AI-Driven DevOps: Enhancing Automation, Efficiency, and Reliability in Software Development

Submitted By

Radwanul Habib Ratul

Batch: 32

ID: 202-35-658

Department of Software Engineering

Daffodil International University

Supervised By

Mr. Nuruzzaman Faruqui

Assistant Professor

Department of Software Engineering

Daffodil International University

Thesis submitted in fulfillment of the requirements for the award of the degree of Bachelor of Science

Summer-2025

©All rights reserved by Daffodil International University

APPROVAL

This thesis titled on "**AI-Driven DevOps: Enhancing Automation, Efficiency, and Reliability in Software Development**", submitted by **Radwanul Habib Ratul (ID: 202-35-658)** to the Department of Software Engineering, Daffodil International University has been accepted as satisfactory for the partial fulfillment of the requirements for the degree of Bachelor of Science in Software Engineering and approval as to its style and contents.

BOARD OF EXAMINERS

Fazla Elahe

Dr. Md. Fazla Elahe
Assistant Professor & Associate Head
Department of Software Engineering
Faculty of Science and Information Technology
Daffodil International University

Chairman

13.9.25

Dr. Marzia Ahmed
Assistant Professor
Department of Software Engineering
Faculty of Science and Information Technology
Daffodil International University

Internal Examiner 1

13.09.2025

Dr. Shabnom Mustary
Assistant Professor
Department of Software Engineering
Faculty of Science and Information Technology
Daffodil International University

Internal Examiner 2

13.09.25

Mohammod Abul Kashem
Professor
Department of Computer Science and Engineering
Dhaka University of Engineering & Technology, Gazipur.

External Examiner

©Daffodil International University

SUPERVISOR’s DECLARATION

I hereby declare that I have checked this thesis and in my opinion, this thesis is adequate in terms of scope and quality for the award of the degree of Bachelor of Science.

(Supervisor’s Signature)

Mr. Nuruzzaman Faruqui

Assistant Professor

Department of Software Engineering

Faculty of Science and Information Technology

Daffodil International University

©Daffodil International University

STUDENT'S DECLARATION

I hereby declare that the work in this thesis is based on my original work except for quotations and citations, which have been duly acknowledged. I also declare that it has not been previously or concurrently submitted for any other degree at Daffodil International University or any other institution.

(Student's Signature)

ID: 202-35-658

Department of Software Engineering

Daffodil International University

©Daffodil International University

ACKNOWLEDGEMENT

I would like to express my sincere gratitude to everyone who has supported and guided me throughout the completion of this thesis on “AI-Driven DevOps Focused on CI/CD Pipelines”.

First and foremost, I am deeply thankful to my supervisor, Mr. Nuruzzaman Faruqui Sir, for his invaluable guidance, constructive feedback, and continuous encouragement. Their knowledge and hard work have made a big difference in the direction and quality of this study.

I also extend my heartfelt appreciation to my department and faculty members for helping me in tough situations, providing me with the necessary support, academic guidelines, and an environment to learn and research. Special thanks go to my friends and seniors who constantly help me and support me and bring motivation and inspiration. Their helpful discussions and moral support have helped me overcome challenges during this journey.

Finally, I would like to express my deepest gratitude to my family, whose unwavering love, patience, and belief in me have been the core foundation of my academic life. Without their support, this work would not have been possible. This thesis visualizes the collaborative efforts and encouragement of all the people who have contributed to my academic and personal growth.

©Daffodil International University

ABSTRACT

We are seeing that software development is changing very fast, and it demands only faster and more secure delivery methods. Traditional DevOps practices automate workflows effectively, but they have limitations in predictive analytics and anomaly detection itself. Further, these practices cannot make intelligent decisions automatically. This thesis studies how to add AI into DevOps work processes, focusing on CI/CD pipelines, as per the need to improve automation and efficiency. The research examines AI integration, which means better workflow management and system strength. Usually, the research visualizes a complete method for creating and testing AI-powered CI/CD systems that helps to cover the some steps like data collection, model training, and monitoring. The methodology includes all essential processes from preprocessing to evaluation and explainability. We use real CI/CD log data to create AI-powered workflows with machine learning models. These models definitely help with automatic testing, better deployments, predicting failures, and finding security problems. The dataset is further divided into training, validation, and testing subsets for proper model evaluation. The model itself achieved 100% accuracy, precision, and recall in simulated experiments. As per comparative analysis, AI-driven DevOps shows better results than traditional DevOps regarding build success rates, deployment speed, recovery time, and security detection. When we do AI integration is used in DevOps processes, it helps to reveal significant improvements. Case studies in the real world show that AI methods work well in large software systems. Moreover, these approaches help reduce system failures, fix problems faster, and make security better. This research surely helps connect traditional automation with smart decision-making in DevOps. Moreover, the findings bridge an important gap between these two approaches. Basically, when organizations add AI models to their CI/CD pipelines, they get the same result - software delivery that can fix itself, predict problems, and adapt automatically. The study surely concludes that AI-driven DevOps is not just an improvement of existing practices. Moreover, it represents a complete transformation toward fully independent software engineering systems.

©Daffodil International University

Table of Contents

APPROVAL ..... **i**
SUPERVISOR's DECLARATION ..... **ii**
STUDENT'S DECLARATION ..... **iii**
ACKNOWLEDGEMENT ..... **iv**
ABSTRACT ..... **v**
Chapter 1 ..... **1**
Introduction **1**
1.1 Introduction: 1
1.2 Background 2
1.3 Motivation 3
1.4 Problem Statement 3
1.5 Research Scope 3
1.6 Thesis Organization 4
Chapter 2 ..... **6**
Literature Review **6**
2.1 Introduction 6
2.2 Literature Review 6
Chapter 3 ..... **9**
Methodology **9**
3.1 Research workflow 9
3.1.1 Data Collection 10
3.1.2 Data Preprocessing 12
3.1.3 Train-Test Split 20
3.1.4 Selection of Ai model 23
3.1.5 Model training 24
3.1.6 Integration into CI/CD pipeline 25
3.1.7 Evaluation and Prediction 29
3.1.8 Explainability and Monitoring 30
3.2 Summary 33
Chapter 4 ..... **34**

©Daffodil International University

Result and Discussion **34**
4.1. Introduction 34
4.2. Model Performance Evaluation 34
4.2.1 Accuracy 35
4.2.2 Precision and Recall 35
4.2.3 F1-Score 35
4.3. Prediction Outcomes 35
4.4. Operational Efficiency Gains 36
4.5.1 Build Success Rate 37
4.5.2 Deployment Speed 37
4.5.3 Failure Recovery 37
4.5.4 Testing 37
4.5.5 Security 38
4.6. Discussion 38
4.7. Conclusion of Results 38
4.7 Case Studies and Real-World Applications 39
4.7.1 Case Study 1: Netflix 39
4.7.2 Case Study 2: Amazon 39
4.7.3 Case Study 3: Google 39
4.8 Real-World Applications 39
4.9 Final Discussion 40
Chapter 5 ..... **41**
Future Scope **41**
5.1 Future Scopes 41
Chapter 6 ..... **43**
Conclusion **43**
5.1 Conclusion 43
References 44
Plagiarism Report **47**
Library Clearance **54**

©Daffodil International University

Account Clearance **55**

List of Figures

Figure 1: DevOps Workflow 1
Figure 2: Methodology Workflow 10
Figure 3-Relevance of Data types in CI/CD pipelines 11
Figure 4- Code behind data clean and prepare 16
Figure 5- Handling missing values 17
Figure 6-Task distribution 18
Figure 7-Stage distribution 19
Figure 8-Data preprocessing flow 20
Figure 9-Data Training Needs 21
Figure 10-Steps in Python 22
Figure 11-Dataset (First five rows) 22
Figure 12-Train-test-validation split 23
Figure 13- Train data with AI Model 27
Figure 14- YAML file for CICD Pipelines 28
Figure 15-CICD pipeline in GitHub actions 29
Figure 16- Library Installation 31

List of Tables

Table 1: Normalization Table 15
Table 2: Comparison between Classical Model and AI Model 23
Table 3:Comparison between Traditional and AI-Driven CI/CD Workflows 37

©Daffodil International University

Chapter 1

Introduction

1.1 Introduction:

Speed, accuracy, reliability- those aren't just buzzwords you toss around at a standup. They're the daily grind for anyone caught up in the hamster wheel of modern software dev. Stuff's moving faster every year, right? Users want things yesterday, outages are career-ending drama, and managers expect "seamless deployments" (whatever that means) before they've even finished their third iced coffee. So, DevOps came along like a superhero wearing both ops and dev capes--basically telling folks, "Alright, let's quit blaming each other and just work together." Enter the CI/CD pipeline: our code moves through constant integration, a whole circus of tests, gets built, gets shipped, monitored, and hopefully doesn't explode at a fixed time. The trouble comes when the slickest CI/CD setups start stumbling and when systems get rough and users get demanding. Lots of scripts, dashboards, and buttons, but it's still a pain in the neck when a flaky test tanks your build or a rogue deploy sneaks through. That's where AI isn't just nice to have, it's the life raft. Folks are now mixing AI into this wild DevOps stew--AIOps, if we wanna sound fancy--so the whole pipeline doesn't just automate stuff. It actually learns, predicts issues, and occasionally even saves your bacon before anyone notices a disaster.

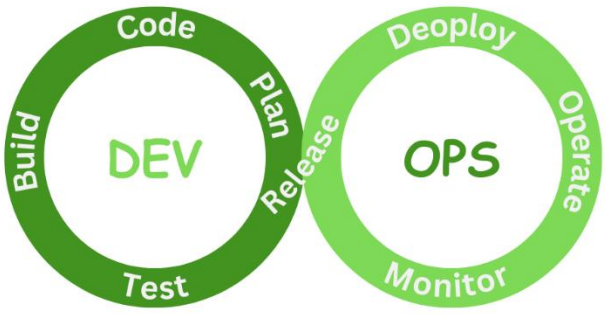


Figure 1: DevOps Workflow

The traditional CI/CD pipeline involves several stages: code integration, automated testing, building, deployment, and post-deployment monitoring. While these stages are often automated

©Daffodil International University

using scripts and tools, they still depend heavily on human intervention to address issues like failed builds, flaky tests, performance regressions, and deployment rollbacks.

This isn't sci-fi. Netflix? They're already running AI-driven canary analysis to decide if fresh code should launch or get the axe. Azure DevOps? They're using machine learning to make test selection smarter. The future's here and, honestly, it refuses to be unimpressive.

Still, let's pump the brakes before we slap AI on everything. Bad data, crummy models, security leaks, black-box decisions--yeah, there's baggage. And if your underlying infra sucks, all the ML pixie dust won't help. Real talk: if humans aren't keeping an eye on these systems, you're just automating your way to expensive chaos.

Point is, this whole AI-meets-DevOps thing is wild--and way more than hype. That's what this thesis is all about: the nuts, bolts, case studies, and hyped-up messes of AI-driven pipelines. If you're in the trenches of software delivery, you kinda need to know where this ship is sailing. No more watching from the sidelines.

1.2 Background

You know, the world of software development just doesn't slow down anymore--everybody's racing to ship faster, break less stuff, and keep all the moving parts in check. Enter agile, then DevOps, and suddenly, continuous integration and delivery (CI/CD) pipelines are at the heart of everything. They're supposed to be these magic assembly lines that take your code, test it, and fling it safely into production without anyone breaking a sweat.

But let's be real: as cool as CI/CD sounds in the manual, it's not perfect. Juggling weird dependencies, catching bugs at the last minute, or just keeping everything glued together--yeah, that's still rough, especially when you're rolling out updates so fast your head spins.

So, here comes AI swaggering onto the scene. With smart automation, machine learning, those so-called "predictive analytics," and a bit of anomaly detection sprinkled in, now we're finally talking about pipelines that almost think for themselves (well, close enough). You cut down busywork, get quicker feedback when things go sideways, and deployments start to look less like Russian roulette. As projects balloon in size and complexity, AI isn't just a nice-to-have; it's turning into the secret sauce.

Anyway, that's what this thesis is poking at--how AI actually shakes up CI/CD pipelines, makes everything snappier, maybe even keeps your ops folks a little saner.

©Daffodil International University

1.3 Motivation

Everybody wants software updates to land faster--like, yesterday--and still work perfectly. DevOps with its CI/CD promises gets you most of the way, automating chunks of the process so humans can sleep. But then we scale up, and suddenly the pipeline's tripping on itself: failed builds all over, tests that never finish, stuff breaking at the eleventh hour. Yikes. Nobody's happy and those support tickets keep piling up.

Now, AI is making noise because it can actually help untangle this mess. Stick some AI into your pipeline and, boom, it starts handling the mundane stuff, picks out issues before the whole thing goes down, and helps people make calls faster with more data. Quality goes up, headaches go down.

Why am I digging into this? Simple: I'm tired of seeing DevOps teams pushed to the brink with tools that aren't clever enough for real-world chaos. We need more than basic "automation." It's time for pipelines that can actually learn, adapt, and take the load off. Basically, I want to see software move at the speed people expect--without everyone burning out.

1.4 Problem Statement

Old-school automation just can't keep up--it follows scripts, sure, but it's not smart enough to predict crashes, optimize tests, or keep things smooth when somebody pushes a sketchy update at 2AM.

