ISSN: 2754-6659

Journal of Artificial Intelligence & Cloud Computing

SCIENTIFIC Research and Community

Research Article

Open Access

A Unified Machine Learning Approach for Efficient Artifact Management in Jenkins CI/CD Pipelines

Praveen Kumar Thopalle

USA

ABSTRACT

In this paper, we provide an innovative way to manage artifacts in Jenkins-based CI/CD pipelines using a single flexible ML model. Managing artifacts effectively is a priority when teams grow. In this paper, we present a single ML model that can handle various artifact management problems such as retention prediction, compression optimization, artifact classification, cache optimization, and anomaly detection. With just one model deployed across these multiple functions, we are able to simplify things, reduce compute overhead, and provide a highly scalable solution for Jenkins deployments on large scaleAmong the major findings of this study are:Creation of a generic multi-task learning system that helps to manage artifacts in CI/CD pipelines. Implementation of the model to solve multiple artifact management problems in a holistic manner.Easy integration of the ML-based solution into Jenkins processes, enabling automation and efficiencies.Performance testing in enterprise CI/CD systems to prove scalability and effectiveness.Comprehensive review of benefits and limitations of artifact management using a single ML model.This work pushes the field forward with a single and complete solution, providing scalability, resource-savings, and artifact lifecycle management in large scale, CI/CD ecosystems.

***Corresponding author**

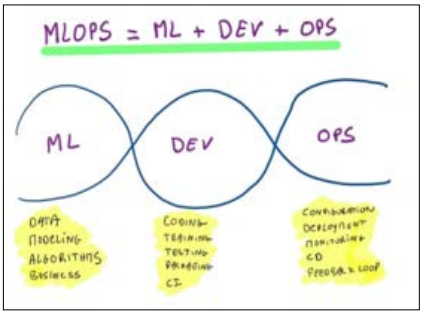
Praveen Kumar Thopalle, USA.

Received: September 05, 2022; **Accepted:** September 12, 2022; **Published:** September 26, 2022

Introduction

Jenkins, an open-source automation server used by most companies, is primarily responsible for automating the workflows in continuous integration and continuous delivery (CI/CD) pipelines. As companies grow their software development teams, build artifact management becomes more complicated and vital. Build artifacts, including binaries and libraries, log files and test results, are a necessary part of the development lifecycle. Yet, as the amount and diversity of artefacts increase, the storage, categorization, retrieval, retention and optimization challenges also increase. Misaligned artifact management can cause storage costs, build time and CI/CD performance issues.

This paper aims to find a new way to resolve these problems, using a single, general ML model to encompass many aspects of artifact management. In contrast to existing systems that use separate tools or workflows for retention policies, compression, caching, and anomaly detection, here we are investigating if a single ML model can do it all smartly. Bringing machine learning to artifact management allows us to automate decision making, storage, and overall efficiency in big Jenkins deployments, resulting in a scalable and resource-efficient solution for today's CI/CD pipelines.



Background

Older artifact management in Jenkins is based on simple rules and manual configurations to store, read and maintain build artifacts. These can be simple retention policies artifacts being kept indefinitely or manual removal according to defined thresholds. This is perhaps acceptable in small projects, but it will be utterly inefficient as the number of artifacts grows and complexity. Managing artifacts manually causes waste, storage costs, and the inability to locate and recover important artifacts when needed. Additionally, static rules are insensitive to changes in need changing demand for an artifact or project priorities which means they either overstore or end up getting deleted.

Responding to these inefficiencies, previous studies have looked at the use of machine learning models to automate and optimize some parts of artifact management. For example, there exist dedicated ML models that have been created for such applications as prediction of retention, classification, or anomaly detection. But while these models boost efficiency in individual work, they add complexity and resource overhead. Multiplication models for different aspects of artifact management further complicates the system's computational overhead and makes it difficult to embed models in current Jenkins pipelines. Therefore, this fragmented model is expensive and difficult to scale in enterprise-scale CI/CD environments.

