GJR Publication
OPEN ACCESS

Global Journal of Research in Engineering & Computer Sciences
ISSN: 2583-2727 (Online)
Volume 06 | Issue 01 | Jan.-Feb. | 2026
Journal homepage: https://gjrpublication.com/gjrecs/

Original Research Article

Predictive Analytics and Self-Healing Systems Using Artificial Intelligence for Software Maintenance

[1]Dr. Mukhtar Ibrahim Bello*, [2]Bello Abubakr Imam, [3]Abdulhamid Mahmoud Shariff, [4]Shehu Hassan Ayagi, [5]Sani Ahmad Muhammad, and [6]Kabir Dalha Kabir

[1,2,3,4,5,6] Department of Computer Science, Kano State Polytechnic

DOI: 10.5281/zenodo.18601001 **Submission Date: 31 Dec. 2025 | Published Date: 10 Feb. 2026**

***Corresponding author:** **Dr. Mukhtar Ibrahim Bello**
Department of Computer Science, Kano State Polytechnic

Abstract

Using artificial intelligence (AI), this study investigated real-time fault prediction and recovery in order to create a framework for software Maintenance and self-healing system. The majority of automation testing frameworks require assistance in anticipating and effectively recovering from errors, despite the expanding significance of software quality. In order to improve the resilience and adaptability of testing frameworks through self-healing mechanisms, this study makes use of AI, particularly machine learning algorithms. The first step in this study is a comprehensive analysis of the constraints in the testing systems and automation testing procedures already in use. The use of AI in software testing is then demonstrated, along with various viewpoints on how machine learning might improve the precision of fault detection and prediction. AI is being investigated in conjunction with predictive maintenance plans and data analysis techniques, which enable the early but accurate identification and avoidance of issues. We concentrate on creating automated recovery procedures that dynamically modify tests based on data collected in real time. To demonstrate real-world uses of self-healing algorithms in various software settings, case studies are provided. Thus, the study identifies implementation hurdles (economic organizational) and technical challenges (such AI integration and scalability). This study offers useful suggestions for implementing AI-driven self-healing frameworks with significant ramifications for industry stakeholders. Lastly, the prospect of an automated future is discussed in this paper's conclusion.

Keywords: *Artificial Intelligence, Self-Healing, Machine Learning, Fault Prediction, Real-Time Recovery, Software Development, Predictive Maintenance.*

1. INTRODUCTION

in an era marked by rapid technological advancement, the complexities of software systems continue to grow, necessitating innovative approaches to their maintenance. As organizations increasingly rely on software to drive their operations, the challenges associated with managing and sustaining these systems become more pronounced. Traditional maintenance strategies often struggle to keep pace with the dynamic nature of software environments, leading to increased downtime and diminished performance. This is where artificial intelligence (AI) enters the equation, offering transformative solutions through predictive analytics and self-healing systems. Predictive analytics harnesses vast amounts of data to identify patterns and forecast potential issues before they disrupt operations, while self-healing systems enable software to autonomously address and rectify malfunctions. Together, these technologies represent a paradigm shift in software maintenance, empowering organizations to enhance efficiency, reduce costs, and ensure seamless user experiences in an increasing digital landscape.

Software maintenance is a critical phase in the Software Development Life Cycle (SDLC) that ensures the continued functionality, reliability, and relevance of software systems after their initial deployment [1]. Unlike the design and

development phases, which focus on creating software, maintenance addresses the ongoing challenges of fixing defects, optimizing performance, adapting to new requirements, and ensuring compatibility with evolving technologies [2].