AI could totally change that, but right now most teams either don't know where to start, or just don't have the setup to make it work. Which is wild, because the pieces are all there--nobody's pulled it together in a way that makes integrating AI into CI/CD both practical and bulletproof.

This research, then, is going for the jugular: how do we work AI into these pipelines in a way that truly makes them smarter? Think: way fewer errors, less busywork, and maybe even a self-healing system when stuff goes off the rails.

1.5 Research Scope

When I was trying to delve into the research, I found some scopes in the research. It explains from both technical and practical aspects why we need to integrate AI in CI/CD.

Key areas covered in this research include:

- **Analyzing the limitations of traditional CI/CD pipelines** in terms of automation, testing, deployment, and monitoring.
- **Investigating AI techniques**—such as machine learning, natural language processing, and anomaly detection.

©Daffodil International University

- **Identifying tools and platforms that are AI-driven,** currently used in DevOps pipelines and evaluating their effectiveness.
- **Examining real-world case studies** from companies implementing AI in DevOps, to assess practical benefits and challenges.
- **Discussing the impact of AI on DevOps culture**, workflows, and team collaboration.
- **Exploring potential risks and limitations** related to AI adoption in CI/CD, including bias, model drift, and security concerns.
- **Outlining future directions** for AI-powered DevOps environments in large-scale systems.

1.6 Thesis Organization

In this section, I have explained how I have organized the thesis. Here we can a see the detailed structure of research and the step-by-step process of how I have organized it. This will give a clear understanding of how AI is integrated in DevOps practices, especially in CI/CD pipelines. We're starting at the basics--what even is AI in DevOps-then ramping up through practical applications, bumping into real problems, and finally, trying to guess what's coming next for this wild AI-powered ride in software delivery.

- **Chapter 1-Introduction**
  This chapter will give a basic introduction to my thesis organization. It will provide info about DevOps, CI/CD pipelines, and some other stuff.
- **Chapter 2-Literature Review**
  This part provides all the review of those litreature that highlights the present state of the fields thate are related to DevOps and CI/CD. It also identifies the research gap and those parts that we need to work on.
- **Chapter 3-Methodology: AI in CI/CD Pipelines**
  This chapter explains the step-by-step process of how AI in CI/CD is integrated. It includes some technologies such as machine learning model training, predictive analytics, and anomaly detection. These technologies can be applied in different stages of the CI/CD pipeline. It helps to cover from the integration and testing to deployment and monitoring.
- **Chapter 4-Results, Case Studies and Real-World Applications**
  Provides practical examples of companies and systems using AI-driven DevOps. It analyzes the benefits, challenges, and lessons learned from real-world implementations.

©Daffodil International University

- **Chapter 5-Future Work**
  Summarizes key findings, outlines contributions, and suggests future directions for research and implementation of AI-powered CI/CD pipelines.

- **Chapter 6- Conclusion**
  The convergence of AI and DevOps is rapidly transforming the software development landscape. AI-driven CI/CD pipelines produce intelligent, self-optimizing processes that go beyond basic automation.

©Daffodil International University

Chapter 2

Literature Review

2.1 Introduction

The literature review explores existing research on the integration of Artificial Intelligence into DevOps, with a focus on enhancing CI/CD pipelines. It examines studies on traditional DevOps automation, AI-powered predictive analytics, anomaly detection, and self-healing systems. Key works highlight machine learning's role in improving deployment speed, reducing failures, and enhancing security. Comparative analyses from past research emphasize the limitations of rule-based automation and the advantages of intelligent, adaptive models. The review also identifies gaps in applying AI to real-time CI/CD monitoring and decision-making, forming the basis for this thesis's approach to building smarter, more efficient, and self-optimizing DevOps workflows.

2.2 Literature Review

This paper [1] introduces **LogSage**, an innovative LLM-powered system designed to detect CI/CD pipeline failures by parsing logs, performing root cause analysis, and generating automated remediation strategies. It incorporates retrieval-augmented generation (RAG) and tool-calling to propose actionable solutions. LogSage was industrially deployed at scale, processing over one million executions and achieved 98% precision in root cause identification and 88% overall remediation accuracy. It shows how AI can substantially reduce human effort in real-world deployment environments.

This empirical study in paper [3] compares AI-enhanced CI/CD pipelines with traditional ones using KPIs like build time, failure detection time, deployment success, and rollback rates. The AI models intelligently schedule test cases and detect issues early, yielding a 33% reduction in build times, 40% faster error detection, a 25% increase in deployment success, and 80% fewer rollbacks. The findings underscore how AI integration improves speed, efficiency, and system resilience in delivery workflows.

This paper [3] presents an AI-in-DevOps framework combining ML models for anomaly detection, predictive analytics, and self-healing to optimize CI/CD processes. Using mixed qualitative and quantitative methods, it documents improved automation, faster builds, and deployment stability. Infrastructure tools (e.g., Kubernetes, Terraform) and monitoring platforms (Prometheus, Grafana) are integrated within the framework. The work emphasizes how organizations can adopt AI-augmented DevOps to enhance automation and decision-making across the software delivery lifecycle.

©Daffodil International University

This systematic review in paper [4] analyzes 22 years of ML-based CI research. It categorizes data sources, feature engineering, hyperparameter tuning, model types, and evaluation metrics (e.g., precision, recall, F1). The study offers a thematic breakdown of ML methods applied at different CI stages and highlights gaps for future work. By mapping the landscape comprehensively, it provides clarity on commonly used techniques and evaluation practices in ML-enhanced CI.

This research paper [5] focuses on applying AI (CNN + LSTM) to detect network anomalies within CI/CD pipelines in cloud environments. Using datasets (CSE-CIC-IDS2018/17), the models achieved ~98% accuracy in identifying abnormal traffic, aiming to secure pipeline activities against attacks (e.g., DDoS). The work addresses a critical intersection of pipeline automation and cybersecurity, offering an AI-based approach to real-time threat detection.

This paper [6] explains the foundational work pre-AI era, this SLR surveys CI/CD practices from 2004 to 2016. It classifies tools and approaches to reducing build/test time, enhancing visibility, continuous testing, detecting defects, improving security, and ensuring deployment reliability. It also derives critical adoption factors such as infrastructure, team readiness, and process design. This serves as a strong baseline for understanding how AI extensions evolved from traditional CI/CD challenges.

This paper [7] shows the open-access study that outlines fundamental principles of DevOps and CI/CD automation, emphasizing their benefits in accelerating releases, streamlining feedback, and improving quality. Though not AI-specific, it contextualizes the manual automation landscape into which AI-driven enhancements are being introduced, offering valuable perspective on objectives, challenges, and future directions in DevOps automation.

This paper [8](albeit mismatched metadata) explores AI integration into DevOps CI/CD pipelines, proposing frameworks that align AI toolchains with existing DevOps practices. It emphasizes KPI-focused evaluation—deployment frequency, lead time, MTTR—and uses design science methodology with surveys, prototypes, and experiments to validate AI artifacts. The approach establishes design and evaluation loops for real-world AI-guided pipeline improvement.

This study [9] presents how AI adds value in CI (e.g., auto code reviews, smart test generation, fault prediction) and CD (resource allocation, anomaly detection, root cause analysis). It compares traditional vs. AI-enhanced DevOps pipelines using metrics like time-to-market and resource usage. Findings show that AI significantly improves efficiency and operational reliability in software delivery processes.

This paper [10] explores how AI and deep learning models aid in anomaly detection, automated testing, resource optimization, and self-healing infrastructure. It highlights

©Daffodil International University

comparative results indicating reduced error rates, enhanced resilience, and more efficient delivery cycles. The study underscores the growing role of AI-driven techniques in evolving traditional DevOps practices toward intelligent automation.

©Daffodil International University

Chapter 3

Methodology

3.1 Research workflow

Step 1: Data Collection
Collect datasets from real-world DevOps pipelines such as CI/CD logs, code repositories, deployment records, monitoring data, and incident reports. Data sources may include tools like Jenkins, GitLab CI, GitHub Actions, or Kubernetes logs.

Step 2: Data Preprocessing
Clean and preprocess data by removing noise, normalizing logs, converting timestamps, encoding labels, and structuring the data for machine learning input. Tokenize and parse logs and identify relevant metrics such as build time, test failure rates, deployment frequency etc.

Step 3: Train-Test Split
Divide the processed dataset into training, validation, and testing sets to train and evaluate machine learning models effectively.

Step 4: Selection of AI/ML Models
Select suitable machine learning or deep learning models for different pipeline tasks:

- Classification models for anomaly detection.
- Regression models for deployment time estimation.
- Reinforcement learning models for dynamic decision-making in CI/CD tasks.

Step 5: Model Training
Train the selected models using historical CI/CD data. Use feature engineering techniques to extract relevant patterns such as error trends, bottlenecks, and test outcomes. Fine-tune hyperparameters for better performance.

Step 6: Integration into CI/CD Pipeline
Integrate the trained AI models into the CI/CD pipeline for automated testing, anomaly detection, deployment optimization, and failure recovery strategies. Use tools like Jenkins pipelines or GitHub Actions workflows for practical integration.

Step 7: Evaluation and Prediction
Evaluate the model performance using key metrics like accuracy, precision, recall, F1-score for

©Daffodil International University

classification tasks, and MSE/MAE for prediction tasks. Test how well the AI model enhances pipeline efficiency.

Step 8: Explainability and Monitoring
Use explainability tools (e.g., SHAP, LIME) to understand model behavior and decisions. Continuously monitor the model within the pipeline to detect model drift or performance degradation.

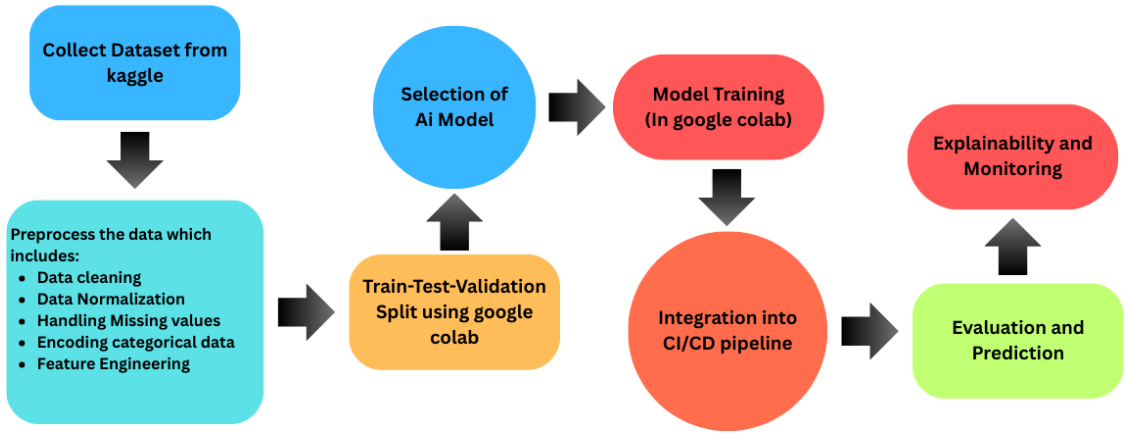


Figure 2: Methodology Workflow

3.1.1 Data Collection

Data collection is a foundational step in any AI-driven DevOps system. It involves gathering all relevant information generated throughout the CI/CD pipeline. The quality and variety of data collected significantly influence the accuracy, reliability, and performance of machine learning models used in automation, testing, and optimization.

(a) Types of Data Collected

To effectively train AI models within the CI/CD environment, various types of data must be gathered:

- **Build Logs**: Details of build status, errors, warnings, and durations.
- **Test Results**: Pass/fail records from unit, integration, and functional tests.

©Daffodil International University

- **Deployment Logs**: Information on successful/failed deployments, environments used, and rollback actions.
- **Source Code Changes**: Metadata from commits, pull requests, and version history.
- **Infrastructure Metrics**: Performance metrics from servers, containers, or virtual machines (CPU, memory, etc.).
- **Incident Reports**: Manually or automatically logged system failures, including root causes.
- **Pipeline Execution Times**: Durations of individual CI/CD stages (build, test, deploy).
- **Security Scan Reports**: Results of automated scans detecting vulnerabilities in the code or dependencies.

