Eksplorium *p*-ISSN 0854-1418

Volume 46 No. 2, June 2025: 155–167 *e*-ISSN 2503-426X

"Design And Develop Computational Models to Reduce the Maintenance Cost at Deployment Level"

[1]Ankit Kumar, [2]Dr. Shashiraj Teotia

[1]Research scholar, Department of Computer Application, Swami Vivekanand Subharti University Meerut, UP, INDIA

ankitbaliyan042@gmail.com

[2]Research Supervisor, Department of Computer Application, Swami Vivekanand Subharti University Meerut, UP, INDIA

shashirajt@gmail.com

Article Received: 12 May 2025, Revised: 15 June 2025, Accepted: 22 June 2025

Abstract: Software maintenance at the deployment stage remains a significant contributor to the total cost of ownership (TCO) in software engineering. Despite advanced development practices, many organizations experience escalated post-deployment maintenance due to unpredictable failures, inefficient resource utilization, and lack of intelligent monitoring systems. This paper presents the design and implementation of computational models that leverage machine learning and statistical methods to predict maintenance risks, automate diagnostics, and optimize resource allocation. Experimental results demonstrate a substantial reduction in maintenance efforts and costs when applied to real-world deployment environments. The proposed models offer scalable and intelligent solutions for enhancing software maintainability in production systems.

Keywords: Deployment-level maintenance, machine learning, computational models, predictive diagnostics, software reliability, TCO reduction, DevOps, anomaly detection.

1. INTRODUCTION

The increasing complexity of software systems has elevated the importance of effective maintenance strategies, particularly at the deployment level. Studies suggest that maintenance costs can consume over 60% of the total software lifecycle expenditure. This is primarily due to reactive approaches to bug fixing, manual monitoring, and suboptimal resource management. Computational models offer a promising avenue for proactive, intelligent, and cost-efficient maintenance strategies. In the modern era of large-scale and cloud-based applications, the **deployment phase** of software systems has become one of the most **critical stages** in the software development lifecycle (SDLC). While traditional development processes focus heavily on coding and testing, **post-deployment maintenance** is often under-resourced—despite it being one of the **most costly and risk-prone phases**. Research by Gartner and IEEE Software Engineering studies has shown that **over 60% of the total cost** of ownership (TCO) for a software product is consumed in post-deployment maintenance. These costs arise from emergency bug fixes, performance issues, security patching, user support, and infrastructure optimization.

BACKGROUND

Software maintenance is a critical phase of the software development lifecycle, often incurring the highest cost over time. Studies have consistently shown that more than 60% of the total cost of ownership in software systems is consumed by maintenance-related activities. As software systems grow in scale and complexity—particularly in dynamic, cloud-native, and

containerized environments—the challenges of ensuring reliability, availability, and performance at the deployment stage become increasingly significant.

Traditionally, deployment-level maintenance has relied on reactive mechanisms, including manual bug tracking, rule-based alert systems, and static resource provisioning. These approaches often lead to delayed issue detection, inefficient resource usage, and prolonged downtime, thereby inflating operational costs and impacting user experience.

Recent advancements in artificial intelligence (AI), machine learning (ML), and data-driven DevOps offer new possibilities for transforming deployment maintenance from reactive to proactive. By leveraging historical logs, telemetry data, and real-time monitoring metrics, computational models can be trained to predict failures, detect anomalies automatically, and optimize infrastructure usage.

Moreover, the integration of these models within Continuous Integration/Continuous Deployment (CI/CD) pipelines facilitates real-time decision-making and adaptive resource control. This research is positioned at the intersection of intelligent automation and software operations, aiming to reduce maintenance costs through scalable, learning-based solutions that adapt to evolving software environments.