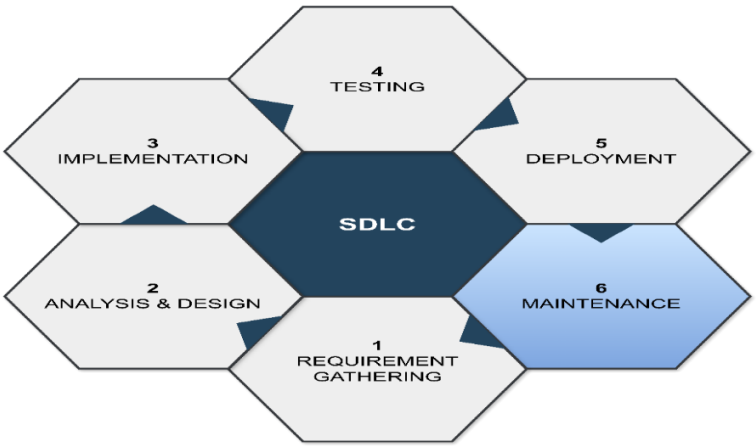

Figure 1: SDLC Phases Highlighting AI Integration in the Maintenance Phase

Traditional software maintenance, however, faces significant challenges. One of the primary issues is the high cost associated with maintenance activities, which can consume a substantial portion of the software's total lifecycle budget [3, 4, 5]. Manual intervention in detecting and resolving issues is another challenge, often leading to delays and human errors [6, 7]. Furthermore, traditional methods struggle to adapt to dynamic and rapidly changing operating environments, making systems prone to failures and inefficiencies [8]. Software maintenance is a crucial phase in the Software Development Life Cycle (SDLC), ensuring that deployed systems remain functional, efficient, and adaptable to changing requirements. However, traditional approaches to software maintenance are fraught with numerous challenges [9]. These include the high cost of maintenance activities, reliance on manual processes, and difficulty in adapting to dynamic environments [10]. These limitations often lead to prolonged system downtimes, increased human error, and inefficiencies that hinder the overall performance and reliability of software systems [11]. In recent years, artificial intelligence (AI) technologies have emerged as transformative tools to address these challenges. AI-driven approaches, such as predictive analytics and self-healing systems, offer innovative solutions to reduce costs, minimize human dependency, and enhance system adaptability. Predictive analytics utilizes historical and real-time data to forecast potential issues, enabling proactive maintenance interventions. Similarly, self-healing systems leverage machine learning algorithms to detect and resolve performance bottlenecks autonomously, ensuring minimal disruption to system operations [12]. Software Maintenance in the SDLC Software maintenance is defined as the process of modifying and updating software applications after their initial deployment to correct faults, improve performance, and adapt to a changing environment [13]. It plays a pivotal role in ensuring the long-term functionality and reliability of software systems. Maintenance encompasses several activities, including error correction, performance optimization, and adapting the software to meet evolving user needs and technological advancements. The significance of software maintenance lies in its ability to extend the lifespan of software systems, ensuring their continued relevance and efficiency. Without proper maintenance, software can become obsolete, leading to operational disruptions, security vulnerabilities, and increased costs associated with system replacements [14].

2. METHODOLOGIES

Research Design We use both qualitative and quantitative research designs in this study's mixed methods approach to provide a more thorough examination of AI-powered self-healing automation testing frameworks. This strategy aims to provide a more comprehensive understanding of the topic by combining the best elements of both approaches. Using case studies and industry reports, the qualitative method gathers insights. Researchers are able to gain a deeper understanding of the subtleties of present practice and the challenges encountered in implementing the self-healing system thanks to this method. By examining real-world instances, the qualitative analysis highlight's themes and patterns that quantitative data can overlook, giving the results more context and nuance. This approach is particularly useful for understanding practitioners' and tech workers' subjective experiences.

2.1 XGBoost and Application

XGBoost is a machine learning algorithm based on gradient boosting, which is an ensemble technique that combines the predictions of multiple base learners (decision trees) to improve accuracy. XGBoost builds models by fitting a sequence of decision trees where each subsequent tree attempts to correct the errors made by the previous trees. Figure 2 below shows the architectural design and components of the XGBoost model.

