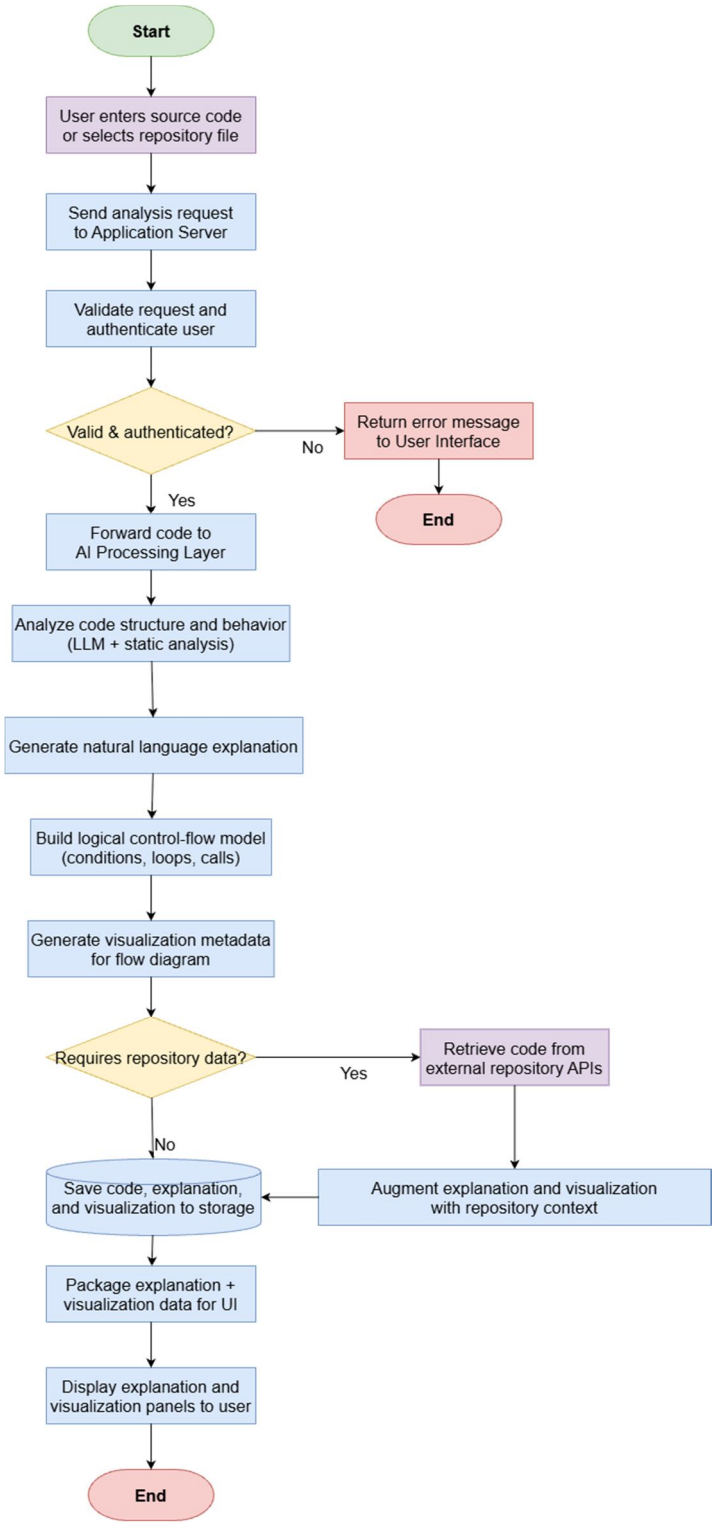


Fig. 2 Workflow of the Proposed CodeMindAI System.

E. Repository Integration

As illustrated in Fig. 2, the system supports integration with external repositories. In the given figure, when repository data is required, the system retrieves code from external repository APIs. The retrieved code is then processed using the same AI-based analysis and visualization pipeline. The system augments explanations and visualizations using repository context, as depicted in the figure. This feature enables users to analyze real-world code examples and understand existing implementations more effectively.

F. Data Management and Security

As shown in Fig. 2, after processing and visualization generation, the system saves code, explanations, and visualization data to storage for future access. Secure Communication Protocols such as HTTPS are used for all user requests and generated outputs, ensuring safe data transmission. Authentication mechanisms verify and restrict user access, as illustrated in the given figure. The backend packages explanation and visualization data and sends it to the user interface for display. These measures ensure reliable, secure, and efficient operation of the system, as represented in the given figure.

