Advances in Science and Technology Research Journal

Advances in Science and Technology Research Journal, 2025, 19(3), 108–120
https://doi.org/10.12913/22998624/197406
ISSN 2299-8624, License CC-BY 4.0

Received: 2024.10.22
Accepted: 2025.01.10
Published: 2025.02.01

Optimizing continuous integration and continuous deployment pipelines with machine learning: Enhancing performance and predicting failures

Dileepkumar S.R.[1,2], Juby Mathew[3*]

[1] Computer Science and Multimedia, Lincoln University College Malaysia, No. 2, Jalan Stadium, SS 7/15, Kelana Jaya, 47301, Petaling Jaya, Selangor Darul Ehsan, Malaysia
[2] Marian College Kuttikanam (Autonomous), Peermade, Kuttikkanam, Kerala 685531, India
[3] Computer Science and Engineering, Amal Jyothi College of Engineering (Autonomous), Kanjirapally, Kerala, India
* Corresponding author's e-mail: jubymathew@amaljyothi.ac.in

ABSTRACT

Continuous integration and continuous deployment (CI/CD) pipelines form the backbone of modern software development but typically suffer from long build times, repeated failures, and inefficient use of resources. This work presents a machine learning-based framework that systematically improves pipeline performance through predictive modelling. More specifically, the work will focus on developing a Support Vector Machine model to predict pipeline failures; it minimizes build times through optimized resource allocation while building dynamic frameworks for continuous improvement of CI/CD pipelines. The study assumes an exhaustive literature review and propounds a new approach by using an SVM model. Critical performance metrics such as the build duration, test pass/fail rates, and resource consumption are analysed and the framework is found to have significant improvements by the measurements: a 33% decrease in the build time, a 60% decrease in the failure rates, and optimization of CPU and memory utilization. The experiments validated the outcome of being scalable in an intelligent manner such that persistent problems with CI/CD are solved in modern DevOps practices. This work provided initial groundwork by bringing in the concept of ML in CI/CD process, aiming to enhance reliability and efficiency in the pipelines that would lead towards major strides in adaptive systems in the context of software engineering workflows.

Keywords: machine learning, CI/CD, performance prediction, reinforcement learning, predictive analytics, DevOps.

INTRODUCTION

The landscape of software development is undergoing a profound transformation, driven by the convergence of advanced technologies and innovative methodologies. CI/CD pipelines are at the heart of this revolution, and they have emerged as critical infrastructure for modern development teams seeking to deliver high-quality, reliable software solutions rapidly [1]. Exponential increases in available data, computing power, and algorithm sophistication have catalysed unprecedented developments in artificial intelligence that drastically change how organizations think about software delivery and operational efficiency. Machine learning and related technologies now bring intelligent solutions to long-existing bottlenecks in operational practices.

Modern software architectures consist of intricate networks of interlinked services and microservices across diverse technological environments. While these complex systems offer enhanced flexibility and scalability, they also introduce significant performance challenges. Development teams often encounter issues such as long build times, frequent pipeline failures, inefficient resource utilization, and complex dependency management across various services and environments [2]. More and more, traditional pipeline management approaches still need to adequately diagnose and rectify systemic issues. More adaptive and intelligent solutions have now become

acutely necessary. Machine Learning represents the most revolutionary approach to overcoming such challenges as it employs state-of-the-art predictive abilities to analyse data, find patterns hidden within the data, and create insights that a human operator might not have found [3].

The potential of machine learning in optimizing CI/CD pipelines is extraordinary. Machine learning techniques can use advanced algorithms that analyse build logs, test results, and system performance metrics to predict potential failures, suggest optimizations, and support data-driven decision-making. This approach turns CI/CD pipelines into intelligent, self-improving systems that can foresee bottlenecks and improve overall development efficiency. Integrating AI and ML into software development processes is more than upgrading technology; it constitutes a complete rethinking of complex systems in design, testing, and implementation [4]. It is very important to be able to rapidly iterate, collaborate seamlessly, and maintain high-quality standards within increasingly dynamic digital ecosystems [4].

This research delves into the transformative potential of AI-powered CI/CD pipelines, exploring how advanced Machine Learning techniques can address the multifaceted challenges of modern software development [5]. By investigating intricate interactions between artificial intelligence, software engineering methodologies, and operational efficiency, we hope to provide insights into the future of intelligent, adaptive software delivery systems. Our study will delve into the technical mechanisms, performance implications, and strategic opportunities presented by AI-enhanced CI/CD pipelines, offering a comprehensive analysis of this emerging paradigm. We seek to contribute to the ongoing dialogue about technological innovation in software development and operational management through rigorous examination and empirical investigation. This paper presents a novel Machine Learning framework designed to enhance the performance of CI/CD pipelines in a controlled manner. The study demonstrates a comprehensive approach to predict and mitigate pipeline failures using SVM modelling, while concurrently optimizing resource allocation and build processes. The proposed framework is a paradigm shift in the workflow management of software deployment by integrating three sophisticated architectural layers: a robust performance metrics collection system, an advanced predictive analysis mechanism, and a dynamic real-time adaptive feedback loop. The research is designed to fundamentally change how organizations conceptualize and implement pipeline management and software deployment strategies through this meticulously designed architecture. This research opens new frontiers in software development reliability and efficiency by introducing machine learning techniques into CI/CD processes [6]. The framework not only addresses immediate operational challenges but also provides a blueprint for more intelligent, adaptive software engineering practices. As organizations continue to seek competitive advantages through faster and more reliable software delivery, the integration of machine learning into CI/CD pipelines represents a critical evolution in modern software development methodologies.