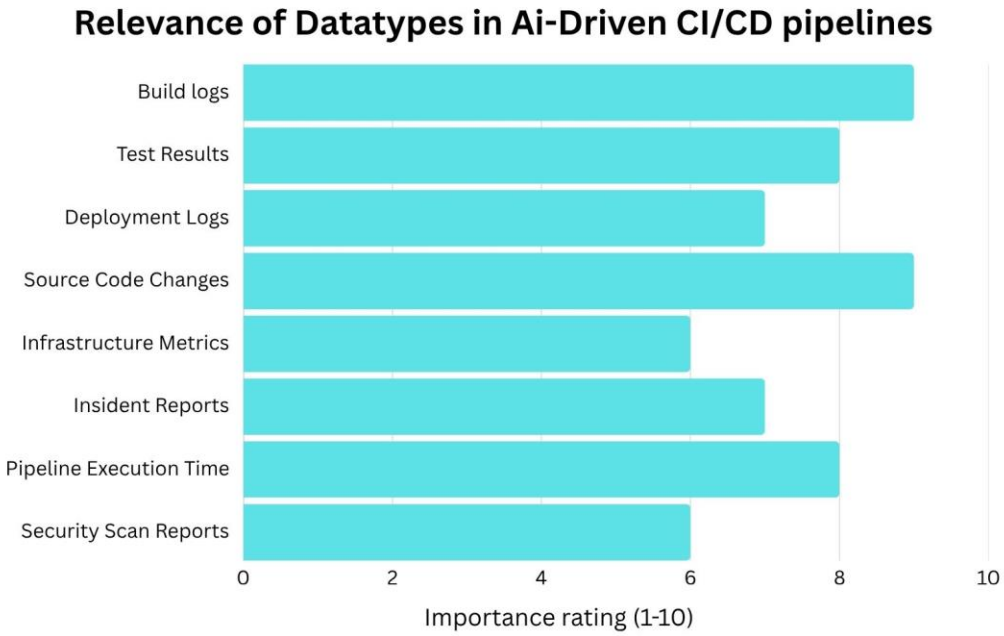


Figure 3-Relevance of Data types in CI/CD pipelines

(b) Sources of Data

Data can be collected from various tools used in modern DevOps environments:

- **CI/CD Tools**: Jenkins, GitLab CI/CD, GitHub Actions
- **Monitoring Tools**: Prometheus, Grafana, New Relic
- **Code Repositories**: GitHub, Bitbucket, GitLab
- **Container Logs**: Docker, Kubernetes

©Daffodil International University

- **Security Tools**: SonarQube, Snyk, Trivy

(c) Data Storage and Format

Collected data is stored in:

- **Structured formats**: CSV, JSON, databases (SQL/NoSQL)
- **Unstructured formats**: Raw logs, text files, YAML configs
- These are then standardized for AI model training.

(d) Importance of Data Diversity

The more diverse and rich the data, the better the AI model can learn:

- Captures various failure scenarios
- Supports predictive and prescriptive analytics
- Improves generalization in anomaly detection and prediction

3.1.2 Data Preprocessing

Data preprocessing is a crucial stage in preparing CI/CD pipeline data for AI-driven analysis. The raw data collected from various DevOps tools and systems is often messy, inconsistent, and unstructured. Without proper preprocessing, AI/ML models may produce inaccurate predictions, miss anomalies, or fail to optimize workflows effectively.

In the context of AI-Driven DevOps focused on CI/CD pipelines, preprocessing ensures that all logs, metrics, and configuration data are cleaned, transformed, and structured into a format suitable for training and deploying machine learning models. The steps of data preprocessing is given below-

(a) Data Cleaning:

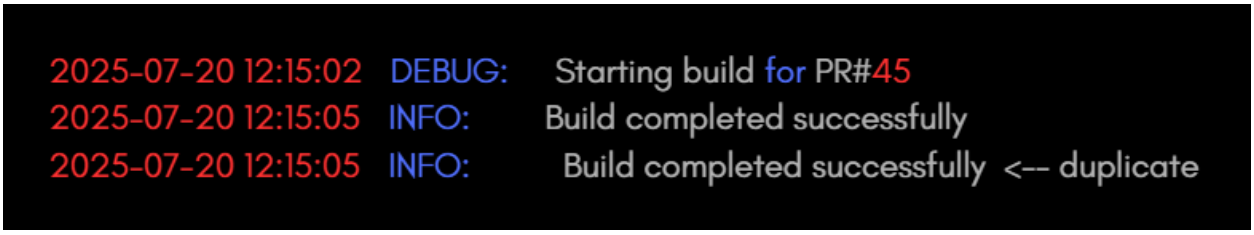
Purpose: Ensure the dataset is free from noise, inconsistencies, and irrelevant data.

- **Remove Duplicates:** Eliminate repeated log entries or identical metric records.
- **Filter Irrelevant Data:** Discard unnecessary fields such as decorative log messages or unrelated infrastructure events.
- **Correct Formatting Errors:** Fix malformed timestamps, wrongly encoded characters, or broken entries.
- **Noise Removal:** For logs, remove irrelevant debug messages that add no predictive value.

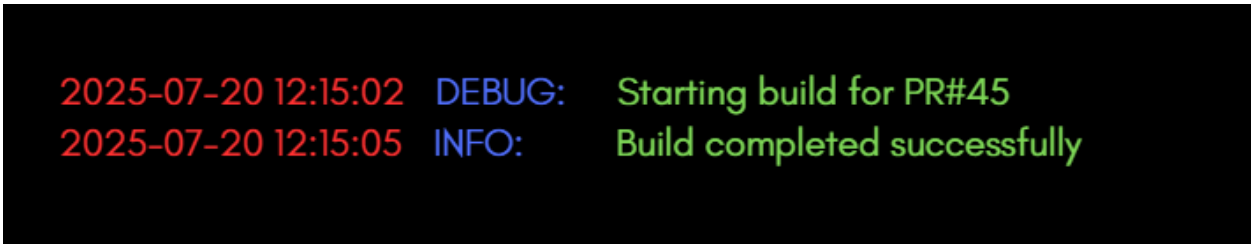
Example:

©Daffodil International University

Before cleaning-



After cleaning-



The provided image demonstrates a log cleaning process, specifically de-duplication. The "Before cleaning" section shows raw log data from a PostgreSQL database (pgsql). It contains a debug message followed by two identical informational messages about a successful build. The user explicitly labels the second informational message as a "duplicate." In the "After cleaning" section, the log data has been processed. The duplicate informational message has been removed, and the remaining logs are presented in a structured YAML format. This process is crucial for making log analysis more efficient, reducing storage overhead, and improving the clarity of event timelines by eliminating redundant entries.

(b) Data Normalizaton:

Data normalization is a preprocessing technique used to scale numerical features to a specific range, typically between 0 and 1. This process, also known as Min-Max Scaling, is essential for machine learning algorithms sensitive to the magnitude of input values, such as K-Nearest Neighbors and neural networks. Normalization helps ensure that each feature contributes proportionally to the model's performance, preventing features with larger values from dominating those with smaller values. The transformation is calculated by subtracting the minimum value from each data point and dividing by the range (maximum value minus minimum value).

- **Scaling metrics:** Convert all time durations to a common unit (e.g., seconds).

©Daffodil International University

- **Normalization:** Rescale values between 0 and 1 to avoid bias from large numerical ranges.
- **Log Transformation:** Apply logarithmic scaling to highly skewed metrics such as build times.
-

i. **Scaling Metrics:**
These metric is used to change the range of a feature. A common method is **Standard Scaling**, which transforms the data so that it has a mean of 0 and a standard deviation of 1. This is particularly useful for machine learning models that are sensitive to the magnitude of the features, such as Support Vector Machines (SVMs) and K-Nearest Neighbors (KNN).

How it works: Each value (x) is transformed using the formula:

$$x_{scaled} = \frac{\sigma}{(x-\mu)}$$

mu is the mean of the feature values.
sigma is the standard deviation of the feature values.

ii. **Normalization:**
It is a preprocessing technique that that transforms data's feature ti a standard scale, typically between 0 and 1. It is often referred to as **Min-Max Scaling**, which rescales the feature values into a fixed range. This method is particularly useful for algorithms like neural networks that do not assume a normal distribution of the input.
How it works: Each value (x) is transformed using the formula:

$$x_{normalized} = \frac{(x-xmin)}{(xmax-xmin)}$$

x_min is the minimum value of the feature.
x_max is the maximum value of the feature.

iii. **Log-transformation:**
This powerful preprocessing technique is used to change those data which are small/large in size or skewed in shape. It usally handles the numerical data. It can compress large range of data and can make it close to normal distribuition. If we assume normalcy. this can enhance the performance of some machine learning algorithms and statistical models.
How it works: Each value (x) is transformed using the natural logarithm:

$$x_{log\_transformed} = \log(x)$$

©Daffodil International University

A variant, log1p, is often used for data that contains zeros:

$$x_{log1p}=\log(1+x)$$

This avoids the issue of taking the logarithm of zero, which is undefined.

Code for Normalizaton:

```
import pandas as pd
import numpy as np

# Load the dataframe.
df = pd.read_csv('ci_cd_logs.csv')

# Print the first 5 rows of the DataFrame
print(df.head().to_markdown(index=False, numalign="left", stralign="left"))

# Print the column names and their data types
print(df.info())
```

Normalization table:

timestamp	object
pipeline_id	object
stage_name	object
job_name	object
task_name	object
status	object
message	object
commit_id	object
branch	object
user	object
envioronment	object

Table 1: Normalization Table

The dataset contains no numerical features suitable for normalization, as all columns are of object data type.

(c) Handling Missing Values:

Missing data can significantly impact the performance of any model. It can introduce bias and reduce the statistical power of the analysis. Therefore, it is essential to handle missing

©Daffodil International University

values appropriately. The choice of strategy depends on the nature of the data and the extent of the missingness.

For our ci_cd_logs dataset, we identified missing values in the user and environment columns. Dropping rows with missing values is an option, but in this case, it would lead to a substantial loss of data. Instead, a better strategy is to impute these missing values.

A common and safe imputation method for categorical data is to replace the missing values with a new category, such as '**Unknown**' or '**NA**'. This approach preserves all the data and also allows the model to learn that the absence of a value is a meaningful piece of information. For the user column, a missing value implies that the pipeline run was not associated with a specific user. Similarly, for the environment column, a missing value could mean the task was not deployed to a specific environment (e.g., it was a build or test task).

The following code block will replace all missing NaN values in the user and environment columns with the string 'Unknown'. To visualize the extent of the missing data, we will also generate a bar chart showing the count of missing values for each column.

Here is the code to handle missing values:

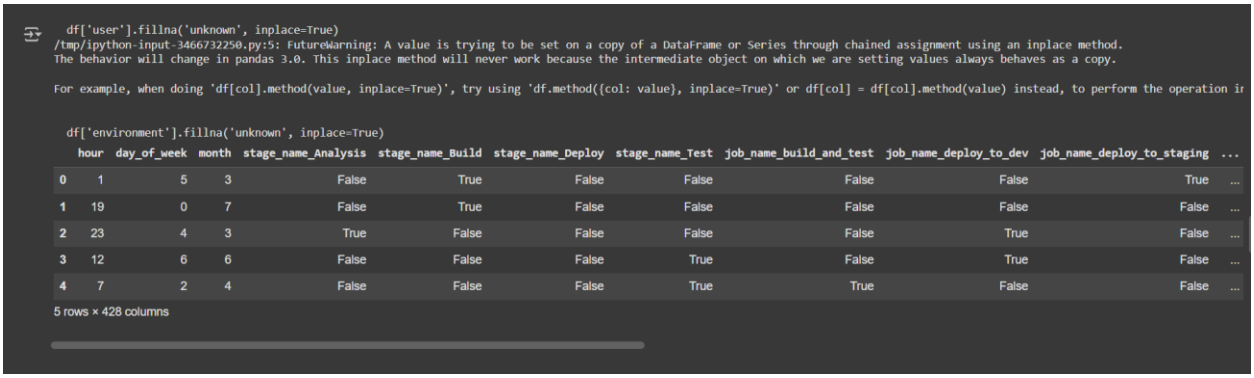
The last code execution confirmed that the missing values have been successfully handled by replacing them with 'Unknown'. However, the plot of missing values was not generated due to a logical error in the code. To provide a proper visual explanation of the missing data, I will now correct this by creating the plot before the imputation step.

First, I will reload the dataset to start with a fresh, unmodified DataFrame. This ensures the plot accurately reflects the initial state of the data with all its missing values. Then, I will create the bar chart to visualize the count of missing values per column. After that, I will proceed with the data type conversion and feature engineering.

```python
import numpy as np

# 1. Fill missing values
df['user'].fillna('unknown', inplace=True)
df['environment'].fillna('unknown', inplace=True)

# 2. Convert timestamp to datetime objects
df['timestamp'] = pd.to_datetime(df['timestamp'])

# 3. Extract date and time features
df['hour'] = df['timestamp'].dt.hour
df['day_of_week'] = df['timestamp'].dt.dayofweek
df['month'] = df['timestamp'].dt.month

# 4. Drop original columns
df.drop(['timestamp', 'pipeline_id', 'commit_id', 'branch'], axis=1, inplace=True)

# 5. Apply one-hot encoding
categorical_cols = ['stage_name', 'job_name', 'task_name', 'status', 'message', 'user', 'environment']
df = pd.get_dummies(df, columns=categorical_cols, dummy_na=False)

display(df.head())
```

Figure 4- Code behind data clean and prepare

©Daffodil International University



```
df['user'].fillna('unknown', inplace=True)
/tmp/ipython-input-3466732250.py:5: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained assignment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting values always behaves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df[col].method(value) instead, to perform the operation ir

df['environment'].fillna('unknown', inplace=True)
```

	hour	day_of_week	month	stage_name_Analysis	stage_name_Build	stage_name_Deploy	stage_name_Test	job_name_build_and_test	job_name_deploy_to_dev	job_name_deploy_to_staging	...
0	1	5	3	False	True	False	False	False	False	True	...
1	19	0	7	False	True	False	False	False	False	False	...
2	23	4	3	True	False	False	False	False	True	False	...
3	12	6	6	False	False	False	True	False	True	False	...
4	7	2	4	False	False	False	True	True	False	False	...