VII. RESULTS AND DISCUSSION

The evaluation of the proposed CodeMindAI implementation was conducted through sample programming cases representing a wide variety of programmatic complexity. Evaluation criteria included the generated explanations' quality, the generated visual representation's clarity of communication, and the generated explanation's overall usability for novice users.

The system was able to provide natural language structured explanations of the behavior of the sample program in an intuitively simple manner. In low-to-moderately complex programs, the generated explanations accurately represent both their control flow and logical operations. The system was able to produce visual representations of the sample program's control flow in flow-oriented diagrams, allowing users to visually trace through the program's execution path. As a result, the user should have been able to develop a much greater understanding of the program from both textual and visual representation, compared to what would result from viewing only the program source code.

The user observations indicated that the integrated development environment would lead to the need for using multiple tools for both explanation and visual representation of a sample program. In addition, the system prioritizes providing its users with an understanding of how and why their code works, compared to other traditional AI-based code completion systems that focus only on generating code. Therefore, the proposed CodeMindAI system will be an effective educational tool for novice programmers, compared to traditional forms of AI-based code completion. When performance limitations occurred during the analysis of larger or more complex files, both the level of detail provided in the explanation, as well as the overall ease of reading the visual diagram decreased. This indicates that both of these issues will require further optimization to adequately support the handling of large-scale programs and improve the modulo of abstraction for visualisation.

Overall, the experimental outcomes indicate that CodeMindAI supports program understanding through the combination of an explanation and a visualisation within an integrated environment. In addition, the discussion confirms that the proposed solution resolves the limitations of existing tools if they are composed of multiple alternative types of learning modalities on a single platform.

VIII. CONCLUSION AND FUTURE WORK

This paper introduces an AI-assisted system called CodeMindAI, which is designed to help programming learners understand the code they write by automatically creating natural language explanations of the code and producing visual representations of the logic contained within the code. It combines code analysis, explanation generation, and visualization into a single environment to resolve some of the limitations present in current tools that focus primarily on generating code and not on creating a conceptual understanding of how the code works.

The results from the conducted experiments show that CodeMindAI improves the clarity of program behaviour and promotes a programming education that is learner-centric. However, while it works effectively for small- and medium-sized programs, it is less effective for larger, more complicated codebases, due to the fact that visual representations can become cluttered, and the detail of the generated explanations can be limited for highly nested structures. These limitations suggest that the system requires improvements in terms of abstraction and scalability.

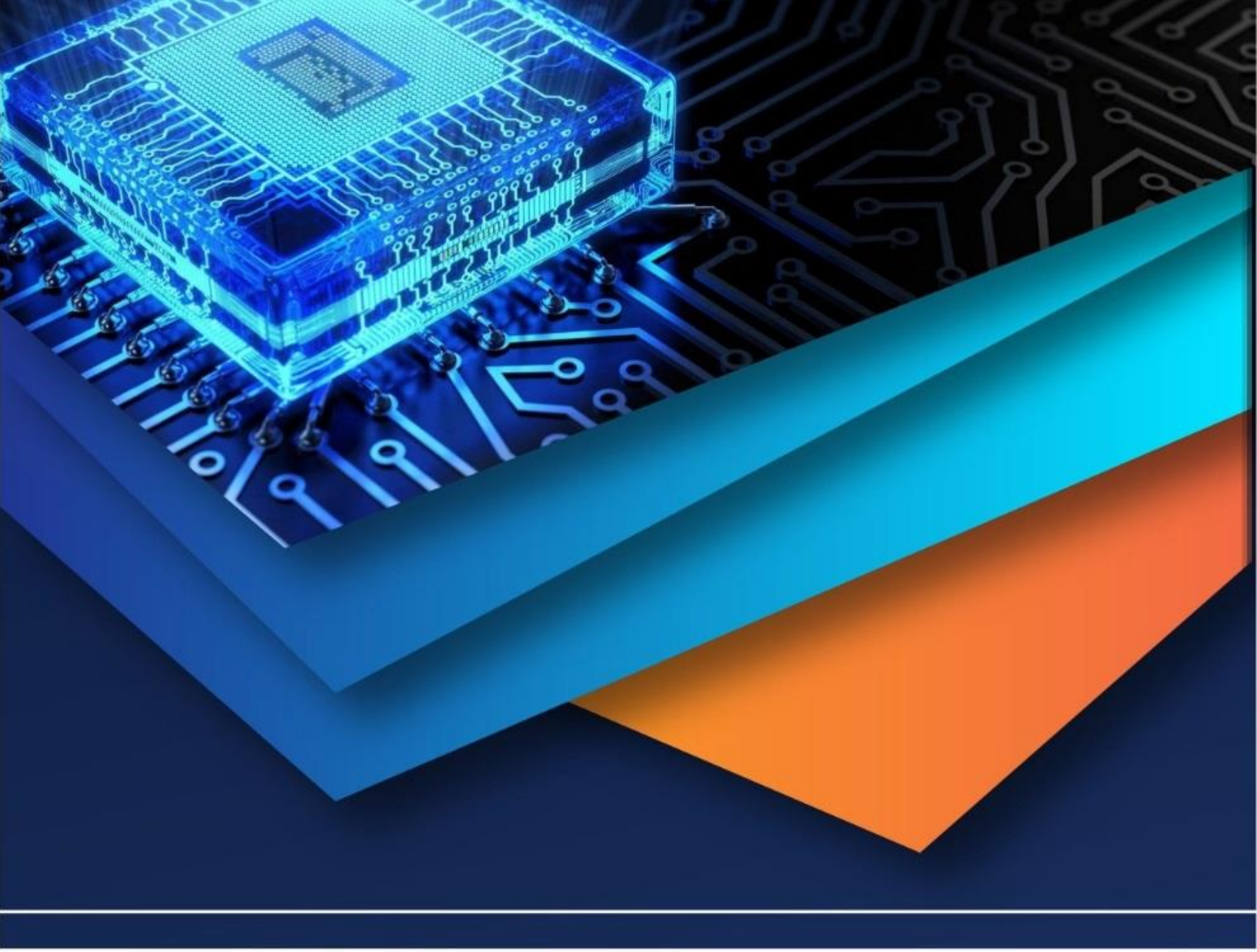
Future work will expand on this project by adding support for additional programming languages and developing improved techniques for visualizing large-scale programs. Future enhancements may include giving users adaptive explanations based on their proficiency level, allowing users to use voice-based interaction with the system, and incorporating assessment modules to measure user comprehension. These enhancements will help establish CodeMindAI as an intelligent educational assistant for programming learners.