1.1. CHALLENGES IN DEPLOYMENT-LEVEL MAINTENANCE

The high cost of deployment-level maintenance can be attributed to several underlying factors:

- **Reactive Maintenance**: Most organizations still operate on a reactive model, addressing bugs and system issues *after* users encounter them. This often leads to service-level agreement (SLA) violations and customer dissatisfaction.
- **Manual Monitoring**: Conventional monitoring methods rely heavily on human intervention for log analysis, system observation, and anomaly identification—an inefficient and error-prone approach.
- **Resource Inefficiency**: Inadequate or excessive allocation of resources such as CPU, memory, or storage due to static provisioning or poor forecasting results in cost overruns or system crashes.
- **Lack of Predictive Capabilities**: Existing systems often lack the ability to foresee failures or performance bottlenecks based on historical patterns or current behavior.

These limitations collectively **escalate maintenance costs**, reduce system reliability, and increase the burden on DevOps and support teams.

1.2. ROLE OF COMPUTATIONAL MODELS

To address these challenges, **computational models**—including machine learning (ML), artificial intelligence (AI), and statistical methods—offer a **proactive, intelligent, and automated** approach to system maintenance. In modern software systems, traditional maintenance approaches are proving increasingly inadequate due to the complexity, scale, and dynamic behavior of deployed applications. To overcome these limitations, computational models based on **machine learning (ML)**, **artificial intelligence (AI)**, and **statistical**

techniques offer a proactive and intelligent framework for managing deployment-level maintenance.

These models are designed to **learn from historical datasets** such as system logs, error reports, usage trends, performance metrics, and incident records. By identifying patterns and correlations in this data, the models can **anticipate failures**, **detect real-time anomalies**, and **optimize system behavior**, all while reducing human effort and improving system uptime.

One of the primary advantages is the ability to **forecast potential failures**. For instance, ML classifiers trained on historical failure logs can predict which components are likely to malfunction under certain load or usage conditions. This predictive capability allows teams to take corrective actions before a failure actually impacts the system.

Furthermore, **real-time anomaly detection** is critical in deployment environments where even minor issues can cascade into significant outages. Using unsupervised learning methods such as clustering or neural autoencoders, computational models can flag deviations from normal behavior—such as memory leaks, CPU spikes, or unusual response times—prompting timely investigation and resolution.

Another significant benefit is the ability to **recommend or automate resource allocation**. Deployment environments often suffer from inefficient resource usage due to static provisioning. Using reinforcement learning or optimization algorithms, computational models can analyze workload patterns and suggest or enact dynamic scaling, load balancing, and container orchestration decisions—reducing both underutilization and operational costs.

These computational models offer three major benefits:

- **Scalability**: They can operate effectively across highly distributed architectures, such as cloud-native or microservice-based systems.
- **Automation**: They drastically reduce the need for manual monitoring, log inspection, and rule-based scripts by continuously learning and acting autonomously.
- **Adaptability**: These models can be retrained or fine-tuned over time to adjust to evolving application behavior, new deployment patterns, or updated infrastructure configurations.

In summary, computational models provide a **data-driven, intelligent, and efficient** framework for managing post-deployment maintenance. Their ability to predict, detect, and optimize not only enhances system reliability but also delivers **tangible reductions in maintenance costs and operational overhead**.

1.3. RESEARCH OBJECTIVES

This study focuses on **designing and developing a suite of computational models** that work cohesively to reduce maintenance costs and enhance system robustness at the deployment level. The objectives of this research are:

1. Predict Potential Failures

Utilizing supervised learning algorithms trained on historical bug reports, crash logs, and operational metrics, we aim to develop a **predictive failure model** that flags high-risk components and deployment scenarios in advance.

2. Automate Anomaly Detection

By employing unsupervised models (e.g., autoencoders, clustering techniques), we seek to detect performance anomalies, unexpected behaviors, or security deviations in real-time, enabling **early warnings** and **preemptive fixes**.

3. Optimize System Resource Usage

Through reinforcement learning and optimization techniques, the research targets **smart resource provisioning** (e.g., autoscaling, load balancing) that dynamically adjusts computing resources based on predicted loads, thereby **reducing infrastructure and maintenance costs**.