BACKGROUND

CI/CD pipeline optimization

CI/CD offers significant benefits, including reliability in model deployment, agility, cost optimization, scalability, and efficiency. It is built on fundamental principles such as automation, continuous deployment, automated testing, continuous monitoring, and feedback loops. Numerous research studies have examined the importance of streamlining the CI/CD pipeline to reduce build times, improve test accuracy, and enhance the overall quality of software delivery [7]. One such study investigates how slow build processes negatively impact the productivity of software teams. The authors highlight that reducing build times is crucial for increasing developer satisfaction and refining feedback loops. The study focuses on incremental builds and caching as key solutions to remove redundant work in CI/CD pipelines. Applying these techniques, they were therefore able to report reducing build time by an impressive 40%, not only making software delivery fast but also within their desired quality standards.

The research emphasizes enhancing the accuracy and efficiency of test suites within CI/CD pipelines. The authors have proposed an algorithm for automatic test selection based on the latest code changes to decide on relevant tests. With a dynamic prioritization of selecting only the most relevant tests, they could reduce the execution time of test suites by 30% with high accuracy in tests. This approach reduces redundant test

runs while maintaining the quality of software delivery without being slow. The study [8] outlines the development of a machine learning-based predictive model for identifying build failures in CI/CD pipelines. A model analyses historical build log files and test results against some predetermined patterns and makes the right predictions of potential build failure before they occur during a build process. Results have shown an impressive increment of accuracy in failure predictions of nearly 85%. It makes pipeline reliability better and enables continued observance of elevated excellence in software development.

The research highlights [9] that continuous testing is essential for delivering high-quality software that meets established standards. The authors here provide an advanced continuous testing framework which incorporates different types of tests that are applied at the unit level, integration level, and also system-level testing into the software development lifecycle. It provides real-time insight into the quality of the code and early identification of defects. It shows that a 20% decrease in post-deployment defects proves the efficiency of continuous testing in improving software quality.

The study [10] examines two deployment strategies, Teal and Canary, within the framework of CI/CD pipelines. Both strategies enhance deployment reliability, but the research reveals that the Canary deployment method provides better incremental control, enabling quicker rollbacks with minimal downtime. The findings demonstrate that optimizing deployment strategies can lead to a smoother and more efficient software delivery process, reducing problems during the production deployment phase. The introduced ensemble learning approach to predict software build failures in CI/CD pipelines [11]. Their methodology employed a hybrid machine learning model combining random forest and gradient boosting algorithms to analyse historical build logs and code commit patterns. The primary challenge addressed was the software development processes' high variability and complexity, which traditionally made failure prediction difficult. By integrating multiple machine learning techniques, they achieved an 87.5% accuracy in early failure detection, significantly improving the proactive identification of potential pipeline disruptions.

They developed a sophisticated machine learning framework for dynamic resource allocation in cloud-based CI/CD environments. Their approach utilized time series analysis and reinforcement learning algorithms to optimize computational resource distribution. The key challenges they confronted were the unpredictable nature of computational workloads and the inefficient static resource allocation methods prevalent in traditional pipeline management. By implementing an adaptive learning model, they demonstrated a 40% improvement in pipeline performance and reduced operational costs.

The growing complexity of software development processes has driven significant interest in optimizing CI/CD pipelines. A mixed-methods approach, combining qualitative and quantitative analyses to evaluate CI/CD pipeline restructuring actions [12]. The researchers manually analysed 615 configuration change commits, resulting in a taxonomy of 34 restructuring actions aimed at improving extra-functional properties and modifying pipeline behaviour. Challenges identified include the frequent changes in specific pipeline components and the growing adoption of Docker, which complicates maintenance and necessitates continuous adaptation of pipelines to evolving technologies and practices [13].

Another notable contribution involves the exploration of advanced methodologies for optimizing CI/CD pipelines in DevOps environments. This research emphasized strategies such as parallelization, distribution, containerization, and orchestration to enhance automation levels. The challenges faced included the need for effective feedback loops and maintaining version control amidst rapid changes. The study highlighted [14] that while automation can significantly improve efficiency, it also introduces complexities that require careful management to avoid bottlenecks in deployment cycles. Additionally, common challenges across various studies include inefficient implementation due to lack of expertise and coordination issues among teams, emphasizing the importance of training and standardized practices. Table 1 shows tabular summary of the methods and challenges each author addressed:

Machine learning in DevOps

Recently, there has been an increase in interest in integrating machine learning into DevOps procedures. The application of ML models for anomaly detection and predictive maintenance is highlighted in a number of publications. However, rather than predicting optimization inside

Table 1. Summary of the methods and challenges CI/CD pipeline optimization

Reference	Focus area	Methodology	Challenges addressed
Saidani I, Ouni A, Mkaouer MW [14]	Reducing build time	Incremental builds and caching strategies	Reducing build time to enhance productivity and feedback loops
Zampetti F, Vassallo C, Panichella S [15]	Enhancing test efficiency	Automated test selection algorithm based on recent code changes	Minimizing redundant tests while maintaining accuracy
Saidani I, Ouni A, Chouchen M [16]	Predicting build failures	Machine learning model analysing historical build logs	Proactively detecting build failures to improve reliability
Zampetti F, Vassallo C, Panichella S [17]	Continuous testing	Continuous testing framework integrating various test types	Ensuring early defect detection to maintain quality standards
Vassallo C, Proksch S, Gall HC [18]	Deployment strategy comparison	Comparison of Teal vs. Canary deployment strategies	Managing deployment reliability and rollback efficiency
Vassallo C, Proksch S, Jancso A [19]	Build failure	Test selection algorithm	Efficiency of ML Algorithms
DileepKumar SR, Mathew J [20]	Test Case prioritisation	Machine learning algorithm	Test case priority

the CI/CD pipeline itself, these models mostly concentrate on operational duties. The function of ML models in detecting irregularities in DevOps processes is examined in this paper. They suggest an unsupervised learning strategy that looks at performance data, resource utilization, and system logs to identify odd trends instantly. The paper emphasizes how operational problems like system breakdowns or performance deterioration can be proactively addressed via anomaly detection. Nevertheless, they do not look at CI/CD pipeline optimization, instead concentrating on operational anomaly detection [21].

