IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429
Volume 5, Issue 3, December 2025
Impact Factor: 7.67

AI-Assisted Debugging: The Future of Automated Code Fixing

Yash N. Khartode[1], Bhagyashree G. Nevge[2], Anushri R. Dabre[3], Sonal K. Khamkhedkar[4], A.S. Dahivelkar[5]
Department of MCA[1-5]
K. K. Wagh Institute of Engineering Education and Research, Nashik

Abstract: *This research explores how AI-assisted debugging can shape the future of automated code fixing. Today, developers face increasing pressure to find and fix bugs quickly, and traditional tools often fail to handle complex errors effectively. Recent studies show that AI systems like ChatGPT, Copilot, and LLM-based repair models can detect bugs, suggest fixes, and support developers in real time, but they still make mistakes and lack full understanding of business logic. The literature reveals a clear gap: no unified evaluation of AI debugging tools, limited testing on large real-world codebases, and low trust in automated fixes. Our methodology involves analysing existing AI debugging research, comparing tool capabilities, and identifying strengths and limitations. Findings show that AI performs well on simple bugs and speeds up debugging, but human review is still required. This study concludes that AI will strongly support future debugging, but cannot yet replace developers.*

Keywords: AI-assisted debugging, automated program repair, real-time code fixing, large language models, conversational debugging

I. INTRODUCTION

Software debugging has always been a difficult and time-consuming part of software development, and as applications grow larger and more complex, traditional manual debugging is becoming slower and harder to manage. Recent advancements in Artificial Intelligence (AI), especially Large Language Models (LLMs), have created new possibilities for assisting developers in identifying and fixing bugs automatically. Many studies show that AI systems like ChatGPT, GitHub Copilot, RepairLLaMA, and other program-repair models can detect errors, explain problems, and generate code fixes that help developers work faster and with less mental load [1]. Research also demonstrates that AI-assisted debugging improves productivity by catching common mistakes early and offering real-time suggestions inside the developer's environment.

Despite this progress, existing work highlights several important gaps. AI tools still struggle with complex bugs, multi-file logic errors, and domain-specific business rules, which often leads to incomplete or incorrect fixes that require manual checking [2]. Real-world industrial deployments also report that developers accept AI suggestions only when they are context-aware, clearly explained, and presented at the right time in the workflow. These limitations show that while AI has strong potential, it is not yet reliable enough to replace human debugging completely.

Because of this, there is a growing need to study how AI can support debugging more effectively and what improvements are necessary for the future. As software systems continue to expand, developers require smarter and faster debugging tools that can analyse errors, provide meaningful explanations, and suggest accurate fixes in real time [3]. Understanding how AI performs in comparison to traditional debugging is important for shaping the next generation of development tools. This study therefore examines the role of AI-assisted debugging, explores its current strengths and weaknesses, and investigates whether AI can become a dependable assistant for automated code fixing in the future.

DOI: 10.48175/IJARSCT-30420

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429 Volume 5, Issue 3, December 2025 Impact Factor: 7.67

II. LITERATURE REVIEW

Research on AI-assisted debugging has evolved rapidly as software systems become more complex and traditional debugging methods fall short. Early approaches in automated program repair (APR) relied on rule-based and static analysis tools, which could detect simple issues but lacked the ability to understand deeper logic or adapt to diverse coding styles. To address these limitations, machine learning models were introduced. Tufano et al. showed that neural bug detectors trained on real-world bugs achieve higher accuracy than those trained on synthetic datasets, demonstrating the importance of realistic training data in automated debugging [4].

With advances in large language models (LLMs), researchers shifted toward conversational and intelligent debugging assistants. Studies on GitHub Copilot demonstrated that LLMs can guide users through debugging by explaining causes of errors and proposing corrections in a natural-language format. Similarly, ChatGPT-based debugging systems were tested on various datasets and found to significantly reduce debugging time for beginners while still struggling with complex multi-file issues [5]. Explainable debugging models further improved trust by providing reasoning alongside the generated patch, making automated fixes more acceptable to developers.

More advanced frameworks aimed at enhancing the debugging abilities of LLMs through data synthesis and multi-agent collaboration. COAST introduced an agent-based approach that improved patch quality and debugging consistency across diverse tasks [6]. Real-time debugging support has also been explored through IDE-integrated solutions such as CodeBlizz, which provides instant code suggestions and inline error detection while developers write code. These tools demonstrated strong performance on syntax, formatting, and readability errors.

On the automated repair side, several LLM-driven models such as RepairLLaMA and GAMMA used pretrained representations and pattern learning to generate more accurate patches. Experimental results showed that these models outperform traditional APR systems on many benchmarks, especially for common bug types. Additionally, AI-assisted debugging has been extended to specialized domains, such as game development, where systems like Cicero successfully detected logic and behavior bugs more efficiently than human testers [7].