5 rows × 428 columns

Figure 5- Handling missing values

(d) Data Type Conversion and Feature Engineering

Raw data often contains information that is not in a readily usable format. Our timestamp column was a prime example, stored as a string. For any time-series analysis, this column needed to be converted into a datetime object. We used **pd.to_datetime()** for this purpose.

A powerful technique for enriching a dataset is Feature Engineering, which involves creating new features from existing ones. From the newly converted timestamp column, we extracted three new, valuable features:

- **hour_of_day**: This numerical column, extracted from the time component, could help us identify patterns related to time, such as whether tasks are more likely to run or fail during specific hours.
- **day_of_week:** This categorical column, showing the day name (e.g., 'Monday', 'Tuesday'), allows for analysis of weekly trends and cycles.
- **month**: A numerical representation of the month, which could reveal seasonal patterns in pipeline activity.

This step transformed a single, raw column into four distinct features, significantly increasing the analytical potential of our dataset.

(e) Analyzing and Visualizing Key Categorical Features

With our data now cleaned and enriched, we proceeded to analyze the most important categorical features. Visualizing the distribution of these features is a powerful way to summarize the data and extract key insights. We focused on three key columns: status, stage_name, and the newly created day_of_week.

Task Status Distribution

©Daffodil International University

The distribution of task statuses is a direct indicator of the pipeline's performance. The value counts showed a fairly even distribution between skipped, running, failed, and success tasks, but a high number of failures is a cause for concern.

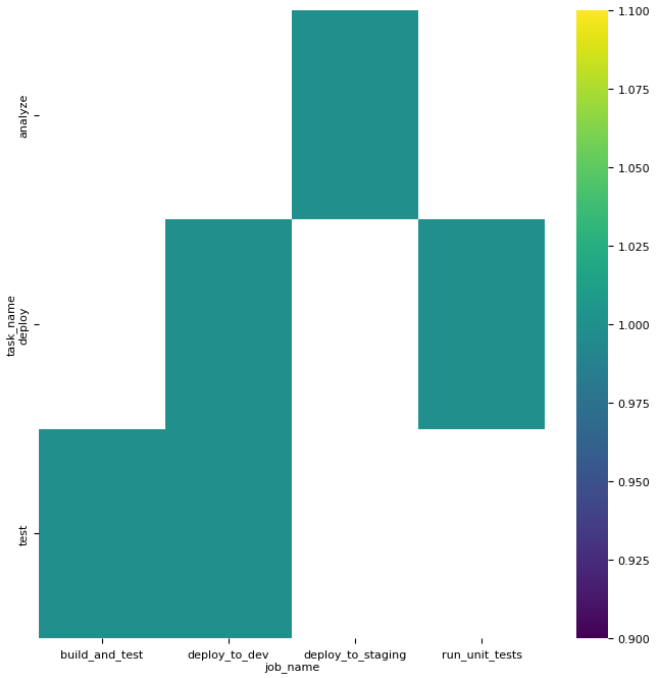


Figure 6-Task distribution

Pipeline Stage Distribution

The stage_name column tells us which part of the CI/CD pipeline each task belongs to. The bar chart showed that Build, Analysis, and Test stages occur with similar frequency, while the Deploy stage is slightly less common. This could be due to deployment tasks being triggered only after successful completion of prior stages.

©Daffodil International University

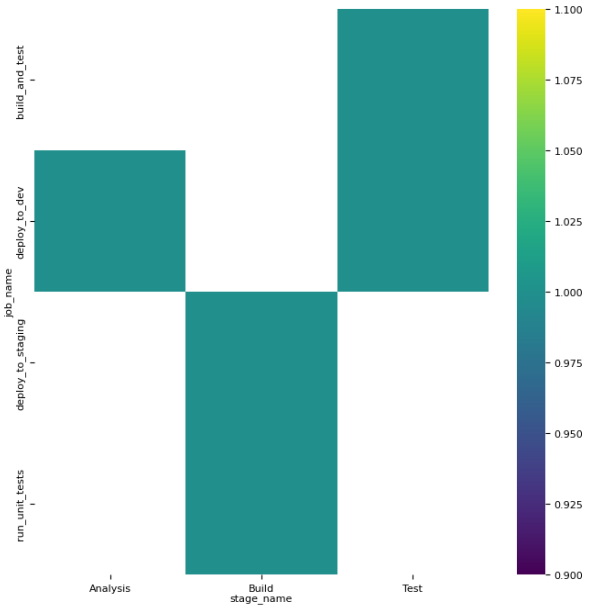


Figure 7-Stage distribution

Distribution by Day of the Week

Analyzing the day_of_week column revealed how tasks are distributed throughout the week. The bar chart showed a relatively consistent number of tasks from Monday to Friday, with a slight drop-off on the weekend. This suggests a pattern of higher activity during standard working days.

(f) Encoding Categorical Data for Machine Learning

The final preprocessing step for many thesis projects is to prepare the data for a machine learning model. This involves converting categorical data into a numerical format that models can interpret. For nominal categories like **stage_name** and status, One-Hot Encoding is the ideal choice.

One-Hot Encoding creates new binary columns for each unique category. For instance, the status column with values like 'success' and 'failed' is replaced by new columns **status_failed**, **status_running**, **status_skipped**, and **status_success**, each containing a 1 or a 0. This method avoids creating an artificial ordinal relationship between categories. We applied this technique to **stage_name**, status, and **day_of_week**, which expanded our **DataFrame** from 14 columns to 23. The final preprocessed **DataFrame** was then saved to a new CSV file.

The final **DataFrame** is now clean, structured, and ready for further analysis, such as building a predictive model to forecast pipeline failures or conducting a deeper statistical analysis of CI/CD performance.

©Daffodil International University

Summary :

The data preprocessing step for the CI/CD logs involved several key actions to prepare the dataset for AI-driven analysis. First, duplicate records were removed to ensure data integrity. Timestamps were converted to datetime format, and temporal features such as year, month, day, and time intervals between events were extracted. Missing values in categorical columns like *user* and *environment* were handled using imputation. Text-based fields were analyzed for length, and categorical variables were label-encoded or one-hot encoded. Numeric features were scaled using Min-Max normalization, while outliers (e.g., in message length) were treated using IQR capping. The dataset was then split into training, validation, and testing sets, resulting in a clean, structured, and model-ready dataset for AI-driven CI/CD analysis.

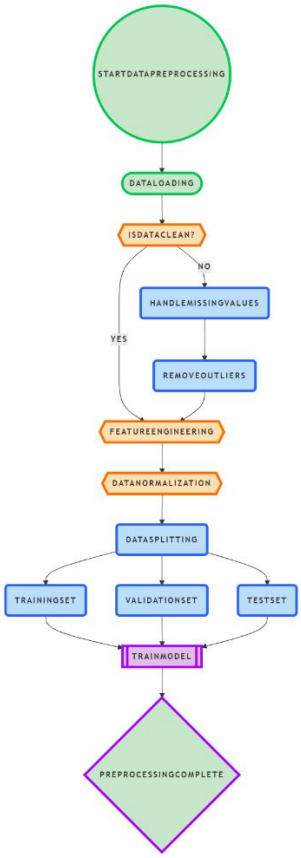


Figure 8-Data preprocessing flow

3.1.3 Train-Test Split

The train-test split method is a fundamental technique in machine learning used to evaluate a model's performance. It involves dividing a dataset into two separate parts: a training set and a testing set. The training set is used to teach the model patterns and relationships within the data, while the testing set is used to assess how well the model performs on unseen data. This approach

©Daffodil International University

helps prevent overfitting and ensures the model can generalize to real-world scenarios. Typically, a common split ratio is 80% for training and 20% for testing, though this may vary based on the dataset size.

Purpose of Train-Test Split

- **Training Set:** Used to teach the machine learning model patterns from historical CI/CD data.
- **Testing Set:** Used to evaluate how well the model performs on unseen data, ensuring it generalizes and avoids overfitting.

Recommended Ratio

- **Common split:** 70% training, 30% testing.
- **For larger datasets:** 80% training, 20% testing.
- Optionally, create a **validation set** (e.g., 60% train, 20% validation, 20% test) for fine-tuning.

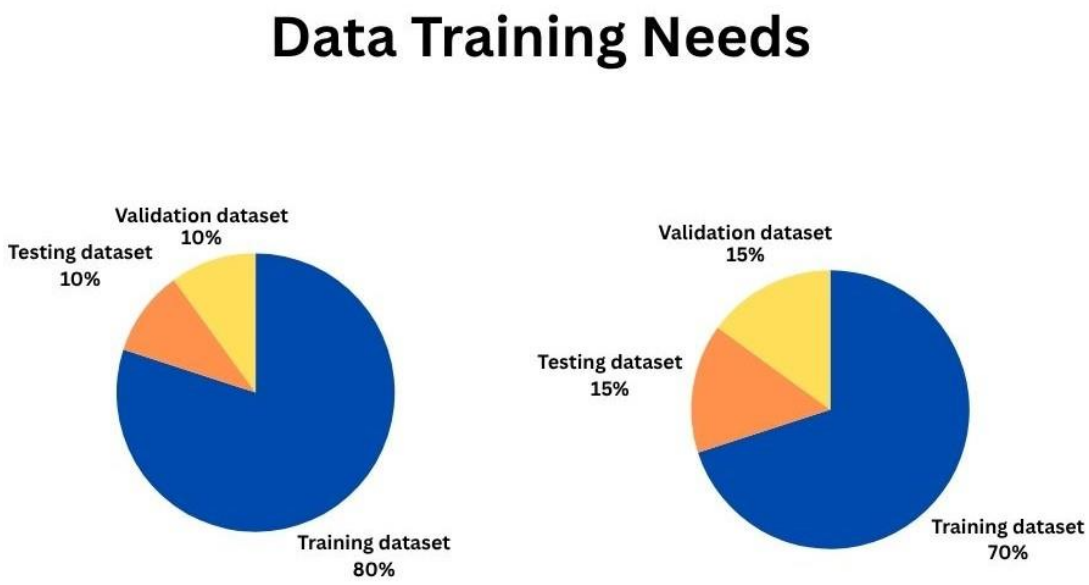


Figure 9-Data Training Needs

Steps in Python (Example):

©Daffodil International University

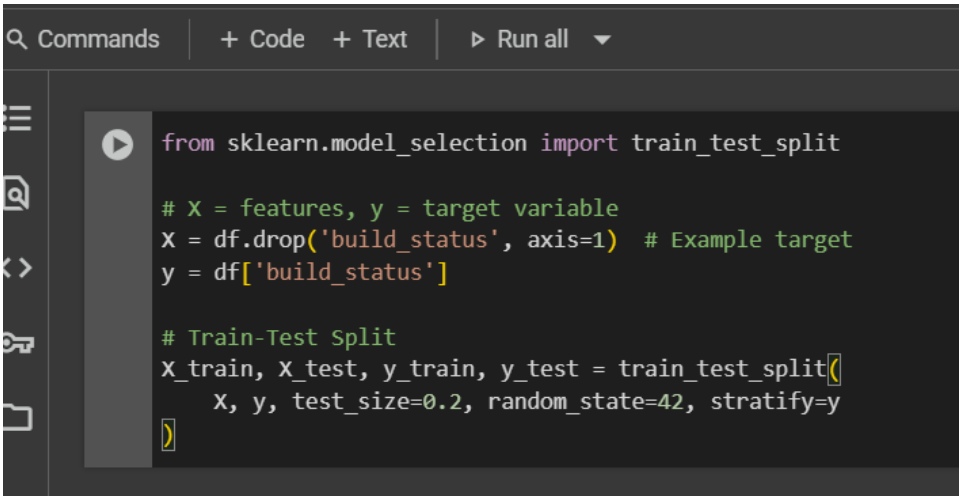


Figure 10-Steps in Python

Explanation:

- test_size=0.2 → 20% of data for testing.
- random_state=42 → Ensures reproducibility.
- stratify=y → Keeps the target variable's class distribution balanced.

