International Journal of
INTELLIGENT SYSTEMS AND APPLICATIONS IN ENGINEERING
ISSN:2147-6799 www.ijisae.org Original Research Paper

An Integrated Container Monitoring Model Using Machine Learning Operations

Zeinab Shoieb Elgamal[1*], Laila Elfangary[2], Hanan Fahmy [3]

Submitted: 12/03/2024 **Revised**: 27/04/2024 **Accepted**: 04/05/2024

Abstract: Machine Learning Operations (MLOps) are designed to accelerate the development of high-quality machine learning (ML) models by reducing the deployment cycle and improving overall efficiency. Despite its promise, the concept of MLOps remains underexplored, with unclear implications for research and practical application. Current research has primarily focused on developing individual ML models, overlooking the complexities of deploying and managing integrated ML systems in real- world scenarios. A comprehensive understanding of system interactions is crucial, particularly when using multi-container services, which necessitate robust and effective monitoring solutions. In response, this study proposes a novel model, the Multi-Containers Monitoring Model, which leverages ML techniques such as Bidirectional Long Short-Term Memory (Bi- LSTM) and State-Action-Reward-State-Action (SARSA) to address these challenges. The proposed model enables the effective scaling and monitoring of MLOps systems by interpreting and managing interactions between containers. It also expands software deployment capabilities across various settings, enhancing software release performance. The results demonstrate that Multi-Containers Monitoring Model improves deployment cycles by up to 24.55%, reduces build length cycles by up to 13%, and decreases response time by up to 50.03%. This study offers a significant advancement in utilizing MLOps for real-world ML system monitoring and deployment.

Keywords: *Machine Learning Operations; Machine Learning; Monitoring; Deployment; Container.*

1. Introduction

The rapid rise in popularity of ML applications has increased the focus on MLOps, which involves continuous integration and deployment (CI/CD) of ML-powered systems. Unlike traditional software, where changes primarily affect code, ML systems also involve model parameters and data, requiring CI/CD automation to expand and support model retraining in production environments [1][2]. However, many real-world ML applications fall short of expectations, as ML field has concentrated largely on building models rather than creating production-ready ML products and coordinating their deployment [3][4]. Moreover, these applications now generate and manage enormous amounts of operational data, requiring real-time monitoring [5]. Without careful monitoring of MLOps model selection and training, applications risk losing market relevance, potentially costing organizations financially and damaging their reputation [6].

This study proposes a multi-container monitoring (MCM) model for software deployment cycles, which tracks communication and container behavior to enable

more frequent releases and reduce production issues. It also addresses MLOps methodology to tackle challenges in developing and monitoring efficient ML. The study adopts a comprehensive perspective to outline key principles, responsibilities, and architectural frameworks involved.

This research provides a valuable contribution to the software industry through:

- Ensure a unified understanding of terms "container," "DevOps," and "MLOps," along with their associated concepts.
- Highlight a variety of studies on ML-based container orchestration techniques and MLOps.
- Propose a new ML model (MCM) to monitor and learn more ML model features based on different software systems to improve software performance.

The structure of this study is as follows: Section 2 compares various ML methods. Section 3 outlines the methodology and introduces the proposed MCM model.

1 Department of Information Systems-Faculty of Computers and Artificial Intelligence, Helwan University, Helwan, Egypt.
ORCID ID: 0000-0002-9446-0133
2 Department of Information Systems-Faculty of Computers and Artificial Intelligence, Helwan University, Helwan, Egypt.
3 Department of Information Systems-Faculty of Computers and Artificial Intelligence, Helwan University, Helwan, Egypt.
ORCID ID: 0000-0002-7247-4825
** Corresponding Author Email: zeinab_elgamal@fci.helwan.edu.eg*

Section 4 details dataset, while Section 5 explains model setup. Section 6 presents the results, followed by a concluding summary in Section 7.

Software engineering uses models like waterfall and agile, aiming to deliver production-ready applications [7]. DevOps replaced old models as traditional lifecycles are unsuitable for dynamic ML projects, as shown in Figure 1.

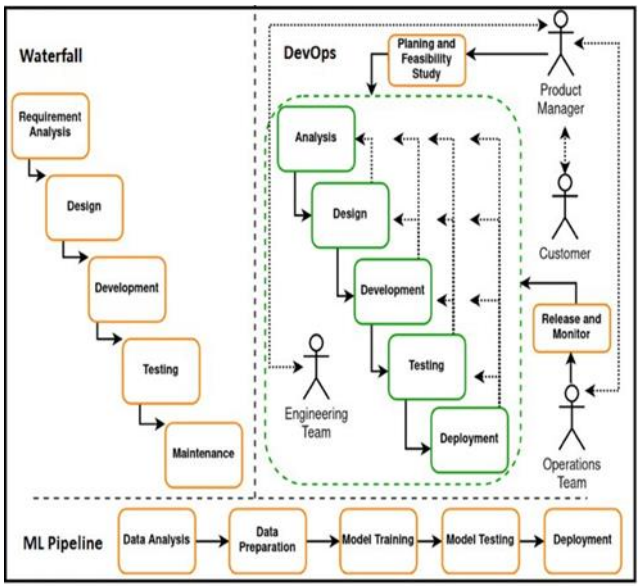

Figure 1. Software Methodologies and ML pipeline [8].