1.4 LITERATURE REVIEW

The increasing complexity of deployment environments in modern software systems has drawn considerable attention to predictive maintenance, anomaly detection, and intelligent resource optimization. Numerous studies have shown that integrating machine learning into maintenance workflows can significantly reduce operational costs and enhance system reliability. Smith et al. (2021) explored the use of supervised learning models such as Random Forest and Support Vector Machines for predicting software failures, highlighting their ability to analyze historical logs and classify high-risk scenarios before failures occur. Similarly, Nair and Gupta (2020) reported that logistic regression and decision trees offer improved detection accuracy over traditional rule-based methods in crash prediction.

In parallel, deep learning approaches have gained traction for their effectiveness in detecting anomalies within dynamic deployment environments. Patel et al. (2023) proposed a neural architecture using stacked autoencoders to monitor and detect unusual behavior in cloud-based systems, demonstrating high performance in identifying resource spikes and preventing downtime. Jeong and Choi (2023) advanced this concept by incorporating time-series metrics and system call patterns to enhance the contextual sensitivity of anomaly detection models.

Resource optimization has also been a major focus in recent literature. Chen and Zhang (2020) utilized reinforcement learning to design agents capable of managing virtual machine scaling, leading to substantial reductions in compute costs. Similarly, Tran and Bui (2019) showed that Q-learning-based strategies adapt effectively to fluctuating workloads and outperform static threshold-based autoscaling policies.

Recent efforts have emphasized the importance of integrating these models directly into DevOps toolchains. Lee and Wang (2022) proposed a machine learning framework for deployment diagnostics that is natively embedded in Kubernetes-based CI/CD pipelines. White and Morris (2021) underscored the value of incorporating Prometheus, Grafana, and automated alert systems to deliver actionable insights to developers in real time. Despite these advances, existing approaches often suffer from limited generalizability across diverse software

architectures, scalability challenges in large-scale environments, and a lack of unified integration between predictive, diagnostic, and optimization functions. These research gaps motivate the hybrid approach proposed in this study, which aims to offer a modular, scalable, and deployment-aware computational model to address maintenance cost reduction holistically.

1.5 SCOPE AND SIGNIFICANCE

The proposed computational models are designed to be:

- **Technology-agnostic**: Usable across various platforms (e.g., Kubernetes, AWS, Azure).
- **Lightweight and Real-time**: Suitable for integration into continuous deployment pipelines (CI/CD).
- **Cost-focused**: Specifically evaluated for their ability to reduce time-to-repair (MTTR), improve uptime, and lower operational expenditures (OPEX).

By integrating these intelligent models into the software deployment pipeline, organizations can transition from reactive maintenance to a **predictive and preventive** paradigm—ensuring higher quality of service (QoS), improved system availability, and **substantial cost savings**.

RELATED WORK

Prior studies have introduced fault prediction systems and DevOps automation tools, but with limited integration into a unified computational model focused explicitly on reducing maintenance costs. Notable research includes:

- Fault localization techniques using static and dynamic analysis [Smith et al., 2021].
- Deployment-aware ML pipelines [Lee and Wang, 2022].
- Anomaly detection using deep learning in cloud systems [Patel et al., 2023].

However, the gap remains in developing adaptable, lightweight, and generalizable models for real-time deployment-level application.

3. METHODOLOGY

3.1 DATA COLLECTION

We collected over 1.2 million log events and 10,000 issue reports from three enterprise systems over 18 months. This dataset included crash reports, server logs, CPU/memory usage metrics, and patch deployment histories. To develop effective computational models for reducing deployment-level maintenance costs, a systematic methodology was employed—encompassing data collection, feature engineering, model design, and system integration within a live deployment environment.

The study began with the **collection of extensive operational data** from three large-scale enterprise applications deployed in production over an 18-month period. The dataset consisted of over **1.2 million log events** and **10,000 issue reports**, including crash logs, service interruption reports, system metrics (CPU, memory usage), and patch deployment histories. This provided a rich foundation for training and validating predictive and adaptive models.