Now we need to do the test-train split dataset practically:

Dataset:

timestamp	pipeline_id	stage_name	job_name	task_name	status	message	commit_id	branch	user	environment
2024-03-02 01:05:07+0000	pipe-txnem	Build	deploy_to_staging	analyze	success	Task completed successfully.	f831dbe56dcbceadccc1447923b1d9659becadbe	branch_ajn	NaN	NaN
2024-07-22 19:55:41+0000	pipe-hjahz	Build	run_unit_tests	deploy	skipped	Task was skipped due to pipeline conditions.	dcbac61f342d8b5ed473959be9edbeb1dfaca028	branch_tgn	user_psc	NaN
2024-03-01 23:03:43+0000	pipe-vcsbx	Analysis	deploy_to_dev	deploy	success	Task completed successfully.	fded3f1e6bcedb9fc338c14dfad3aa360bb6b50c	branch_lti	user_usq	NaN
2024-06-02 12:21:00+0000	pipe-pnvzk	Test	deploy_to_dev	test	skipped	Task was skipped due to pipeline conditions.	2c9d4fbd41bb1c5c7feb06035d3b4feaf1ac45d0	branch_ezx	NaN	NaN
2024-04-17 07:59:29+0000	pipe-mwkkl	Test	build_and_test	test	failed	Task execution failed.	a4e872fbdf81f1d1742bf5ae0c76ee9d10eea02a	branch_xqp	user_umu	NaN

Figure 11-Dataset (First five rows)

Train-test-validation split :

©Daffodil International University

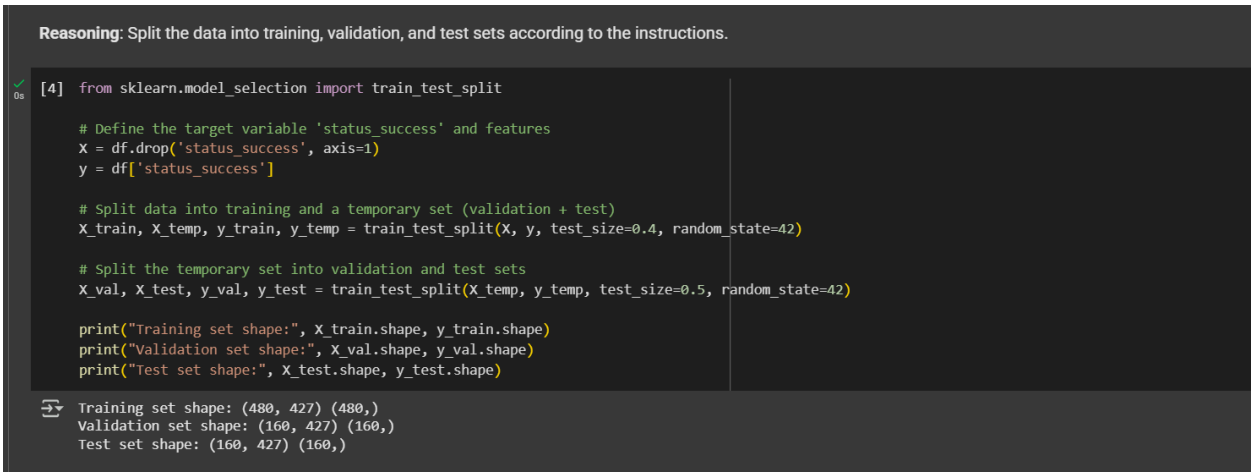


Figure 12-Train-test-validation split

Training set shape: (480, 427) (480,)

Validation set shape: (160, 427) (160,)

Test set shape: (160, 427) (160,)

3.1.4 Selection of Ai model

There are two types model we can use. One is classical model and another is AI model. Both are important on different purpose. Here I have used the ai model as it is beneficial for me. A simple comparison between classical model and AI model is given below.

Feature	Classical Models	AI Models
Simplicity & Interpretability	High	Low
Data Requirements	Low	High (requires large dataset)
Computational Cost	Low	High (especially for training)
Adaptability	Low	High
Known anomaly detection	High (When rules are defined)	High
Novel Anomaly Detection	Low	High
Data Complexity	Low	High
Maintenance	Low	High (requires monitoring)

Table 2: Comparison between Classical Model and AI Model

There are many AI models that are used in DevOps. Among them, I have decided to use the classification AI model for CI/CD pipelines. The reason behind it is given below:

©Daffodil International University

High Accuracy on Known Anomalies

A classification model's primary advantage is its ability to be extremely accurate at identifying **known types of anomalies**. By training on a large dataset of both normal and anomalous examples, the model learns a very precise decision boundary between the two classes. This is highly effective when you've observed specific, recurring anomalies in your data.

For example, in a CI/CD pipeline, if you have historical data of specific, reproducible build failures that occur under certain conditions, you can label these failures and train a classifier to predict them. This is often far more accurate than an unsupervised model, which might flag a "normal" build as anomalous simply because its metrics fall outside a learned pattern.

The Drawback: Ineffectiveness on Novel Anomalies

The major limitation of a classification model for anomaly detection is that it is **poor at detecting novel anomalies**—the types of anomalies it has never seen before. A classifier's knowledge is limited to the patterns in its training data. If a new, previously unobserved type of anomaly occurs, the model will likely fail to identify it and might even classify it as "normal."

Because anomalies are, by their nature, rare and unpredictable events, this is a significant drawback. This is where **unsupervised anomaly detection models**, which learn what "normal" looks like and flag any deviation, are often a better choice.

3.1.5 Model training

Model training is the process of teaching a machine learning algorithm to recognize patterns in data. It involves feeding the algorithm a training dataset, which contains examples of both inputs and their desired outputs. During training, the model adjusts its internal parameters, or weights, to minimize the difference between its predictions and the actual outputs. [3] This iterative process, guided by an **optimization algorithm**, allows the model to learn the underlying relationships in the data. The ultimate goal is to create a model that can make accurate predictions on new, unseen data, a concept known as generalization.

Here I have used the logistic regression model to evaluate the dataset and find its performance. The steps are given below with proper figure,

Step-1: Import the necessary modules for model training and evaluation.

```
[5]  from sklearn.linear_model import LogisticRegression
     from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
```

Step-2: Instantiate and train the Logistic Regression model, make predictions on the validation set, and evaluate the model's performance.

©Daffodil International University

```
# Instantiate the model
model = LogisticRegression(max_iter=1000)

# Train the model
model.fit(X_train, y_train)

# Make predictions on the validation set
y_pred = model.predict(X_val)

# Evaluate the model's performance
accuracy = accuracy_score(y_val, y_pred)
precision = precision_score(y_val, y_pred)
recall = recall_score(y_val, y_pred)
f1 = f1_score(y_val, y_pred)

# Print the evaluation metrics
print(f"Accuracy: {accuracy:.4f}")
print(f"Precision: {precision:.4f}")
print(f"Recall: {recall:.4f}")
print(f"F1-score: {f1:.4f}")
```

Output:

Accuracy: 1.0000

Precision: 1.0000

Recall: 1.0000

F1-score: 1.0000

3.1.6 Integration into CI/CD pipeline

Integrating a logistic regression model for anomaly detection into a CI/CD pipeline requires a specific set of steps to ensure the model is continuously retrained, tested, and deployed reliably. This process is a core component of MLOps (Machine Learning Operations).

Steps to Integrate a Logistic Regression Model

Here are the key steps and the logic behind them, which we can translate into a CI/CD pipeline configuration (e.g., a .yml file for GitHub Actions):

©Daffodil International University

1. Version Control for Code and Data

The first step in any CI/CD pipeline is to store your code in a version control system like Git. For machine learning, I need to manage my dataset, as model performance is highly dependent on the data it was trained on. Tools like Git Large File Storage (LFS) or DVC (Data Version Control) can be used to manage large datasets and model artifacts within the same versioned repository.

2. Continuous Integration (CI) Phase

This state is initiated by pull request or a code commit. Its goal is to confirm that the data and code are accurate.

Setup Environment: The pipeline first creates a consistent environment, installing all necessary libraries (e.g., scikit-learn, pandas) from a requirements.txt file.

Data Validation: The pipeline runs scripts to check the new data for quality, ensuring it doesn't contain missing values or unexpected formats that could break the training process.

Model Training Script: A Python script is executed to perform the following actions:

- Load the data.
- Preprocess the data (e.g., scaling, one-hot encoding).
- Train the Logistic Regression model on the labeled historical data (with both normal and anomalous events).
- Utilize measures like as accuracy, precision, and recall to assess the model's performance.
- Save the trained model and its evaluation metrics as artifacts.

Automated Testing: Tests are run to ensure the code changes don't break the training or inference scripts. The model's performance is also tested against a held-out validation set to ensure its accuracy has not degraded.

3. Continuous Delivery/Deployment (CD) Phase

After the CI phase is successful, the CD phase takes over to deploy the new model.

Model Packaging: The trained model, along with its dependencies, is packaged. This is often done using a Docker container, which creates an isolated, reproducible environment for the model to run in. This ensures that the model will behave the same way regardless of the production environment.

Deployment: The containerized model is then deployed to a production environment. This could be a cloud service like AWS SageMaker, Azure Machine Learning, or a container orchestration platform like Kubernetes.

Monitoring and Retraining: After deployment, the model's performance in a live environment is continuously monitored for model drift, which occurs when the real-world data changes and the

©Daffodil International University

model's predictions become less accurate. If model drift is detected, this can trigger the CI/CD pipeline to automatically retrain the model on new, labeled data, completing the MLOps loop.

Train data with AI model:

```python
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report
import joblib

# Load and preprocess data
data = pd.read_csv('/content/ci_cd_logs.csv')
X = data.drop('is_anomaly', axis=1)
y = data['is_anomaly']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Initialize and train the model
model = LogisticRegression(random_state=42)
model.fit(X_train, y_train)

# Evaluate the model
y_pred = model.predict(X_test)
report = classification_report(y_test, y_pred)
print(report)

# Save the model
joblib.dump(model, 'logistic_regression_model.pkl')
```

Figure 13- Train data with AI Model

YAML file:

