ISSN: 2633-4828 Vol. 6 No.1, January, 2024

International Journal of Applied Engineering & Technology

# PREDICTIVE MODELING AND MACHINE LEARNING TECHNIQUES FOR BOTTLENECK IDENTIFICATION AND OPTIMIZATION IN VERSION CONTROL AND CI/CD

**Amarjeet Singh and Alok Aggarwal**
School of Computer Science, University of Petroleum & Energy Studies, Dehradun, Uttarakhand

## ABSTRACT

*In modern software development, Continuous-Integration and Continuous-Deployment pipelines are crucial for efficient and reliable software delivery. However, identifying bottlenecks or performance issues within the pipeline can be challenging. Traditional analysis methods often rely on manual inspection or ad hoc approaches, making it difficult to pinpoint specific areas that require optimization. This work aims to address this problem by proposing a predictive model that identifies bottlenecks in the Continuous-Integration and Continuous-Deployment pipeline and provides insights for improvement. It has been observed that during the SDLC lifecycle, software developers never could predict the optimal time to deploy the microservices which results in financial losses. The proposed work effectively predicts the optimal timing for executing pull requests, push requests, and deployments within the context of software development projects for microservices in DevOps culture. By analyzing and assimilating historical data encompassing various aspects such as code modifications, team activity levels, and project milestones, the aim is to equip developers with actionable insights that can significantly enhance project planning and coordination. The workload in Continuous-Integration and Continuous-Deployment pipelines can fluctuate significantly, with varying amounts of code changes, parallel test executions, and deployment scenarios. This dynamic nature adds further complexity to bottleneck identification and necessitates the need for automated, data-driven analysis techniques. Out of the three models; Random forest, Logistic regression, and Light GBM; it was observed that Light GBM performs best in terms of accuracy, AUC, and F1 score with 81%, 77%, and 82% respectively. In terms of the build stage bottleneck, it is observed that there are on an average 10 bottlenecks with the highest number of bottlenecks in the build 4 stage of the pipeline which recorded 15 bottlenecks. In terms of test stage bottleneck, there are on an average 7 bottlenecks with the highest number of bottlenecks in the unit test stage of the pipeline which recorded 15 bottlenecks. In terms of deployment stage bottlenecks, there are on an average 4 bottlenecks with the highest number of bottlenecks in the deployment orchestration which recorded 15 bottlenecks.*

*Keywords: Random Forest Regressor, Gradient Boosting Regressor, LSTM, RMSE, DevOps, Git, Subversion, micro-service*

## 1. INTRODUCTION

The complexity of modern Continuous-Integration and Continuous-Deployment (CI/CD) pipelines, with their interconnected stages and dependencies involving various tools, frameworks, and infrastructure components, makes it impractical to perform manual analysis [1]-[3]. The dynamic nature of workloads further complicates the identification of bottlenecks, as the fluctuating levels of code changes, parallel test executions, and deployment scenarios require automated and data-driven analysis techniques [4]. Without a reliable and efficient method to identify bottlenecks, software development teams face several challenges [5]-[9]. They may experience delays in software delivery due to inefficient resource allocation or lengthy build and test cycles. Performance issues may go unnoticed until they impact the overall system, resulting in poor user experience or even system failures. Additionally, the lack of actionable insights on bottleneck locations hampers the team's ability to prioritize and implement targeted optimizations, leading to suboptimal pipeline performance [10]. Therefore, there is a critical need to develop a predictive model that can accurately identify bottlenecks in CI/CD pipelines and provide actionable insights for improvement. Such a model would enable development teams to proactively identify and address performance issues, optimize resource allocation, streamline the software delivery process, reduce cycle times, and enhance overall software delivery efficiency and reliability. By bridging the gap between manual inspection and a data-driven approach, the predictive model would empower software development teams to gain

a deeper understanding of their CI/CD pipelines' performance, identify potential bottlenecks with precision, and make informed decisions regarding optimization strategies [11]-[15]. This advancement would significantly contribute to the continuous improvement of software delivery processes and enable organizations to deliver high-quality software with increased efficiency and reliability. Figure 1 shows the architectural flow of build and deploy of microservices from development to business production servers.