DevOps combines software development and IT operations, aiming for rapid releases, team communication, automation, and continuous integration, delivery, and monitoring [9][10]. Continuous integration (CI) automates code integration from multiple developers, promoting frequent merges to speed development and improve quality [11]. Continuous delivery (CD) aims to deliver new features quickly by ensuring software is always production-ready [12]. Continuous deployment (CDE) is commonly mistaken for CD. With continuous deployment, all software changes are automatically deployed to production. Nevertheless, some companies have policies in place to get external clearance before making a new version of an application available to users. Continuous deployment is therefore optional and can be omitted, yet continuous delivery is deemed required in specific situations [8].

2. Machine Learning Techniques

The reviewed studies, which published from 2018 to 2024, covered a variety of ML techniques that have been used to container orchestration, including workload modeling and reinforcement learning decision-making. To improve prediction accuracy and computing efficiency, the increasing use of ML solutions seeks to

Continuous monitoring uses cloud services to evaluate app performance against business criteria [13]. The ML pipeline automates the ML lifecycle, reducing human involvement [14].

MLOps integrates DevOps with ML practices, covering the entire lifecycle from data design to deployment, with a focus on automation and continuous monitoring [15][16], as illustrated in Figure 2.

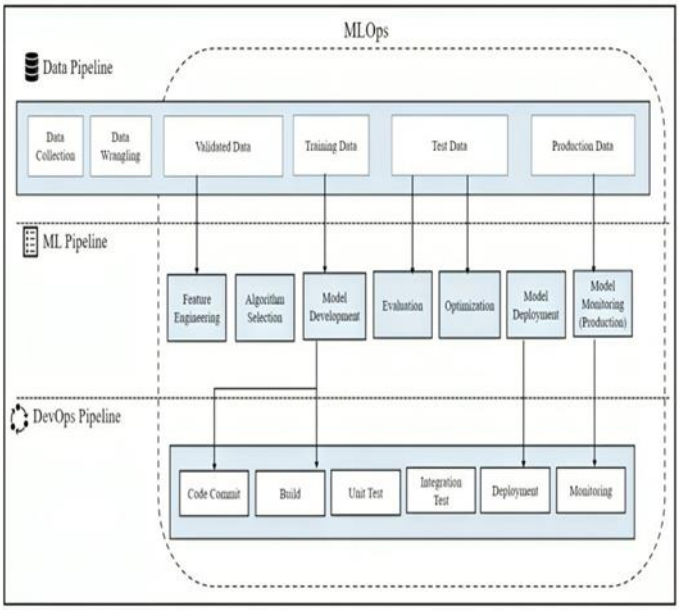

Figure 2. MLOps High-level Process [14].

Containerization, dockerizing, and other roles and concepts are often associated with MLOps stages [14][17]. Containerization is already a common practice for settings that provide the on-demand, transient execution of computing operations [18][19]. Containers can be expanded horizontally and can be transparently replaced, reused, and updated [20]. Dockerizing simplifies hosting and execution various software

applications, platforms, middleware, databases and provides an efficient way to isolate networks and limit container resource usage [21].

Microservices architecture (MSA) helps monitor ML projects during development, unlike monolithic design [22]. Microservices decompose applications into core functions, where each function operates as its own "service" within a container and communicates with other containers over the network [23]. In contrast, monolithic design suits only small-scale systems with simple internal structures [24].

integrate numerous contemporary ML techniques to create a complete orchestration pipeline, which includes resource provisioning and multi-dimensional behavior modeling. The development of ML models also makes it easier to expand various cloud infrastructures and application designs. Tables 1 and 2 provide the goals and matrices for each of these algorithms.

Table 1. Container orchestration objectives based on ML- approach.

Objective	performance analysis	dependency analysis	scaling	scheduling	estimate task arrival rates	forecast resource utilization	forecast resource needs	estimate request arrival rates	anomaly detection	computation offloading	Ref
CNN + BT	√	√	√		√						[25]
K-means	√		√	√		√			√	√	[26]
Model-based RL	√		√	√		√			√	√	[27] [26]
GRU	√		√	√		√			√	√	[28]
LASSO	√		√	√		√			√	√	[29]
DT	√		√	√		√			√	√	[30]
PR	√		√	√		√			√	√	[26]
DRL	√		√	√		√			√	√	[31]
Q-Learning		√	√				√		√		[32]
SARSA	√			√				√	√		[33]
MDP + Q-Learning	√		√	√		√			√	√	[33]
MDP + SARSA	√	√	√								[27]
SVM	√			√				√	√		[29] [34]
Actor-Critic	√		√	√		√			√	√	[31]
SVM + Actor-Critic	√	√	√								[29] [31]
LSTM		√	√				√		√		[28]
BI-LSTM	√			√				√	√		[35]
BI-LSTM + SARSA	√	√	√		√						[36]
Gradient Boosting Regression		√	√				√		√		[37]
ANN	√			√				√	√		[38] [31] [35]
NN						√					[28]
NB	√			√				√	√		[35]
ARIMA						√					[39]
SVR	√			√				√	√		[38]
RF	√			√				√	√		[35]
LR	√			√				√	√		[38]

Table 2. Container orchestration matrix based on ML- approach.