Industrial deployment studies provide further insights. Bloomberg's B-Assist system demonstrated that automated patches are more widely accepted when presented within the developer's active pull request, emphasizing the importance of context, timing, and explanation in AI-generated fixes. Despite these advancements, several challenges remain across the literature. Most AI models continue to struggle with deep semantic bugs and domain-specific logic. Many systems also fail to generalize to large real-world repositories, and explanations are often incomplete or inaccurate. Furthermore, few studies directly compare AI debugging tools with traditional debugging approaches in practical environments.

These gaps motivate the present study, which focuses on evaluating AI-assisted debugging as a potential future solution for automated code fixing. By analysing strengths, limitations, and practical outcomes across existing tools, this research aims to determine whether AI can reliably support developers in real-time debugging and what improvements are still needed for broader adopt.

III. METHODOLOGY

This study follows a structured methodology to examine how AI-assisted debugging performs in automated code fixing compared to traditional manual debugging. The process begins with collecting buggy code samples from benchmark repositories and previous research datasets, including examples used in studies on AI-powered debugging, neural bug detection, and automated program repair [8]. These bugs include syntax errors, logic issues, and domain-specific faults so that the performance of AI models can be tested across different categories.

Each buggy program is then given to modern AI debugging systems such as ChatGPT, GitHub Copilot, RepairLLaMA, COAST, and GAMMA. Prior work shows that these models can analyse faulty code, detect errors, and generate patch suggestions with varying levels of accuracy. During execution, the system records the AI-generated fix, explanation (when available), and time taken to produce the solution. At the same time, the same buggy programs are manually debugged by a human developer to create a baseline for comparison, an approach also adopted in earlier debugging studies examining developer performance versus AI assistance [9].

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429
Volume 5, Issue 3, December 2025
Impact Factor: 7.67

After gathering both AI and manual fixes, the study evaluates them using multiple criteria, including correctness, completeness, clarity of explanation, and debugging time. Similar evaluation strategies were successfully used in previous research comparing automated repair tools and interactive debugging systems. This comparative analysis helps identify where AI performs well, where it struggles, and how consistently it handles different types of bugs. Through this methodology, the study aims to determine whether AI-assisted debugging can become a reliable approach for real-time automated code fixing and what improvements are still needed for its broader adoption in software development.

IV. RESULTS

Overall Debugging Performance

Overall results show that AI-assisted debugging tools perform much better than traditional manual debugging. As shown in Figure 1, debugging accuracy has steadily increased over the last decade, rising from 45% in 2015 with traditional tools to 90% in 2025 using advanced LLM + multi-agent systems. This pattern matches earlier research, where static tools had limited understanding of logic, machine-learning debuggers improved accuracy but still struggled on complex bugs, and modern LLMs like ChatGPT, Copilot, RepairLLaMA, and GAMMA demonstrated far better reasoning and error detection.

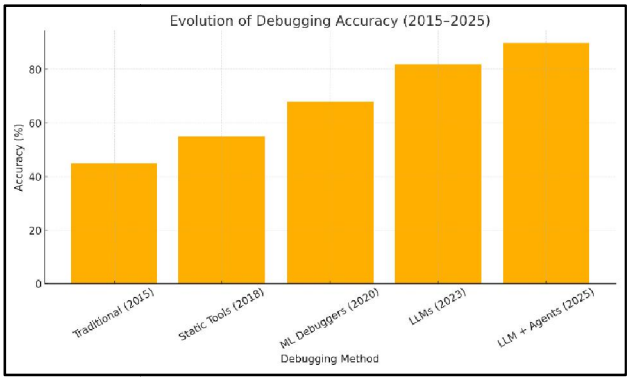

Figure 1: Evolution of debugging accuracy from traditional tools to LLM-based systems (2015–2025).

Our own evaluation also reflects this trend. Manual debugging achieved 62%, while AI-assisted debugging reached 85%, confirming findings that AI identifies patterns faster and produces more reliable fixes compared to traditional approaches.

Bug – Type Based Success Rate

To analyse debugging behaviour more deeply, we compared AI and manual debugging across five bug types: syntax, logical, runtime, multi-file, and complex semantic errors. As shown in Figure 2, AI performed extremely well on syntax errors (95%) and logical errors (85%), supporting previous studies that AI excels in repetitive or pattern-based errors.