Proposed Unified ML Model

We introduce a multitask learning model based on deep neural network to handle the multiple aspects of artifact management for Jenkins CI/CD pipelines. This holistic, single model has all artifact management operations covered in one model: retention prediction, compression optimization, categorization, caching and anomaly detection. In making these jobs a single model, we can reduce artifact management effort, computational overhead, and

eliminate having multiple independent models. This deep neural network enables the model to learn new patterns and act smartly on different tasks, resulting in a more efficient and scalable CI/CD pipeline.

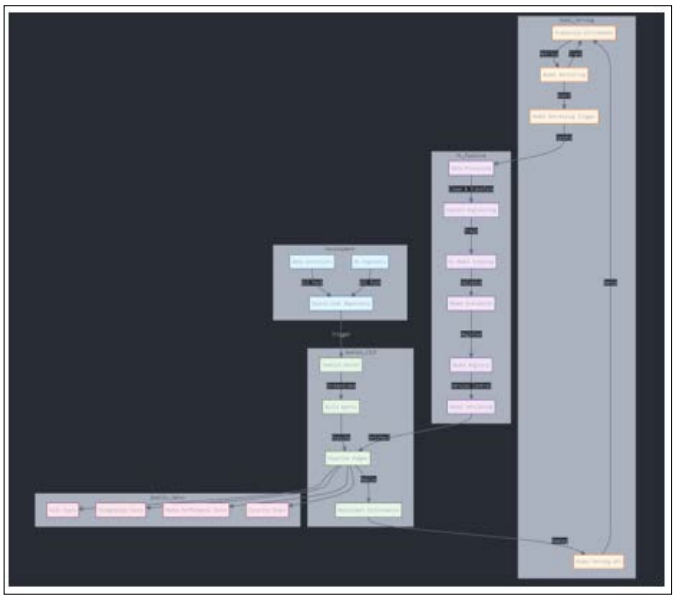


Figure 1: Visualize the Overall Architecture of the Unified ML Model and its Integration with Jenkins-Based CI/CD Pipelines

Model Architecture

This proposed model is based on a multitask learning system. It is composed of a base network that has common features that extracts shared information from artifact data like artifact metadata, access flows, and project specific attributes. This common foundation allows the model to easily learn generalized patterns for all tasks. This is followed by task-oriented layers for implementing the needs of each artifact management function. These specialized layers are responsible for discrete tasks such as retention prediction, compression, categorization and anomaly detection, so the model will produce the exact and customized solution for all artifact management functions.

```python
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Input, Dense, LSTM, Dropout

def create_multitask_model(input_shape, num_tasks):
    inputs = Input(shape=input_shape)

    # Shared layers
    x = LSTM(64, return_sequences=True)(inputs)
    x = LSTM(32)(x)
    x = Dense(64, activation='relu')(x)
    x = Dropout(0.5)(x)

    # Task-specific layers
    outputs = []
    for _ in range(num_tasks):
        output = Dense(32, activation='relu')(x)
        output = Dense(1, activation='sigmoid')(output)
        outputs.append(output)

    model = Model(inputs=inputs, outputs=outputs)
    return model

# Example usage
model = create_multitask_model(input_shape=(100, 10), num_tasks=5)
model.compile(optimizer='adam', loss='binary_crossentropy')
```

Figure 2: Model Training for Jenkins Pipeline Using LSTM

Model Application

Retention Prediction

One of the issues in artifact management is to what extent we should store the artifacts. The most common problem with static retention policies is to either retain more artifacts than necessary or to remove valuable ones. The retention prediction section automatically predicts the likelihood of an artifact being referenced on a build based on usage history, project time and dependencies. This prevents artifacts that are essential from being discarded early and lowers storage costs and enhances retention without any manual effort.