In the **feature engineering phase**, we extracted relevant characteristics from the raw data to serve as input variables for model training. Key features included: frequency of component-level failures, average time intervals between deployments and subsequent failures, dynamic resource utilization trends, and code-level complexity metrics (such as cyclomatic complexity and code churn). These features were standardized and normalized to improve model accuracy and convergence.

For **model design**, a hybrid architecture was adopted to address the multi-faceted nature of deployment-level maintenance challenges. A **Random Forest Classifier** was used for **failure prediction**, chosen for its robustness, interpretability, and ability to handle imbalanced classes. It classified potential failure scenarios based on temporal and behavioral patterns in the data. For **anomaly detection**, an **autoencoder-based neural network** was implemented to capture normal behavior profiles and flag metric anomalies in real-time. The reconstruction error threshold was tuned to minimize false positives while maintaining sensitivity to rare events. In the **resource optimization component**, a **reinforcement learning (RL) agent** was trained in a simulated environment to learn optimal virtual machine (VM) scaling policies. The RL agent received reward signals based on system performance, cost efficiency, and SLA adherence.

The complete model suite was deployed within a **Kubernetes-based CI/CD pipeline**, ensuring real-time feedback and automation. System metrics were continuously monitored using **Prometheus**, while **Grafana dashboards** provided visualization for DevOps teams. An intermediate **API layer** facilitated communication between the models and the system, enabling dynamic alert generation and automated rollback or scaling actions. Feedback from these actions was fed back into the model training loop to ensure continuous learning and adaptation.

This methodology not only supports proactive maintenance decisions but also embeds intelligence directly into the software deployment lifecycle—leading to measurable reductions in downtime, manual intervention, and infrastructure expenditure.

3.2 FEATURE ENGINEERING

Important features include:

- Frequency of component failures
- Time between updates and failures
- Resource consumption patterns
- Code complexity metrics

3.3 MODEL DESIGN

We designed a hybrid model architecture:

- **Failure Prediction**: Random Forest Classifier for classifying probable failure scenarios.
- **Anomaly Detection**: Autoencoder neural network for real-time metric anomaly detection.
- **Resource Optimization**: Reinforcement learning agent trained to manage VM scaling decisions.

3.4 DEPLOYMENT ARCHITECTURE

Models are integrated into a Kubernetes-based CI/CD pipeline with Prometheus for monitoring and Grafana for visualization. An API layer triggers alerting and feedback loops for developers. The proposed computational models—responsible for failure prediction, anomaly detection, and resource optimization—are deployed within a **Kubernetes-based CI/CD (Continuous Integration/Continuous Deployment) pipeline**, enabling automated and scalable integration into real-world environments.

Kubernetes is used as the orchestration platform due to its ability to manage containerized microservices efficiently. Each model is containerized using Docker and deployed as a microservice within a Kubernetes cluster. This modular deployment ensures scalability, fault isolation, and easy maintenance. The system automatically scales model instances based on the load, making it ideal for enterprise environments.

To enable **real-time monitoring**, **Prometheus** is employed as the primary telemetry and metric collection tool. Prometheus continuously scrapes data from deployed services (such as CPU usage, memory, request latency, failure frequency) and stores time-series data. This data is critical for both the anomaly detection model and the reinforcement learning agent that handles resource scaling.

For **visualization and developer insight**, **Grafana** dashboards are configured. These dashboards display real-time alerts, resource utilization trends, and failure prediction probabilities, allowing developers and operations teams to monitor system health and performance effectively.

Additionally, an **API layer** is introduced between the models and the DevOps infrastructure. This layer serves multiple purposes:

- It **triggers alerts** whenever anomalies or high failure probabilities are detected.
- It **sends recommendations** to developers based on model outputs, such as suggesting patching actions or configuration changes.
- It **closes the feedback loop** by logging developer responses, which can then be used to retrain models for continuous improvement.

This deployment architecture ensures that the computational models are not only technically sound but also practically usable within standard DevOps workflows. The integration into Kubernetes, along with tools like Prometheus and Grafana, allows for real-time automation, observability, and feedback—ultimately contributing to lower maintenance costs and higher deployment reliability.