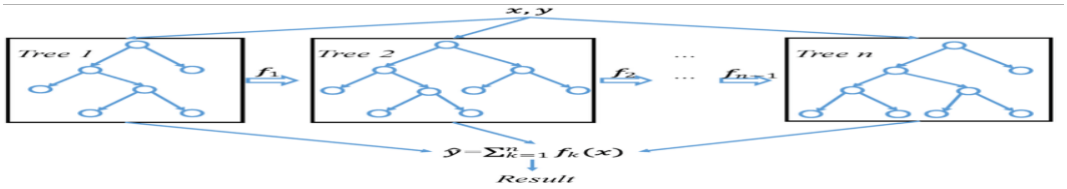

Figure 2: XGBoost architectural design

2.2 Gradient Boosting Framework

The gradient boosting algorithm is formulated as follows: $\hat{y}_i = \sum_{k=1}^{K} f_k(x_i)$

Where:

$\hat{y}_i$ is the predicted output for the *i*-th instance,
x_i is the feature vector of the *i*-th instance,
$f_k(x_i)$ is the output of the *k*-th base learner (decision tree),
K is the total number of base learners.

The key idea in gradient boosting is to iteratively add new models (trees) to minimize a loss function. At each iteration, the algorithm fits the next model to the residual errors (the difference between actual and predicted values) produced by the existing ensemble of trees.

2.3 Training Procedure

1. Data Splitting: The dataset is first split into training, validation, and test sets, typically using a 70-15-15 split.
2. Feature Engineering: Time-series data is aggregated to form features such as moving averages, sliding window statistics, and frequency domain features (Fourier transforms) to capture both short-term and long-term trends.
3. Hyperparameter Tuning: Using cross-validation, key hyperparameters such as the learning rate, max depth, number of trees, and subsampling rate are optimized to minimize overfitting and improve generalization.

3. RESULT

Predictive analysis in the context of software maintenance leverages historical data to forecast future system behaviors and identify potential issues before they occur. By utilizing machine learning models, such as XGBoost, this approach aims to enhance the efficiency of software systems by providing actionable insights. In this section, we explore the application of predictive analytics to sensor data, evaluating the performance of the trained model and its ability to accurately predict future states, ultimately supporting the development of self-healing systems.

Figure 5 and 6 presents the sensor readings for the first 100 cycles, along with the predicted values generated by the XGBoost model (represented by the red lines). The model was trained for 100 epochs and fine-tuned using both L1 and L2 regularization techniques to enhance its generalization capabilities and prevent overfitting. The plot shows that the model's predictions closely follow the actual sensor data, demonstrating a high degree of accuracy and reliability. No significant discrepancies or abnormalities are observed, indicating that the model is effectively capturing the underlying patterns in the time-series data.