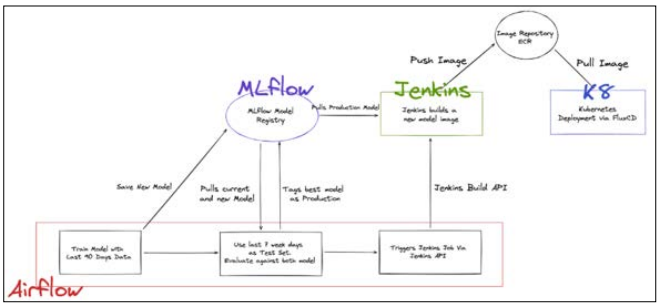


Figure 3: Model Serving for Jenkins with ML and Continuous Retraining

Compression Optimization

Effective compression is needed for reducing storage costs for large-scale CI/CD environments. The model determines the most appropriate compression for each artifact, given its size, type and occurrence. This makes artifacts such as logs, binaries, and test results compressible, in a way that maximizes storage savings and retrieval performance, while providing rapid access to frequently used artifacts and saving storage space.

Automated Categorization

Organizing artifacts into pre-defined categories is crucial to managing them. Manual sorting is prone to mistakes and inefficiencies. The unified model will classify artifacts automatically according to type, usage, and pipeline significance utilizing both static and dynamic metadata. This facilitates the ease of retrieval, and also streamlines the other processes like retention and caching to give priority artifacts proper attention.

Caching Strategy

Caching is essential for both speeding up build and overall CI/CD. But bad caching practices can waste time and provide poor performance. The model predicts what artifacts should be cached based on accesses, their size, and importance, providing a more intelligent cache that allows you to quickly find commonly visited artifacts while minimizing the cache size.

Anomaly Detection

Variations in artifact production, usage or storage can be used to hint at a problem such as configuration errors or security problems. The centralized model constantly records artifact activity for anomalies, including sudden increases in usage or sudden deletions. By finding and alerting teams to these anomalies early, the model halts the cascade of issues before they become major, ensuring greater security and stability of the CI/CD pipeline.

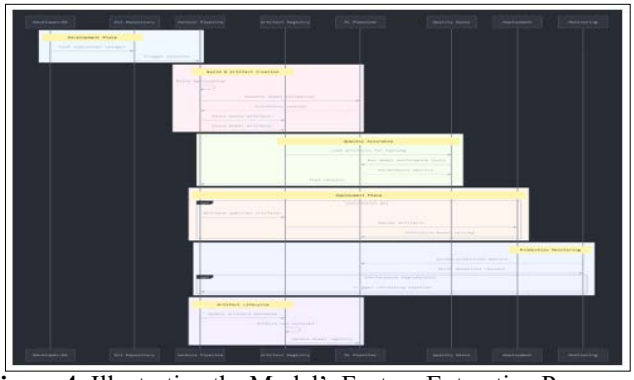


Figure 4: Illustrating the Model's Feature Extraction Process and the Training Pipeline

Feature Engineering

The effectiveness of the single machine learning model depends on the quality and variety of training features. As an artifact manager, mining the feature from artifact and build metadata is important for obtaining the correct answer for retention prediction, compression, categorization, caching, and anomaly detection. The model interprets a comprehensive feature set that defines many different elements of artifact behavior and the connection with build itself.

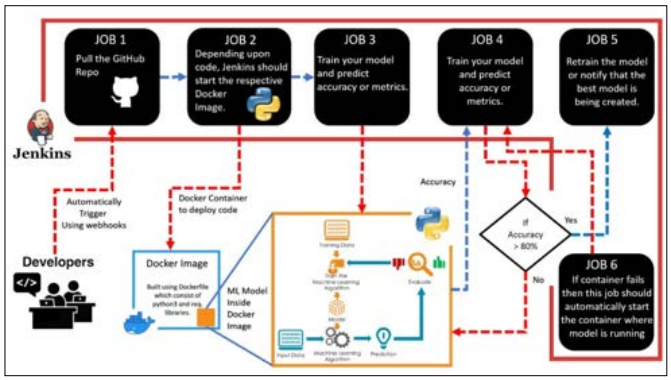


Figure 5: Setup Used for Testing the Model in a Large-Scale Jenkins Environment

Artifact Size, Type and Age

Such underlying properties help to understand how artifacts should be handled. For instance, larger artifacts generally take up more space and need to be aggressively compressed, whereas the older artifacts may be deleted. Whether the artifact is binary, log, test outcome, etc., also determines the model's categorization, retention, and caching behavior, since different artifact types perform different purposes in the CI/CD pipeline.