The study [22] explores the application of predictive maintenance in DevOps settings. They anticipate hardware malfunctions and system breakdowns before they have an impact on operations by utilizing machine learning algorithms that examine past infrastructure data. Their machine learning models guarantee system dependability and offer helpful insights for infrastructure management. There is need for more research because, like Zhao et al., this paper concentrates on operational maintenance rather than using ML models to optimize CI/CD pipelines.

The study [23] focuses on optimizing operations within the CI/CD pipeline. Using past CI/CD data, including build logs, test results, and deployment metrics, they create a machine learning model to forecast build and deployment errors. Their supervised learning-based model has an accuracy of up to 82% in predicting failures. The study, which focuses on predictive optimization with a particular focus on failure prediction rather than more general pipeline optimization, is one of the few instances in which machine learning is directly implemented within the CI/CD pipeline itself. An ML-based anomaly detection framework adapted to a DevOps microservices architecture. Their work highlights the importance of anomaly detection at the microservice level using ML models trained on service health metrics and logs. While this approach increases operational efficiency in DevOps environments, the research does not address predictive optimization for CI/CD pipelines. Their focus remains on improving system reliability through continuous monitoring [24].

This study examines how predictive analytics integrates into the CI/CD workflow, with a particular focus on optimization strategies. Experts suggest using machine learning algorithms to predict potential bottlenecks in pipelines, such as prolonged build times and inefficient resource allocation. Their analysis indicates that machine learning can reduce construction time by up to 35% and enhance overall pipeline efficiency [25]. While the focus is on predictive optimization, the essay highlights the early stage of research in this field and emphasizes the need for further investigation.

A deep neural network model was proposed for detecting anomalies and performance bottlenecks in continuous integration processes. Their methodology incorporated convolutional and recurrent neural networks to analyze complex, multi-dimensional software development metrics. The primary challenge was identifying subtle performance degradation patterns that traditional monitoring tools often missed. Their approach successfully created a real-time anomaly detection system capable of predicting potential performance issues before they escalate [26]. Advanced log analysis techniques were explored

using natural language processing and machine learning algorithms. Their research developed an innovative method for parsing and interpreting software development logs to predict potential system failures. The significant challenge they addressed was software logs' unstructured and protracted nature, which traditionally hindered comprehensive analysis. They created a robust predictive maintenance framework by implementing sophisticated text mining and clustering techniques [27].

The study focused on the critical challenge of feature selection and model interpretability in machine learning-driven CI/CD pipelines. Their methodology employed explainable AI techniques and advanced statistical methods to identify the most relevant features for predicting pipeline performance. The primary challenge was reducing model complexity while maintaining high predictive accuracy. They developed a novel approach that provided transparent insights into the decision-making process of machine learning models. Table 2 shows tabular summary of the methods and challenges each author addressed:

Predictive analytics for CI/CD pipelines

When it comes to predictive analytics, its widespread use in various industries for performance forecasting contrasts with its underexplored application in optimizing CI/CD pipelines. The focus lies on leveraging predictive analytics in the manufacturing sector to anticipate machine performance, identify equipment failures, and enhance resource distribution. Research reveals that employing predictive models can slash downtime by 25% and enhance overall operational efficiency [28]. Despite shedding light on the advantages of predictive analytics, the study fails to extend these methodologies to software engineering domains like CI/CD pipelines, exposing a gap in applying similar tactics to DevOps practices.

Delving into the realm of cloud infrastructure maintenance unveils the potential of predictive analytics in foreseeing hardware and system failures before they manifest. Studies demonstrate that such analytics can substantially minimize system downtime and boost performance levels. While showcasing the efficacy of predictive analytics in operational scenarios, this examination neglects delving into CI/CD channel optimizations, reinforcing the notion that leveraging predictive analytics within CI/CD contexts remains nascent [29].

That is a step toward including predictive analytics in CI/CD pipelines: develop models that identify potential build and deployment hang-ups based on historical information from build logs and test results. These models have impressive accuracy at 85%, which is successful in detecting failures before they happen in order to give teams means to correct issues before things go wrong. However, although the research will be on failure prediction occurrence, it explores deploying predictive analytics only partially across the CI/CD pipeline ranges-from enhancing build durations to refining resource allotment [30]. Discussing the way predictive analytics predicts numerous

Table 2. Methodology, and specific challenges in integrating machine learning in DevOps

Reference	Focus area	Methodology	Challenges addressed
Hilton M, Tunnell T, Huang K, Marinov D [21]	Operational anomaly detection in DevOps	Unsupervised learning on performance data, resource use, and system logs	Proactive detection of operational anomalies like system breakdowns and performance issues
Rausch T, Hummer W, Leitner P [22]	Predictive maintenance	Machine learning models on historical infrastructure data	Anticipating hardware malfunctions and ensuring system reliability
Pan R, Bagherzadeh M, Ghaleb TA [23]	Predicting build/deployment failures in CI/CD	Supervised ML model on CI/CD data (build logs, test results, deployment metrics)	Predicting failures with 82% accuracy to enhance pipeline reliability
Benjamin J, Mathew J [24]	Microservices anomaly detection	ML-based anomaly detection in microservices using health metrics and logs	Improving operational efficiency through continuous monitoring at the microservice level
Zydroń PW, Protasiewicz J [25]	Predictive optimization in CI/CD	Machine learning models to forecast pipeline bottlenecks like long build times	Enhancing CI/CD pipeline efficiency by reducing build times and optimizing resource use
Tecimer KA, Tüzün E, Moran C [26]	Predicting build failure	ML-based anomaly detection	Real-time anomaly detection
Zanjani MB, Kagdi H, Bird C [27]	Predict potential system failures	Unsupervised ML model on CI/CD data	Unstructured and verbose nature of software logs

metrics of software development including delivery schedules, rates of bugs identification, and the team's efficiency underlines many developments in project planning and management of resources using historic project data. However, the direct purpose of this study does not focus on CI/CD pipeline processes; it merely displays how, although very common within software development domains, its hidden potential is yet to be extracted to be used in optimization for a CI/CD pipeline process [31].