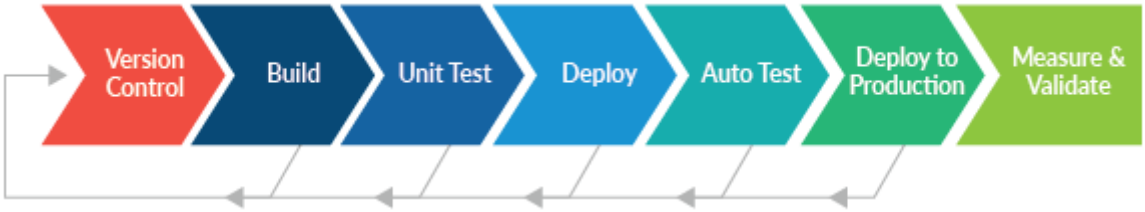


**Fig. 1:** Architectural Flow of Build and Deploy of Microservices from Development to Business Production Servers

This work proposes a detailed and comprehensive approach that integrates machine learning techniques with historic software development data to predict the optimal timing of pull requests, push requests, and deployments. By leveraging the power of machine learning, it aims to provide developers with actionable insights to enhance project planning and coordination, ultimately improving the efficiency and productivity of software development teams. To deal with the aforementioned problem, this work proposes the usage of gadget-getting-to-know techniques to gain insights into bottlenecks in the modern CI/CD pipeline. The answer involves amassing and analyzing historic records from the CI/CD gadget, which includes construct logs, test outcomes, code metrics, deployment logs, and overall performance monitoring facts. By training models on these records bottlenecks in the pipeline can be expected and diagnosed.

Rest of the paper is organized as follows. Section 2 gives the methodology and the experimental description including the various steps used for vulnerabilities detection in the cloud. Results and discussion are given in section 3. Section 4 concludes the work.

## 2. METHODOLOGY AND EXPERIMENT DESCRIPTION

Containerized applications generally make extensive use of cluster networks [16]-[18]. To understand how an application interacts and identifies anomalous communications, active network traffic is to be observed [19]-[22]. At the same time, if active traffic is compared to allowed traffic, network policies can be identified that are not actively used by cluster workloads. This information can be used to further strengthen the allowed network policy, removing unneeded connections to reduce the attack surface [23]. Kubernetes nodes must be on a separate network and should not be exposed directly to public networks. If possible, direct connections should be avoided to the general corporate network [24]-[25]. This is only possible if Kubernetes control and data traffic are isolated. Otherwise, both flow through the same pipe, and open access to the data plane implies open access to the control plane. Ideally, nodes should be configured with an ingress controller, set to only allow connections from the master node on the specified port through the network access control list (ACL).

The proposed methodology consists of the following steps:

***1. Dataset Description:*** The primary data source is the archive from Git Hub that is available for querying via the Amazon RDS database as the data is stored in table format. In RDS, the data stored is then extracted into tabular format and used for further processing and analysis. Some of the data regarding collaborators and teams was extracted using the GitHub API. The data extracted consisted of 1.2 million observations and 10 features. The data consisted of the pipeline information from January 1, 2023 to May 25, 2023.

***2. Data Collection:*** Gather historical data from the CI/CD system, including build logs, test results, code metrics, deployment logs, and performance monitoring data. This data should cover a significant period and include information about build times, test coverage, code quality, and deployment success rates. Figure 2 shows the data collection for microservices applied for machine learning approaches.

```
USER  PID %CPU %MEM    VSZ   RSS TTY   STAT START   TIME COMMAND
root    1  0.0  0.0   4336   796 ?     Ss   18:17   0:00 /bin/sh -c node server.js
root    5  0.1  0.5 772124 22700 ?     Sl   18:17   0:00 node server.js
```

**Fig. 2:** Data Collection for microservices applied for machine learning approaches

***3. Data Processing:*** After the dataset is generated using Amazon RDS, the data is exported in parquet format and finally it is processed in Python using Pyspark.