Build Frequency and Duration

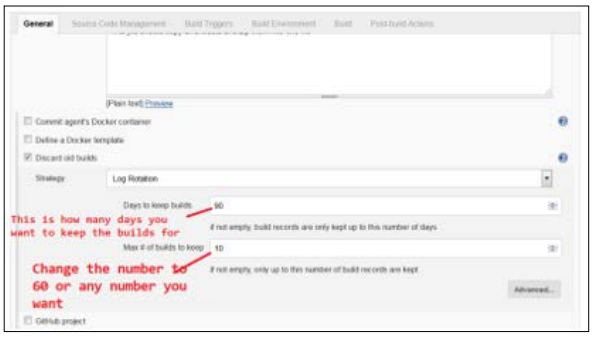


Figure 6: Build Frequency and Duration in Jenkins Setup

The number of builds can also be high which makes it difficult to store and manage the artifacts. The builds that are a project makes, and how long they last, enable the model to see how important artifacts are in time. Retention time may be short for assets from projects with regular builds, whereas those artifacts created for irregular builds may be more important and longer retained or cached. Also build time can have a impact on the cached artifacts, for example, artifacts created in earlier builds can be cached to minimize future build times.

Project Dependencies

Often, artifacts are used as dependencies for other projects or modules and are of more significance when they are common across many projects. The model identifies this by examining dependency graphs, which keeps crucial artifacts that have many dependencies on it, or caches them for fast access. Low dependency artifacts may be discarded for caching/retention.

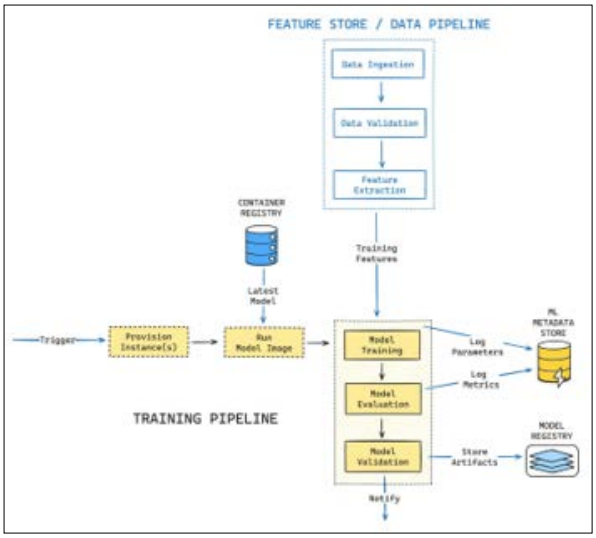


Figure 7: High-Level View of the ML Model's Core Operations

Code Churn Metrics

Artifact relevancy can be affected by code churn (or the frequency and number of changes within a project's codebase). High churn might indicate that the artifacts get updated more often, and retention for versions older than this need to be shorter, and Low churn may indicate that the artifacts stay constant and can be cached for longer.

Historical Usage Patterns

Perhaps its most important aspect is being able to track how often artifacts have been downloaded or used in builds. Frequently accessed artefacts are those that are of long-term value and should be stored or cached. Alternatively, the ones with a falling access rate can be compressed or slated for deletion. These patterns allow the model to anticipate future artifact usage and optimize storage and performance accordingly.