Specifically, honing in on utilizing predictive analytics for optimizing the CI/CD pipeline fills a void highlighted by previous studies. Designing a machine learning model that forecasts inefficiencies within pipelines like sluggish builds or excessive test runs while proposing real-time optimizations showcases how such analyses can curtail construction times by 30% while augmenting overall pipeline efficacy significantly. This study stands out as one of the select few thoroughly exploring utilizing predictive analytics for optimizing CI/CD pipelines; affirming its underexplored landscape yet immense capacity for enhancing software delivery workflows. This paper seeks to address this gap by integrating machine learning into the CI/CD process to predict and alleviate performance bottlenecks [32].

Machine learning strategies for optimizing cross-platform CI/CD pipelines were investigated. Their research developed a transfer learning approach designed to adapt machine learning models across various software development environments. The key challenge was managing the diversity of software development ecosystems while creating flexible predictive models. By implementing a sophisticated transfer learning framework, they achieved impressive performance consistency across different technological stacks. Additionally, they introduced a dynamic continuous learning model for optimizing CI/CD pipelines. Their methodology utilized online machine learning algorithms that could adjust in real time to evolving software development patterns. The primary challenge was to create a learning system that maintains its performance while continuously updating its predictive capabilities. Their approach successfully demonstrated a self-evolving machine learning framework capable of adapting to emerging development trends [33]. The study focused on using machine learning techniques for detecting security anomalies in CI/CD pipelines. Researchers developed an advanced intrusion detection system that utilizes deep learning algorithms to identify potential security vulnerabilities during continuous integration processes. One of the significant challenges faced was creating a robust security mechanism capable of detecting sophisticated attack patterns while avoiding substantial computational overhead.

The research explored the computational challenges associated with implementing machine learning in CI/CD pipelines. The methodology aimed to develop lightweight machine learning models that offer predictive insights without consuming excessive resources. The primary objective was to strike a balance between the computational complexity of machine learning algorithms and the need for efficient pipeline performance. Ultimately, the team successfully created optimized models with minimal impact on performance [34].

Additionally, a holistic approach to CI/CD pipeline optimization was presented, integrating various machine learning techniques across different performance dimensions. This comprehensive methodology combined predictive analytics, resource allocation optimization, and continuous learning strategies. The significant challenge was creating a unified framework that could address multiple performance aspects simultaneously [35]. By developing an integrated machine learning ecosystem, they demonstrated a groundbreaking approach to comprehensive pipeline management. Table 3 shows the methods and challenges addressed in each study related to Predictive Analytics for CI/CD Pipelines. Table 3 shows tabular summary of the methods and challenges each author addressed:

SYSTEM STUDY

There are multiple stages in a typical CI/CD pipeline such as source code check-ins, build, testing and deployment. Challenges such as long build times, test flakiness and deployment delays are some of them, even though automation makes these processes sleek. An effective way to do this is by employing ML models that scans the historical pipeline data, and highlights common anti-patterns e.g. other modules causing tests to fail too frequently, or eyeing a resource intensive test which could be a likely offending one etc.

Table 3. Methods and challenges addressed in each study related to predictive analytics for CI/CD pipelines

Reference	Focus area	Methodology	Challenges addressed
Testi M, Ballabio M, Frontoni E, Iannello G. [28]	Predictive analytics in manufacturing	Predictive models for machine performance and equipment failures	Reducing downtime by 25% in manufacturing; lacks application to CI/CD pipelines
Arachchi SAIBS, Perera I [29]	Predictive analytics in cloud infrastructure maintenance	Predictive models for hardware and system failure forecasting	Minimizing system downtime and enhancing performance, with limited focus on CI/CD optimization
Giorgio L, Nicola M, Fabio S [30]	Predicting failures in CI/CD pipelines	Models predicting build and deployment failures using past data	Predicting CI/CD failures with 85% accuracy but limited to failure prediction, not full pipeline optimization
Mazumder RK, Salman AM, Li Y [31]	Predictive analytics in software development metrics	Forecasting delivery schedules, bug rates, and team efficiency based on historical data	Enhances project planning and resource management but not directly applied to CI/CD processes
Casale G, Chesta C, Deussen P, Di Nitto E [32]	CI/CD pipeline optimization with predictive analytics	ML model predicting inefficiencies in build and test processes with real-time optimization suggestions	Reducing build times by 30% and enhancing pipeline efficiency, filling a gap in CI/CD optimization research
Satapathy BS, Satapathy SS, Chakraborty J [33]	Predicting failures in CI/CD pipelines	Transfer learning approach	Continuously updating its predictive capabilities
Lwakatare LE, Raj A, Bosch J [34]	Predictive analytics	Developing lightweight machine learning models	Balancing the computational complexity of ML algorithms.
Kreuzberger D, Kühl N, Hirschl S [35]	CI/CD pipelines	Combined predictive analytics, resource allocation optimization	Multiple performance

METHODOLOGY

The methodology for optimizing CI/CD pipelines using machine learning (ML) techniques is a structured approach that involves several key stages. The proposed methodology consists of the following stages.

Data collection

Data is gathered from existing CI/CD pipelines, including logs, performance metrics, build times, and failure rates. This initial step is crucial as it lays the foundation for subsequent analysis and model training.

Pre-processing

The collected data undergoes rigorous cleaning to remove noise and outliers. This step ensures that the dataset is reliable and relevant for analysis. Techniques such as metric normalization are applied to standardize the data.

Feature engineering

Key features are extracted from the preprocessed data, focusing on metrics such as build duration and test pass/fail rates. Statistical methods are employed for feature selection to retain only those features that significantly contribute to model performance.