Matrix	resource demands	application latency	propagation timeouts	image's size of the container	power usage	duplicate sizes	time of response	request arrival rate	request type	package loss rate	task completion time	Ref
CNN + BT	√	√										[25]
K-means	√		√	√	√	√	√					[26]
Model-based RL	√		√	√	√	√	√					[27] [26]
GRU	√		√	√	√	√	√					[28]
LASSO	√		√	√	√	√	√					[29]
DT	√		√	√	√	√	√					[30]
PR	√		√	√	√	√	√					[26]
DRL	√		√	√	√	√	√					[31]
Q-Learning	√											[32]
SARSA	√									√	√	[33]
MDP + Q-Learning	√		√	√	√	√	√					[33]
MDP + SARSA	√					√	√	√	√			[27]
SVM	√									√	√	[29] [34]
Actor-Critic	√		√	√	√	√	√					[31]
SVM + Actor-Critic	√					√	√	√	√			[29] [31]
LSTM	√											[28]
BI-LSTM	√									√	√	[35]
BI-LSTM + SARSA	√	√										[36]
Gradient Boosting Regression	√											[37]
ANN	√									√	√	[38] [31] [34]
NN	√											[28]
NB	√									√	√	[34]
ARIMA	√											[39]
SVR	√									√	√	[38]
RF	√									√	√	[34]
LR	√									√	√	[38]

3. MCM Model

Figure 3 illustrates the proposed MCM model, which consists of four different layers (Development layer, MLOps and container layer, Monitoring layer and Tools and Automation layer).

The proposed model initiates with developer's code commit and culminates in its deployment and monitoring across multiple environments. Next sections present a detailed description of layers' components.

3.1 Development Layer

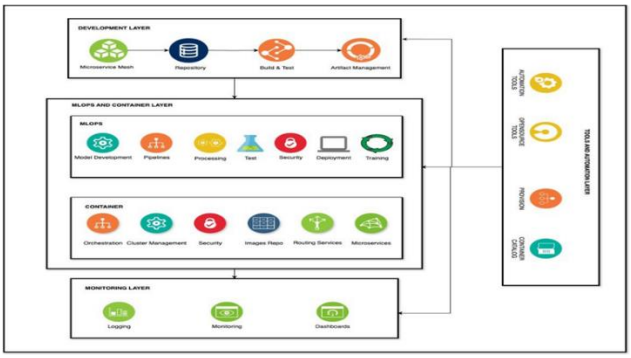

Figure 3. The Proposed MCM Model.

Multiple containers make up the container layer. Consequently, the identical procedures will be followed by every container in this

The development layer, the first component of the MCM model, covers application structure (microservices) and code lifecycle. Its goal is a well- designed framework, with actions focusing on managing code changes and implementing CI to minimize conflicts.

Code lifecycle processes ensure proper versioning and a stable code version in the main repository. After a successful build, test cases are run, and the system artifact is prepared for release and deployment.

3.2 MLOps and Container Layer

This layer has two sublayers: MLOps on top, followed by container layer. In MLOps, MCM model is developed, pipelines are built, tests are prepared, configuration is done, and security gates are added. The most important stage is determining the application type since it forms the foundation for the other steps. Understanding application type to identify and analyze the received data, is the second step. After data is received, the third stage, data transformation, starts. Setting up some security validation over the model pipeline and configuring all required pipeline deployment environments constitute the fourth phase. MCM's model training phase is the fifth step. The validation phase comes in at number six. The final phase at MLOps layer, retraining step, is dependent on MCM model validation process results.

tier. Microservices mesh, image repository, cluster management, container security, and container orchestration are among the steps. Along with deployment, auto-scaling, health monitoring, migration, and load balancing, this procedure also includes container's network configuration, security, and resource allocation.

3.3 Monitor Layer

This layer includes dashboards for monitoring MCM model stages, validating and addressing anomalies. It stores internal logs for pipeline troubleshooting and external logs for data application tracking, helping identify malfunctions and ensure smooth operation.

3.4 Tools and Automation Layer

This layer provides tools to support MCM model layers, ensuring reliable statistics and smooth transitions. It uses open- source and automation tools for provisioning and container cataloging.

4. Dataset

MCM model's dataset gathers unstructured data from sources like source control software, software engineers' comments, workload patterns, system configurations and historical performance data. It involves two phases: collecting and organizing data, then determining if labeling is needed. K-means clustering is used to label the dataset into layers and sub-layers based on unlabeled attributes.

The clustering strategy was described in Algorithm 1. Dataset was made up of a set of attributes A = {a1,..., at}, and each attribute was given a primary label L as well as a set of auxiliary labels P if necessary. A set of labels $L \neq \emptyset$ and a set of tuples T with the same cardinality as A, that is

T= {(a1, l1, p1),...,(at, lt, pt)}, where li $\in$ L for all $1 < i \leq t$, are what remains at the end of the operation. Without diverting its attention to a different label, the algorithm seeks to increase the number of qualities that have the same label applied to them consecutively. The goal is to reduce the cost of labeling multiple qualities at once and aid in model identification by combining attribute analysis.