***4. Feature Engineering:*** Extract relevant features from the collected data, such as build duration, test coverage percentage, code complexity metrics, and deployment failure rates. Transform the raw data into a suitable format for machine learning algorithms.

***5. Model Training:*** Apply machine learning algorithms, such as classification or regression models, to train on historical data. Split the data into training and validation sets, ensuring that the model captures the patterns and relationships within the data.

***6. Model Evaluation:*** Assess the performance of the trained model using appropriate evaluation metrics, such as accuracy, precision, recall, or mean absolute error. Fine-tune the model parameters to optimize its predictive accuracy.

***7. Bottleneck Identification:*** Utilize the trained model to predict and identify bottlenecks within the CI/CD pipeline. Analyze the model's outputs and generate insights on specific areas that require optimization or improvement.

A holistic flow of the methodology used in the work is shown in figure 3.

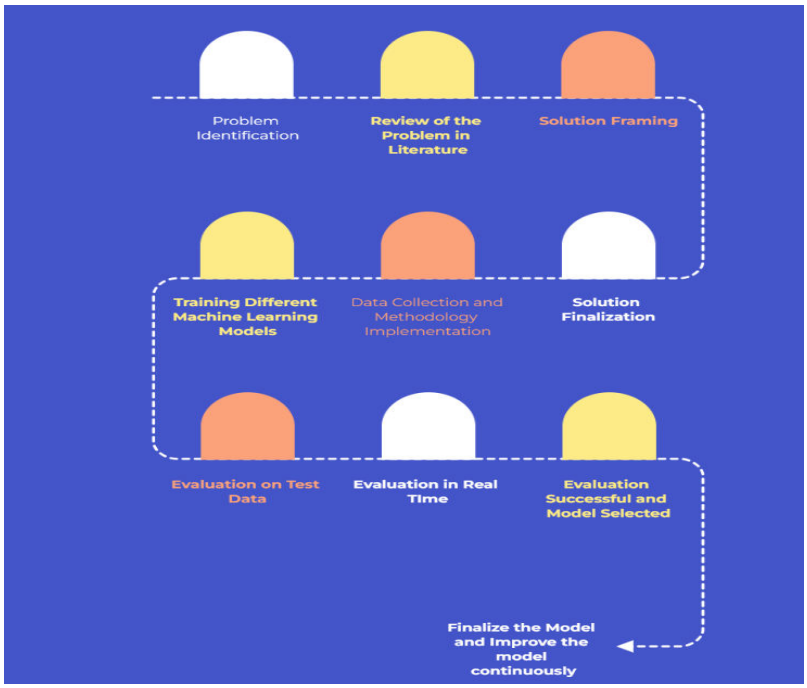


**Fig. 3:** A Holistic Flow of the Methodology Used in the Work

## 3. RESULTS AND DISCUSSION

Table 1 shows the performance metrics (Accuracy, AUC, F1) of different models; Random Forest, Logistic Regression, and LightGBM. The results demonstrate that LightGBM works better than Random Forest and Logistic Regression.

**Table 1:** Performance Metrics of Different Models

| Model | Accuracy | AUC | F1 |
|---|---|---|---|
| Random Forest | 0.72 | 0.71 | 0.75 |
| Logistic Regression | 0.64 | 0.58 | 0.62 |
| LightGBM | 0.81 | 0.77 | 0.82 |

***Build Stage Bottlenecks:***

A pipeline is taken that has 5 build stages and the number of bottlenecks predicted by LightGBM algorithm is computed. It is observed that there are on an average 10 bottlenecks and the highest number of bottlenecks have been observed in the build 4 stage of the pipeline. Figure 4 shows the build stage bottleneck. Out of the three models; Random forest, Logistic regression and LightGBM; it was observed that LightGBM performs best in terms of accuracy, AUC and F1 with 81%, 77% and 82% respectively.

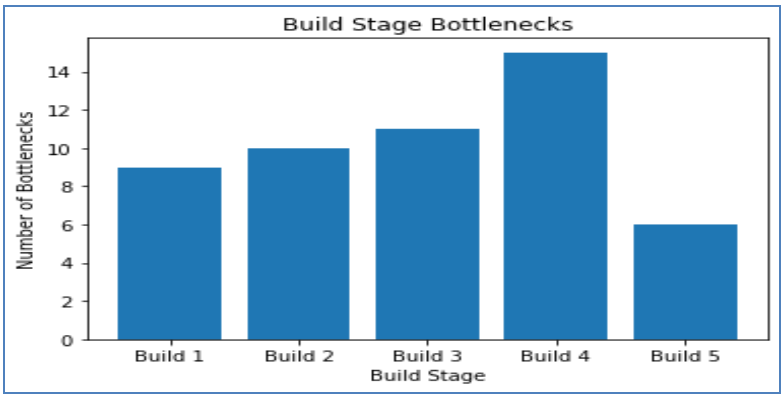


**Fig. 4:** Build Stage Bottleneck

***Test stage Bottlenecks:***

A pipeline is taken that has 4 test stages namely unit tests, integration tests, system tests and performance tests; and the number of bottlenecks predicted by the LightGBM algorithm is computed. It is observed that there are on an average 7 bottlenecks and the highest number of bottlenecks have been observed in the build unit test stage of the pipeline. Figure 5 shows the test stage bottleneck.

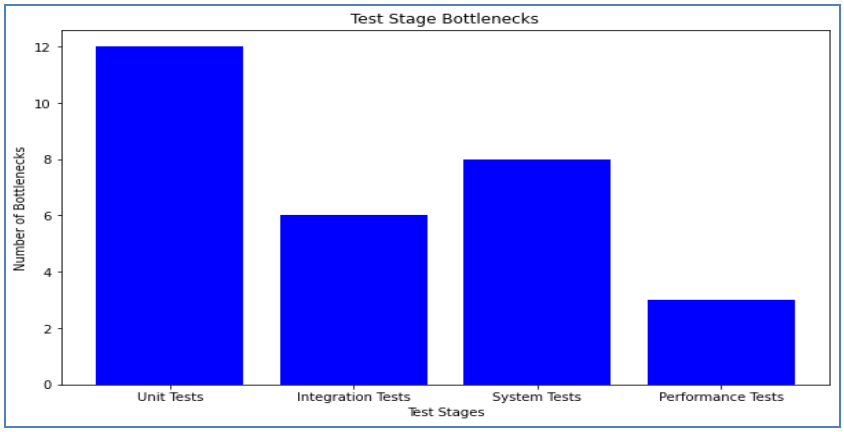


**Fig. 5:** Test Stage Bottleneck

***Deployment Stage Bottlenecks:***

A pipeline is taken that has 4 deployment stages namely; build artifacts, configuration management, deployment orchestration and validation; and the number of bottlenecks predicted by Light GBM algorithm is computed. It is observed that there are on an average 4 bottlenecks and the highest number of bottlenecks have been observed in the deployment orchestration stage of the pipeline. Figure 6 shows the build stage bottlenecks.

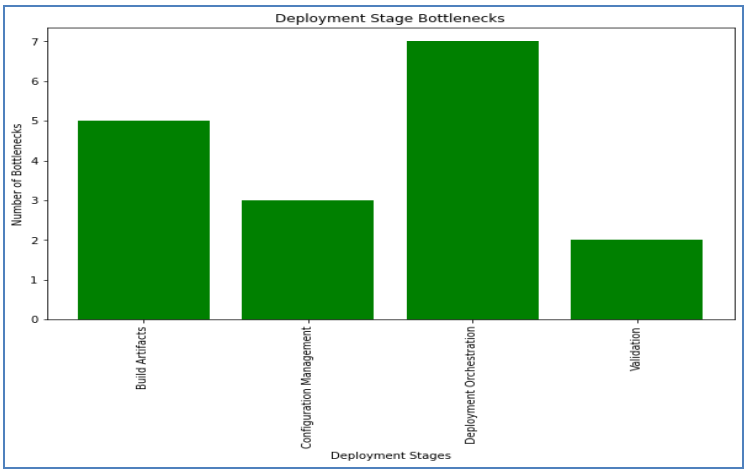


**Fig. 6:** Build Stage Bottlenecks

For the evaluation of the model, three months were consumed using 3 projects information from March 2023 to June 2023. Initially, the bottlenecks in the small-scale projects were detected easily due to the size of the project. After trying to improve the bottlenecks and re-simulating the project, the efficiency of the project was improved by 20% with the standard error of 5%. For the medium scaled projects, it improved by 10% with the standard error of 2%. For the large-scaled projects, the improvement was 5% with a standard error of 2%.

To select the features, an RFE (Recursive Feature Elimination) algorithm was used along with Boruta Python and performed a combination of forward and shadow feature searches on the data to determine the best features possible. Multiple models were used for experimentation to find the one that best predicts the data. For any given model, stratified k-fold cross-validation (k=10) was employed with a 30% test set and a 70% training set, utilizing a dataset of 1.2 million observations. The accuracy scores, AUC scores, and F1 were calculated of each classifier to compare and contrast their performance. When comparing the results across all the models, it was observed that the LGBM classifier achieves the highest accuracy and AUC score. However, other algorithms were either overfitting or having high bias. To explore the effect of the model, models were evaluated for the different project deployments. For this analysis, the random forest classifier and logistic regression were excluded due to high overfitting and high bias. When predicting the bottlenecks, the projects were sized into small-scale, medium-scale, and large-scale. The small-scale project bottlenecks were evaluated by implementing the algorithm on the previous small-scale projects and correcting all the bottlenecks in the current project. The productivity in terms of time and efficiency was improved by 20-30%. For the moderate-scale projects, the evaluation was done in the same way but only few bottlenecks available were tested due to the limitation of time, and this improved the efficiency by 10-12%. For the large-scale projects, efficiency improved by 7% as only partial testing was possible due to the limitation of time.

## 4. CONCLUSION

This work addressed the challenges of identifying bottlenecks in CI/CD pipelines and proposed a predictive model that leverages machine learning techniques to provide insights for improvement. The model is trained on historical data from the CI/CD system, including build logs, test results, code metrics, deployment logs, and performance monitoring data. The evaluation of the model demonstrates its effectiveness in identifying bottlenecks at different stages of the CI/CD pipeline. By addressing these bottlenecks, developers can optimize

the software delivery process and increase efficiency. The results show that small-scale projects can achieve efficiency improvements of up to 20-30%, moderate-scale projects can achieve improvements of around 10-12%, and even large-scale projects experience an efficiency boost of approximately 7%. By utilizing the insights provided by the predictive model, developers can make informed decisions to improve the CI/CD process. This leads to shorter delivery cycles, reduced development time, and increased overall efficiency of the software development. The integration of machine learning techniques in the CI/CD pipeline contributes to more effective software delivery, enabling organizations to stay competitive in the rapidly evolving software development landscape. Overall, the proposed predictive model offers a valuable solution to the challenge of bottleneck identification in CI/CD pipelines. It bridges the gap between manual inspection and data-driven analysis, empowering development teams to proactively address performance issues, optimize resource allocation, and streamline the software delivery process. With the ability to accurately identify bottlenecks and provide actionable insights, organizations can enhance their software delivery efficiency and reliability, ultimately leading to improved competitiveness in the market.