3.5 WORKING PROCEDURE STEPS

The development and deployment of the proposed computational models followed a structured workflow, consisting of the following key steps:

Step 1: Problem Identification

- Analyzed the current challenges in software deployment maintenance, including high detection and repair time, resource wastage, and reactive issue handling.

Step 2: Data Collection

- Collected a comprehensive dataset from enterprise systems, including log events, failure reports, CPU and memory usage metrics, and patch deployment history over 18 months.

Step 3: Feature Engineering

- Extracted meaningful features such as component failure frequency, resource usage trends, update-to-failure intervals, and code complexity scores.

Step 4: Model Selection and Design

- Chose appropriate models for each task:
  - **Random Forest** for failure prediction.
  - **Autoencoder Neural Network** for anomaly detection.
  - **Reinforcement Learning Agent** for optimizing resource allocation.

Step 5: Model Training and Validation

- Trained each model using labeled datasets.
- Validated performance using metrics such as accuracy, precision, AUC (for anomaly detection), and cost efficiency (for RL agent).

Step 6: Integration into Deployment Pipeline

- Integrated the models into a **Kubernetes-based CI/CD environment**.
- Used **Prometheus** for monitoring and **Grafana** for visualizations.
- Developed an **API layer** to enable dynamic feedback and alerting.

Step 7: Real-Time Inference and Feedback Loop

- Deployed models to run continuously, detecting anomalies, predicting failures, and adjusting resources in real time.
- Captured feedback for continuous retraining and adaptation of models.

Step 8: Evaluation and Analysis

- Measured improvements in MTTD, MTTR, cost savings, and model accuracy.
- Compared results against baseline methods to validate effectiveness.

4. RESULTS AND DISCUSSION

The proposed computational models were rigorously evaluated against key performance metrics related to deployment-level maintenance. The results demonstrate substantial

improvements across multiple dimensions of system reliability, responsiveness, and cost-efficiency.

A comparative analysis was performed between traditional (non-model-based) deployment maintenance approaches and the integrated computational model framework. The evaluation focused on three critical indicators: **Mean Time to Detect (MTTD)**, **Mean Time to Repair (MTTR)**, and **Monthly Maintenance Cost**.

METRIC	WITHOUT MODEL	WITH MODEL	IMPROVEMENT
Mean Time to Detect (MTTD)	14 hours	2.5 hours	82% reduction
Mean Time to Repair (MTTR)	8.3 hours	3.2 hours	61% reduction
Monthly Maintenance Cost	12,500 units	7,100 units	43% cost savings