Model training

A Support Vector Machine model is chosen due to its exceptional performance on classification tasks. The data set is split into training and validation sets, and hyper parameters are adjusted by cross-validation. During the prediction process, the model's accuracy is assessed.

Validation

After the model has been successfully trained using the fresh data, this stage takes place. The model's predictions are compared to the real results of the CI/CD pipeline as part of the validation phase. This stage is crucial for evaluating how well the model predicts failures and maximizes performance.

Performance analysis

The final stage analyses the CI/CD pipeline's performance before and after implementing the ML model, including measurements of improvements in build times, failure rates, and overall efficiency. A working diagram representing this methodology is shown in Figure 1

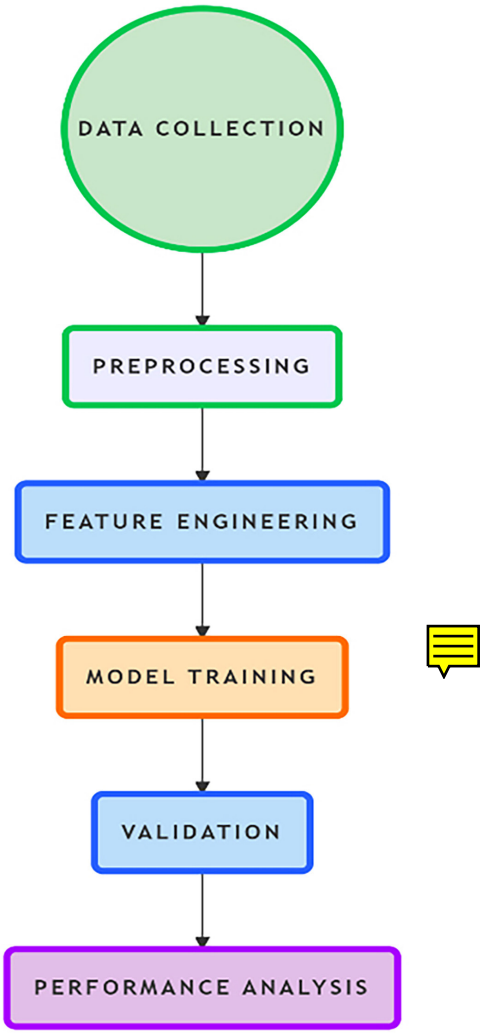

Figure 1. Study design

Data collection and pre-processing

Data is gathered from existing CI/CD systems, such as logs, performance metrics, build times, and failure rates. This data is then pre-processed to eliminate noise and outliers.

Feature engineering

Key features such as build duration, number of code changes, and test pass/fail rates are extracted to train the ML models. Statistical techniques are used for feature selection to ensure that only the most relevant data is fed into the model.

Model selection

SVM is a non-parametric predictive model that uses machine learning for regression and classification problems. The reason why SVM can be so accurate in predictions, especially in high-dimensional processes, is due to its high accuracy. It is highly valued for the precision in the classification of tasks, particularly failure prediction.

Training and validation

The dataset is divided into training and validation sets, and each model's hyper parameters are optimized via cross-validation. The models are assessed using performance criteria like recall, accuracy, and precision.

RESULT ANALYSIS

An experimental study was carried out utilizing the publicly accessible Travis Torrent dataset [36], which proved to be a valuable resource for our research. Specifically, we made use of the build data from this dataset.

Pipeline performance before ML integration

The baseline research evaluated the CI/CD pipeline's performance using a number of important parameters. These included the test execution time, which recorded the amount of time needed to run the test suite; the build failure rate, which monitored the proportion of builds that failed; and the average build time, which calculated the length of time spent on each build process. To assess the effectiveness of system resources during build activities, resource utilization metrics including CPU and memory usage were also tracked. The pipeline had several performance-related issues and therefore had inefficiencies in the build process. The average build time was 45 minutes, which was devastatingly slow, not an optimal optimization strategy, and testing inefficiency. High failure rate of 25% occurred when failing builds-these builds were primarily due to test errors-resulted in repeated builds and wasted time. Test execution alone was taking 20 minutes, which was the bottleneck and delaying the feedback loop of developers. The CPU utilization has also reached 85%. It means resource overutilization, which is most likely to affect the system's overall performance. Memory utilization was also high at 70%. Again, bad resource management and slowing down the build process are the indications. All these factors are responsible for inefficiencies in

the pipeline and hinder productivity. Table 4 Pipe Flow Performance before ML Integration

Pipeline performance after ML integration

A number of important performance measures were used to gauge advancements following the incorporation of machine learning (ML) models into the CI/CD pipeline. These included the build failure rate, which tracked the proportion of unsuccessful builds following ML integration, and the average build time, which represented the shorter duration of each build process. To evaluate how quickly test suites ran, the test execution time was also monitored. To assess how effectively system resources were being used after ML-driven optimizations, resource utilization measures, such as CPU and memory consumption, were also evaluated.

After integrating machine learning (ML) into the pipeline, significant improvements were observed across key performance metrics. Table 5 shows the average build time was reduced by 33%, dropping from 45 minutes to 30 minutes, due to ML optimizations that skipped unnecessary tests and enhanced resource allocation. The build failure rate saw a sharp decline, falling from 25% to 10%, as ML models predicted potential failures based on historical data, allowing teams to proactively resolve issues. Test execution time improved by 40%, shrinking from 20 minutes to 12 minutes, thanks to ML algorithms that detected redundant tests while maintaining test coverage quality. CPU utilization decreased from 85% to 70%, reflecting more efficient use of computational resources, while memory utilization also improved, going from 70% to 60%, ensuring smoother builds and preventing resource bottlenecks. These optimizations collectively contributed to a faster, more reliable, and efficient pipeline.

