RADemics

Generative AI and Code Synthesis Frameworks for Automated Software Engineering and Debugging

Jayashree Pradip Tamkhade, Ananthi. M, Arati V. Deshpande
VISHWAKARMA INSTITUTE OF TECHNOLOGY, V.S.B. ENGINEERING COLLEGE, VISHWAKARMA INSTITUTE OF TECHNOLOGY

Generative AI and Code Synthesis Frameworks for Automated Software Engineering and Debugging

[1]Jayashree Pradip Tamkhade, Assistant Professor, Electronics and Telecommunications, Vishwakarma Institute of Technology, Kondhwa Campus, Laxminagar, Kondhwa, Pune-46. jayashree.tamkhade@vit.edu

[2]Ananthi. M, Assistant Professor, Information Technology, V.S.B. Engineering College, Karur. ananthivsb@gmail.com

[3]Arati V. Deshpande, Assistant Professor, Computer Engineering, Vishwakarma Institute of Technology, Pune, arati.deshpande1@vit.edu

Abstract

The rapid advancement of Artificial Intelligence (AI) has brought transformative changes to software engineering, particularly in the areas of debugging and code synthesis. This chapter explores the integration of AI-driven techniques within the software development lifecycle, with a focus on automated bug fixing, debugging complex systems, and continuous integration (CI/CD). As software systems grow more intricate and performance-sensitive, the traditional methods of bug detection and resolution struggle to keep pace. AI-based tools, through machine learning and deep learning models, offer significant improvements by automating the identification of defects, optimizing code generation, and enhancing real-time debugging processes. Key applications, such as anomaly detection, root cause analysis, and predictive debugging, are discussed in the context of large-scale systems, real-time environments, and high-performance applications. The chapter also highlights the challenges in achieving accuracy, reducing false positives, and ensuring seamless integration of AI tools into existing development workflows. By leveraging AI in continuous deployment, this work underscores the potential for more efficient and reliable software production, allowing for faster bug identification, reduced downtime, and improved software quality. The evolving role of AI in debugging is examined through case studies, exploring its real-world impact on enhancing the agility and robustness of modern software systems. As AI continues to evolve, it is poised to redefine software engineering paradigms, making automated debugging and code synthesis indispensable for the future of software development.

Keywords: Artificial Intelligence, Automated Bug Fixing, Debugging, Continuous Integration, Anomaly Detection, Predictive Debugging.

Introduction

The landscape of software engineering has undergone a monumental transformation over the past few decades [1]. As technology advances and software systems become increasingly complex, traditional debugging methods are no longer sufficient to maintain the performance and stability of modern applications [2]. In the past, debugging was largely a manual, error-prone process,

where developers combed through lines of code to identify and resolve issues [3]. The emergence of Artificial Intelligence (AI) in the software development lifecycle has revolutionized this process, providing advanced techniques that can automate bug detection, optimize code, and ensure system reliability with greater accuracy [4]. As modern software systems grow larger, more dynamic, and interconnected, there is an increasing need for AI-powered tools to handle the complexity and scale that traditional methods cannot manage effectively [5].

At the heart of AI's application in debugging is machine learning (ML), a subset of AI that enables systems to learn from data and make predictions without explicit programming [6]. Machine learning algorithms are now being applied to identify patterns and anomalies in software behavior, significantly enhancing the bug detection process [7]. By analyzing large datasets of system logs, error reports, and application performance metrics, AI models can learn what constitutes normal system behavior and flag deviations that might indicate a bug [8]. The power of machine learning lies in its ability to uncover hidden patterns, making it possible to detect complex, non-obvious bugs that traditional static analysis tools might miss [9]. As a result, AI techniques such as anomaly detection and predictive debugging are becoming indispensable for modern software development [10].

AI-based debugging is not confined to detecting bugs in isolation; it can also address issues related to the scalability and performance of complex, distributed systems [11]. The rise of cloud-native applications and microservices has introduced new challenges for developers, including issues with resource management, system dependencies, and real-time data processing [12]. AI-powered debugging tools are adept at analyzing these large-scale systems, where understanding the interactions between various components and identifying bottlenecks can be overwhelming [13]. By continuously monitoring the system's behavior, AI tools can predict potential failures and proactively alert developers about underlying issues before they cause system outages or degrade user experience [14]. This predictive capability significantly improves system reliability, reduces downtime, and enhances overall operational efficiency [15].

In the context of continuous integration (CI) and continuous deployment (CD), AI has proven to be an invaluable asset [16]. The CI/CD pipeline is designed to streamline software development by automating the testing and deployment of code, ensuring faster release cycles and higher-quality software [17]. The rapid pace of releases in CI/CD environments can introduce challenges in debugging, as issues may not be detected until after deployment in a production environment [18]. AI models integrated within the CI/CD pipeline can continuously monitor code during development, detect potential bugs in real time, and even suggest fixes before the code is deployed to production [19]. This integration allows for a proactive debugging approach, where bugs are resolved during the development process, preventing costly post-deployment failures and minimizing the need for manual interventions [20].