Algorithm 1: Cluster Labeling Algorithm

Notations: T // Set of labeled attributes along their labels, D // Dataset, A // Attribute in the data set, L// Label, P// additional label, K// unvisited attributes
Input: Set of A in D
Output:
1) A set of pairs, T (attribute, label, additional label).
2) K keeps unvisited attributes.
Steps:
1) T ← ∅
2) Repo VAL
3) Code merge

```
While (D has unlabeled attributes) do
    // Sequence of exploration
    a ← pick unlabeled attribute from A
    Manually inspect a to assign L_P
    T ←T ∪ {(a, l, p)}
    K ← {a}
    While (K is not empty) do
        a ← pop (K)
        // Similar attribute retrieval
        F ← search for non-labelled attribute a1, ..., an sorted by similarity to a;
        Manually inspect a1, ..., an to assign L_P
        A = {(a1, l),....,( an,l)}
        T←T∪D
        // Attribute retrieval improvements
        a' ← pick relevant attribute from D
        add a' to K
        End
    End
```

Dataset was separated into 6 primary layers and 8 sub-layers for each of the 158 attributes after the algorithm was run. Table 3 displays the algorithm's result.

Table 3. Cluster labeling algorithm's result.

Layer	Sub-Layer	Attributes
Owner	-	1. username 2. user privilege
Data System	-	1. collection name 2. commit ID 3. commit description 4. number of reviewers on pull request 5. source control system type
App/Web and Database	-	1. application type 2. application language 3. number of databases 4. databases names 5. detailed component names 6. third-party application 7. application behavior details in terms of error codes 8. number of APIs 9. topological dependencies
Pipeline	Agent	1. agent pool name 2. pool owner 3. pool information 4. Pool consumption report 5. name 6. demands 7. execution plan 8. capability name 9. capability value 10. username 11. user role 12. user access 13. last run 14. status 15. version 16. maintenance job 17. maintenance history 18. approvals and checks
	Pipeline	1. type 2. name 3. pool 4. parameters 5. set of tasks 6. predefined variables 7. variable groups 8. triggers 9. branch filters 10. failure rate 11. pre-defined tasks 12. post-defined tasks 13. last run 14. run time rate 15. pipeline logs 16. traces /incidents descriptions 17. list of all the dependencies for each pipeline. 18. error symptoms 19. conditions (setting option offline) 20. additional parameters 21. output variables 22. created date 23. changed date 24. change type 25. changed by 26. change comment 27. relationships across different pipelines
	Build	1. description 2. number format 3. status 4. automatically link new defects 5. defect related branch 6. image URL 7. image URL for the branch 8. Docker file path 9. Docker image to build 10. Docker image to deploy 11. markdown link 12. job authorization scope 13. job timeout (minutes) 14. job cancel timeout (minutes)
	Code	1. source of software code 2. team project 3. repo name 4. branch 5. tag sources 6. cleaning option type
	Task	1. count 2. version 3. link settings 4. architecture 5. arguments 6. configuration 7. platform 8. number of retries 9. control option 10. specification 11. timeout 12. goals 13. search patterns 14. symbols folder path 15. indexing type 16. SSH connection 17. source path 18. target path 19. machines 20. admin login 21. admin password 22. tool name to run
	Artifact	1. name 2. path to publish 3. publish location 4. max artifact size 5. tar archive 6. source 7. default version

		8. source alias 9. artifact downloaded
	Test	1. JUnit test results 2. test results files 3. test run title 4. code coverage tool 5. code analysis type 6. test filter criteria 7. platform version 8. test run parameters 9. batch tests 10. test framework 11. authentication method
	Stage	1. stage name 2. retention policy 3. release name format 4. integrations 5. Pre-deployment conditions 6. Post-deployment conditions
Container	-	1. name 2. hash 3. size 4. number 5. trace IDs 6. networks 7. storage systems 8. storage available 9. capacity metric 10. operating system 11. registry name 12. registry type 13. registry username 14. container app name 15. container app environment 16. ingress setting 17. YAML path 18. registry connection 19. startup command 20. namespace flag 21. secrets container tag
Package	-	1. name 2. version 3. type 4. image tag 5. image size

5. Configuration

The first major step in MCM model configuration phase is developing application, followed by setting up ML model.

5.1 Application Development

Application development divides functionality into services, integrates DevOps and MLOps pipelines, and uses containerization for scalability, management, and automated feature rollout.

1)
ransition from Monolithic to Containerized-Microservices Applications

This shift demands careful planning and execution for a successful migration. To enable resource scaling for microservices, including autoscaling capabilities, RedHat OpenShift and Kubernetes were selected as the orchestration platforms. Additionally, GitOps tools, the platform CLI, and Azure pipelines were utilized to execute the deployment process.

Four primary stages made up the transition process:

- Determine components: using business functionalities as a guide, divide large application into more manageable, smaller parts.
- Define clear interfaces: set up well-defined interfaces between microservices.
- Data management: define how data will be handled and shared across microservices.
- Technology stack: choose the most appropriate technologies and frameworks for developing and deploying each microservice.

2) Containerization with Docker

- Microservice packaging: to ensure consistency and portability across many contexts, containerize each microservice and its dependencies using Docker.
- Dependency management: by encapsulating dependencies, Docker containers remove conflicts and guarantee that every microservice runs independently.
- Simplified deployment: Docker streamlines deployment of microservices.

