International Journal of Advanced Research and Review

AI-POWERED CODE DEBUGGING ASSISTANT

[1]Dr. D. Nagesh Babu, [2]Polani Keerthi Varshini, [3]Palamarthy Lakshmi Supriya, [4]Katta Lakshmi Prasanna.

[1]Associate professor, COMPUTER SCIENCE AND ENGINEERING, St.Ann's College Of Engineering and Technology, Nayunipalli (V), Vetapalem (M), Chirala, Bapatla Dist, Andhra Pradesh – 523187, India

[2,3,4]U. G Student, Dept COMPUTER SCIENCE AND ENGINEERING, St.Ann's College Of Engineering and Technology, Nayunipalli (V), Vetapalem (M), Chirala, Bapatla Dist, Andhra Pradesh – 523187, India

ABSTRACT

An intelligent web-based tool called the AI Powered Code Debugging Assistant was created to make debugging easier and better for developers. Both novices and experts find traditional debugging techniques difficult because they are frequently time-consuming and demand a great deal of skill. By using an AI-driven methodology to provide real-time feedback, mistake explanations, and ideas for updated code, this project seeks to address these issues. Constructed with a Python backend for local hosting and HTML, JavaScript, and Tailwind CSS for a responsive UI, the system incorporates Cohere's Command R+ model to carry out code analysis and intelligent reasoning. Python, JavaScript, Java, C++, C#, and PHP are among the programming languages it supports, guaranteeing broad use.

KEYWORDS

AI Powered Code Debugging Assistant, Intelligent web - based tool, Developer productivity, AI powered debugging, Code analysis, Real - time feedback, Error detection, Cohere Command R+, Python backend, Multilanguage support.

INTRODUCTION

In today's technology-driven world, debugging remains one of the most persistent challenges faced by programmers —from beginners who are still learning to experienced professionals managing complex projects. Traditional debugging techniques, such as manual code inspection or compiler-generated error messages, are often slow, confusing, and fail to provide deeper insights into the real causes of errors. This not only delays software development but also limits a programmer's ability to learn from mistakes and improve coding efficiency.

The AI Powered Code Debugging Assistant addresses this problem by introducing an intelligent, AI driven approach to code correction and understanding. The primary objective of this project is to simplify the debugging process through automation, provide clear explanations of errors, and deliver optimized code solutions that follow best programming practices.

LITERATURE SURVEY

A literature review is a study of existing research related to a project. It helps to understand previous work, identify their drawbacks, and show how the current project provides a better solution. Several researchers have explored AI based debugging systems to enhance code correction and analysis. Kyla Levin et al. (2024) in their paper ChatDBG: An AI Powered Debugging Assistant introduced a system integrating large language models with traditional debuggers to answer developer queries in natural language and suggest fixes, though it had limited multi language support. Bhavya Chopra etal. (2024) proposed Robin, a conversational debugging assistant that improved developer AI interaction and bug localization speed, but it focused mainly on interaction design rather than code education or correction depth. Md. Asraful Haque and Shuai Li (2024) examined The Potential Use of ChatGPT for Debugging and Bug Fixing, showing ChatGPT's promise in identifying and fixing simple bugs, though it lacked contextual understanding and reliable integration with code execution.

RELATED WORK

The AI Powered Code Debugging Assistant builds upon the growing advancements in artificial intelligence and natural language processing to transform the way developers identify and resolve coding errors. Unlike traditional debuggers that rely on manual inspection or compiler outputs, this system intelligently interprets code, detects errors, and provides meaningful explanations with optimized correction suggestions. The project employs a web-based architecture developed using HTML, JavaScript, and Tailwind CSS, offering an interactive and responsive interface that enhances user experience. A Python-based backend ensures smooth execution and local deployment, facilitating real-time code analysis and response generation. Through this integration, the system not only identifies syntax and logical issues but also explains the reasoning behind corrections, bridging the gap between debugging and learning. By providing instant, AI-driven feedback, the project advances existing debugging methodologies, and accessible to both novice and experienced programmers.

EXISTING METHOD

The existing method by Kyla Levin et al. (2024) in ChatDBG: An AI Powered Debugging Assistant integrates large language models with traditional debuggers to answer developer queries in natural language and suggest fixes. Although it improved bug detection and user interaction, the system had several limitations. It relied heavily on external debuggers, making setup complex and less flexible. It supported only a few

programming languages, mainly Python and C++, limiting its applicability. The system focused on identifying errors rather than providing detailed educational explanations or optimized code corrections. Additionally, it lacked a web based interface, reducing accessibility and ease of use. These drawbacks indicate the need for a more versatile debugging assistant that supports multiple languages, provides instant AI-driven feedback.