**REFERENCES**

1. V. Singh et al.: Performance analysis of middleware distributed and clustered systems (PAMS) concept in mobile communication devices using Android operating system. In: PDGC, pp. 345-349 (2014).
2. Khanahmadi, M. et al.: Detection of microservice-based software anomalies based on OpenTracing in cloud. Softw: Pract Exper. 53(8), 1681–1699, 2023.
3. Guo, Z., et al.: Deep Federated Learning Enhanced Secure POI Microservices for Cyber-Physical Systems. *IEEE Wireless Communications* 29 (2), 22-29 (2022).
4. Singh, Vinay et al.: A digital Transformation Approach for Event Driven Micro-services Architecture residing within Advanced vcs. In: CENTCON, pp. 100-105, (2021).
5. A. Vishnoi et al.: An improved cryptographic technique using homomorphic transform. *Proc. ICICICT,* 1451-1454 (2022).
6. Jian, Chen et al.: TraceGra: A trace-based anomaly detection for microservice using graph deep learning. Computer Communications 204, 109-117 (2023).
7. Singh et al.: Improving Business deliveries using Continuous Integration and Continuous Delivery using Jenkins and an Advanced Version control system for Microservices-based system. *Proc. IMPACT*, 2022, pp. 1-4.
8. Chakradar M. et al.: Feature selection for insulin resistance using random forest based approach. J Arch. Egyptol. 18 (4), 4861-4879 (2021).
9. Asma, Belhadi et al.: Reinforcement learning multi-agent system for faults diagnosis of mircoservices in industrial settings. Computer Communications 177, 213-219 (2021).
10. Muhammad, Abdullah et al.: Unsupervised learning approach for web application auto-decomposition into microservices. Journal of Systems and Software 151, 243-257 (2019).
11. A. Vishnoi et al.: Image Encryption Using Homomorphic Transform. *Proc. ICICICT,* 1455-1459 (2022).
12. Mishra, S., et al.: Analysis of security issues of cloud-based web applications. Journal of Ambient Intelligence and Humanized Computing 3 (1), 50 (2020).
13. Mahmudul, Hasan wt al.: Attack and anomaly detection in IoT sensors in IoT sites using machine learning approaches. Internet of Things, 7, 100059 (2019).

14. Singh V. et al.: Improving Business deliveries using Continuous Integration and Continuous Delivery using Jenkins and an Advanced Version control system for Microservices-based system. In: IMPACT, pp. 1-4 (2022).
15. Madam, Chakradar et al.: A Non-invasive Approach to Identify Insulin Resistance with Triglycerides and HDL-c Ratio using Machine learning. Neural Processing Letters (NEPL) 55 (1), 93-113 (2023).
16. Nobre, J.: Anomaly Detection in Microservice-Based Systems. *Appl. Sci. 13,* 7891 (2023).
17. Singh. V. et al.: A holistic, proactive and novel approach for pre, during and post migration validation from subversion to git. Computers, Materials & Continua 66 (3), 2359-2371 (2021).
18. Thullier, F. et al.: LE2ML: a microservices-based machine learning workbench as part of an agnostic, reliable and scalable architecture for smart homes. J Ambient Intell Human Comput 14, 6563–6584 (2023).
19. Singh, A. et al.: Event Driven Architecture for Message Streaming data driven Microservices systems residing in distributed version control system. In: ICISTSD, pp. 308-312 (2022).
20. Maha, Driss et al.: Microservices in IoT Security: Current Solutions, Research Challenges, and Future Directions. Procedia Computer Science 192, 2385-2395 (2021).
21. Singh, V. et al.: Event Driven Architecture for Message Streaming data driven Microservices systems residing in distributed version control system. In: ICISTSD, pp. 308-312 (2022).
22. A. Vishnoi et al.: A Cryptosystem analysis for text messages using Homomorphic Transform. *Proc. ICICICT,* 1445-1450 (2022).
23. Madam Chakradar et al.: A Regression based machine learning model to estimate the missing fat-mass parameter in CALERIE study dataset. J Arch. Egyptol, 17 (12), 1533-1546 (2021).
24. Ankit Vishnoi et al.: Text encryption for lower text size: Design and implementation. Materials Today: Proceedings, 79 (2), 278-281 (2023).
25. A. Vishnoi, A. Aggarwal, A. Prasad, M. Prateek and S. Aggarwal: Text encryption for lower bandwidth channels: Design and implementation. *2022 Third International Conference on Intelligent Computing Instrumentation and Control Technologies (ICICICT),* 1460-1464 (2022).