3) Orchestration with Kubernetes

- Automated deployment: Kubernetes automates deployment process which includes scheduling, scaling, and monitoring.
- Service discovery and load balancing: Kubernetes includes built- in capabilities for service discovery and load balancing, ensuring effective communication between microservices.
- Fault tolerance: Kubernetes provides features such as self- healing and replication, which guarantee high availability and resilience of microservices in the event of failures.
- Scaling: Kubernetes supports horizontal scaling of microservices according to resource utilization.

4) CI/CD Pipelines

- Implement CI/CD pipelines: to automate build, test, and deployment of microservices application to guarantee that updates are delivered quickly and reliably.
- Version control system selection: proposed model used Git to manage the source code, facilitating collaboration and tracking changes.

- Apply pull request policy over the pipeline.

- Store Artifact: The outcomes of running CI/CD pipeline called "Artifact". Artifact contains container images which will be used for different environments' deployments. Storing process depends on pushing locally produced image to shared registry (RedHat Quay and/ or Docker Hub) by using image ID.

Application development process illustrates how the refactored microservices code can be automatically built using Docker as shown in Figure 4.

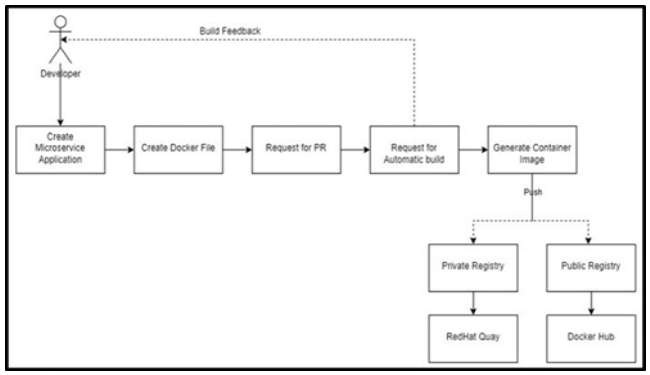

Figure 4. Microservice-Container Based Application Development.

5.2 Setting Up

The MCM model integrates Bi-LSTM with SARSA algorithms to improve ML performance and reduce deployment times for containerized applications. Bi-LSTM, a recurrent neural network with multiple LSTM layers, works with SARSA reinforcement learning to optimize container performance. This approach is embedded in CI/CD pipelines and monitoring systems for continuous optimization. By processing streaming data, Bi- LSTM and SARSA adjust to workload changes, optimizing resource use. Performance is evaluated with backpropagation through time (BPTT) and the adaptive moment estimation (Adam) optimizer. The integration architecture is shown in Figure 5.

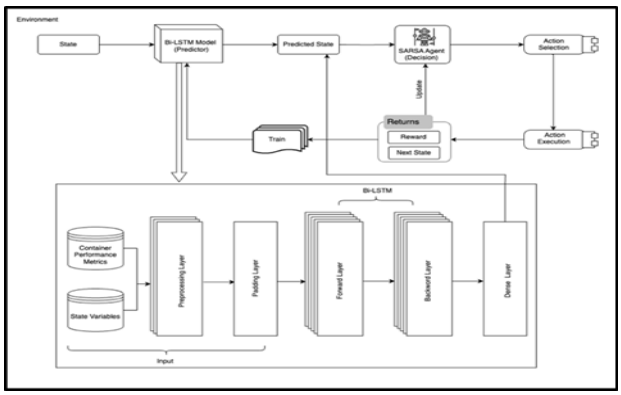

Figure 5. Proposed Model Integration Architecture.

The integrated operation in MCM's model architecture is represented mathematically by the following equations:

5) Define state x_t at time t as a vector containing performance metrics and resource usage data:

x_t = [CPU utilization$_t$, memory utilization$_t$, network throughput$_t$, latency$_t$, request rate$_t$, error rate$_t$].

6) Initialize Bi-LSTM Model with BPTT

- Forward Pass is denoted in Eq. from (1) to (3)

By given a sequence of input states x_{t-k+1}, x_{t-k+2}, …, x_t where k is the sequence length. For each time step i, compute hidden states h_i using Bi-LSTM is denoted as in Eq. (1):

$$h_i = \text{BiLSTM}(x_i, h_{i-1}) \quad (1)$$

Where W_O is the wight matrix and b_O is the bias of the output layer, the output y' at each time step is computed as in Eq. (2) and loss L is computed as the mean squared error (MSE) between the predicted y' and the actual y is denoted as in Eq. (3):$y'_i = W_O h_i + b_O$ (2)$L = \frac{1}{4^N} (y_i - y'_i)^2$ (3)Backward Pass is denoted in Eq. from (4) to (6)For updating model weights, backpropagate the gradient through time hence the gradient of Loss with the consideration of the output is computed as in Eq. (4): $\frac{\partial L}{\partial y'_i} = 2(y' - y)$ (4)

3) For each time step i from t to $t-k+1$, the gradients of the loss with the consideration of Bi-LSTM parameters θ are computed as in Eq. (5):

Integration of Bi-LSTM and SARSA

- At each time step t, predict future state by use Bi- LSTM is computed in Eq. (12):

$$S'_{t+1} = \text{BiLSTM}(x_t, h_{t-1}) \quad (12)$$

- SARSA agent uses predicted state S'_{t+1} to select action and update M-values as denoted in Eq. (13):

$$a_t = arg\ max_a\ M(S'_{t+1}, a) \quad (13)$$

- Collect new sequences of data periodically and retrain Bi-LSTM using BPTT.