PROPOSED METHOD

The proposed method enhances existing AI debugging systems by integrating Cohere's Command R+ model for intelligent code analysis across multiple languages such as Python, JavaScript, Java, C++, C#, and PHP. Unlike previous methods, it provides not only error detection but also detailed explanations, optimized code suggestions, and learning points. The web- based frontend, built with HTML, JavaScript, and Tailwind CSS, ensures a responsive and user-friendly interface. A Python backend handles processing and local deployment for real-time feedback. By combining AI reasoning, multi-language support, and educational guidance, this method overcomes the limitations of traditional debuggers and earlier AI systems.

SYSTEM ARCHITECTURE

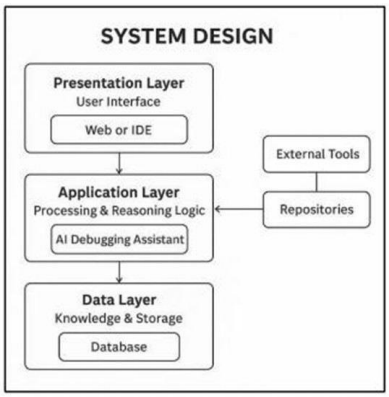

Fig.1: AI-Powered Code Debugging Assistant

METHODOLOGY DESCRIPTION

Input Collection: Users input code snippets into the system through a web- based interface or integrated development environment (IDE). They can also select the desired programming language (Python, Java, C++, C#, JavaScript, or PHP) for analysis.

Code Parsing: The submitted code is parsed to identify its syntax structure, keywords, and logical flow. This stage helps the system understand the programming context and prepares it for accurate analysis.

Error Detection: The AI model powered by Cohere's Command R+ analyzes the parsed code to detect syntax, runtime, and logical errors. It evaluates the structure and semantics to pinpoint the exact lines where issues occur.

Error Explanation: Once errors are detected, the system generates detailed, human-readable explanations. Each explanation clarifies the cause of the issue
—such as incorrect indentation, invalid syntax, or data type mismatch—helping users understand the underlying problem.

Code Correction and Optimization: The system suggests corrected code by implementing AI-based reasoning and best programming practices. It not only fixes the identified errors but also optimizes the code for efficiency and readability.

Output Generation: The corrected code, along with the explanation, is displayed in a separate output section. Users can review both the solution and reasoning and directly copy the working code for testing.

Learning Assistance: The platform promotes learning by highlighting the difference between the user's original and corrected code, enabling beginners to improve their coding skills through comparison and understanding.

Multilanguage Support: The assistant supports multiple programming languages, ensuring adaptability across diverse coding environments and making it a versatile tool for both students and professionals.

System Integration: The backend (developed in Python) processes requests and manages interactions with the AI model, while the frontend (HTML, JavaScript, Tailwind CSS) ensures a responsive and user- friendly interface.

Outcome: This structured methodology enables real-time debugging, accurate code correction, and interactive learning. It reduces debugging time, improves developer productivity, and provides an intelligent, educational debugging experience.

RESULTS AND DISCUSSION

A brief introduction to the assistant, outlining how it helps developers identify, understand, and resolve coding errors faster

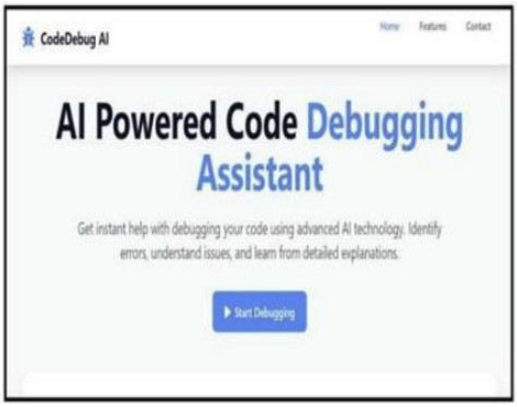

Fig.2: Application Home Page

User-Friendly Design: Clean interface with intuitive buttons making it easy for first time users to start debugging without prior setup.

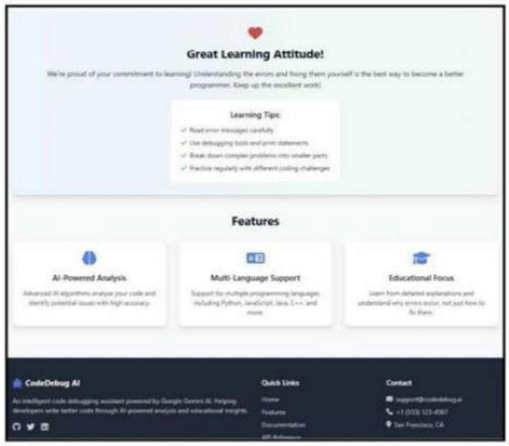

Fig.3: Input code page

User-Friendly Design: Clean interface with intuitive buttons making it easy for first time users to start debugging without prior setup.

Fig.4 entering input details

Code Editor / Text Area: A clean, syntax highlighted editor where users can paste or type their code directly. Error Highlighting: The system pinpoints errors or suspicious parts of the code for easy identification. Includes references to relevant programming concepts, libraries, or functions. Acts like a mentor, guiding users toward better coding practices.