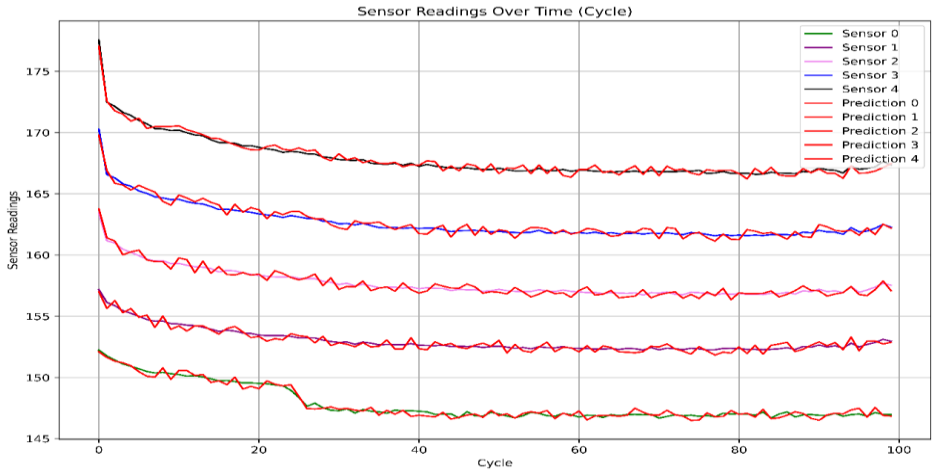

Figure 5: First 100 Cycles of the sensor readings

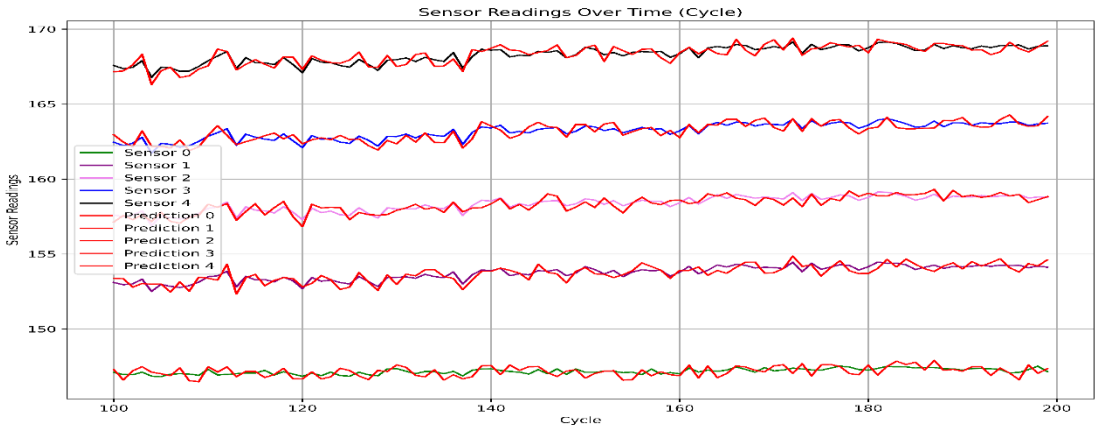

Figure 6: Second 200 Cycles of the sensor readings

4. CONCLUSION

In conclusion, the integration of artificial intelligence in software maintenance, particularly through predictive analytics and self-healing systems, marks a significant evolution in how organizations manage their IT infrastructure. By utilizing advanced data processing and machine behavior models, predictive maintenance enhances equipment reliability and operational efficiency, ultimately reducing downtime. The increasing reliance on interconnected machines and the flow of data necessitates a robust response; thus, the adoption of intelligent systems becomes indispensable for maintaining competitive advantage in today's fast-paced technological landscape [15]. Furthermore, as we transition towards a Computing Continuum, where cloud, edge, and IoT converge, the potential for AI-driven solutions will only expand, providing organizations with novel opportunities to optimize their maintenance strategies [16]. Embracing these advancements will define the future trajectory of software maintenance, making it more proactive and resilient to evolving challenges.

5. RECOMMENDATIONS

1. **Further Optimization of the RL Agent**
   Although the RL agent performed well in fault detection and mitigation, there is room for further optimization. Enhancements to the agent's learning algorithm could enable it to handle more complex fault scenarios, including those involving multiple concurrent failures or catastrophic system breakdowns. Exploring alternative reinforcement learning techniques, such as deep reinforcement learning or multi-agent systems, could potentially improve the agent's decision-making capabilities.
2. **Implementation in Real-World Environments**
   While the proposed model demonstrated success in a simulated environment, real-world testing would help validate its practicality and robustness. Deploying the self-healing system in an actual hydraulic rig would provide further insights into the system's performance under varying operational conditions and identify areas for improvement.
3. **Continuous Monitoring and Feedback Loop**
   Implementing a continuous monitoring system, which gathers feedback from real-time operations, would allow the RL agent to adapt dynamically to evolving system conditions. By incorporating new data and feedback, the agent can continuously improve its decision-making and healing process, leading to better long-term performance.