6. Discussion

Initially, experiments were conducted on a local Windows Server 22 OS machine with 32 GB RAM to train and test MCM model. MCM model was applied on different code bases (Java and .Net) projects. Java projects have been developed with SpringBoot and .Net applications have been developed with .net framework versions 4.8.0,

4.8.1, and 6. Lift-and-Shift and refactoring migration methods were chosen for MCM model.

The pipelines are implemented by YAML. CD pipeline included integration, quality control, security testing, UAT, load testing, packaging, and pre- production environments. RedHat-OpenShift and Kubernetes were chosen as orchestration platforms for MCM

$$\frac{\partial L}{\partial \theta} = \sum_{t=t-k+1} \frac{\partial L}{\partial y'_i} \cdot \frac{\partial y'_i}{\partial h_i} \cdot \frac{\partial h_i}{\partial \theta}$$

(5) model to scale resources for microservices and even to simply enable

autoscaling. Platform CLI and Azure pipelines were used for deployment execution. RedHat Quay and Docker Hub registries were

Where α is the learning rate, the model weights are updating using Adam's optimizer as in Eq. (6):

used to push images using image ID to be ready for deployment.

MCM model performance was evaluated and evaluated using BPTT. Adam optimizer was selected to improve performance and

$$\Delta\theta = -\alpha \frac{\partial L}{\partial \theta}$$

(6) accelerate convergence of Bi-LSTM. Further, Adam's optimizer was selected to modify Bi-LSTM weights during training to minimize the

3) Initialize SARSA agent algorithm

- Initialize M-values is computed as in Eq. (7):

$$M(s,a) = 0 \quad \forall s \in S, \forall a \in A \qquad (7)$$

- Select action by using ϵ-greedy policy as denoted in Eq. (8):

loss function. Experiments are implemented using Python 3.9.12. Some of Python libraries is imported into the MCM model code to build it as shown in Figure 6.

$$a_t = \begin{cases} \text{random action} & \text{with probability } \epsilon \\ arg\,max_a\ M(s_t, a) & with\ probability\ 1-\epsilon \end{cases} \qquad (8)$$

- Execute action a_t, observe reward r_{t+1} and next state s_{t+1}.

Select next action by using the same ϵ-greedy policy as denoted in Eq. (9):

$$a_{t+1}$$

```
import pandas as pd
import numpy as np
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import MinMaxScaler
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense, Bidirectional
import matplotlib.pyplot as plt
```

Figure 6. Example of Python Libraries.

Figure 7. examines how the build and deployment frequencies have improved in seconds across the various environments, with build

$$a_{t+1} = \begin{cases} \text{random action} & \text{with probability } \epsilon \\ arg\,max_a\ M(s_t + 1, a) & with\ probability\ 1-\epsilon \end{cases} \qquad (9)$$

durations reaching up to 13% and deployment durations reaching up to 24.55%.

- Where discount factor is denoted by η and learning rate by α, update M-values is computed in Eq. (10):

$$M(s_t, a_t) \leftarrow M(s_t, a_t) + \alpha\,[r_{t+1} + \eta M(s_{t+1}, a_{t+1}) - M(s_t, a_t)] \qquad (10)$$

- Gradually reduce exploration rate as denoted in Eq. (11):

$$\epsilon = \max(\epsilon_{min}, \epsilon \cdot \epsilon_{decay}) \qquad (11)$$

methodologies or results. Thus, to fill this gap, this research proposes an integrated ML model using Bi-LSTM and SARSA algorithms, which leverages automated multi-container architecture and microservices principles throughout build and deployment stages of application development lifecycle.

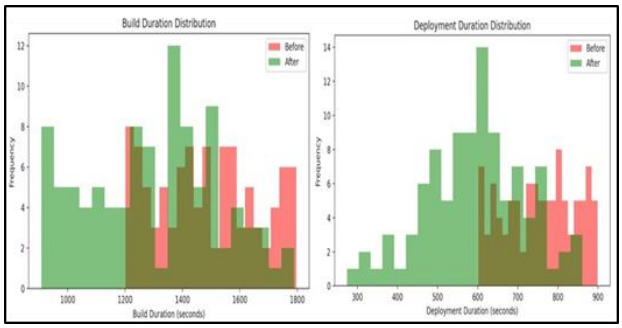

Figure 7. Model Enhancements for Build and Deployment Duration.

The ability to extend the number of containers in the case of a failure was quick, as shown in Figure 8. Figure 9 summarizes the average improvements for the performance metrics used by the MCM model.