REFERENCES

[1] M. Kazemitabaar, R. Ye, X. Wang, A. Z. Henley, P. Denny, M. Craig, and T. Grossman, "Codeaid: Evaluating a classroom deployment of an llm-based programming assistant that balances student and educator needs," in Proceedings of the 2024 CHI Conference on Human Factors in Computing Systems, 2024, pp. 1–20.

[2] E. Poitras, B. G. C. Crane, D. Dempsey, T. A. Bragg, A. A. Siegel, and M. P.-C. Lin, "Cognitive apprenticeship and artificial intelligence coding assistants," in Navigating Computer Science Education in the 21st Century. IGI Global Scientific Publishing, 2024, pp. 261–281.
[3] P.-W. Lin, S.-H. Yu, and C.-H. Lai, "Dynamic program analysis and visualized learning system in university programming courses," Engineering Proceedings, vol. 98, no. 1, p. 30, 2025.
[4] C. C. Le, H.-C. Truong-Vinh, H. N. Phan, D. D. Le, T. N. Nguyen, and N. D. Bui, "Visualcoder: Guiding large language models in code execution with fine-grained multimodal chain-of-thought reasoning," arXiv preprint arXiv:2410.23402, 2024.
[5] Y. Wang, E. Shi, L. Du, X. Yang, Y. Hu, S. Han, H. Zhang, and D. Zhang, "Cocosum: Contextual code summarization with multi-relational graph neural network," arXiv preprint arXiv:2107.01933, 2021.
[6] P. Lima, J. Melegati, E. Gomes, N. S. Pereira, E. Guerra, and P. Meirelles, "Cadv: A software visualization approach for code annotations distribution," Information and Software Technology, vol. 154, p. 107089, 2023.
[7] N. Krieger, S. Speth, and S. Becker, "Hylimo: A hybrid live-synchronized modular diagramming editor as ide extension for technical and scientific publications," in Proceedings of the 1st ACM/IEEE Workshop on Integrated Development Environments, 2024, pp. 70–73.
[8] S. Elnaffar, F. Rashidi, and A. Z. Abualkishik, "Teaching with ai: a systematic review of chatbots, generative tools, and tutoring systems in programming education," arXiv preprint arXiv:2510.03884, 2025.

©IJRASET: All Rights are Reserved | SJ Impact Factor 7.538 | ISRA Journal Impact Factor 7.894 |



10.22214/IJRASET

45.98

IMPACT FACTOR: 7.129

IMPACT FACTOR: 7.429

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Call : 08813907089 (24*7 Support on Whatsapp)