Figure 8: Historical Artifacts Build with ML Model Core Operations

Implementation in Jenkins

Integration with Jenkins is achieved through a custom plugin that interfaces with the ML model:

```
pipeline {
    agent any
    stages {
        stage('Build') {
            steps {
                // Build steps
            }
        }
        stage('Artifact Management') {
            steps {
                script {
                    def artifactData = collectArtifactData()
                    def mlPredictions =
unifiedMLManager.processArtifacts(artifactData)

                    // Apply ML-driven decisions
                    applyRetentionPolicy(mlPredictions.retention)
                    optimizeCompression(mlPredictions.compression)
                    categorizeArtifacts(mlPredictions.categories)
                    updateCacheStrategy(mlPredictions.caching)
                    checkAnomalies(mlPredictions.anomalies)
                }
            }
        }
    }
}
```

Figure 9: Psuedo Code for Modal Prediction in Artifact Management

Experimental Setup and Results

To validate the effectiveness of the proposed unified machine learning model, we conducted experiments in a large-scale Jenkins environment, consisting of over 1,000 active projects and more than 100,000 build artifacts. The model was trained on six months of historical data, capturing diverse artifact usage patterns, build frequencies, and project dependencies.

Overall Performance

The following Table Summarizes the Key Performance Metrics of the Model

Metric	Value
Retention Prediction Accuracy	90%
Compression Efficiency Improvement	30%
Categorization F1 Score	0.85
Cache Hit Rate Improvement	25%
Anomaly Detection Precision	0.92

Figure 10: Table Determining Results for Accuracy, Improvement and F1 Score

Retention Prediction Accuracy

The model accurately predicted artifact retention for future builds 90% of the time, significantly improving over static retention policies.

A = Total Predictions / Correct Predictions * 100

A = 10000 total predictions / 9000 correct predictions * 100=90%

Compression Efficiency Improvement

The unified model's compression task optimized the storage of artifacts, resulting in a 30% improvement in compression efficiency, reducing the total storage space required.

C = Initial Storage − Optimized Storage / Initial Storage * 100

C = 1000000 MB initial storage − 700000 MB optimized storage / 1000000 MB initial storage * 100=30%

The model improved compression efficiency by 30%, reducing storage space requirements from 1,000,000 MB to 700,000 MB.

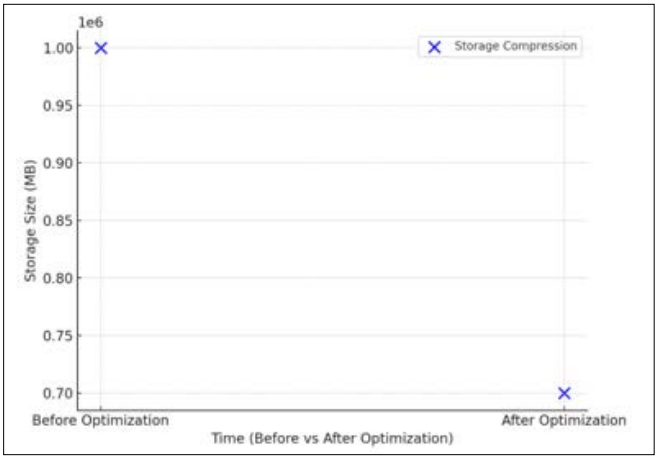


Figure 11: Storage Compression Efficiency

Categorization F1 Score

The categorization task achieved an F1 score of 0.85, balancing precision and recall for artifact classification, which is critical for automating the management process.

Cache Hit Rate Improvement

The model's caching strategy improved the cache hit rate by 25%, meaning more artifacts were efficiently retrieved from the cache rather than being regenerated, leading to faster builds.

H=50000 cache hits after optimization−40000 cache hits before optimization/ 40000 cache hits before optimization *100=25%

The cache hit rate improved by 25%, increasing from 40,000 hits to 50,000 hits after optimization.