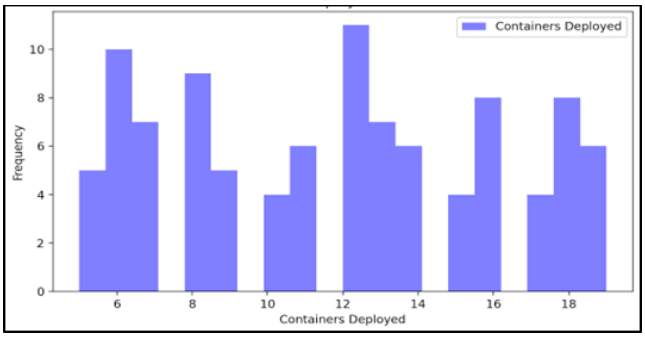

Figure 8. MCM Model AVG Containers Deployed.

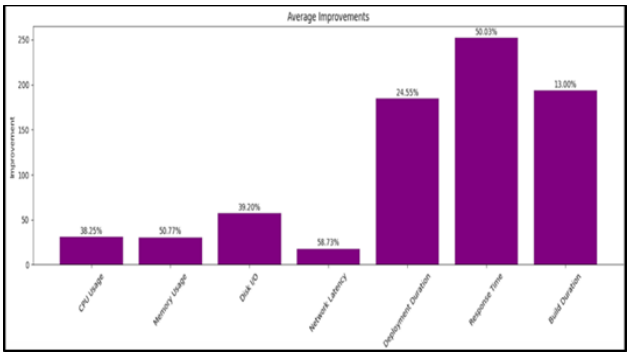

Figure 9. MCM Model AVG Improvements.

7. Conclusions and Future work

The increasing demand for innovation has led to the development of more ML systems than ever, necessitating enhanced monitoring and analysis skills for ML models. Despite this, only a limited number of proofs of concept advance to production deployment. In practice, data scientists continue to manage most ML operations manually. MLOps paradigm seeks to address these challenges. Existing literature highlights some approaches to monitoring MLOps applications, but it lacks studies that specifically address multi- container and microservices architectures, making it difficult to directly compare proposed MCM model with similar MCM model aims to increase frequency of software deployments across various environments, enhance software release performance, and reduce the need for redevelopment and redeployment. By employing MLOps, MCM model improves software deployment cycles by up to 24.55%, reduces build duration cycles by up to 13%, and decreases response times by up to 50.03%. The findings improve deployment rates of existing methods in software systems. Also, provides an effectively monitoring approach for ML model features using MLOps. Additionally, the research highlights broader implications for optimizing software development processes, enhancing operational efficiency, and supporting scalable, robust solutions for industry-wide adoption. Future work should involve more experimental studies to assess MLOps pipelines and their impact on the overall software development cycle. It is also important to implement MCM model on various datasets to monitor its effectiveness and conduct further experiments to compare its performance against baseline approaches or alternative optimization strategies.

Data Availability

The data presented in this study are available on request from the corresponding author.

Conflicts of Interest

The authors declare no conflict of interest.

References

[1] F. Calefato, et al., A Preliminary Investigation of MLOps Practices in GitHub, Association for Computing Machinery,2022, doi:10.1145/3544902.3546636.

[2] D. Kreuzberger,et al., “Machine Learning Operations (MLOps): Overview, Definition, and Architecture,” 2022, [Online],

http://arxiv.org/abs/2205.02302.

[3] Wiggerthale and Julius, “Explainable Machine Learning in Critical Decision Systems: Ensuring Safe Application and Correctness ”, MDPI, 2024.

[4] Z.Shoieb et al., “The Impact of using MLOps and DevOps on Container based Applications: A Survey”,Informatics Bulletin,Faculty of Computers and Artificial Intelligence,2024.

[5] E. Calikus, Self-Monitoring using Joint Human-Machine Learning: Algorithms and Applications, no. 69.

[6] P. Liang et al., “Automating the training and deployment of models in MLOps by integrating systems with machine learning”, Proceedings of the 2nd International Conference on Software Engineering and Machine Learning, 2024, doi: 10.54254/2755- 2721/67/20240690.

[7] B. Karlaš et al., “Building Continuous Integration Services for Machine Learning,” Proc. ACM SIGKDD Int. Conf. Knowl. Discov. Data Min., 2020, doi: 10.1145/3394486.3403290.

[8] P. Ruf, M. Madan, C. Reich, and D. Ould-Abdeslam, “Demystifying mlops and presenting a recipe for the selection of open-source tools,” Appl. Sci., 2021, doi: 10.3390/app11198861.

[9] Z. Shoieb et. al,“Enhancing Software Deployment Release Time Using DevOps Pipelines”, IJSER, vol.11, no. 3, 2020, ISSN: 2229-5518.

[10] M. Rowse and J. Cohen, “A survey of DevOps in the South African software context,” Proc. Annu. Hawaii Int. Conf. Syst. Sci., 2021, doi:

10.24251/hicss.2021.814.

[11] A. Sajid et al., "AI-Driven Continuous Integration and Continuous Deployment in Software Engineering" 2nd International Conference on Disruptive Technologies (ICDT), 2024.

[12] Bollineni and Satyadeepak, "Devops Approaches To Managing And Deploying Machine Learning Devops Approaches To Managing And Deploying Machine," International Journal of

Business Quantitative Economics and Applied Management Research, 2024.

[13] Buttar, Ahmed Mateen et.al "Optimization of DevOps Transformation for Cloud-Based Applications," Electronics (Switzerland),2023.

[14] N. Hewage and D. Meedeniya,"Machine Learning Operations: A Survey on MLOps Tool Support," 2022, doi:10.48550/arXiv.2202.10169.