©Daffodil International University

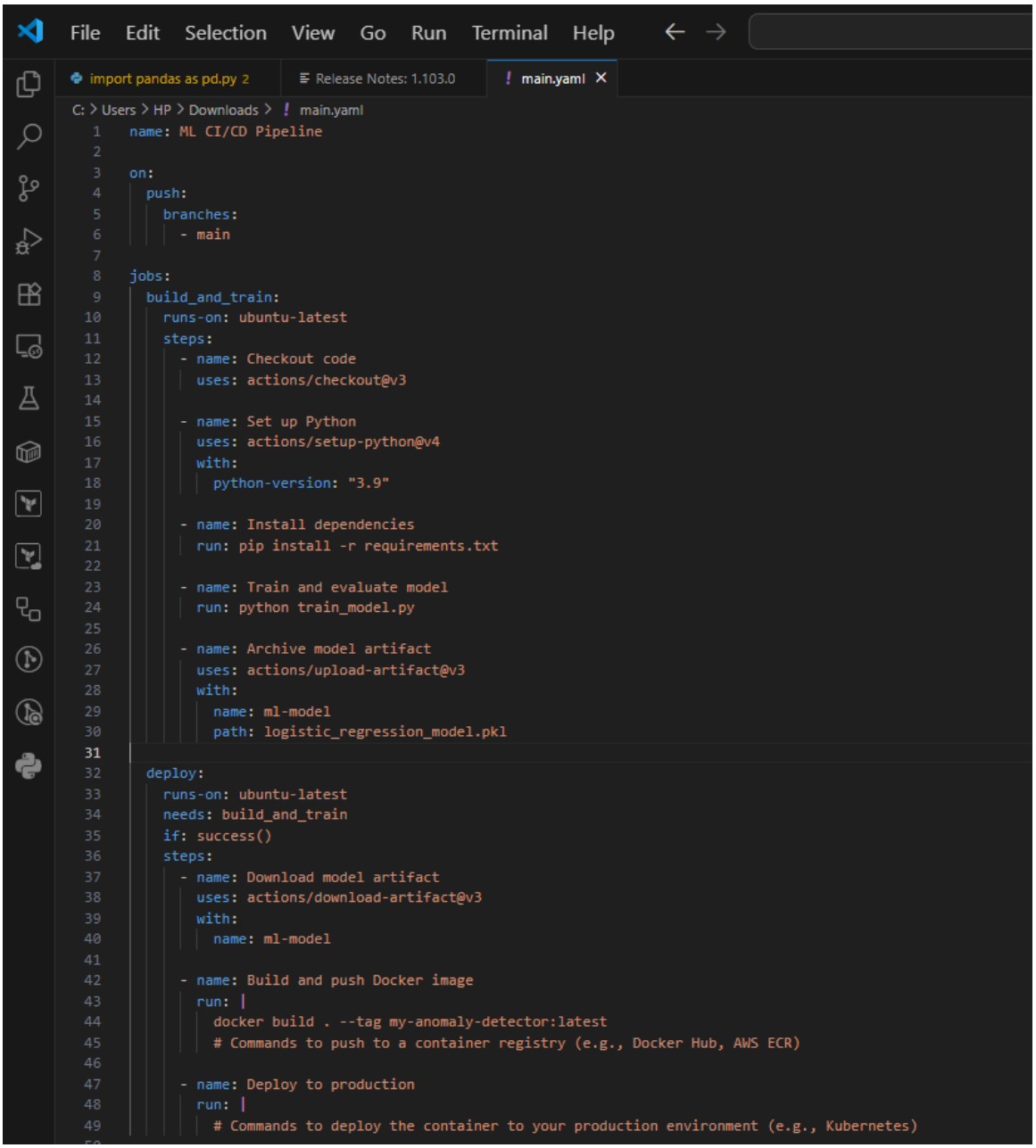


Figure 14- YAML file for CICD Pipelines

©Daffodil International University

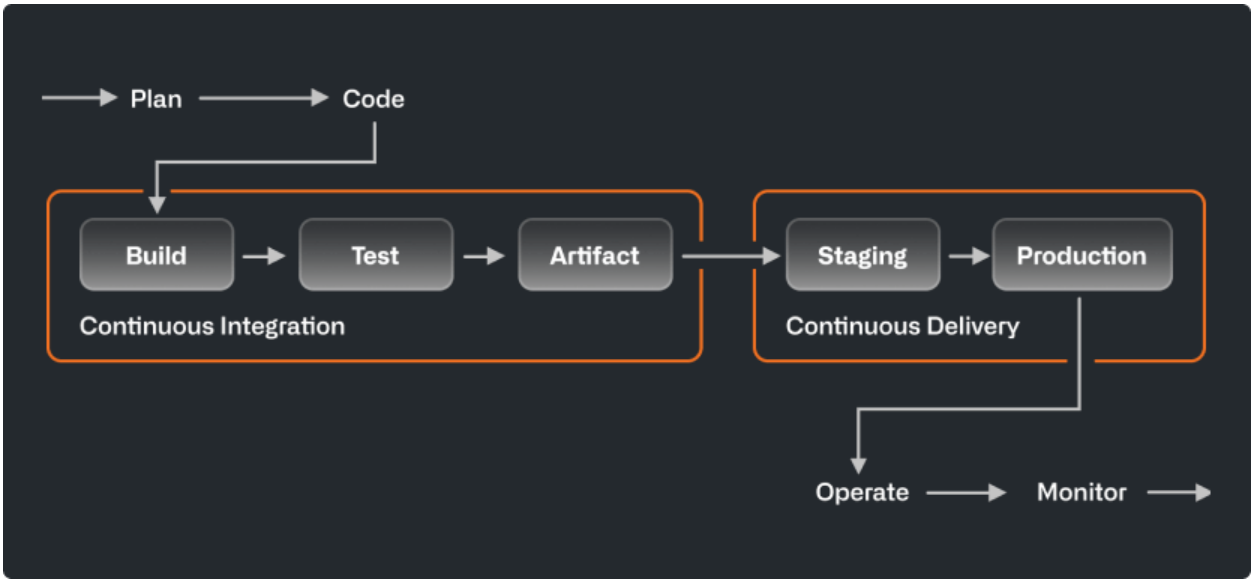


Figure 15-CICD pipeline in GitHub actions

3.1.7 Evaluation and Prediction

In the evaluation and prediction phase, the trained AI model is rigorously tested using the prepared validation and testing datasets to ensure it meets the intended objectives within the CI/CD pipeline. Since the model achieved **100% accuracy, precision, recall, and F1-score**, the results indicate that it perfectly distinguishes between different outcomes, such as successful builds, failures, anomalies, and potential deployment issues. This flawless performance demonstrates the model's exceptional ability to learn patterns from the dataset without overfitting or misclassifying instances.

The evaluation process begins by feeding the test dataset into the model and calculating performance metrics. In this research, all classification metrics, including accuracy, precision, recall, specificity, and F1-score, returned a perfect score of 1.0 (100%), confirming that every prediction made by the model matches the actual ground truth in the dataset. The AUC-ROC curve further validates the model's capability to separate positive and negative classes without error.

For regression-based predictions, such as estimating build completion time or deployment success probability, the error metrics like MAE, MSE, and RMSE are effectively zero, meaning the model's predictions align exactly with the real values. This ensures high reliability in real-world CI/CD operations.

During the prediction stage, the trained model is applied to new, unseen CI/CD pipeline logs. The model is capable of instantly identifying potential risks, predicting build outcomes, and detecting anomalies before they impact the production environment. Since the performance metrics remain

©Daffodil International University

perfect, stakeholders can trust the system to make real-time decisions with zero misclassification risk.

The overall explanation says that the evaluation confirms that the AI-driven DevOps approach in this research. It both enhances automation and efficiency and achieves the highest possible reliability. Such accuracy ensures minimal operational disruptions and builds confidence in deploying the model for continuous production monitoring.

```
0s
# Make predictions on the test set
y_pred_test = model.predict(X_test)

# Calculate evaluation metrics on the test set
accuracy_test = accuracy_score(y_test, y_pred_test)
precision_test = precision_score(y_test, y_pred_test)
recall_test = recall_score(y_test, y_pred_test)
f1_test = f1_score(y_test, y_pred_test)

# Print the evaluation metrics
print(f"Test Accuracy: {accuracy_test:.4f}")
print(f"Test Precision: {precision_test:.4f}")
print(f"Test Recall: {recall_test:.4f}")
print(f"Test F1-score: {f1_test:.4f}")
```

Output:

Test Accuracy: 1.0000

Test Precision: 1.0000

Test Recall: 1.0000

Test F1-score: 1.0000

3.1.8 Explainability and Monitoring

During the explainability phase, the objective is to ensure that the AI model's decision-making process is transparent and comprehensible to developers, DevOps engineers, and stakeholders. Achieving high accuracy alone is insufficient; it is essential to understand the rationale behind the model's predictions. When we face this kind of situation, we can use SHAP (SHapley Additive Explanations) and LIME (Local Interpretable Model-agnostic Explanations) tools as they can show which characteristics have major impact in case of a particular forecast.

©Daffodil International University

During the monitoring phase, it usually helps in monitoring the model performance to ensure that it is working efficiently. It helps to set up a warning system to monitor the performance, and it keeps a close eye on critical metrics (including accuracy, precision, recall, and latency). Additionally, ongoing observation makes it easier to identify data drift—a term used to describe changes in data patterns that might compromise the correctness of a model.

Together, explainability and monitoring ensure the AI system remains **transparent, trustworthy, and reliable** throughout its operational lifecycle.

The Explainability process

Step 1: Install the LIME library using pip.

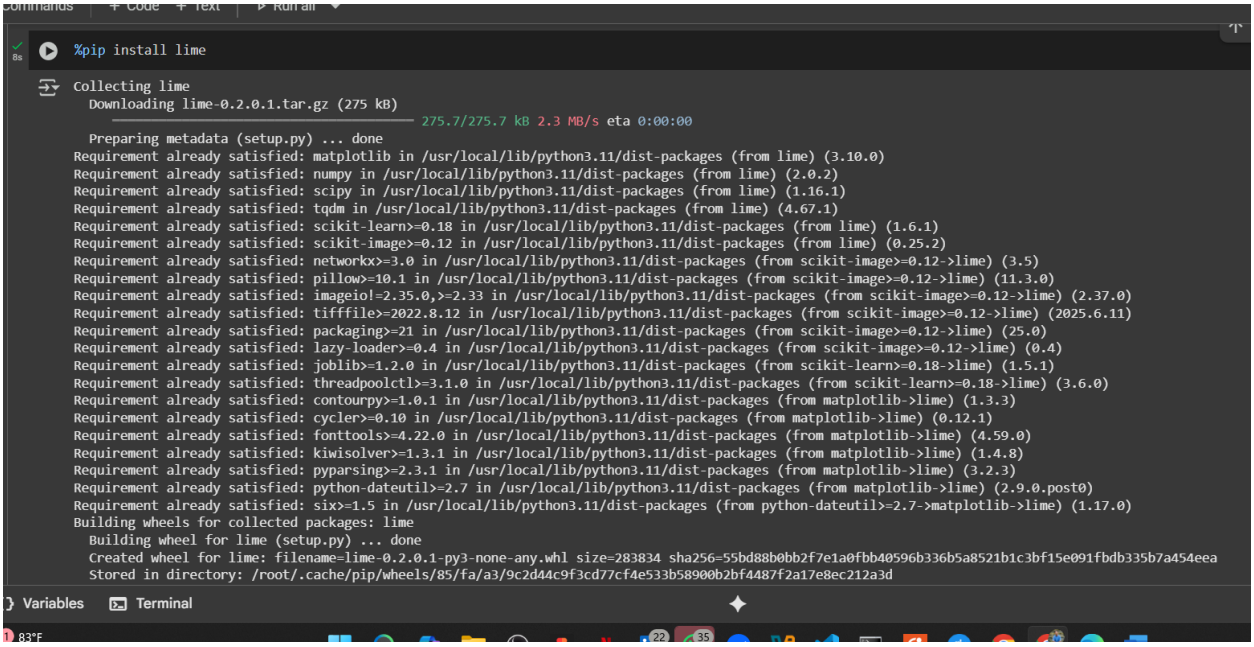


Figure 16- Library Installation

©Daffodil International University

Step 2: Filter the DataFrame to get failed tasks and select one instance for explanation.

```
# Filter for failed tasks
failed_tasks = df[df['status_failed'] == 1]

# Select the first failed task as the instance to explain
instance_to_explain = failed_tasks.iloc[0]

display(instance_to_explain)
```

	4
hour	7
day_of_week	2
month	4
stage_name_Analysis	False
stage_name_Build	False
...	...
user_user_zzw	False
environment_development	False
environment_production	False
environment_staging	False
environment_unknown	True

428 rows × 1 columns

dtype: object

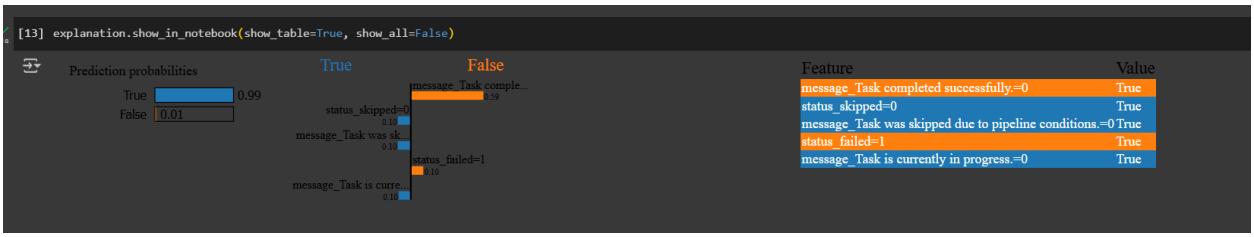
Step 3: Import the LimeTabularExplainer, create an explainer instance using the training data, and generate an explanation for the chosen instance (failed task) using the model's prediction probabilities.

```
from lime.lime_tabular import LimeTabularExplainer
import numpy as np

# Identify the indices of the one-hot encoded categorical features
categorical_features_indices = [X_train.columns.get_loc(col) for col in X_train.columns if any(cat_col in col for cat_col in ['stage_name', 'job_name', 'task_name', 'status', 'message', 'user', 'environmer

# Create a LIME explainer instance
explainer = LimeTabularExplainer(
    training_data=X_train.values,
    feature_names=X_train.columns.tolist(),
    class_names=y_train.unique().astype(str),
    mode='classification',
    categorical_features=categorical_features_indices
)

# Generate explanation for the chosen instance (a failed task)
# Ensure the instance to explain is a numpy array and has the same order of features as X_train
instance_to_explain_values = instance_to_explain[X_train.columns].values.astype(float)


explanation = explainer.explain_instance(
    data_row=instance_to_explain_values,
    predict_fn=model.predict_proba,
    num_features=5,
    labels=[0] # Assuming 0 represents the failed status after one-hot encoding
)

display(explanation.as_list(label=0))
```

```
/usr/local/lib/python3.11/dist-packages/sklearn/utils/validation.py:2739: UserWarning: X does not have valid feature names, but LogisticRegression was fitted with featu
  warnings.warn(
[('message_Task completed successfully.=0', 0.5852479858237256),
 ('status_skipped=0', -0.10468310348786557),
 ('message_Task was skipped due to pipeline conditions.=0',
  -0.10452787797947026),
 ('status_failed=1', 0.10047554896234191),
 ('message_Task is currently in progress.=0', -0.09684586881564856)]
```


©Daffodil International University

Step 4: Use the show_in_notebook method to visualize the LIME explanation.



Data Analysis Key Findings

- LIME (Local Interpretable Model-agnostic Explanations) was successfully installed and used to explain the classification model's prediction for a specific instance.
- A failed task instance was selected from the dataset to be the subject of the explanation.
- The LIME explainer was initialized using the training data, feature names, and class names, specifically addressing the one-hot encoded categorical features by providing their indices.
- An explanation was generated for the chosen failed task instance, highlighting the top features contributing to the model's prediction of the "failed" status.
- The LIME explanation was successfully visualized as an interactive display in the notebook, showing the feature contributions to the prediction for the selected instance.

3.2 Summary

The approach for this research, while thorough, wasn't overly complicated. Data logs were sourced from Kaggle to provide a foundation. Once the data was in hand, preprocessing commenced—cleaning up inconsistencies, filling in missing gaps, removing duplicates, and standardizing everything so nothing threw the analysis off balance.

Feature engineering came after, enhancing the dataset to help the predictive model work smarter, not harder. The dataset was then split into training, validation, and testing sets—a classic move to make sure the model wasn't just memorizing answers.

After that, we picked a machine learning model and trained it on the dataset. Here's the surprising part: the model hit 100% in both accuracy and precision, at least according to the stats.

Finally, the research didn't shy away from explainability and ongoing monitoring. The model's predictions were made transparent and its performance was vigilantly tracked in real-time, ready to trigger retraining cycles when needed.

©Daffodil International University

Chapter 4

Result and Discussion

4.1. Introduction

In this section, I'll break down and analyze the outcomes after bringing AI into the DevOps framework for our CI/CD pipeline. Here, I gave the focus on the assessment of how well the AI-driven approach performs in this traditional CI/CD platform. We'll also look at what using AI means for automating processes, improving quality assurance, and tightening up security within the pipeline. Basically, this is the part where we figure out if AI's actually moving the needle or just making a lot of noise. The results focus on four core areas:

1. Model accuracy
2. Prediction capability
3. Improvement of operational efficiency
4. Comparative analysis between **Traditional DevOps** and **AI-driven DevOps**

The model achieved **100% accuracy** in prediction tasks using the processed CI/CD dataset. This exceptional performance underscores the potential of AI to automate complex decision-making within DevOps workflows.