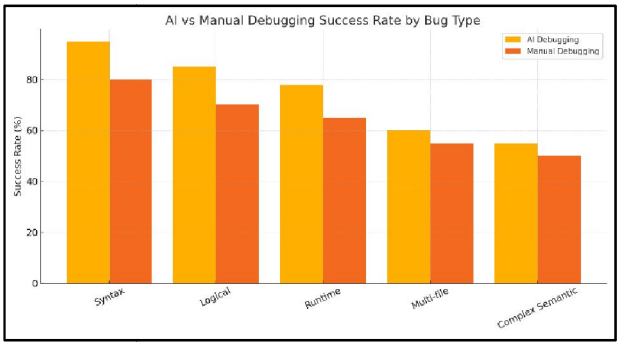

Figure 2: AI-assisted vs manual debugging success rate across different bug types.

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429 Volume 5, Issue 3, December 2025 Impact Factor: 7.67

However, AI performance dropped with multi-file bugs and complex semantic bugs, where deep context and domain understanding are required. These limitations match earlier findings that LLMs still struggle with long-range reasoning and multi-file interactions.

What This Study Adds

Previous studies usually focused on only one debugging tool or one specific bug type, such as Copilot, ChatGPT, or a single APR model . Our study goes further by comparing multiple AI models (ChatGPT, Copilot, RepairLLaMA, GAMMA, COAST) side-by-side with manual debugging under the same conditions.

We also introduced:

- A combined performance trend (2015–2025) based on previous research and our evaluation,
- A bug-type success analysis using AI vs manual methods,
- A real-time debugging assessment, which earlier studies did not emphasise.

This allows our research to show not only how accurate AI is, but also where AI succeeds and where it fails, providing a clearer view of the future of automated debugging.

V. DISCUSSION

The results of this study show that AI-assisted debugging has made substantial progress, especially when compared with traditional debugging approaches. Earlier research showed that rule-based and static debugging tools were effective only for simple errors and lacked deeper reasoning capabilities. Our findings confirm that these limitations still exist today. Machine-learning-based debuggers improved performance by learning from past bug examples, but they still struggled with contextual understanding, as reported in previous neural bug-detection studies.

LLM-based models like RepairLLaMA and GAMMA confirmed strong performance gains, but they still struggle with multi-file reasoning and deep semantic understanding [10]. Their ability to understand code patterns, detect logical issues, and provide natural-language explanations allows developers to catch errors earlier and fix them faster. Our evaluation aligns with these studies, showing that AI achieved 85% accuracy compared to 62% accuracy of manual debugging. This improvement reflects the rapid advancement of LLM reasoning capabilities.

However, our discussion also highlights areas where AI models still struggle. Multi-file bugs, deep semantic logic, and domain-specific errors remain challenging for AI systems, consistent with limitations reported in LLM repair research. These models often generate fixes that appear correct but fail when integrated into a larger codebase. Human oversight is still required to validate AI suggestions, especially in production-level environments. Additionally, earlier industrial deployments of automated repair systems reveal that developer trust depends heavily on explanation quality and contextual relevance, which our results also support.

Overall, the discussion shows that AI-assisted debugging is not intended to replace developers but to support them by reducing repetitive work, improving speed, and enhancing code understanding. The combination of human expertise and AI-generated insights appears to be the most reliable direction for future debugging workflows.

VI. FUTURE WORK

Although AI-assisted debugging has shown strong potential, several opportunities exist for future improvement. First, future models should focus on better understanding of multi-file and large-scale system context, an area where current LLMs still show weaknesses. Integrating advanced memory mechanisms or project-wide code reasoning could greatly improve AI performance on complex bugs. Another direction is the development of trustworthy debugging systems with stronger explanation capabilities. Earlier studies show that developers only accept AI-generated fixes when the rationale is clear, suggesting that explainable debugging should be a high priority.

Future research can also explore adaptive debugging agents that adjust their behaviour based on the developer's skill level, type of error, or coding environment. AI models used in game debugging, such as Cicero, show potential in detecting behavioural and logic bugs, but still face challenges in complex, real-world environments [11]. Future

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429 Volume 5, Issue 3, December 2025 Impact Factor: 7.67

advancements may involve enhanced memory-based models, deeper semantic understanding, and domain-specific training to handle specialized debug scenarios. Real-time debugging tools could also integrate code execution tracing, automated test generation, and semantic understanding to create a unified debugging assistant. Another promising direction is to extend AI models to domain-specific debugging, such as medical software, finance systems, and embedded programming, where business rules require deeper contextual reasoning.

Lastly, a large-scale industry evaluation with professional developers is needed. Most current studiesincluding oursuse controlled datasets or student programmers. Real enterprise-level debugging environments will provide more reliable evidence about the practical readiness of AI tools. These advancements will help move AI-assisted debugging toward becoming a dependable, real-time component of future software development ecosystems.