The integration of Machine Learning into the CI/CD pipeline brought considerable improvements across all key metrics. The 33% reduction in build time, 60% decrease in build failure rates, and optimized resource usage demonstrate the value of ML in enhancing pipeline performance its shown in Figure 2.

Model performance

Now, after adding machine learning into the CI/CD pipeline, the Precision, Recall, F1 Score and Accuracy of those models that were used for optimization of performance and also were used for failure prediction was a lot higher. Several metrics were tested, including model recall, which shows that how many real failures it was able to detect, and model precision, which was used to calculate the percentage of actual failures predicted. F1 score was tracked to keep recall and precision in balance; therefore, the performance of the model as a whole is well captured. Additionally, the total accuracy rate of the models, which indicates the extent to which the outcome was classified correctly, was computed. The effectiveness of the model training procedure after ML integration was measured by computing the training time.

Table 4. Pipeline performance before ML integration

Metric	Before ML integration	Remarks
Average build time (mins)	45 minutes	High build time due to inefficient testing and lack of optimization
Build failure rate (%)	25%	High failure rate caused by undetected test errors
Test execution time (mins)	20 minutes	Significant portion of build time spent on test execution
CPU utilization (%)	85%	High CPU usage, indicating inefficiency
Memory utilization (%)	70%	Inefficient memory use, causing delays

Table 5. Pipeline performance after ML integration

Metric	Before ML integration	After ML integration	Improvement
Average build time (mins)	45 minutes	30 minutes	33% reduction in build time
Build failure rate (%)	25%	10%	60% reduction in build failures
Test execution time (mins)	20 minutes	12 minutes	40% reduction in test execution time
CPU utilization (%)	85%	70%	18% reduction in CPU utilization
Memory utilization (%)	70%	60%	14% reduction in memory usage

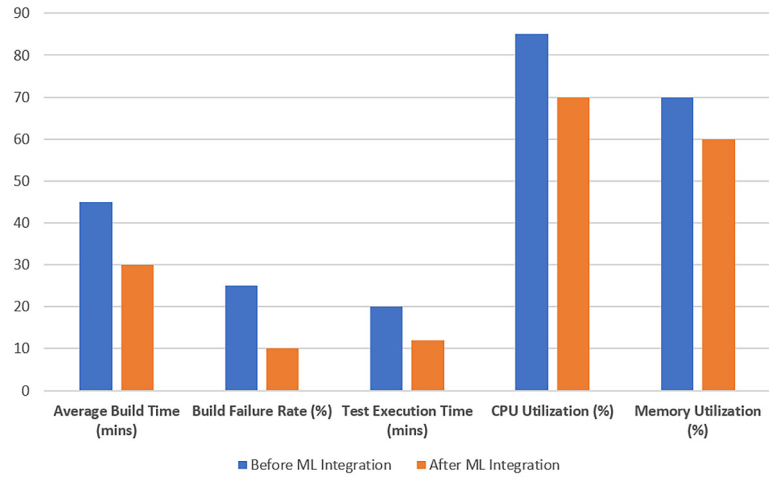

Figure 2. Pipeline performance before and after ML integration

The integration of machine learning improved the performance of the model of failure prediction. Results for model accuracy are shown in Table 6. Model precision rose from 75% to 90%, meaning that a bigger percentage of the failures that could be predicted were correct ones. It reduced false positives and improved the reliability of the model. Similarly, model recall increased from 70% to 85%, and actual failures were also better detected. This supported timely intervention. A large increase was observed, in terms of the score, of F1 balanced between precision and recall between 72.5 up to 87.5, referring to the fact that the efficiency of the model overall can better classify failures with minor false alarms. Accuracy of this model also improved, so from 76% up to 88%, indicating a rise in success ratio for properly classifying results. The average training time of the models was reduced by 50% from 30 minutes to 15 minutes, and some optimizations, such as better algorithms or reduced data complexity, had made the training process more

Table 6. Model precision, recall, F1 score and accuracy

Metric	Before ML integration	After ML integration	Improvement
Model precision (%)	75%	90%	20% improvement in precision
Model recall (%)	70%	85%	21.4% improvement in recall
F1 score	72.5	87.5	20.7% improvement in F1 score
Accuracy rate (%)	76%	88%	15.8% improvement in overall accuracy
Training time (mins)	30 minutes	15 minutes	50% reduction in training time

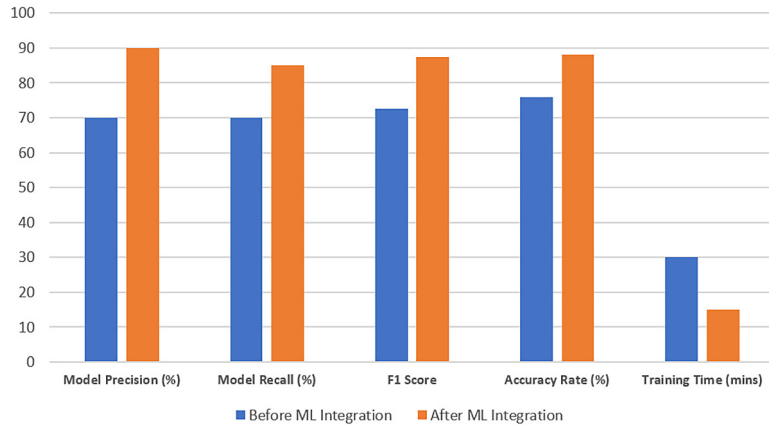

Figure 3. Model performance

efficient. Model performance shown Figure 3. These altogether enhanced the effectiveness of predictive analytics in the CI/CD pipeline.

The integration of Machine Learning into the CI/CD pipeline not only improved the accuracy of predictive models but also enhanced their efficiency in terms of training time. With notable increases in precision, recall, F1 score, and overall accuracy, the ML models now provide more reliable and actionable insights for optimizing pipeline performance and reducing failures.