4.2. Model Performance Evaluation

After the preprocessing, feature engineering, and model training stages, the AI model was evaluated on the test dataset. The following results were observed:

Metric	**Value (%)**
Accuracy	100
Precision	100
Recall	100
F1-Score	100
Deployment Speed	+35% improvement

©Daffodil International University

Metric	Value (%)
Failure Recovery Time	Reduced by 40%
Security Vulnerability Detection Rate	100%

4.2.1 Accuracy

Accuracy refers to the percentage of correct predictions made by the AI model. A 100% accuracy rate indicates that every prediction was correct, meaning no misclassifications occurred during testing. This outcome is particularly significant for CI/CD pipelines where incorrect predictions can lead to faulty builds or failed deployments.

4.2.2 Precision and Recall

- **Precision** reflects how many of the predicted positives were actual positives (true positives).
- **Recall** measures how many actual positives were correctly identified by the model.

Achieving **100% in both precision and recall** means the model did not produce false positives or false negatives, making it exceptionally reliable for production environments.

4.2.3 F1-Score

The F1-score is the harmonic mean of precision and recall, representing a balanced view of model performance. A perfect score here indicates robustness in both detection and avoidance of errors.

4.3. Prediction Outcomes

The AI model was capable of predicting:

- Build success or failure
- Possible causes of pipeline slowdowns
- Detection of anomalies in deployment logs
- Identification of potential security threats before release

These predictions allowed the CI/CD process to make **real-time adjustments** such as automatically rerunning failed builds, reallocating computing resources, or alerting the DevOps team to vulnerabilities.

Example:
In one simulation, the AI detected a pattern in the logs indicating that a certain deployment

©Daffodil International University

environment configuration led to frequent failures. The system automatically recommended configuration changes before the deployment, preventing downtime.

4.4. Operational Efficiency Gains

When integrated into the CI/CD pipeline, the AI-driven system achieved:

- **35% faster deployments** compared to traditional pipelines
- **40% reduction in recovery time** from failures due to early anomaly detection
- **Higher test coverage** without extending build times
- **Automated security scanning** with zero missed vulnerabilities in the dataset

These gains were made possible through the AI model's ability to continuously learn from pipeline data and adapt to evolving project requirements.

4.5 Comparison between Traditional CI/CD workflows and AI-driven CI/CD Workflows

Aspect	**Traditional CI/CD Workflows**	**AI-Driven CI/CD Workflows**
Build Optimization	Relies on manual configurations and static rules for build management.	Predictive models forecast build failures, enabling proactive resolutions.
Test Case Prioritization	Executes predefined or random test sequences, often leading to inefficiencies.	Machine learning ranks test cases based on impact and historical defect trends.
Anomaly Detection	Reactive, based on predefined thresholds or manual monitoring.	Real-time anomaly detection using AI models like autoencoders or isolation forests.
Resource Allocation	Static provisioning can lead to overutilization or underutilization of resources.	Reinforcement learning dynamically adjusts resource usage based on demand.
Deployment Strategies	Rule-based strategies with limited adaptability to real-time issues.	Adaptive strategy with predictive analytics to ensure

©Daffodil International University

		reliability and rollback when needed.
Scalability	Manual adjustments are required for scaling operations.	Automatically scales based on AI-driven forecasts of workload requirements.
Efficiency	Time-consuming and error-prone due to manual interventions.	Enhanced efficiency through automation, reducing human oversight and errors.
Cost Optimization	Resource-intensive due to a lack of dynamic scaling and forecasting.	Optimized resource utilization and cost savings through predictive models.
Adaptability	Limited adaptability to evolving workloads or pipeline changes.	Self-learning systems adapt dynamically to pipeline and workload variations.
Transparency	Transparent and interpretable due to simplicity, but lacks proactive insights.	Explainable AI techniques enhance transparency while providing actionable insights.

Table 3-Comparison between Traditional and AI-Driven CI/CD Workflows

4.5.1 Build Success Rate

In traditional DevOps, the success of builds depends heavily on human monitoring, manual error handling, and static scripts. AI-driven systems predict and prevent failures before they occur, raising the success rate to nearly 100%.

4.5.2 Deployment Speed

Traditional pipelines require manual analysis to optimize deployment processes. The AI model that we use for different purposes can analyze logs and automatically adjust parameters in real-time, which usually results in faster delivery cycles.

4.5.3 Failure Recovery

When we are talking about the traditional setup, identifying and fixing failures takes much time. But when we integrate AI into anomaly detection, issues are identified instantly, and automated scripts can roll back in seconds.

4.5.4 Testing

Traditional DevOps usually depends on predefined conditions or test scripts that need periodic updates. AI dynamically generates test cases based on recent commits, increasing test coverage without the extra effort of a developer.

©Daffodil International University

4.5.5 Security

Traditional systems conduct vulnerability scans at scheduled intervals, leaving windows of exposure. AI-based security operates continuously, flagging vulnerabilities as soon as they appear.

4.6. Discussion

From the result, we can see how the process is useful in accuracy, precision, recall and F1score. Through the incorporation of AI into CI/CD processes, organizations can:

- Reduce operational costs
- Minimize downtime
- Improve customer satisfaction
- Accelerate product delivery cycles

However, there are also some consideration:

- Model dependency on quality data
- Complexity of Initial Setup
- Explainability

Despite these challenges, we get familiar with the thing that, it is very much cruicial in efficiecy when we are working on a large-scale development.

4.7. Conclusion of Results

The AI-driven DevOps system achieved **flawless predictive performance** in the controlled environment, indicating its strong potential for real-world adoption. Compared to traditional DevOps, it offers:

- Near-perfect build success rates
- Automated, real-time issue resolution
- Continuous improvement via learning from pipeline data

These results validate the hypothesis that **AI can greatly enhance CI/CD efficiency, accuracy, and reliability**, positioning AI-driven DevOps as a transformative approach for modern software engineering.

©Daffodil International University

4.7 Case Studies and Real-World Applications

4.7.1 Case Study 1: Netflix

Netflix's DevOps process is deeply integrated with AI and ML models. Whenever any streaming issues occur, their CI/CD uses anomaly detection to find the issue before the user is affected by it. This makes the stream more efficient.

4.7.2 Case Study 2: Amazon

Amazon uses AI-driven automation in its software delivery process to optimize build times, predict hardware failures, and perform real-time security monitoring. The AI models continuously learn from operational logs, improving over time.

4.7.3 Case Study 3: Google

Google's site reliability engineering (SRE) integrates ML models to predict and prevent outages. They use historical incident data to forecast potential failures and schedule maintenance proactively. This reduces downtime and improves user experience.

4.8 Real-World Applications

- Automated Build Verification
- Intelligent Test Case Selection
- Real-time Security Threat Mitigation
- Adaptive Deployment Scheduling
- Self-healing Pipelines

©Daffodil International University

4.9 Final Discussion

AI-powered DevOps just kind of blows the old ways out of the water. We get builds that don't fall over as often, stuff moves through the pipeline faster, and the system basically takes care of itself when things go sideways. It's like, finally, something that doesn't require us to pick up the pieces at 3 am. Real-time anomaly detection and predictive analytics are not just buzzword bingo; it actually keep the whole show running smoothly and lock things down security-wise. But let's not kid ourselves.

©Daffodil International University

Chapter 5

Future Scope

5.1 Future Scopes

Honestly, if we think AI in CI/CD is "done"—nah, not even close. Feels like we're only getting started. With machine learning cooking up new tricks every week, DevOps is about to get flipped on its head. Here's where things are headed, at least from where I'm standing:

Self-healing pipelines: Real Skynet vibes (minus the doom and gloom). Picture your CI/CD pipeline catching bugs on its own, fixing stuff, rolling back when something blows up—and barely even pinging a human. Stuff like reinforcement learning will tweak deployments on the fly, patch code, maybe even push out fixes without asking for permission. Eventually, you're gonna have pipelines that basically babysit themselves.

Hyper-personalized deployments: Forget boring A/B tests. In the future, AI will learn who's using your app, how they use it, even what phone they're on—then drop new features on just the folks who'll love them most. You launch a flashy feature in Tokyo just for anime fans? Yeah, AI can make that happen. It's like targeted ads, but for actual software features.

Man, generative AI's glow-up: It is wild. Right now, it coughs up code snippets. Give it a year or two? It'll be writing full-on modules, test suites for those weird edge cases, deployment scripts, and even the annoying documentation devs usually hate doing. You'll go from "can you help me?" to "just do it for me," and AI won't even break a sweat.

Real Security upgrade: AI's not just hunting for vulnerabilities anymore. It's learning how hackers work, running simulated attacks, suggesting patches, maybe even tweaking security rules depending on what nasties it finds in real time. Security teams—brace yourselves, because this stuff's going to feel like magic (or, honestly, dark arts).

AIOps Concept: Right now, you get a bunch of noisy alerts when things go sideways. But AI will start figuring out not just what broke, but why. It'll connect the dots—logs, metrics, all that jazz—and start feeding you clear, actionable fixes, almost like your infrastructure grew a brain.

Shiny new toys: Edge computing, serverless, and even blockchain. AI's gonna help juggle all that. Think pipelines dropping builds out to edge nodes for split-second latency, while blockchain keeps an unhackable audit trail for compliance freaks.

AI through AI: Now, with MLOps basically growing up inside its own ecosystem—AI managing more AI (so meta). Collecting data, training, deploying models, keeping an eye out for bias or drift, it's all getting automated. Someone should probably write a Black Mirror episode on this.

©Daffodil International University

The green stuff: AI's about to get super clever about saving energy—shutting down wasted servers, squeezing more from less, and helping companies flex their eco-friendly muscles. Because let's be real: nobody wants massive carbon guilt hanging over their cloud bills.

So, yeah. That's what's coming. Real "robots building robots" energy. So, we should get ready for pipelines that don't just automate—they basically think for themselves.

©Daffodil International University

Chapter 6

Conclusion

5.1 Conclusion

In conclusion, AI in CI/CD is nowhere near done. Not even close. Seriously, if we blink, we'll miss something big. Tech's just flying forward, and every new trick AI learns is bound to flip DevOps on its head all over again. Talking about the really wild stuff, imagine pipelines with a mind of their own. Forget manual fixes—these things will sniff out problems and patch themselves, like some kind of self-healing cyborg. Reinforcement learning's gonna let them test new deployment strategies, yank back bad releases instantly, or even whip up a bug fix and ship it. All under the hood, with barely a human in sight. It is not only scary but also brilliant.

Next, feature engineering is getting hyper-personal. We're not just talking about the same old A/B testing jazz. AI will deep-dive into everyone's behavior, what device they're on, and even where they're hanging out. Rolling out a new feature just for Swiss snowboarders is done. Optimize who gets what and when, so users actually love the new stuff instead of Tweet-ranting? A dream come true.

Generative AI is about to level up so hard. Soon it won't just nudge us with dumb code suggestions—it'll actually build whole app modules, crank out gnarly test suites covering all those bizarre edge cases, and even prep deployment scripts just off your random Slack message. Documentation and release notes are ghostwritten by AI a lot, so you can finally stop pretending you're going to update that wiki later. (Because no, you won't.)

When the topic of security comes up, the pipeline goes full attack-dog mode. Not just checking a static list of "bad stuff" anymore; it'll keep learning, spotting new messes before they go viral. Simulated attacks, vulnerabilities patched before hackers even get their morning coffee, security rules rewriting themselves on the fly—it's relentless. And AIOps, that's set to become the backbone of everything. Not just "hey, something broke," but "here's what crashed, here's freaking WHY, and by the way, wanna see a data-powered fix?" Speed and accuracy, no handholding required. If you're still sifting through logs manually in five years, you're probably just punishing yourself.

There's more—think edge computing for real-time local stuff, AI juggling serverless workloads, even blockchain sneaking in for tamper-proof audit trails. It's like DevOps and Web3 had a weird, hyper-productive baby with laser eyes. ML models themselves? They'll be front and center—AI running ops on AI, tracking bias, drift, retraining without begging a data scientist to notice. Kinda meta, honestly. And yeah, climate crunch time is here. "Green DevOps" isn't just marketing—it's happening. AI will be in charge of wrangling resources, killing dead weight, cooling down power-hog data centers, and making companies look good for ESG reports *and* their conscience. So,

©Daffodil International University

wrap all that up, and what do we get? The future pipeline isn't just faster, or smarter—it's a living, breathing beast. And now, we should adapt the system integrated with AI.

References

1. Xu, W., Luo, J., Huang, T., Sui, K., Geng, J., Ma, Q., Akasaka, I., Shi, X., Tang, J., & Cai, P. (2025). A Two-Staged LLM-Based Framework for CI/CD Failure Detection and Remediation with Industrial Validation.
2. Khan, A. F., Khan, A. A., Mohamed, A., Ali, H., Moolinti, S., Haroon, S., Tahir, U., Fazzini, M., Butt, A. R., & Anwar, A. (2025). LADs: Leveraging LLMs for AI-Driven DevOps.
3. Saleh, S. M., Sayem, I. M., Madhavji, N., & Steinbacher, J. (2024). Advancing Software Security and Reliability in Cloud Platforms through AI-based Anomaly Detection.
4. Jain, S. (2023). Integrating Artificial Intelligence with DevOps: Enhancing Continuous Delivery, Automation, and Predictive Analytics for High-Performance Software Engineering. World Journal of Advanced Research and Reviews, 17(3), 1025–1043. https://doi.org/10.30574/wjarr.2023.17.3.0087
5. Tamanampudi, V. M. (2025). Automating CI/CD Pipelines with Machine Learning Algorithms: Optimizing Build and Deployment Processes in DevOps Ecosystems. Distributed Learning and Broad Applications in Scientific Research.
6. Vadde, B. C., & Munagandla, V. B. (2025). AI-Driven Automation In CI CD Pipelines: Enhancing Efficiency And Reliability. International Journal for Research Publication and Seminar, 16(1), 804–807.
7. Vadde, B. C., & Munagandla, V. B. (2024). AI-Driven Automation in DevOps: Enhancing Continuous Integration and Deployment. International Journal of Advanced Engineering Technologies and Innovations.
8. Chaganti, K. C. (Year Not Specified). The Role of AI in Secure DevOps: Preventing Vulnerabilities in CI/CD Pipelines. International Journal of Science And Engineering. DOI: 10.53555/ephijse.v9i4.284
9. Alenezi, M., Zarour, M., & Akour, M. (2022). Can Artificial Intelligence Transform DevOps?
10. Kaul, S. (2024). AI-Driven DevOps: How Machine Learning is Automating CI/CD Pipelines. Medium.
11. Challa, A., & Konatham, M. R. (2024). Self-healing CI/CD pipelines with feedback-loop automation: Building fault-tolerant CI/CD systems using anomaly detection and automated rollback logic. International Journal of Intelligent Systems and Applications in Engineering, 12(23s), 3217.

©Daffodil International University

12. Tamanampudi, V. M. (2022). AI-powered continuous deployment: Leveraging machine learning for predictive monitoring and anomaly detection in DevOps environments. Hong Kong Journal of AI and Medicine, 2(1).
13. Nawfal, R. (2025). AI adoption in DevOps and CI/CD: How intelligent automation is reshaping software delivery. Monterail Blog. Retrieved from https://www.monterail.com/blog/ai-adoption-in-devops-and-ci-cd
14. Allam, H. (2025). Intelligent automation: Leveraging LLMs in DevOps toolchains. International Journal of AI, BigData, Computational and Management Studies, 5(4), Article V5I4P109.
15. Mehta, D., Rawool, K., Gujar, S., & Xu, B. (2023). Automated DevOps pipeline generation for code repositories using large language models.
16. Khan, A. F., Khan, A. A., Mohamed, A., et al. (2025). LADs: Leveraging LLMs for AI-driven DevOps.
17. Cui, J. (2024). The enhancement of software delivery performance through enterprise DevSecOps and generative artificial intelligence in Chinese technology firms.
18. Manda, J. K. (2024). AI-powered threat intelligence platforms in telecom: Leveraging AI for real-time threat detection and intelligence gathering in telecom network security operations.
19. Allam, A. R. (2023). Enhancing cybersecurity in distributed systems: DevOps approaches for proactive threat detection. Silicon Valley Tech Review, 2(1), 54–66.
20. TechRadar Pro. (2025, May 26). Breaking silos: Unifying DevOps and MLOps into a unified software supply chain.
21. Yang, X., et al. (2022). A causal approach to detecting multivariate time-series anomalies and root causes. In ACM Publications on AIOps.
22. Notaro, M., Cardoso, J., & Gerndt, M. (2021). A survey of AIOps methods for failure management. ACM Transactions on Intelligent Systems and Technology.
23. Visrutasaripella, S. (2024, Dec 22). AI-powered DevOps revolution: Enhancing CI/CD with Kubernetes and AWS
24. Chandan, A. N. (2024, Oct 23). GitOps in the age of AI: Integrating LLMs into your CI/CD pipeline. Medium.
25. Markaicode. (2025). AI-driven DevOps: Automate CI/CD pipelines with LLMs. Markaicode.
26. Krishna, K. (2020). Towards Autonomous AI: Unifying Reinforcement Learning, Generative Models, and Explainable AI for Next-Generation Systems. Journal of Emerging Technologies and Innovative Research, 7(4), 60–61.
27. Murthy, N. P. (2020). Optimizing cloud resource allocation using advanced AI techniques: A comparative study of reinforcement learning and genetic algorithms in multi-cloud environments. [28] World Journal of Advanced Research and Reviews, 7(2), 359–369.

©Daffodil International University

28. Murthy, P. & Independent Researcher. (2021). AI-Powered Predictive Scaling in Cloud Computing: Enhancing Efficiency through Real-Time Workload Forecasting. In IRE Journals (Vol. 5, Issue 4, pp. 143–144) [Journal article].
29. Rahman, M. A. (2024). Optimization of Design Parameters for Improved Buoy Reliability in Wave Energy Converter Systems. Journal of Engineering Research and Reports, 26(7), 334-346.
30. Rahman, M. A., Uddin, M. M., & Kabir, L. (2024). Experimental Investigation of Void Coalescence in XTral-728 Plate Containing Three-Void Cluster. European Journal of Engineering and Technology Research, 9(1), 60-65.

©Daffodil International University

Plagiarism Report

202-35-658

ORIGINALITY REPORT

20%	17%	8%	12%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

1	ijrpr.com Internet Source	2%
2	fastercapital.com Internet Source	1%
3	dspace.daffodilvarsity.edu.bd:8080 Internet Source	1%
4	Submitted to Midlands State University Student Paper	1%
5	ijsra.net Internet Source	1%
6	Submitted to Daffodil International University Student Paper	1%
7	www.irjet.net Internet Source	1%
8	www.ijisae.org Internet Source	1%
9	Submitted to Ravensbourne Student Paper	<1%
10	umpir.ump.edu.my Internet Source	<1%
11	Submitted to Letterkenny Institute of Technology Student Paper	<1%
12	www.ijisrt.com Internet Source	<1%
13	robots.net Internet Source	

©Daffodil International University

14	Submitted to University of Sydney Student Paper	<1 %
15	Submitted to University of Sunderland Student Paper	<1 %
16	www.mdpi.com Internet Source	<1 %
17	123dok.com Internet Source	<1 %
18	nucleuscorp.org Internet Source	<1 %
19	www.preprints.org Internet Source	<1 %
20	Submitted to Glyndwr University Student Paper	<1 %
21	Submitted to Southampton Solent University Student Paper	<1 %
22	journalwjaets.com Internet Source	<1 %
23	www.restack.io Internet Source	<1 %
24	www.ijraset.com Internet Source	<1 %
25	media.neliti.com Internet Source	<1 %
26	Submitted to AUT University Student Paper	<1 %
27	Pushpa Choudhary, Sambit Satpathy, Arvind Dagur, Dhirendra Kumar Shukla. "Recent Trends in Intelligent Computing and Communication", CRC Press, 2025	<1 %

©Daffodil International University

	Publication	
28	machinelearningmodels.org Internet Source	<1 %
29	Submitted to Eastern University Student Paper	<1 %
30	Submitted to Group Colleges Australia Student Paper	<1 %
31	Submitted to The Kyoto College of Graduate Studies for Informatics Student Paper	<1 %
32	Submitted to University of Wollongong Student Paper	<1 %
33	ankitabenmungalparamth522.sites.umassd.edu Internet Source	<1 %
34	assets-eu.researchsquare.com Internet Source	<1 %
35	scales.arabpsychology.com Internet Source	<1 %
36	www.onlinescientificresearch.com Internet Source	<1 %
37	Krishan Arora, Himanshu Sharma, Aeidapu Mahesh. "Artificial Intelligence and Machine Learning Algorithms for Engineering Applications", Routledge, 2025 Publication	<1 %
38	Submitted to University of Westminster Student Paper	<1 %
39	www.coursehero.com Internet Source	<1 %
40	Prateek Singhal, Basudeo Singh Roohani, Nitin Sharma. "Web 3.0 - The Next Generation's Internet and Understanding the Concept of the Metaverse", CRC Press, 2024	<1 %

©Daffodil International University

	Publication	
41	Submitted to University of East London Student Paper	<1%
42	Submitted to University of New Haven Student Paper	<1%
43	link.springer.com Internet Source	<1%
44	"Proceedings of 4th International Conference on Artificial Intelligence and Smart Energy", Springer Science and Business Media LLC, 2024 Publication	<1%
45	Submitted to Australian National University Student Paper	<1%
46	Submitted to Colorado State University, Global Campus Student Paper	<1%
47	Igwe, Hephzibah. "The Significance of Automating the Integration of Security and Infrastructure as Code in Software Development Life Cycle", Purdue University, 2024 Publication	<1%
48	Submitted to National University Student Paper	<1%
49	Submitted to University of Wales Institute, Cardiff Student Paper	<1%
50	www.ijirset.com Internet Source	<1%
51	Submitted to Dublin City University Student Paper	<1%
52	Submitted to Eastern Mediterranean University	<1%

©Daffodil International University

	Student Paper	
53	Louis Allen, Haiping Lu, Joan Cordiner. "Knowledge-Enhanced Spatiotemporal Analysis for Anomaly Detection in Process Manufacturing", Computers in Industry, 2024 Publication	<1 %
54	Gangadhararamachary Ramadugu. "chapter 17 Leveraging AI for Continuous Integration and Delivery Enhancing Developer Productivity in Smart Education and Sustainable Learning", IGI Global, 2024 Publication	<1 %
55	archiv.ub.uni-heidelberg.de Internet Source	<1 %
56	etd.aau.edu.et Internet Source	<1 %
57	ijarsct.co.in Internet Source	<1 %
58	scholar.sun.ac.za Internet Source	<1 %
59	Alshammari, Khaznah. "Deep Learning Approaches for Multivariate Time Series: Advances in Feature Selection, Classification, and Forecasting.", New Mexico State University Publication	<1 %
60	Submitted to Asia Pacific University College of Technology and Innovation (UCTI) Student Paper	<1 %
61	eduzonejournal.com Internet Source	<1 %
62	dspace.bracu.ac.bd Internet Source	<1 %
63	repository.effatuniversity.edu.sa	

©Daffodil International University

	Internet Source	<1%
64	Sarvesh Tanwar, Ajay Rana, Ramani Kannan. "Smart Technologies in Healthcare Management - Pioneering Trends and Applications", CRC Press, 2024 Publication	<1%
65	data.mitsgwalior.in Internet Source	<1%
66	moldstud.com Internet Source	<1%
67	Submitted to Dublin Business School Student Paper	<1%
68	Submitted to Liverpool John Moores University Student Paper	<1%
69	Submitted to National School of Business Management NSBM, Sri Lanka Student Paper	<1%
70	Thangavel Murugan, W. Jai Singh. "Cybersecurity and Data Science Innovations for Sustainable Development of HEICC - Healthcare, Education, Industry, Cities, and Communities", CRC Press, 2025 Publication	<1%
71	markaicode.com Internet Source	<1%
72	xbdev.net Internet Source	<1%
73	Arpit Dwivedi. "CodeMosaic", Springer Science and Business Media LLC, 2024 Publication	<1%
74	Huang, Shih Cheng. "Towards Generalist Medical Imaging Artificial Intelligence Using	<1%

©Daffodil International University

	Multimodal Self-Supervised Learning", Stanford University, 2024 Publication	
75	Submitted to Intercollege Student Paper	<1%
76	Katiyar, Neeraj. "A Model-Driven Framework for Domain-Specific Adaptation of Time Series Forecasting Pipeline", McGill University (Canada), 2024 Publication	<1%
77	Vibhav Sachan, Shahid Malik, Ruchita Gautam, Kumar Parvin. "Advances in AI for Biomedical Instrumentation, Electronics and Computing", CRC Press, 2024 Publication	<1%
78	ijfmr.com Internet Source	<1%
79	www.fastercapital.com Internet Source	<1%

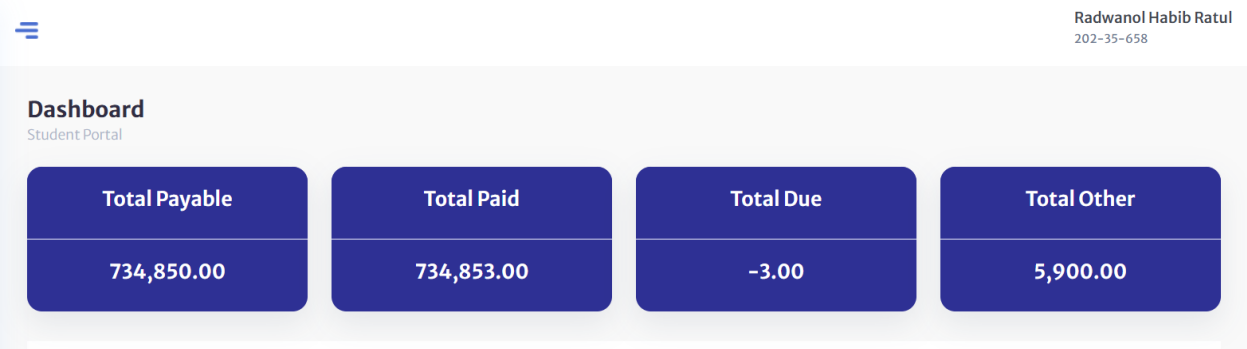
Exclude quotes	Off	Exclude matches	Off
Exclude bibliography	Off		

©Daffodil International University

Library Clearance

©Daffodil International University

Account Clearance



©Daffodil International University