6. REFERENCES

1. de Vicente Mohino, J., Bermejo Higuera, J., Bermejo Higuera, J. R., & Sicilia Montalvo, J. A. (2019). The application of a new secure software development life cycle (S-SDLC) with agile methodologies. *Electronics, 8*(11), 1218. https://doi.org/10.3390/electronics8111218
2. Muller, A., Crespo Marquez, A., & Iung, B. (2008). On the concept of e-maintenance: Review and current research. *Reliability Engineering & System Safety, 93*(8), 1165–1187. https://doi.org/10.1016/j.ress.2007.08.006
3. Galar, D., Sandborn, P., & Kumar, U. (2017). *Maintenance costs and life cycle cost analysis.* CRC Press.
4. Banker, R. D., Datar, S. M., Kemerer, C. F., & Zweig, D. (1993). Software complexity and maintenance costs. *Communications of the ACM, 36*(11), 81–95. https://doi.org/10.1145/163359.163375
5. Levitt, J. (2009). *The handbook of maintenance management.* Industrial Press Inc.
6. Nagaria, B. (2021). *An investigation of human error in software development* (Doctoral dissertation). Brunel University London.
7. Makoondlall, Y. (2020). *A requirement-driven approach for modelling software architectures* (Doctoral dissertation). Kingston University.

8. Igwe, H. (2024). *The significance of automating the integration of security and infrastructure as code in software development life cycle* (Doctoral dissertation). Purdue University.
9. Kumar, B. (2022). AI implementation for predictive maintenance in software releases. *International Journal of Research and Review Techniques, 1*(1), 37–42.
10. Dwivedi, Y. K., Hughes, L., Ismagilova, E., Aarts, G., Coombs, C., Crick, T., Duan, Y., et al. (2021). Artificial intelligence (AI): Multidisciplinary perspectives on emerging challenges, opportunities, and agenda for research, practice and policy. *International Journal of Information Management, 57,* 101994. https://doi.org/10.1016/j.ijinfomgt.2019.101994
11. McMillan, L. (2023). *Artificial intelligence–enabled self-healing infrastructure systems* (Doctoral dissertation). University College London, University of London (United Kingdom).
12. Ammar, M., Haleem, A., Javaid, M., Bahl, S., & Verma, A. S. (2022). Implementing Industry 4.0 technologies in self-healing materials and digitally managing the quality of manufacturing. *Materials Today: Proceedings, 52,* 2285–2294. https://doi.org/10.1016/j.matpr.2021.10.381
13. Ibrahim, M. S., & Saber, S. (2023). Machine learning and predictive analytics: Advancing disease prevention in healthcare. *Journal of Contemporary Healthcare Analytics, 7*(1), 53–71.
14. Nama, P., Reddy, P., & Pattanayak, S. K. (2024). Artificial intelligence for self-healing automation testing frameworks: Real-time fault prediction and recovery. *Artificial Intelligence, 64*(3S).
15. Shahane, V. (2024). Towards real-time automated failure detection and self-healing mechanisms in cloud environments: A comparative analysis of existing systems. *Journal of Artificial Intelligence Research and Applications, 4*(1), 136–158.
16. Ibidunmoye, O., Hernández-Rodriguez, F., & Elmroth, E. (2015). Performance anomaly detection and bottleneck identification. *ACM Computing Surveys (CSUR), 48*(1), 1–35. https://doi.org/10.1145/2791120

CITATION
Bello, M. I., Imam, B. A., Shariff, A. M., Ayagi, S. H., Muhammad, S. A., & Kabir, K. D. (2026). Predictive Analytics and Self-Healing Systems Using Artificial Intelligence for Software Maintenance. In Global Journal of Research in Engineering & Computer Sciences (Vol. 6, Number 1, pp. 109–113).
https://doi.org/10.5281/zenodo.18601001