Fig.5 Solution and Explanation

This illustrates the functioning of the AI Powered Code Debugging Assistant, which identifies coding errors, provides a step-by- step explanation of the detected issues, and generates a corrected version of the code. In the example shown, the system analyses the given Python snippet, detects syntax and logical errors, and produces an improved version with appropriate corrections.

CONCLUSION

The AI-powered Code Debugging Assistant integrates static and dynamic program analysis with LLMs to provide accurate root-cause explanations and generate validated code patches. Experimental results show it reduces debugging time, minimizes errors, and improves developer productivity

FUTURE SCOPE

Future enhancements include multi-language support, explainable fixes, IDE integration, collaborative debugging, and AI-assisted code refactoring for broader usability and efficiency.

REFERENCES

1. Harini, D. P. (2012/9). codes: A Collaborative spam Detection system with a novel E-mail abstraction scheme. IDSR Journal of Engineering.
2. Ray, B., Hellendoorn, V. J., Godhane, S., Tu, Z., Bacchelli, A., & Devanbu, P. (2016). *Naturalness of Buggy Code.* Proceedings of the 38th International Conference on Software Engineering (ICSE), 428–439.
3. Tufano, M., Watson, C., Bavota, G., Penta, M. D., White, M., & Poshyvanyk, D. (2019). *An Empirical Study on Learning Bug-Fixing Patches in the Wild via Neural Machine Translation.* ACM Transactions on Software Engineering and Methodology, 28(4), 1–29.

4. Chen, Z., Kommrusch, S., Tufano, M., Poshyvanyk, D., & Monperrus, M. (2019). *SequenceR: Sequence-to- Sequence Learning for End-to-End Program Repair*. IEEE Transactions on Software Engineering, 47(9), 1943– 1959.
5. Fan, J., & Jiang, S. (2020). *A Deep Learning Framework for Automatic Bug Localization and Repair.* Journal of Systems and Software, 168, 110643.
6. Svyatkovskiy, A., Deng, S. K., Fu, S., & Sundaresan, N. (2020). *Intellicode Compose: Code Generation Using Transformer.* Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE), 1433–1443.
7. Li, Z., & He, H. (2021). *Neural Program Repair with Semantic Feedback.* Proceedings of the 35th AAAI Conference on Artificial Intelligence, 8816–8824.
8. Ahmed, F., & Devanbu, P. (2021). *Learning to Localize Bugs in Software Code Using Transformer Models.* IEEE/ACM International Conference on Automated Software Engineering (ASE), 981–992.
9. Chen, M., Tworek, J., Jun, H., Yuan, Q., et al. (2021). *Evaluating Large Language Models Trained on Code.* arXiv:2107.03374.
10. Joshi, T., & Saha, D. (2022). *AI-Driven Debugging Tools: Leveraging Machine Learning to Detect and Repair Code Errors.* Journal of Software: Evolution and Process, 34(7), e2449.
11. Xu, Z., Zhao, Y., & Wang, T. (2022). *A Hybrid AI Framework for Real-Time Code Debugging and Recommendation.* IEEE Access, 10, 66742–66755.
12. Liang, P., Chen, Y., & Chen, Q. (2023). *Intelligent Debugging Assistant Based on Code Semantics and Transformer Models.* International Journal of Intelligent Systems, 38(5), 8219–8236.
13. Kim, D., Nam, J., Song, J., & Kim, S. (2013). *Automatic Patch Generation Learned from Human-Written Patches.* Proceedings of the 35th International Conference on Software Engineering (ICSE), 802–811.
14. White, M., Tufano, M., Vendome, C., & Poshyvanyk, D. (2019). *Sorting and Transforming Program Repair Ingredients via Deep Learning Code Representations.* Empirical Software Engineering, 24(6), 3269–3306.
15. Xie, X., Zhao, Y., & Hao, D. (2023). *Survey on Intelligent Software Debugging: From Heuristics to AI-Driven Systems.* ACM Transactions on Software Engineering and

Methodology, 32(3), 1–45.
16. Zhu, H., Li, X., & Wang, S. (2022). *LLM4Debug: Leveraging Large Language Models for Automated Debugging.* IEEE International Conference on Software Maintenance and Evolution (ICSME), 450–462.
17. Nguyen, T. T., & Nguyen, H. A. (2023). *A Comprehensive Review of AI-Assisted Debugging Tools for Modern Software Development.* Journal of Software Engineering Research and Development, 11(2), 98–115.
18. Wang, P., & Zhang, L. (2024). *Enhancing Software Reliability through AI-Driven Static and Dynamic Code Analysis.* Future Generation Computer Systems, 154, 132–147.
19. OpenAI. (2023). *GPT-4 Technical Report: Multimodal Model for Code Understanding and Reasoning.* arXiv:2303.08774.
20. Google DeepMind. (2023). *AlphaCode: Using Large-Scale Language Models for Competitive Programming and Debugging.* Science, 379(6639), 202–210.