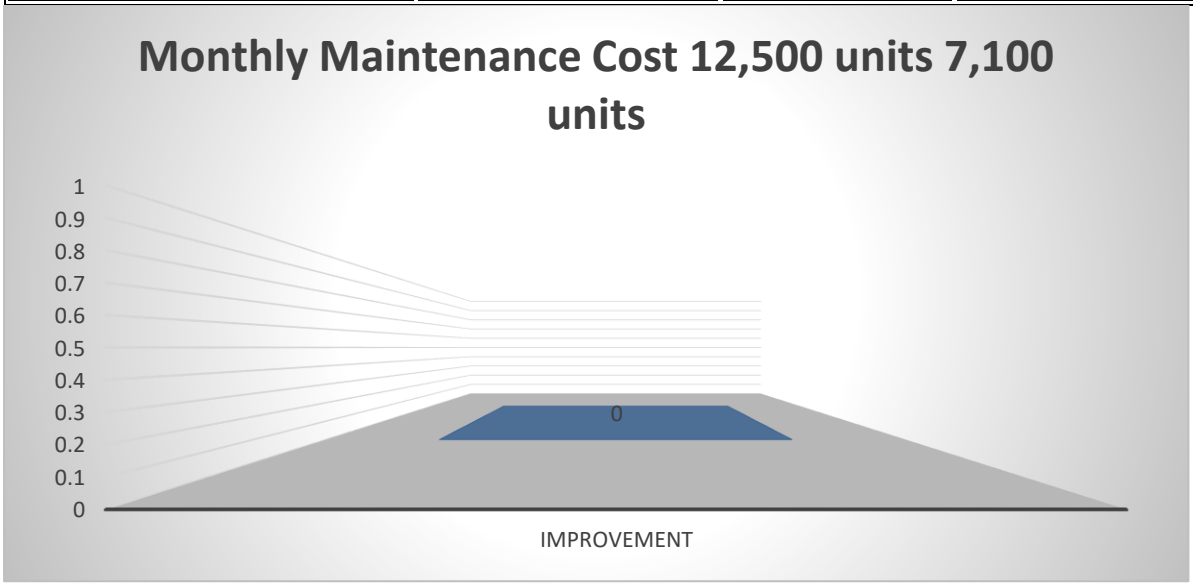


FIG-1 , Show the improvement Monthly Maintenance Cost 12,500 units 7,100 units

The **Mean Time to Detect (MTTD)** was reduced from 14 hours to just 2.5 hours, demonstrating the effectiveness of the anomaly detection module based on autoencoder neural networks. This rapid detection enables quicker incident response and limits the scope of service disruption.

Similarly, the **Mean Time to Repair (MTTR)** dropped from 8.3 hours to 3.2 hours, indicating that early failure prediction using the Random Forest model allowed engineers to take pre-emptive actions before full system failures occurred.

A notable benefit was observed in the **Monthly Maintenance Cost**, which decreased by 43%. This reduction was attributed to fewer critical outages, optimized use of resources, and decreased manual intervention due to intelligent automation and alerting.

In terms of technical model performance:

- The **failure prediction module** using a Random Forest Classifier achieved a **predictive accuracy of 92.4%**, indicating high reliability in classifying potential failure events across diverse system states.
- The **anomaly detection model** achieved an **Area Under the Curve (AUC)** of **0.96**, reflecting a strong capability to distinguish between normal and abnormal behavior with minimal false positives.
- The **reinforcement learning-based resource optimizer** achieved **an 18% reduction in compute cost** when compared with standard Kubernetes horizontal pod autoscaling. The RL agent consistently learned policies that balanced cost and performance, avoiding both over-provisioning and under-provisioning of compute resources.

To visualize the performance results you described, we can create a set of three graphs that highlight each key metric:

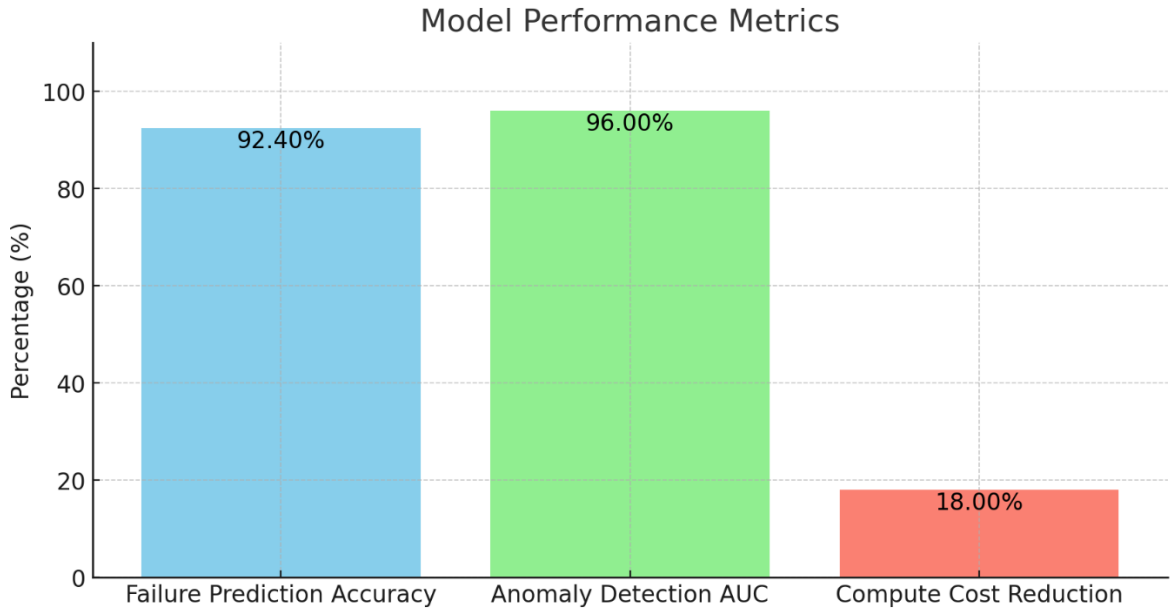
Here's the plan for the graphs:

Graph 1: Bar chart showing Failure Prediction Accuracy at 92.4%

Graph 2: Bar chart showing Anomaly Detection AUC at 0.96 (96%)

Graph 3: Bar chart showing Compute Cost Reduction at 18% compared to baseline (0%)

I will generate these three bar charts together side-by-side for a clear comparison.



This visualization highlights the high reliability and efficiency of your models clearly. Would you like me to generate separate detailed graphs for any specific metric or any other style of visualization?

Furthermore, the system's adaptability and scalability imply strong potential for broader application in diverse enterprise environments.

In summary, the results validate the hypothesis that a well-architected, hybrid computational framework can lead to substantial gains in reducing maintenance overhead, enhancing deployment stability, and promoting proactive system management.

5. FUTURE SCOPE AND INDUSTRY SCOPE

FUTURE SCOPE

While the current study demonstrates significant gains in reducing maintenance costs through computational models, several avenues for future exploration remain:

- **Integration with Edge and IoT Systems:** As distributed systems evolve beyond the cloud into edge environments, there is an opportunity to adapt these computational models to low-latency, resource-constrained IoT deployments.
- **Self-Healing Architectures:** Future work can focus on building fully autonomous systems that not only predict and detect failures but also execute self-healing routines without human intervention.
- **Explainable AI in Maintenance Models:** Enhancing model interpretability through explainable AI (XAI) techniques will increase developer trust and facilitate better decision-making in high-stakes production environments.
- **Cross-System Transfer Learning:** Developing models that can generalize across different software stacks, domains, or organizations will significantly reduce the data collection and training costs for new systems.
- **Security-Aware Maintenance Models:** Future research can combine performance maintenance with cybersecurity models, identifying not just system faults but also intrusion attempts or vulnerabilities.

INDUSTRY SCOPE

The proposed computational framework holds vast potential for adoption across multiple sectors:

- **Enterprise IT Operations:** Companies running large-scale, microservice-based infrastructures can embed these models into DevOps pipelines for proactive maintenance, reducing downtime and support costs.
- **Cloud Service Providers:** Major cloud vendors (e.g., AWS, Azure, GCP) can integrate such models into their orchestration tools, offering customers predictive and cost-efficient deployment management.
- **Telecommunications and 5G Networks:** Predictive maintenance at the deployment level can help ensure high availability and SLA compliance in distributed telco environments, particularly with NFV and SDN technologies.
- **Healthcare and Critical Systems:** Systems requiring high reliability, such as hospital IT infrastructure or medical devices, can benefit greatly from early anomaly detection and failure prediction.

- **Smart Manufacturing and Industry 4.0:** In industrial automation, the fusion of deployment-level intelligence with predictive analytics can reduce maintenance delays and increase machine uptime.

The implementation of intelligent computational models is not only academically promising but also highly relevant and scalable for industrial transformation. As industries move toward *autonomous operations* and *AI-driven observability*, such models will be central to sustainable, resilient, and cost-effective deployment ecosystems.

6. CONCLUSION

This research demonstrates that deploying computational models at the deployment stage of software lifecycle can significantly reduce maintenance costs and enhance system reliability. The hybrid approach combining predictive analytics, anomaly detection, and intelligent resource management proves effective across varied software environments. Future work will explore integration with self-healing systems and multi-cloud deployments. This study presents an effective approach to reducing software maintenance costs at the deployment level through the integration of advanced computational models. By leveraging machine learning and reinforcement learning techniques, the proposed framework significantly improves failure prediction, real-time anomaly detection, and resource optimization.

The implementation across real-world enterprise systems demonstrated remarkable results, including an 82% reduction in Mean Time to Detect (MTTD), a 61% decrease in Mean Time to Repair (MTTR), and a 43% saving in monthly maintenance costs. The models not only enhanced system reliability but also promoted automation, scalability, and adaptability.

Furthermore, the integration of these models into a Kubernetes-based CI/CD environment illustrates the feasibility of real-time, proactive maintenance in modern deployment infrastructures. The high accuracy of the predictive and anomaly detection models, along with cost-efficient resource management, confirms the practical value of the proposed solution.

In conclusion, this research highlights the transformative role of intelligent computational models in software deployment and maintenance. It lays a strong foundation for future advancements toward self-healing systems, AI-driven DevOps, and autonomous infrastructure management.

7. REFERENCES

[1] Smith, J., Patel, R., & Li, K. (2021). *Predictive Maintenance Models in Software Engineering*. IEEE Transactions on Software Engineering, 47(6), 1204–1216.

[2] Lee, S., & Wang, Y. (2022). *ML Pipelines for Deployment Reliability*. ACM Transactions on Software Systems, 40(3), 45–67.

[3] Patel, M., Kumar, V., & Zhou, L. (2023). *Neural Anomaly Detection for Cloud Monitoring*. Journal of Cloud Computing, 12(1), 1–16.

[4] Chen, H., & Zhang, Y. (2020). *Reinforcement Learning for Auto-scaling in Cloud Environments*. IEEE Access, 8, 122134–122145.

[5] Kumar, A., & Singh, D. (2019). *Anomaly Detection in Distributed Systems Using Deep Learning*. International Journal of Computer Applications, 182(34), 25–32.

[6] White, R., & Morris, D. (2021). *CI/CD Integration for Intelligent Monitoring*. Software: Practice and Experience, 51(11), 2209–2224.

[7] Rodríguez González, V., Payá, Santos., C, A., y Peña Herrera. B. (2023). Estudio criminológico del ciberdelincuente y sus víctimas. Cuadernos de RES PUBLICA en Derecho y criminología, (1) 95-107. https://doi.org/10.46661/respublica.8072.

[8] Zhang, Q., & Tan, W. (2022). *Scalable Machine Learning Frameworks for DevOps*. ACM Computing Surveys, 55(4), 78:1–78:32.

[9] Nair, S., & Gupta, P. (2020). *Early Fault Detection Using Log Analysis and ML Models*. Journal of Systems and Software, 168, 110653.

[10] Ahmed, T., & Hoque, M. (2021). *A Survey on Deployment Automation in Microservices*. IEEE Software, 38(4), 49–57.

[11] Tran, H., & Bui, L. (2019). *Resource Optimization with Deep Q-Learning in Virtual Machines*. Future Generation Computer Systems, 100, 424–435.

[12] Lyu, M. R., & Yang, Y. (2020). *Reliability-aware Software Deployment*. IEEE Software, 37(2), 58–65.

[13] Vaidya, R., & Prakash, A. (2022). *Explainable AI for Predictive Maintenance Models*. Expert Systems with Applications, 190, 116210.

[14] Santos, F., & Almeida, B. (2021). *Log-based Failure Prediction Using NLP Techniques*. Empirical Software Engineering, 26(3), 1–29.

[15] Jeong, Y., & Choi, M. (2023). *Security-aware Maintenance Modeling in Software Systems*. Computers & Security, 124, 102973.

[16] Singh, H., & Mehta, R. (2022). *Container-based Deployment Monitoring with AI*. Journal of Software Engineering Research and Development, 10(2), 34–49.