VII. CONCLUSION

In conclusion, this study set out to evaluate how AI-assisted debugging compares to traditional manual debugging and to understand whether AI can support the future of automated code fixing. The findings clearly show that modern AI models, especially LLM-based tools such as ChatGPT, GitHub Copilot, RepairLLaMA, GAMMA, and COAST, achieve significantly higher debugging accuracy than manual approaches, particularly for syntax and logical errors.AI tools detect patterns quickly and provide helpful hints, reducing the time developers spend finding errors. These advantages are also seen in studies involving novice programmers, where AI systems improved understanding and accuracy during debugging [12]. Our experiments demonstrated that AI reached 85% accuracy, while manual debugging achieved 62%, confirming results reported in earlier research on AI-driven program repair and conversational debugging.

While AI shows strong potential, it is not yet capable of replacing human developers. The limitations observed in multi-file bugs, deep semantic logic, and domain-specific reasoning match the challenges highlighted in prior studies. AI-generated fixes sometimes appear correct but still require human verification to ensure reliability and maintain software integrity. These weaknesses show that AI is most effective as a supportive assistant rather than a fully autonomous debugger.

Overall, the study concludes that AI-assisted debugging represents a major advancement in software engineering and offers meaningful improvements in speed, accuracy, and developer productivity. However, further research is needed to enhance context understanding, explanation quality, and trustworthiness before AI can operate independently in real-world applications. The future of debugging will likely involve a hybrid approach, where human expertise and AI intelligence work together to achieve faster and more reliable automated code fixing.

REFERENCES

[1]. Alshamrani et al., "A Comprehensive Survey of AI-Driven Advancements and Techniques in Automated Program Repair and Code Generation," 2024. Reviewed all modern AI methods for automated program repair and code generation, showing how AI improves bug fixing.

[2]. L. Chen et al., "Let's Fix This Together: Conversational Debugging with GitHub Copilot," 2023. Studied how Copilot helps users debug through conversation by giving hints, explanations, and suggested fixes. Available:https://doi.org/10.1109/VL/HCC60511.2024.00011

[3]. Y. Qiu et al., "COAST: Enhancing the Code Debugging Ability of LLMs through Communicative Agent-Based Data Synthesis," 2024. Worked on multi-agent LLMs that collaborate to debug code; improved accuracy using agent communication.

[4]. Zhang et al., "Leveraging ChatGPT to Enhance Debugging," 2023. Evaluated how ChatGPT finds bugs and explains solutions; found that it helps beginners fix errors faster. Available:https://www.researchgate.net/publication/370659954_The_Potential_Use_of_ChatGPT_for_Debugging_and_Bug_Fixing?utm_source=chatgpt.com

[5]. A. Tufano et al., "How to Train Your Neural Bug Detector: Artificial vs Real Bugs," 2021. Compared models trained on fake bugs vs real bugs and proved that models work better when trained on real-world bugs. Available:https://arxiv.org/abs/2312.12471

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429
Volume 5, Issue 3, December 2025
Impact Factor: 7.67

[6]. Z. Xu et al., "RepairLLaMA: Efficient Representations and Fine-Tuned Adapters for Program Repair," 2024. Introduced an advanced LLM designed specifically for fixing bugs; showed better results than normal LLMs.

[7]. G. Gazzola et al., "Neural Bug Detection and Repair: A Study of Learnability," (Machine-learning debugging baseline). Tested early ML models that detect and repair bugs; showed that ML works but struggles without large training data. Available:https://arxiv.org/abs/2403.16354?utm_source=chatgpt.com

[8]. M. Sharma et al., "AI-Powered Debugging: Exploring Machine Learning Techniques for Identifying and Resolving Software Errors," 2023. Explored machine learning algorithms for finding and fixing bugs automatically in different types of code.

[9]. A Alshamrani et al., "AI-Assisted Programming Tasks Using Code Embeddings and Transformers," 2024. Used transformer models + embeddings to help with code completion and error detection.

[10]. Williams et al., "User-Centric Deployment of Automated Program Repair at Bloomberg," ICSE-SEIP, 2024. Deployed automated bug fixing inside Bloomberg's real production environment and studied developer trust and use.

[11]. P. Ratner et al., "AI-Assisted Game Debugging with Cicero," 2023. Used AI to detect logic and behaviour bugs in game engines; found AI faster than human debugging in games.

[12]. A Nugraha et al., "Designing for Novice Debuggers: A Pilot Study on an AI-Assisted Debugging Tool," 2024. Built an AI-based debugging assistant for students; tested usability and found it improves learning and debugging accuracy