[15] S. Alla and S. K. Adari, Beginning MLOps with MLFlow, 2021, doi:10.1007/978-1-4842- 6549-9.

[16] G. Recupito et al., "A Multivocal Literature Review of MLOps Tools and Features," 2023, doi: 10.1109/seaa56994.2022.00021.

[17] L. E. L. B, I. Crnkovic, R. Ellinor, and J. Bosch, "From a Data Science Driven Process to a Continuous Delivery Process for Machine Learning Systems," Proceedings- PROFES- 21st Int. Conf., 2020.

[18] C. Segarra et al., "Serverless Confidential Containers: Challenges and Opportunities" 2024.

[19] C. Segarra et al., "Serverless Confidential Containers: Challenges and Opportunities" 2024.

[20] B. Burns,"Design patterns for container- based distributed systems".

[21] E.Summary,"PRINCIPLES OF CONTAINER- BASED".

[22] Abhishek M Nair et al., "Dockerized Application with Web Interface," International Journal of Scientific Research in Computer Science, Engineering and Information Technology, 2023.

[23] J. Brier and lia dwi jayanti, "DevSecOps of Containerization,"2020,[Online].http://journal.um-surabaya.ac.id/index.php/JKM/article/View/2203.

[24] A. Mahesar et al., "Efficient microservices offloading for cost optimization in diverse MEC cloud networks". J Big Data, 2024. https://doi.org/10.1186/s40537-024-00975-w.

[25] Z. Zhong et al.,"Machine Learning-based Orchestration of Containers: A Taxonomy and Future Directions," ACM Comput. Surv., 2022, doi: 10.1145/3510415.

[26] Y. Zhang, W. Hua, Z. Zhou, G. E. Suh, and B. Delimitrou, "Sinan: ML- based and QoS-aware resource management for cloud microservices," Int. Conf. Archit. Support Program. Lang. Oper. Syst.- ASPLOS, 2021, doi: 10.1145/3445814.3446693.

[27] S. Venkateswaran and S. Sarkar, "Fitness- Aware Containerization Service Leveraging Machine Learning," IEEE Trans. Serv. Comput.,2021,doi:10.1109/TSC.2019.2898666.

[28] H. Sami, A. Mourad, H. Otrok, and J. Bentahar, "FScaler: Automatic Resource Scaling of Containers in Fog Clusters Using Reinforcement Learning,"Int. Wirel. Commun. Mob. Comput. IWCMC,2020, doi:10.1109/IWCMC48107.2020.9148401.

[29] Lasitha Maduranga, and Gayan Dharmaratne, "Leveraging Generative Adversarial Networks to Improve LSTM and GRU Models Performances for Stock Price Prediction," International Statistics Conference, 2024.

[30] H. Qiu et al.,"Firm: An intelligent fine- grained resource management framework for SLO-Oriented microservices," Proc. 14th USENIX Symp. Oper. Syst. Des. Implementation, OSDI 2020.

[31] J. E. Dartois et al., "Investigating Machine Learning Algorithms for Modeling SSD I/O Performance for Container-Based Virtualization," IEEE Trans. Cloud Comput., 2021, doi: 10.1109/TCC.2019.2898192.

[32] Y. Bao et al.,"Deep Learning-Based Job Placement in Distributed Machine Learning Clusters With Heterogeneous Workloads,"IEEE/ACMTrans. Netw.,2023,doi:10.1109/TNET.2022.3202529.

[33] Elina Guzueva et al.,"Optimisation tool: Q-learning and its application in various fields," E3S Web of Conferences 515, 2024.

[34] F. Rossi, M. Nardelli, and V. Cardellini, "Horizontal and vertical scaling of container- based applications using reinforcement learning," IEEE Int. Conf. Cloud Comput. CLOUD,2019, doi:10.1109/CLOUD.2019.0006.

[35] H. Dawid, CORDIC Algorithms and Architectures, 2018, doi:10.1201/9781482276046- 22.

[36] X. Tang et al.,"Fisher: An efficient container load prediction model with deep neural network in clouds," IEEE Int. Symp. Parallel Distrib. Process.

with Appl. 17th IEEE Int. Conf. Ubiquitous Comput. Commun. 8th IEEE Int. Conf. Big Data Cloud Comput. 2018, doi:10.1109/BDCloud.2018.00041.

[37] M. Yan et al.,"HANSEL: Adaptive horizontal scaling of microservices using Bi- LSTM,"Appl. Soft Comput., 2021,doi: 10.1016/j.asoc.2021.107216.

[38] David Jobst,"Gradient-Boosted Mixture Regression Models for Postprocessing Ensemble Weather Forecasts," arxiv, 2024, doi: 10.48550/arXiv.2412.09583.

[39] K. Ye and Y. Kou,"Modeling Application Performance in Docker Containers Using Machine Learning Techniques," Proc. Int. Conf. Parallel Distrib. Syst.- ICPADS, 2018, doi:10.1109/PADSW.2018.8644581.

[40] Hongrong Cheng and Miao Zhang, "Predictive Performance Evaluation of ARIMA and Hybrid ARIMA-LSTM Models for Particulate Matter Concentration," JOIN (Jurnal Online Informatika), 2024.