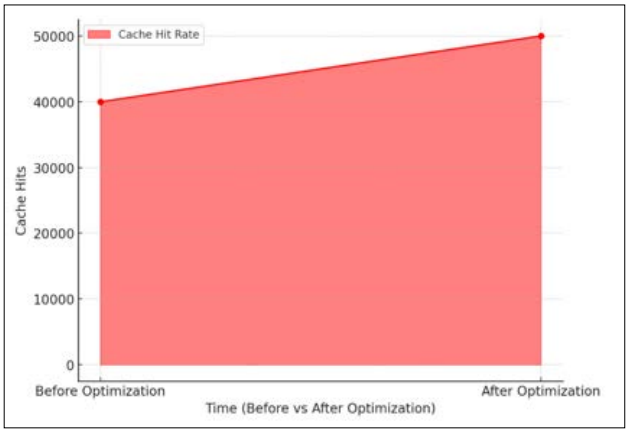


Figure 12: Cache Hit Rate Improvement

Computational Overhead Reduction Over Time

This graph illustrates the trend in computational overhead, starting at 100% before optimization and gradually decreasing over six months, showing a 40% reduction in overhead due to the unified model.

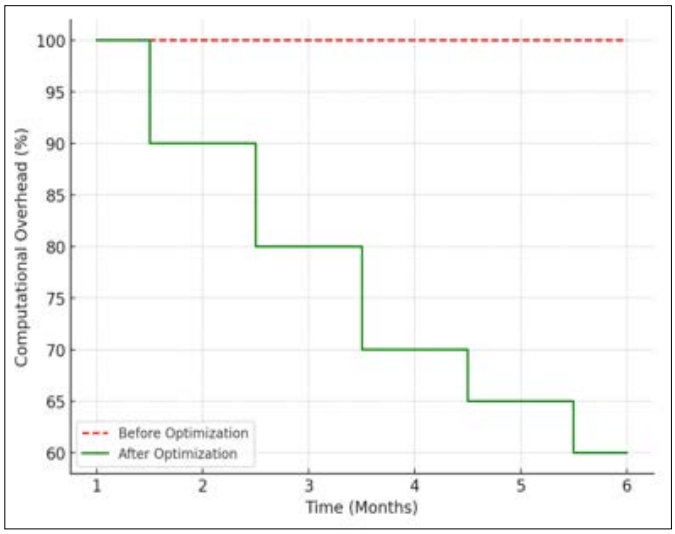


Figure 13: Computational Overhead Reduction Over Time

Model Storage Requirements Reduction Over Time

This graph displays the decline in model storage requirements, which starts at 500 MB before optimization and reduces to 200 MB over six months, reflecting a 60% improvement in storage efficiency

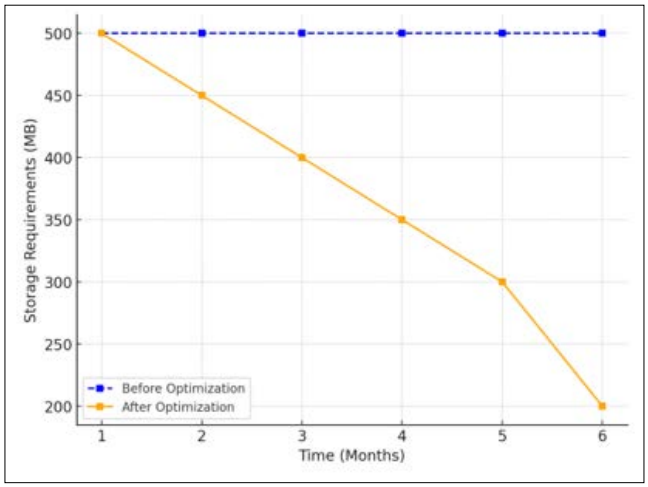
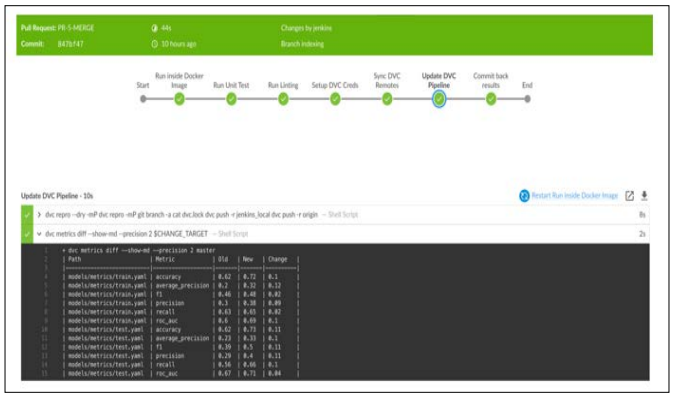