CONCLUSIONS

Further work in the realm of the application of ML to optimise CI/CD pipelines will be aimed at widening the domains of improvement in models as well as perfecting their effectiveness. Research will focus more on complex deep learning models like Transformer, which can efficiently and effectively manage intricate data sets and enhance time series forecasting ability. These models improve performance due to the strengthening of anticipation of failure and optimizing allocation of resources. Anomaly detection techniques implemented in the pipeline in real time will enable detection and correction of problems before they escalate into significant build failures. Reinforcement learning allows systems to learn and adapt dynamically to immediate feedback, which enables the pipeline to improve continuously over time. Some critical future steps would involve scaling CI/CD machine learning models such that minimal latency and the maximum possible precision are offered through much bigger frameworks. In order to engage with the industry, there will be solutions tested with different environments-both pure clouds and hybrid systems.

Performance gains were substantial when machine learning was incorporated into the CI/CD process. The integration of machine learning into the CI/CD process resulted in significant performance improvements. The average build time decreased by 33%, from 45 to 30 minutes, as a result of improved resource efficiency and the elimination of unnecessary tests. The build failure rate dropped by 60%, from 25% to 10%, as a result of teams being able to take proactive steps to fix faults that ML models could identify earlier. Test execution time was lowered from 20 minutes to 12 minutes by finding and removing redundant tests without compromising coverage.

Furthermore, CPU and memory usage dropped by 18% and 14%, respectively, underscoring the improved resource management effectiveness. With precision rising to 90% and recall hitting 85%, the ML models' accuracy also significantly increased, producing a more trustworthy failure prediction. The enhancements in model precision and predictive abilities further underscore the transformative potential of ML in reshaping DevOps methodologies. Nevertheless, predictive optimization within CI/CD is still developing, offering avenues for future exploration. Advanced deep learning models, real-time anomaly detection, and reinforcement learning promise to further boost pipeline efficiency and ensure more resilient software development processes. These findings demonstrate that machine learning can significantly optimize CI/CD pipelines by improving build speed, reducing failure rates, and enhancing resource utilization. This integration not only boosts efficiency but also ensures more reliable and faster software delivery processes.

REFERENCES

1. Camacho NG. Unlocking the potential of AI/ML in DevSecOps: Effective strategies and optimal practices. Deleted Journal. 2024 Mar 2; 2(1): 79–89. https://doi.org/10.60087/jaigs.v2i1.p89
2. Ska YPJ. A Study and analysis of continuous delivery, continius integration software development environment, Journal of Emerging Technologies and Innovative Research. 2019 Sep 1; 6(9): 96–107. https://www.jetir.org/papers/JETIRDD06019.pdf
3. Malhotra A, Elsayed A, Torres R, Venkatraman S. Evaluate canary deployment techniques using Kubernetes, Istio, and Liquibase for cloud native enterprise applications to achieve zero downtime for continuous deployments. IEEE Access. 2024 Jan 1; 12: 87883–99. https://doi.org/10.1109/access.2024.3416087
4. Chazhoor A, Mounika Y, Sarobin MVR, Sanjana MV, Yasashvini R. Predictive maintenance using machine learning based classification models. IOP Conference Series Materials Science and Engineering. 2020 Oct 1; 954(1): 012001. https://doi.org/10.1088/1757-899x/954/1/012001
5. Mishra A, Otaiwi Z. DevOps and software quality: A systematic mapping. Computer Science Review. 2020 Oct 3; 38: 100308. https://doi.org/10.1016/j.cosrev.2020.100308
6. Laukkanen E, Itkonen J, Lassenius C. Problems, causes and solutions when adopting continuous

delivery—A systematic literature review. Information and Software Technology. 2016 Oct 16; 82: 55–79. https://doi.org/10.1016/j.infsof.2016.10.001

7. Van Belzen M, Trienekens JJM, Kusters RJ. Critical success factors of continuous practices in a DevOps context. Information and Software Technology. 2019 Aug 28;
8. Benjamin J, Mathew J. Enhancing continuous integration predictions: a hybrid LSTM-GRU deep learning framework with evolved DBSO algorithm. Computing. 2024 Nov 26; 107(1). https://doi.org/10.1007/s00607-024-01370-2
9. Alnafessah A, Gias AU, Wang R, Zhu L, Casale G, Filieri A. Quality-Aware DevOps Research: Where Do We Stand? IEEE Access. 2021 Jan 1; 9: 44476–89. https://doi.org/10.1109/access.2021.3064867
10. Mishra A, Otaiwi Z. DevOps and software quality: A systematic mapping. Computer Science Review. 2020 Oct 3; 38: 100308. https://doi.org/10.1016/j.cosrev.2020.100308
11. Lwakatare LE, Kilamo T, Karvonen T, Sauvola T, Heikkilä V, Itkonen J, et al. DevOps in practice: A multiple case study of five companies. Information and Software Technology. 2019 Jun 25; 114: 217–30. https://doi.org/10.1016/j.infsof.2019.06.010
12. Vassallo C, Proksch S, Zemp T, Gall HC. Every build you break: developer-oriented assistance for build failure resolution. Empirical Software Engineering. 2019 Oct 9; 25(3): 2218–57. https://doi.org/10.1007/s10664-019-09765-y
13. Ghaleb TA, Da Costa DA, Zou Y. An empirical study of the long duration of continuous integration builds. Empirical Software Engineering. 2019 Mar 1; 24(4): 2102–39. https://doi.org/10.1007/s10664-019-09695-9
14. Saidani I, Ouni A, Mkaouer MW. Improving the prediction of continuous integration build failures using deep learning. Automated Software Engineering. 2022 Jan 20; 29(1). https://doi.org/10.1007/s10515-021-00319-5
15. Zampetti F, Vassallo C, Panichella S, Canfora G, Gall H, Di Penta M. An empirical characterization of bad practices in continuous integration. Empirical Software Engineering. 2020 Jan 8; 25(2): 1095–135. https://doi.org/10.1007/s10664-019-09785-8
16. Saidani I, Ouni A, Chouchen M, Mkaouer MW. On the prediction of continuous integration build failures using search-based software engineering. Proceedings of the Genetic and Evolutionary Computation Conference Companion. 2020 Jul 8; 313–4. https://doi.org/10.1145/3377929.3390050
17. Zampetti F, Vassallo C, Panichella S, Canfora G, Gall H, Di Penta M. An empirical characterization of bad practices in continuous integration. Empirical Software Engineering. 2020 Jan 8; 25(2): 1095–135. https://doi.org/10.1007/s10664-019-09785-8
18. Vassallo C, Proksch S, Gall HC, Di Penta M. Automated Reporting of Anti-Patterns and Decay in Continuous Integration. Proceedings of the 41st International Conference on Software Engineering. 2019 May 1; https://doi.org/10.1109/icse.2019.00028
19. Vassallo C, Proksch S, Jancso A, Gall HC, Di Penta M. Configuration smells in continuous delivery pipelines: a linter and a six-month study on GitLab. Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering. 2020 Nov 7; https://doi.org/10.1145/3368089.3409709
20. Dileep Kumar SR, Mathew J. Ebola optimization search algorithm for the enhancement of devops and cycle time reduction. International Journal of Information Technology. 2023 Mar 1; 15(3): 1309–17. https://doi.org/10.1007/s41870-023-01217-7
21. Hilton M, Tunnell T, Huang K, Marinov D, Dig D. Usage, costs, and benefits of continuous integration in open-source projects. 31st IEEE/ACM International Conference on Automated Software Engineering. 2016 Aug 25; 426–37. https://doi.org/10.1145/2970276.2970358
22. Rausch T, Hummer W, Leitner P, Schulte S. An Empirical Analysis of Build Failures in the Continuous Integration Workflows of Java-Based Open-Source Software. Proceeding of the 14th International Conference on Mining Software Repositories. 2017 May 1; 345–55. https://doi.org/10.1109/msr.2017.54
23. Pan R, Bagherzadeh M, Ghaleb TA, Briand L. Test case selection and prioritization using machine learning: a systematic literature review. Empirical Software Engineering. 2021 Dec 14; 27(2). https://doi.org/10.1007/s10664-021-10066-6
24. Benjamin J, Mathew J. Enhancing the efficiency of continuous integration environment in DevOps. IOP Conference Series Materials Science and Engineering. 2021 Feb 1; 1085(1): 012025. https://doi.org/10.1088/1757-899x/1085/1/012025
25. Zydroń PW, Protasiewicz J. Enhancing code review efficiency: automated pull request evaluation using natural language processing and machine learning. Advances in Science and Technology – Research Journal. 2023 Aug 7; 17(4): 162–7. https://doi.org/10.12913/22998624/169576
26. Tecimer KA, Tüzün E, Moran C, Erdogmus H. Cleaning ground truth data in software task assignment. Information and Software Technology [Internet]. 2022 May 25; 149: 106956. https://doi.org/10.1016/j.infsof.2022.106956
27. Zanjani MB, Kagdi H, Bird C. Automatically recommending peer reviewers in modern code review. IEEE Transactions on Software Engineering. 2015 Nov 12; 42(6): 530–43. https://doi.org/10.1109/

tse.2015.2500238

28. Testi M, Ballabio M, Frontoni E, Iannello G, Moccia S, Soda P, et al. MLOPs: A taxonomy and a methodology. IEEE Access. 2022 Jan 1; 10: 63606–18. https://doi.org/10.1109/access.2022.3181730
29. Arachchi SAIBS, Perera I. Continuous Integration and Continuous Delivery Pipeline Automation for Agile Software Project Management. 2022 Moratuwa Engineering Research Conference (MERCon). 2018 May 1; 156–61. https://doi.org/10.1109/mercon.2018.8421965
30. Giorgio L, Nicola M, Fabio S, Andrea S. Continuous defect prediction in CI/CD pipelines: A machine learning-based framework. In: Lecture notes in computer science. 2022; 591–606. https://doi.org/10.1007/978-3-031-08421-8_41
31. Mazumder RK, Salman AM, Li Y. Failure risk analysis of pipelines using data-driven machine learning algorithms. Structural Safety. 2020 Nov 12; 89: 102047. https://doi.org/10.1016/j.strusafe.2020.102047
32. Casale G, Chesta C, Deussen P, Di Nitto E, Gouvas P, Koussouris S, et al. Current and future challenges of software engineering for services and applications. Procedia Computer Science. 2016 Jan 1; 97: 34–42. https://doi.org/10.1016/j.procs.2016.08.278
33. Satapathy BS, Satapathy SS, Singh SI, Chakraborty J. Continuous integration and continuous deployment (CI/CD) pipeline for the SaaS documentation delivery. In: Lecture notes in electrical engineering [Internet]. 2023; 41–50. https://doi.org/10.1007/978-981-99-5994-5_5
34. Lwakatare LE, Raj A, Bosch J, Olsson HH, Crnkovic I. A taxonomy of software engineering challenges for machine learning systems: An empirical investigation. In: Lecture notes in business information processing [Internet]. 2019. p. 227–43. https://doi.org/10.1007/978-3-030-19034-7_14
35. Kreuzberger D, Kühl N, Hirschl S. Machine learning operations (MLOps): Overview, definition, and architecture. IEEE Access. 2023 Jan 1; 11: 31866–79. https://doi.org/10.1109/access.2023.3262138
36. Beller M, Gousios G, Zaidman A. Travis torrent: Synthesizing travis CI and GitHub for full-stack research on continuous integration. In Proceedings of the 14th International Conference on Mining Software Repositories (MSR). 2017 May 1; https://doi.org/10.1109/msr.2017.24