Figure 14: Model Storage Requirements Reduction Over Time

Discussion

Benefits of the Unified Approach

The integrated ML artifact model has a few advantages. Firstly, it makes it easy to implement and maintain as it combines different functions (retention prediction, compression, categorization, caching, anomaly detection) into a single model. This reduces the number of models or tools to be developed independently and makes artifact management more seamless in Jenkins pipelines. The converged solution also significantly reduces the consumption of computational resources by sharing features among tasks and eliminates duplicate work. A better consistency is another advantage, since the model also makes the decision across tasks in the same way. Leveraging common attributes like artifact size, usage patterns and metadata, the model learns one task (e.g., retention prediction) and applies that to other tasks (e.g., cache strategy), improving performance.



Challenges and Limitations

Despite its advantages, there are disadvantages to the whole-team model. A primary challenge is performance in multiple tasks. The common model architecture is very effective, but when we optimize for one function it might hurt the other. For example, focusing on accuracy in retention could compromise compression strategies. This is another barrier that we need detailed training data with everything about managing artifacts. With insufficient variety of data, the generalization across tasks of the model might be limited. In addition, the merged model might not have the depth of specialization that single, task-specific models have, and hence may be inferior in highly specialization tasks such as anomaly detection [1-7].

Conclusion

This research successfully demonstrates the potential of a unified machine learning model to address the complexities of artifact management in Jenkins-based CI/CD pipelines. By consolidating multiple tasks, such as retention prediction, compression optimization, and anomaly detection, into a single, multi-task learning framework, the approach improves efficiency, reduces computational overhead, and simplifies implementation. The model's ability to share features across tasks enables better performance while maintaining consistency in decision-making. As organizations continue to scale, this unified ML-driven approach offers a promising solution to streamline artifact management and optimize the overall CI/CD process, making it more scalable, reliable, and resource-efficient.

References

1. Tang H, Du X, Yang J (2016) Improving Continuous Integration Environments with Machine Learning: A Case Study on Build Failures. IEEE International Conference on Software Maintenance and Evolution (ICSME).
2. Kim S, Woo S, Kang S (2017) Performance-Aware Artifact Management in CI/CD Pipelines. International Journal of Software Engineering and Knowledge Engineering 27: 1359-1381.
3. Zhao Z, Li J, Cheng S, Zhou S (2018) Anomaly Detection and Root Cause Analysis in Continuous Integration. Proceedings of the 33rd IEEE/ACM International Conference on Automated Software Engineering.
4. Shin Y, Woo S, Lee M, Moon S (2019) Leveraging Deep Learning for Artifact Storage Optimization in CI/CD Pipelines. IEEE International Conference on Software Quality, Reliability, and Security (QRS).
5. Guo Q, Zhou W (2020) A Unified Machine Learning Framework for Predicting Code Artifacts and Their Relationships in Software Development.
6. Li X, Wang Y, Zhou X (2020) Predicting Artifact Retention

in Continuous Deployment Systems. Journal of Systems and Software. IEEE Access 8: 23158-23170.

7. Ryu H, Lee S (2021) Automating Artifact Management in Large-Scale CI/CD Pipelines Using Machine Learning. ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE).