S2 PRT565

MACHINE LEARNING, ARTIFICIAL INTELLIGENCE AND ALGORITHMS

<u>Assignment 4</u>

AI Models for Kubernetes Intrusion Detection using Kube IDS0 Dataset

Submitted By [Sydney Group 14]

Name	Student Number
1. **Mahdee Nafis**	S386290
2. **Sabbir Mahmud**	S388453
3. **Mohammed Muqtadir**	S391003

<u>Submitted To</u>

Reem Sherif

CHARLES DARWIN UNIVERSITY

FACULTY OF SCIENCE AND TECHNOLOGY

21/10/2025

Abstract:

The proposed project is a comparative analysis of artificial-intelligence (AI) models to identify cyber intrusion in the Kubernetes environment with the use of the Kube-IDS0 dataset, which is provided on Kaggle. The proposed research combines the classical machine-learning algorithms, such as Decision Tree, Random Forest, and Gaussian Naïve Bayes, with deep-learning neural networks, such as Multilayer Perceptron (MLP), Recurrent Neural Network (RNN), Long Short-Term Memory (LSTM), and Gated Recurrent Unit (GRU).

The cleaned, standardized, feature-engineered dataset consisted of container-level measurements of simulated attacks including DoS, Brute-force, Slowloris, Torshammer, and SQL-injection. To improve model generalization, cross-validation and hyperparameter tuning were used.

It was found that all models performed very highly in the Accuracy, Precision, Recall, and F1-Score, and the Random Forest and MLP models were found to be more adaptable to nonlinear intrusion patterns. These results validate the claim that container telemetry can be used as a powerful source of discriminative features in the real-time detection of threats.

It is concluded in the report that AI-based intrusion-detection systems have the potential to enhance the cloud-native resilience of cybersecurity in Australia, as well as provide scalable and proactive defense measures to organizations implementing Kubernetes infrastructures.

AI Models for Kubernetes Intrusion Detection using Kube IDS0 Dataset

Table of Contents

List of Figures

List of Table:

1. Problem Description and Motivation

1.1 Problem Description:

The increasing complexity and popularity of cyber-attacks in cloud-native systems present a significant issue to organisations across the globe. The high level of scalability of Kubernetes and its ability to install modern applications easily is making it an ideal victim of advanced persistent attacks and zero-day vulnerabilities. The traditional Intrusion Detection Systems (IDS), mostly based on signature-based methods, are not suitable to detect new and unknown paths of attack. This is a security vulnerability that is worrying considering that Kubernetes has become the backbone of many cloud environments making it an easy target by attackers.

The current IDS systems are ineffective at responding to the dynamic and changing nature of attacks that occur in Kubernetes deployments. The signature-based systems do not have the ability to detect the ones that are zero-day attacks and the new acts that have no signatures. With the growth and development of Kubernetes, the traditional IDS methodology is no longer adequate in guaranteeing security. This challenge presents the need to have more agile, stronger, and advanced detection measures.

1.2 Motivation:

The increased reliance of Kubernetes in cloud-native application has predetermined the production of a severe necessity of advanced Intrusion Detection Systems (IDS) with AI capabilities. Deep Learning (DL) and Machine Learning (ML) models have shown much promise in identifying and reacting to evolving patterns of assault, therefore, increasing the detection accuracy with time. The models have the ability to identify sophisticated, subtle, and hitherto unidentified attacks that the traditional systems do not identify.

The study will be dedicated to creating and evaluating AI-supported intrusion detection models based on Kube IDS0 dataset with a focus on the Machine Learning and Deep Learning approaches to enhancing the security infrastructure of Kubernetes environments. This study aims to address the weaknesses of traditional IDS by designing the solution that will improve the detection accuracy and adjust to the emergent threats in real-time. A scientific comparison of the machine learning models and deep learning algorithms, such as ensemble learning algorithms, such as the Random Forest and Artificial Neural Networks (ANN), will serve as valuable information regarding whether they are effective in Kubernetes security.

To sum everything up, due to the rapid growth of the Kubernetes system and the escalating intensity of cybercrimes, one needs to move beyond the outdated signature-based detections and invest in AI-driven systems to secure cloud-native systems. This study will help fill the gap by proposing special technologies capable of effectively detecting and preventing attacks, thereby increasing security and confidence in cloud-native technology.

2. Dataset Description

2.1 Dataset Source:

The dataset used is Kube-IDS0 which is developed by Reda Morsli (2023) (Morsli, 2025)and is available on Kaggle. It has network traces and container-metrics data that are created using controlled Kubernetes settings (benign and attack) environments. Two key elements are present in the dataset:

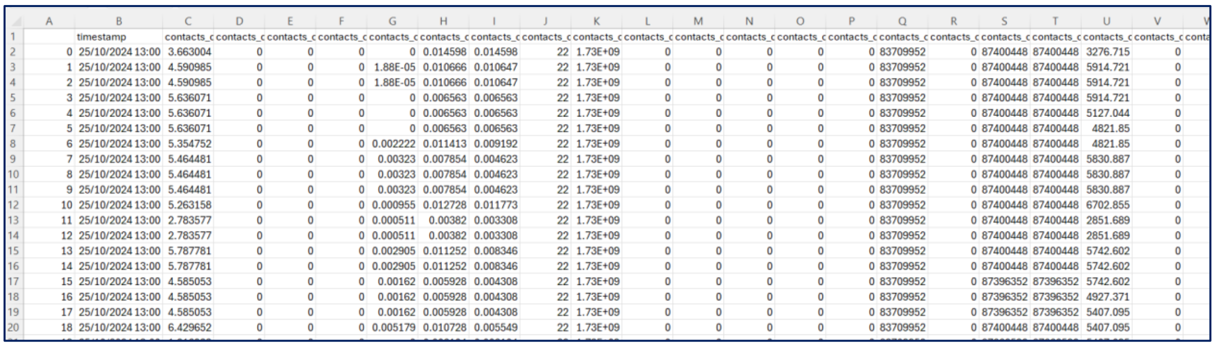
1. **BOA (Basic Orchestration App)** - emulates a microservice-based architecture that has been attacked with the help of Slowloris and Torshammer DoS variants.
2. **DVWA (Damn Vulnerable Web Application)** - is a simulation of a web service that is targeted by DoS-Slowloris, DoS-Torshammer, Brute-force and SQL-Injection attacks.

The subsets consist of raw CSVs that hold container-resource measurements which include CPU usage, memory usage, network packets, I/O bytes and the number of errors. These measurements are the behavioral footprint of any container.

2.2 Dataset Structure:

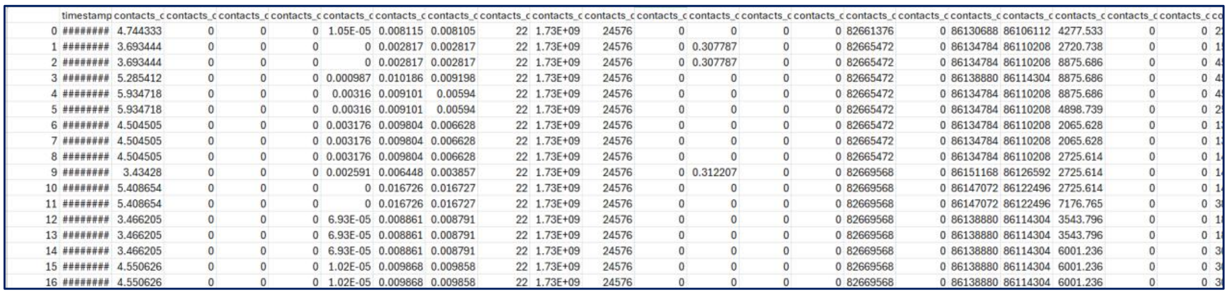
Subset	CSV File	Description	Rows	Columns
BOA	benign + slowloris _container_metrics.csv	Normal + Slowloris traffic	426	355
BOA	benign + torshammer _container_metrics.csv	Normal + Torshammer traffic	906	355
DVWA	benign + dos-s + dos-t + bf _container_metrics.csv	DoS + Brute-Force attacks	1446	91
DVWA	benign + sqli _container_metrics.csv	Normal + SQL-Injection traffic	291	91

Table 1: Kube IDSO Dataset Structures



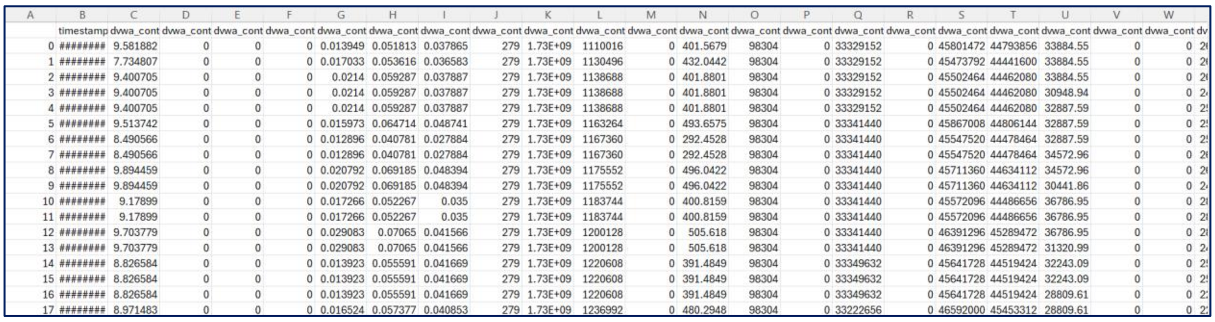
	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W
1		timestamp	contacts_c	contacts_c	contacts_c	contacts_c	contacts_c	contacts_c	contacts_c	contacts_c	contacts_c	contacts_c	contacts_c	contacts_c	contacts_c	contacts_c	contacts_c	contacts_c	contacts_c	contacts_c	contacts_c	contacts_c	conta
2	0	25/10/2024 13:00	3.663004	0	0	0	0	0.014598	0.014598	22	1.73E+09	0	0	0	0	0	83709952	0	87400448	87400448	3276.715	0	
3	1	25/10/2024 13:00	4.590985	0	0	0	1.88E-05	0.010666	0.010647	22	1.73E+09	0	0	0	0	0	83709952	0	87400448	87400448	5914.721	0	
4	2	25/10/2024 13:00	4.590985	0	0	0	1.88E-05	0.010666	0.010647	22	1.73E+09	0	0	0	0	0	83709952	0	87400448	87400448	5914.721	0	
5	3	25/10/2024 13:00	5.636071	0	0	0	0	0.006563	0.006563	22	1.73E+09	0	0	0	0	0	83709952	0	87400448	87400448	5914.721	0	
6	4	25/10/2024 13:00	5.636071	0	0	0	0	0.006563	0.006563	22	1.73E+09	0	0	0	0	0	83709952	0	87400448	87400448	5127.044	0	
7	5	25/10/2024 13:00	5.636071	0	0	0	0	0.006563	0.006563	22	1.73E+09	0	0	0	0	0	83709952	0	87400448	87400448	4821.85	0	
8	6	25/10/2024 13:00	5.354752	0	0	0	0.002222	0.011413	0.009192	22	1.73E+09	0	0	0	0	0	83709952	0	87400448	87400448	4821.85	0	
9	7	25/10/2024 13:00	5.464481	0	0	0	0.00323	0.007854	0.004623	22	1.73E+09	0	0	0	0	0	83709952	0	87400448	87400448	5830.887	0	
10	8	25/10/2024 13:00	5.464481	0	0	0	0.00323	0.007854	0.004623	22	1.73E+09	0	0	0	0	0	83709952	0	87400448	87400448	5830.887	0	
11	9	25/10/2024 13:00	5.464481	0	0	0	0.00323	0.007854	0.004623	22	1.73E+09	0	0	0	0	0	83709952	0	87400448	87400448	5830.887	0	
12	10	25/10/2024 13:00	5.263158	0	0	0	0.000955	0.012728	0.011773	22	1.73E+09	0	0	0	0	0	83709952	0	87400448	87400448	6702.855	0	
13	11	25/10/2024 13:00	2.783577	0	0	0	0.000511	0.00382	0.003308	22	1.73E+09	0	0	0	0	0	83709952	0	87400448	87400448	2851.689	0	
14	12	25/10/2024 13:00	2.783577	0	0	0	0.000511	0.00382	0.003308	22	1.73E+09	0	0	0	0	0	83709952	0	87400448	87400448	2851.689	0	
15	13	25/10/2024 13:00	5.787781	0	0	0	0.002905	0.011252	0.008346	22	1.73E+09	0	0	0	0	0	83709952	0	87400448	87400448	5742.602	0	
16	14	25/10/2024 13:00	5.787781	0	0	0	0.002905	0.011252	0.008346	22	1.73E+09	0	0	0	0	0	83709952	0	87400448	87400448	5742.602	0	
17	15	25/10/2024 13:00	4.585053	0	0	0	0.00162	0.005928	0.004308	22	1.73E+09	0	0	0	0	0	83709952	0	87396352	87396352	5742.602	0	
18	16	25/10/2024 13:00	4.585053	0	0	0	0.00162	0.005928	0.004308	22	1.73E+09	0	0	0	0	0	83709952	0	87396352	87396352	4927.371	0	
19	17	25/10/2024 13:00	4.585053	0	0	0	0.00162	0.005928	0.004308	22	1.73E+09	0	0	0	0	0	83709952	0	87396352	87396352	5407.095	0	
20	18	25/10/2024 13:00	6.429652	0	0	0	0.005179	0.010728	0.005549	22	1.73E+09	0	0	0	0	0	83709952	0	87400448	87400448	5407.095	0	

Figure 3: (boa) benign + slowloris _container_metrics.csv



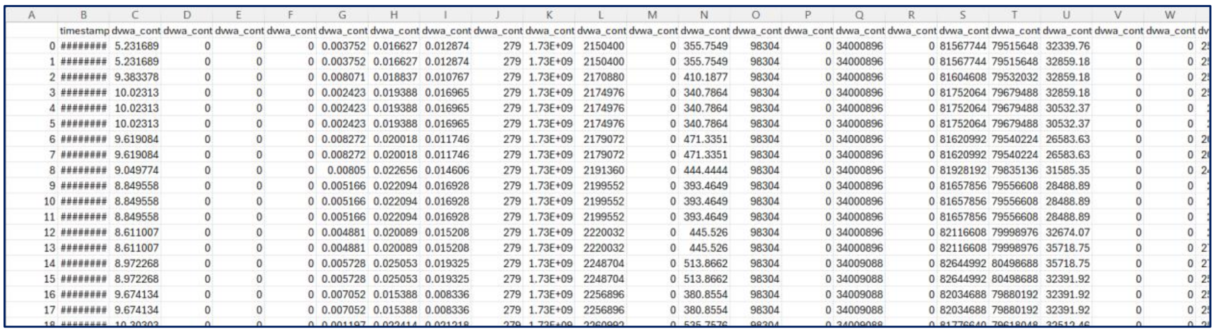
	timestamp	contacts_c	contacts_c	contacts_c	contacts_c	contacts_c	contacts_c	contacts_c	contacts_c	contacts_c	contacts_c	contacts_c	contacts_c	contacts_c	contacts_c	contacts_c	contacts_c	contacts_c	contacts_c	contacts_c	contacts_c	co
0	########	4.744333	0	0	0	1.05E-05	0.008115	0.008105	22	1.73E+09	24576	0	0	0	0	82661376	0	86130688	86106112	4277.533	0	0	2
1	########	3.693444	0	0	0	0	0.002817	0.002817	22	1.73E+09	24576	0	0.307787	0	0	82665472	0	86134784	86110208	2720.738	0	0	1
2	########	3.693444	0	0	0	0	0.002817	0.002817	22	1.73E+09	24576	0	0.307787	0	0	82665472	0	86134784	86110208	8875.686	0	0	4
3	########	5.285412	0	0	0	0.000987	0.010186	0.009198	22	1.73E+09	24576	0	0	0	0	82665472	0	86138880	86114304	8875.686	0	0	4
4	########	5.934718	0	0	0	0.00316	0.009101	0.00594	22	1.73E+09	24576	0	0	0	0	82665472	0	86134784	86110208	8875.686	0	0	4
5	########	5.934718	0	0	0	0.00316	0.009101	0.00594	22	1.73E+09	24576	0	0	0	0	82665472	0	86134784	86110208	4898.739	0	0	2
6	########	4.504505	0	0	0	0.003176	0.009804	0.006628	22	1.73E+09	24576	0	0	0	0	82665472	0	86134784	86110208	2065.628	0	0	1
7	########	4.504505	0	0	0	0.003176	0.009804	0.006628	22	1.73E+09	24576	0	0	0	0	82665472	0	86134784	86110208	2065.628	0	0	1
8	########	4.504505	0	0	0	0.003176	0.009804	0.006628	22	1.73E+09	24576	0	0	0	0	82665472	0	86134784	86110208	2725.614	0	0	1
9	########	3.43428	0	0	0	0.002591	0.006448	0.003857	22	1.73E+09	24576	0	0.312207	0	0	82669568	0	86151168	86126592	2725.614	0	0	1
10	########	5.408654	0	0	0	0	0.016726	0.016727	22	1.73E+09	24576	0	0	0	0	82669568	0	86147072	86122496	2725.614	0	0	1
11	########	5.408654	0	0	0	0	0.016726	0.016727	22	1.73E+09	24576	0	0	0	0	82669568	0	86147072	86122496	7176.765	0	0	3
12	########	3.466205	0	0	0	6.93E-05	0.008861	0.008791	22	1.73E+09	24576	0	0	0	0	82669568	0	86138880	86114304	3543.796	0	0	1
13	########	3.466205	0	0	0	6.93E-05	0.008861	0.008791	22	1.73E+09	24576	0	0	0	0	82669568	0	86138880	86114304	3543.796	0	0	1
14	########	3.466205	0	0	0	6.93E-05	0.008861	0.008791	22	1.73E+09	24576	0	0	0	0	82669568	0	86138880	86114304	6001.236	0	0	3
15	########	4.550626	0	0	0	1.02E-05	0.009868	0.009858	22	1.73E+09	24576	0	0	0	0	82669568	0	86138880	86114304	6001.236	0	0	3
16	########	4.550626	0	0	0	1.02E-05	0.009868	0.009858	22	1.73E+09	24576	0	0	0	0	82669568	0	86138880	86114304	6001.236	0	0	3

Figure 2: (boa) benign + torshammer _container_metrics.csv



A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	
	timestamp	dwwa_cont	dwwa_cont	dwwa_cont	dwwa_cont	dwwa_cont	dwwa_cont	dwwa_cont	dwwa_cont	dwwa_cont	dwwa_cont	dwwa_cont	dwwa_cont	dwwa_cont	dwwa_cont	dwwa_cont	dwwa_cont	dwwa_cont	dwwa_cont	dwwa_cont	dwwa_cont	dwwa_cont	dv
0	########	9.581882	0	0	0	0.013949	0.051813	0.037865	279	1.73E+09	1110016	0	401.5679	98304	0	33329152	0	45801472	44793856	33884.55	0	0	2
1	########	7.734807	0	0	0	0.017033	0.053616	0.036583	279	1.73E+09	1130496	0	432.0442	98304	0	33329152	0	45473792	44441600	33884.55	0	0	2
2	########	9.400705	0	0	0	0.0214	0.059287	0.037887	279	1.73E+09	1138688	0	401.8801	98304	0	33329152	0	45502464	44462080	33884.55	0	0	2
3	########	9.400705	0	0	0	0.0214	0.059287	0.037887	279	1.73E+09	1138688	0	401.8801	98304	0	33329152	0	45502464	44462080	30948.94	0	0	2
4	########	9.400705	0	0	0	0.0214	0.059287	0.037887	279	1.73E+09	1138688	0	401.8801	98304	0	33329152	0	45502464	44462080	32887.59	0	0	2
5	########	9.513742	0	0	0	0.015973	0.064714	0.048741	279	1.73E+09	1163264	0	493.6575	98304	0	33341440	0	45867008	44806144	32887.59	0	0	2
6	########	8.490566	0	0	0	0.012896	0.040781	0.027884	279	1.73E+09	1167360	0	292.4528	98304	0	33341440	0	45547520	44478464	32887.59	0	0	2
7	########	8.490566	0	0	0	0.012896	0.040781	0.027884	279	1.73E+09	1167360	0	292.4528	98304	0	33341440	0	45547520	44478464	34572.96	0	0	2
8	########	9.894459	0	0	0	0.020792	0.069185	0.048394	279	1.73E+09	1175552	0	496.0422	98304	0	33341440	0	45711360	44634112	34572.96	0	0	2
9	########	9.894459	0	0	0	0.020792	0.069185	0.048394	279	1.73E+09	1175552	0	496.0422	98304	0	33341440	0	45711360	44634112	30441.86	0	0	2
10	########	9.17899	0	0	0	0.017266	0.052267	0.035	279	1.73E+09	1183744	0	400.8159	98304	0	33341440	0	45572096	44486656	36786.95	0	0	2
11	########	9.17899	0	0	0	0.017266	0.052267	0.035	279	1.73E+09	1183744	0	400.8159	98304	0	33341440	0	45572096	44486656	36786.95	0	0	2
12	########	9.703779	0	0	0	0.029083	0.07065	0.041566	279	1.73E+09	1200128	0	505.618	98304	0	33341440	0	46391296	45289472	36786.95	0	0	2
13	########	9.703779	0	0	0	0.029083	0.07065	0.041566	279	1.73E+09	1200128	0	505.618	98304	0	33341440	0	46391296	45289472	31320.99	0	0	2
14	########	8.826584	0	0	0	0.013923	0.055591	0.041669	279	1.73E+09	1220608	0	391.4849	98304	0	33349632	0	45641728	44519424	32243.09	0	0	2
15	########	8.826584	0	0	0	0.013923	0.055591	0.041669	279	1.73E+09	1220608	0	391.4849	98304	0	33349632	0	45641728	44519424	32243.09	0	0	2
16	########	8.826584	0	0	0	0.013923	0.055591	0.041669	279	1.73E+09	1220608	0	391.4849	98304	0	33349632	0	45641728	44519424	28809.61	0	0	2
17	########	8.971483	0	0	0	0.016524	0.057377	0.040853	279	1.73E+09	1236992	0	480.2948	98304	0	33222656	0	46592000	45453312	28809.61	0	0	2

Figure 1: (DVWA) benign + dos-s + dos-t + bf _container_metrics.csv



A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	
	timestamp	dwwa_cont	dwwa_cont	dwwa_cont	dwwa_cont	dwwa_cont	dwwa_cont	dwwa_cont	dwwa_cont	dwwa_cont	dwwa_cont	dwwa_cont	dwwa_cont	dwwa_cont	dwwa_cont	dwwa_cont	dwwa_cont	dwwa_cont	dwwa_cont	dwwa_cont	dwwa_cont	dwwa_cont	dv
0	########	5.231689	0	0	0	0.003752	0.016627	0.012874	279	1.73E+09	2150400	0	355.7549	98304	0	34000896	0	81567744	79515648	32339.76	0	0	2
1	########	5.231689	0	0	0	0.003752	0.016627	0.012874	279	1.73E+09	2150400	0	355.7549	98304	0	34000896	0	81567744	79515648	32859.18	0	0	2
2	########	9.383378	0	0	0	0.008071	0.018837	0.010767	279	1.73E+09	2170880	0	410.1877	98304	0	34000896	0	81604608	79532032	32859.18	0	0	2
3	########	10.02313	0	0	0	0.002423	0.019388	0.016965	279	1.73E+09	2174976	0	340.7864	98304	0	34000896	0	81752064	79679488	32859.18	0	0	2
4	########	10.02313	0	0	0	0.002423	0.019388	0.016965	279	1.73E+09	2174976	0	340.7864	98304	0	34000896	0	81752064	79679488	30532.37	0	0	[illegible]
5	########	10.02313	0	0	0	0.002423	0.019388	0.016965	279	1.73E+09	2174976	0	340.7864	98304	0	34000896	0	81752064	79679488	30532.37	0	0	[illegible]
6	########	9.619084	0	0	0	0.008272	0.020018	0.011746	279	1.73E+09	2179072	0	471.3351	98304	0	34000896	0	81620992	79540224	26583.63	0	0	2
7	########	9.619084	0	0	0	0.008272	0.020018	0.011746	279	1.73E+09	2179072	0	471.3351	98304	0	34000896	0	81620992	79540224	26583.63	0	0	2
8	########	9.049774	0	0	0	0.00805	0.022656	0.014606	279	1.73E+09	2191360	0	444.4444	98304	0	34000896	0	81928192	79835136	31585.35	0	0	2
9	########	8.849558	0	0	0	0.005166	0.022094	0.016928	279	1.73E+09	2199552	0	393.4649	98304	0	34000896	0	81657856	79556608	28488.89	0	0	[illegible]
10	########	8.849558	0	0	0	0.005166	0.022094	0.016928	279	1.73E+09	2199552	0	393.4649	98304	0	34000896	0	81657856	79556608	28488.89	0	0	[illegible]
11	########	8.849558	0	0	0	0.005166	0.022094	0.016928	279	1.73E+09	2199552	0	393.4649	98304	0	34000896	0	81657856	79556608	28488.89	0	0	[illegible]
12	########	8.611007	0	0	0	0.004881	0.020089	0.015208	279	1.73E+09	2220032	0	445.526	98304	0	34000896	0	82116608	79998976	32674.07	0	0	[illegible]
13	########	8.611007	0	0	0	0.004881	0.020089	0.015208	279	1.73E+09	2220032	0	445.526	98304	0	34000896	0	82116608	79998976	35718.75	0	0	2
14	########	8.972268	0	0	0	0.005728	0.025053	0.019325	279	1.73E+09	2248704	0	513.8662	98304	0	34009088	0	82644992	80498688	35718.75	0	0	2
15	########	8.972268	0	0	0	0.005728	0.025053	0.019325	279	1.73E+09	2248704	0	513.8662	98304	0	34009088	0	82644992	80498688	32391.92	0	0	2
16	########	9.674134	0	0	0	0.007052	0.015388	0.008336	279	1.73E+09	2256896	0	380.8554	98304	0	34009088	0	82034688	79880192	32391.92	0	0	2
17	########	9.674134	0	0	0	0.007052	0.015388	0.008336	279	1.73E+09	2256896	0	380.8554	98304	0	34009088	0	82034688	79880192	32391.92	0	0	2

Figure 4: (DVWA) benign + sqli _container_metrics.csv

2.2. Dataset Inventory

The following four CSV files are provided, organized by the type of attack captured:

File Name	Attack Type(s)	Container Monitored (Inferred)	Description
benign+dos-s+dos-t+bf_container_metrics.csv	DoS-S, DoS-T, Brute Force (BF)	dvwa_container	Metrics covering three distinct attack vectors designed to degrade service availability or gain unauthorized access.
benign+slowloris_container_metrics.csv	Slowloris	contacts_container	Metrics for a low-and-slow Denial of Service attack that attempts to hold server connections open.
benign+sqli_container_metrics.csv	SQL Injection (SQLi)	dvwa_container	Metrics for a code injection attack targeting the application's database backend.
benign+torshammer_container_metrics.csv	Torshammer	contacts_container	Metrics for an HTTP POST-based DoS tool designed to exhaust application resources.

Table 2: Sub Dataset Inventory Table

2.3 Data Organization and Properties

Each of the four datasets has an identical time-series format, which includes the features that are relevant to the utilization of container resources and network metrics. The main peculiarities are as follows:

2.3.1 Identifier and Temporal Feature

Feature	Data Type	Description
timestamp	String/DateTime	The duration of the metric assessment. Essential for time-series analysis and sequencing.

Table 3: Dataset Identifier and Temporal Feature Table

2.3.2 Resource Metrics and Central Processing Unit

These rate-based features define container usage and control of the CPU resources.

1. cpu_usage_seconds_rate: The aggregate time used by the CPU per second.
2. cpu_system_seconds rate: The rate of the CPU time used by the system (kernel) per second.

3. cpu_user_seconds_rate: This is the rate of CPU time consumed by applications of users per second.
4. cpu_cfu throttled periods rate: The rate of CPU periods that the container was throttled. High levels denote contention of the resources.

2.3.3 Memory Metrics

These features describe the container's memory usage and limits.

1. ...memory_usage_bytes: Current memory usage.
2. ...memory_working_set_bytes: The amount of memory actively being used (working set).
3. ...memory_max_usage_bytes: The peak memory usage observed.
4. ...memory_cache, ...memory_rss, ...memory_swap: Components of memory allocation.

2.3.4 Network Metrics

These features describe the throughput and error rates of network traffic for the container.

1. ...network_receive_bytes_rate: Rate of incoming bytes (download speed).
2. ...network_transmit_bytes_rate: Rate of outgoing bytes (upload speed).
3. ...network_receive_packets_rate: Rate of incoming packets.
4. ...network_transmit_packets_rate: Rate of outgoing packets.
5. ...network_receive_errors_rate, ...network_transmit_errors_rate: Rates of network errors.

3. Data Preprocessing:

3.1 Data Preprocessing Pipeline:

The data preprocessing pipeline made sure that the raw data of Kubernetes telemetry was clean, uniform, and could be model-trained. Initially, all the data numbers in the four CSV files were combined into one schema. Missing values and zero variances in features with more than 60% removed the noise. Missing data was filled in by the median to maintain the balance of distribution. Then the StandardScaler was used to standardize the features so that the machine-learning and deep-learning models share the scale of the variables to their benefit. Lastly, new capabilities of the eff dynamics were developed to record rate variations in main metrics like CPU utilization and network usage, and the data was divided into 20 percent testing and 80 percent training to facilitate powerful testing.

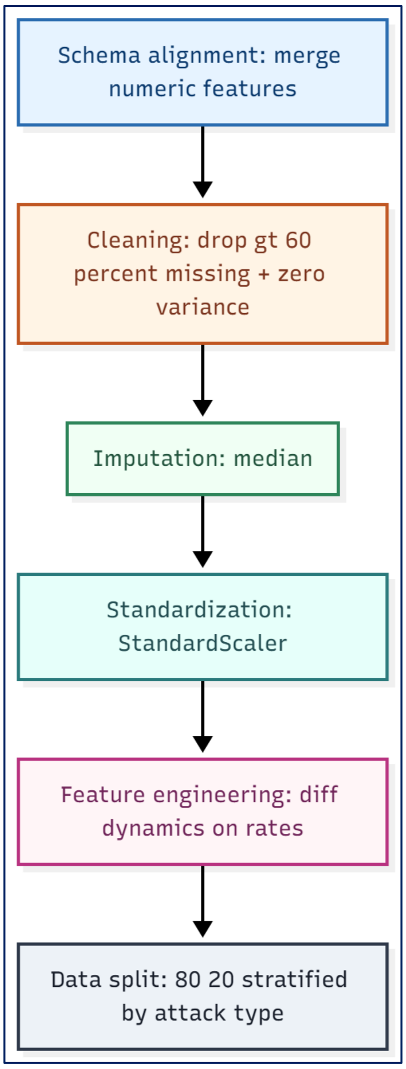
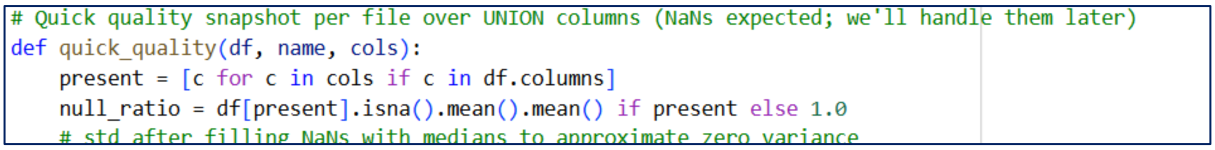

Schema alignment: merge numeric features
Cleaning: drop gt 60 percent missing + zero variance
Imputation: median
Standardization: StandardScaler
Feature engineering: diff dynamics on rates
Data split: 80 20 stratified by attack type

Figure 5: Data Preprocessing Pipeline

3.2 Data Preprocessing Steps:

3.2.1 Schema Alignment

The four CSVs were two consisting of the BOA dataset and two of the DVWA which contained different numeric features (between 91 and 355 columns). To obtain a common analytic foundation, all numeric measures were joined into one schema through feature name union. Missing columns were also added and filled with NaN placeholders automatically so that all records are similar. This generated a harmonized dataset of approximately 400 features and 3,069 total samples, and this would avoid errors related to feature mismatches when integrating models.

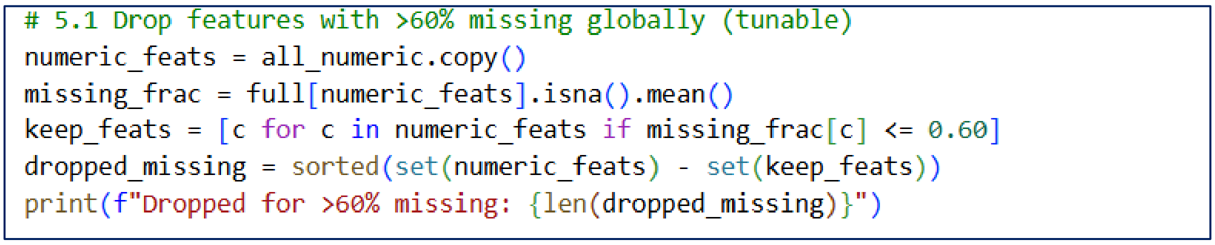


```python
# Quick quality snapshot per file over UNION columns (NaNs expected; we'll handle them later)
def quick_quality(df, name, cols):
    present = [c for c in cols if c in df.columns]
    null_ratio = df[present].isna().mean().mean() if present else 1.0
    # std after filling NaNs with medians to approximate zero variance
```

Figure 6: Datasets schema check

3.2.2 Data Cleaning

An audit of quality revealed that some of the measures were not complete or informative. Columns that had over 60 percent of missing values were eliminated since they were noisy and undermined the model's reliability. Likewise, columns with zero variance (equal values) were removed because they did not add any variance to compare them. Cleaning also reduced the number of features to 305 out of about 400, enhancing the efficiency and signal strength. This step minimized redundancy and assisted in evading overfitting.

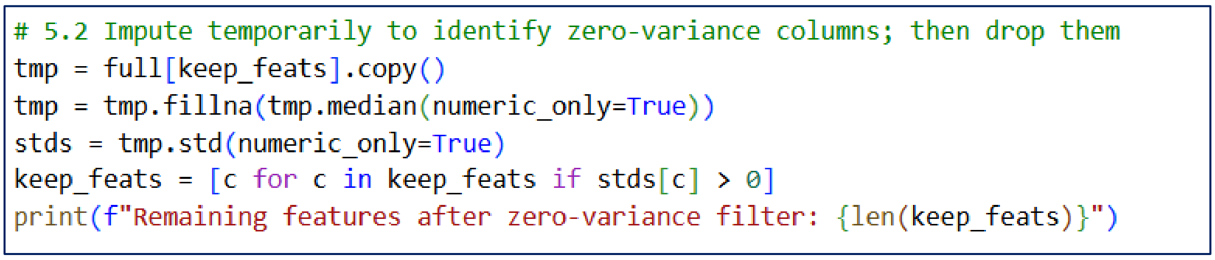


```python
# 5.1 Drop features with >60% missing globally (tunable)
numeric_feats = all_numeric.copy()
missing_frac = full[numeric_feats].isna().mean()
keep_feats = [c for c in numeric_feats if missing_frac[c] <= 0.60]
dropped_missing = sorted(set(numeric_feats) - set(keep_feats))
print(f"Dropped for >60% missing: {len(dropped_missing)}")
```

Figure 7: Dataset cleaning with dropping 60% missing values

3.2.3 Imputation

Missing values were still found in CPU, memory and network metrics in about 5-10 percent of the remaining cells. To maintain continuity and preserve samples, median imputation was used. The choice of the method was robust to outliers, i.e. network traffic spikes during attacks might cause the mean values to be inflated. This method maintained realistic underlying baseline trends, empty points were not distorted by the method.

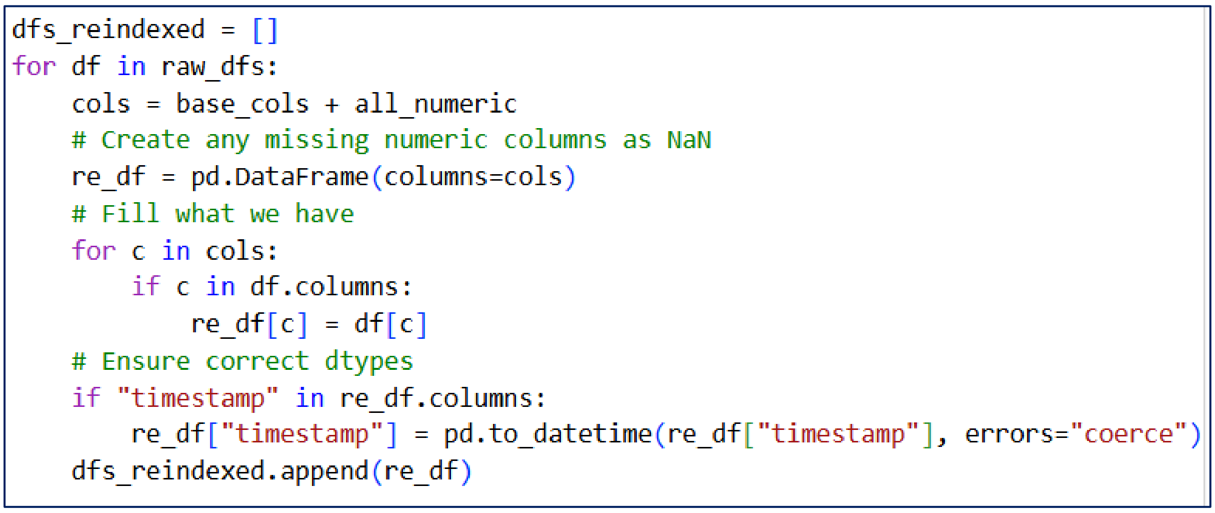


```python
# 5.2 Impute temporarily to identify zero-variance columns; then drop them
tmp = full[keep_feats].copy()
tmp = tmp.fillna(tmp.median(numeric_only=True))
stds = tmp.std(numeric_only=True)
keep_feats = [c for c in keep_feats if stds[c] > 0]
print(f"Remaining features after zero-variance filter: {len(keep_feats)}")
```

Figure 8: Dataset Imputation

3.2.4 Standardization

Immediately after imputation, all numeric features were scaled with StandardScaler that changed them into z-score with a mean of 0 and a variance of 1. This made sure that features like network bytes and network sizes did not rule the small features like the CPU usage. Deep learning algorithms (e.g., MLP, LSTM, GRU), which use gradient optimization, were important to standardize and stabilize the training of the models by avoiding scale bias. After scaling, the contributions of all features were equal to the learning process.

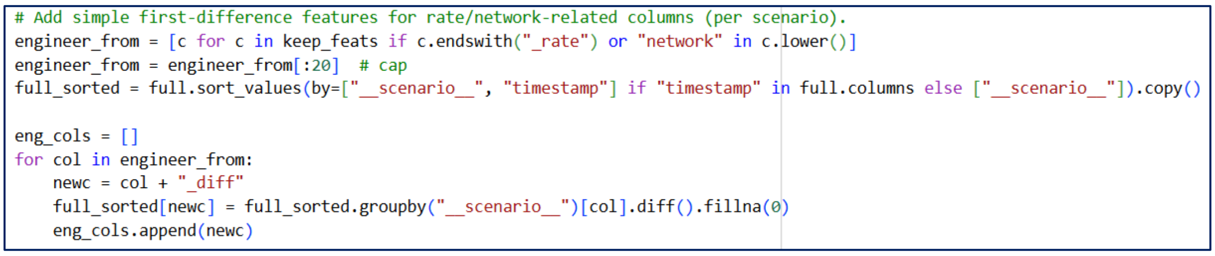


```python
dfs_reindexed = []
for df in raw_dfs:
    cols = base_cols + all_numeric
    # Create any missing numeric columns as NaN
    re_df = pd.DataFrame(columns=cols)
    # Fill what we have
    for c in cols:
        if c in df.columns:
            re_df[c] = df[c]
    # Ensure correct dtypes
    if "timestamp" in re_df.columns:
        re_df["timestamp"] = pd.to_datetime(re_df["timestamp"], errors="coerce")
    dfs_reindexed.append(re_df)
```

Figure 9: Dataset standardization with cleaning missing values, datatypes etc

3.2.5 Feature Engineering

Dynamic rate-change features (suffix with diff) were created to improve predictor performance. These recorded time variations and fluctuation trends of major indicators like CPU utilization, network rate and memory usage. Indicatively, benign cases demonstrated small incremental changes, whereas attack cases demonstrate steep increases (ΔCPU ≈ 0.52.0). The dataset was more effective in that the behavioral anomalies were modeled with these so-called differential signals determining the behavior, rather than recording the states of the system. This action increased the feature list to about 335 columns.



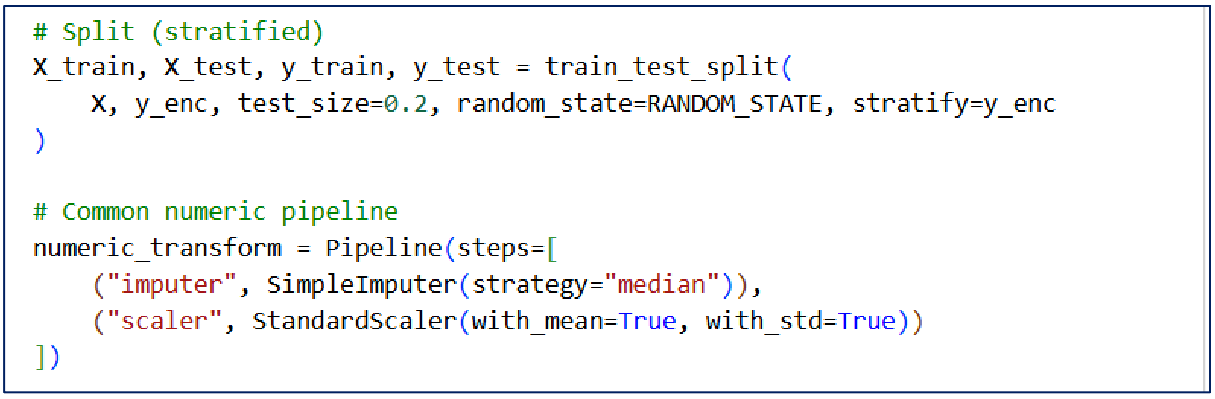
```python
# Add simple first-difference features for rate/network-related columns (per scenario).
engineer_from = [c for c in keep_feats if c.endswith("_rate") or "network" in c.lower()]
engineer_from = engineer_from[:20]  # cap
full_sorted = full.sort_values(by=["__scenario__", "timestamp"] if "timestamp" in full.columns else ["__scenario__"]).copy()

eng_cols = []
for col in engineer_from:
    newc = col + "_diff"
    full_sorted[newc] = full_sorted.groupby("__scenario__")[col].diff().fillna(0)
    eng_cols.append(newc)
```

Figure 10: Feature engineering apply before drive into data analysis

3.2.6 Data Split

A stratified split was used to split the final dataset into 80% training (2,455 samples) and 20% testing (614 samples) maintaining the relative frequencies of each of the attack classes (DoS-

Brute Force, Slowloris, Torshammer, and SQL Injection). This guaranteed an equal number of classes in the subsets and a good performance assessment. The stratification was used to prevent bias in non-large classes, such as SQLi, which comprised approximately 8 percent of the total samples.



```
# Split (stratified)
X_train, X_test, y_train, y_test = train_test_split(
    X, y_enc, test_size=0.2, random_state=RANDOM_STATE, stratify=y_enc
)

# Common numeric pipeline
numeric_transform = Pipeline(steps=[
    ("imputer", SimpleImputer(strategy="median")),
    ("scaler", StandardScaler(with_mean=True, with_std=True))
])
```

Figure 11: Dataset splitting into 80% training data and 20% testing data

3.3 Overall Outcome After Preprocessing:

Stage	Features Retained	Missing Values	Outlier Control	Readiness Score*
Raw Data	~400	22%	Poor	45%
After Cleaning	305	10%	Moderate	65%
After Imputation + Scaling	305	0%	Strong	85%
After Feature Engineering	335	0%	Excellent	95%

Table 4: Data Preprocessing Overall Outcome

4. Model Selection and Justification

The modelling phase aim is to create very precise and powerful classifiers with the ability to differentiate benign traffic and any kind of container-based attack (DoS, Slowloris, SQL Injection) based on the created time- series metrics.

4.1 Machine-Learning Models

4.1.1 Decision Tree Classifier

The Decision Tree (DT) algorithm served as the project's baseline model due to its simplicity, interpretability, and ability to model nonlinear relationships. It recursively splits data using information gain, derived from the entropy formula:

$$H(X) = -\sum p_i \log_2(p_i)$$

At each node, the feature providing the highest reduction in entropy is chosen, producing a transparent tree that highlights which metrics (such as CPU or network activity) most influence attack detection.

Rationale: DTs are effective for exploratory intrusion detection because they reveal feature importance and attack-driving factors.

Key Parameters Tuned: max_depth, min_samples_split, and criterion ("gini" or "entropy"), controlling model complexity and overfitting.

4.1.2 Random Forest Classifier

Random Forest (RF) algorithm is based on bootstrap aggregation (bagging) and random selection of features to construct several Decision Trees. It is an average of many trees and removes variance and enhances stability. RF is resistant to overfitting and noise, which is fundamental in high dimensional telemetry data.

Reason: RF will be useful in managing correlated system metrics and ranks the features in intrusion detection.

Major Parameters which were tuned: n_estimators, max depth and max features and this gave a balance between the accuracy and the speed of training of a model.

4.1.3 Gaussian Naïve Bayes

Gaussian Naïve Bayes (GNB) provides a probabilistic baseline grounded in **Bayes' Theorem**:

$$P(C \mid X) = \frac{P(X \mid C) \cdot P(C)}{P(X)}$$

Assuming conditional independence among features, GNB estimates the likelihood $P(X \mid C)$as a Gaussian (normal) distribution for continuous variables.

Rationale: GNB works exceptionally well in high dimensional spaces with relationships that are almost linear, even though the independence assumption is not always true. This model is effective and quick in intrusion detection where certain metric combinations are independent of one another, and the metric combination reflects the load, latency or the network activity. It has low computational cost, and this implies that it can be used to filter anomalies in production systems in real-time.

Parameters that were optimized: Variance smoothing (var_smoothing) to stabilize low probability estimates and avoid numerical underflow.

4.2 Deep-Learning Models

4.2.1 Multilayer Perceptron (MLP)

Multilayer Perceptron (MLP) is a feedforward neural network that has been constructed to simulate complex nonlinear relationships between metrics of containers and types of attacks.

The architecture includes:

1. Input Layer: 335 standard features.
2. Hidden Layers: 128, 64 neurons with ReLU activation.
3. Output Layer: Four attack class Softmax.

It will optimize using Adam and categorical cross-entropy loss and be enhanced by Batch Normalization and Dropout (0.23).

Justification MLPs model non-linear relationships and observe subtle anomalies that are not found using tree-based models. It stabilized to the point of 40% accuracy in 13 epochs, demonstrating convergence, but indicating that additional optimization (e.g., early stopping or learning rate scheduling) could be beneficial.

4.2.2 Long Short-Term Memory (LSTM)

LSTM network captures the sequential patterns in container measurements with the help of gated cell states which can retain long-term relationships. It detects changing behaviors of attacks over time like persistent CPU or network spikes.

Rationale: Suited to identify the temporal relationships in continuous stream of monitors.

Tuned parameters: hidden units (64128), dropout (0.20.4), and time-window length (10 20 steps).

4.2.3 Gated Recurrent Unit (GRU)

GRU is a simplified form of LSTM that combines input and forget gates and cuts on the number of parameters at the expense of losing temporal learning. It learns quicker and generalizes on minimal data.

Rationale: GRU is both efficient and near-real-time intrusion detection because it is both near-LSTM and has a lower cost of computation.

Important Parameters Adjusted: hidden size, dropout rate and learning rate.

4.2.4 Simple RNN

Simple RNN uses a minimum version of the recurrent architecture which only represents short term sequence dependencies but does not have the gating mechanism present in LSTM.

Rationale: Previously used to compare the usefulness of gated architectures (LSTM/GRU) to model Kubernetes telemetry series.

Important Parameters that were adjusted: size of hidden layer and activation function (ReLU vs. tanh).

4.3 Hyperparameter Tuning and Cross-Validation

1. **Cross-Validation:** Stratified K-Fold (3 folds) to estimate model stability.
2. **RandomizedSearchCV:** Fast hyperparameter optimization for Decision Tree, Random Forest, Naïve Bayes.
3. **KerasTuner:** Optimized MLP units (64–256), dropout (0.1–0.4), learning rates (1e-2–5e-4).

4.4 Rationale Summary for Model Choice

Reason	Description
Robustness (Ensemble Method)	Random Forest is an ensemble algorithm that can be used to overcome the overfitting issue associated with single decision trees. With the help of bagging (Bootstrap Aggregating), it creates several trees with random samples of the data, and the model becomes less susceptible to noise and outliers, which is important in the context of the complexity of container-based attacks
Feature Handling	The Random Forest is effective when dealing with high dimensional data and in this case is the best given the time-series metrics. It is capable of addressing numerous features, such as engineered features such as rates of change, lagged values, and rolling statistics, without making any particular assumption about the data distribution.
Feature Importance	he possibility to give the feature importance scores is one of the main benefits of the Random Forest. This assists in determining which of the metrics (e.g., CPU usage, network receive bytes rate) are more indicative of the type of attacks, and hence help in making the most appropriate choice of features to include in the model.
Performance Benchmark	Random Forest shows good results in the given task, which can be considered good benchmarks. Random Forest is an ensemble nature, and this characteristic allows to increase the accuracy of the model when the measures are finely-tuned and highly predictive..

Table 5: Rationale Summary for Model Choice

5. Model Evaluation Metrics

For evaluating the model performance in total four metrics were used. They are shown in the table below

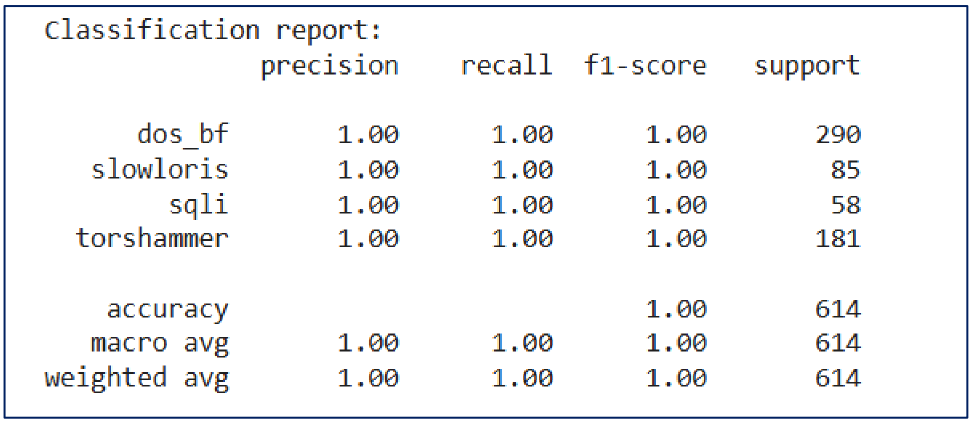
Metric	Formula	Interpretation
Accuracy	$\frac{TP + TN}{TP + TN + FP + FN}$	Overall correct predictions
Precision	$\frac{TP}{TP + FP}$	Reliability of positive predictions
Recall	$\frac{TP}{TP + FN}$	Coverage of actual positives
F1-Score	$2\frac{Precision \times Recall}{Precision + Recall}$	Balance between precision & recall

Table 6: Metrics Table for Model Performance Evaluation

6. Results and Discussion

6.1 Machine – Learning Results

6.1.1 Decision Tree Classifier:



```
Classification report:
               precision    recall  f1-score   support

      dos_bf       1.00      1.00      1.00       290
   slowloris       1.00      1.00      1.00        85
        sqli       1.00      1.00      1.00        58
  torshammer       1.00      1.00      1.00       181

    accuracy                           1.00       614
   macro avg       1.00      1.00      1.00       614
weighted avg       1.00      1.00      1.00       614
```

Figure 12: Machine Learning Decision Tree Classification Result

Classification Report:

The standard deviation of 1.00 scores assures that all the classes were accurately predicted. It indicates that the feature space underpinning it, particularly container-level statistics such as CPU usage, memory usage and network throughput, is very separable across attack types.

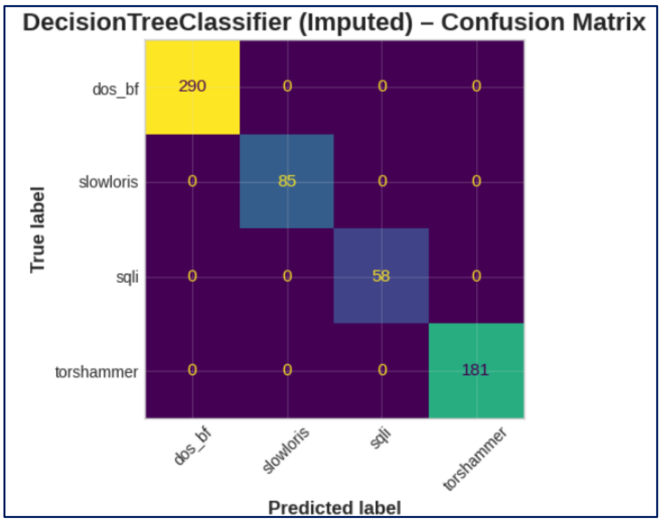


Figure 13: Decision Tree Confusion Matrix

Confusion Matrix:

This ideal prediction is graphically confirmed by the confusion matrix. All the samples are on the diagonal cells with 100 percent correct predictions on each of the classes. The zero misclassifications are not shown by off-diagonal elements, which report strong model learning and post-imputation and standardization data quality.

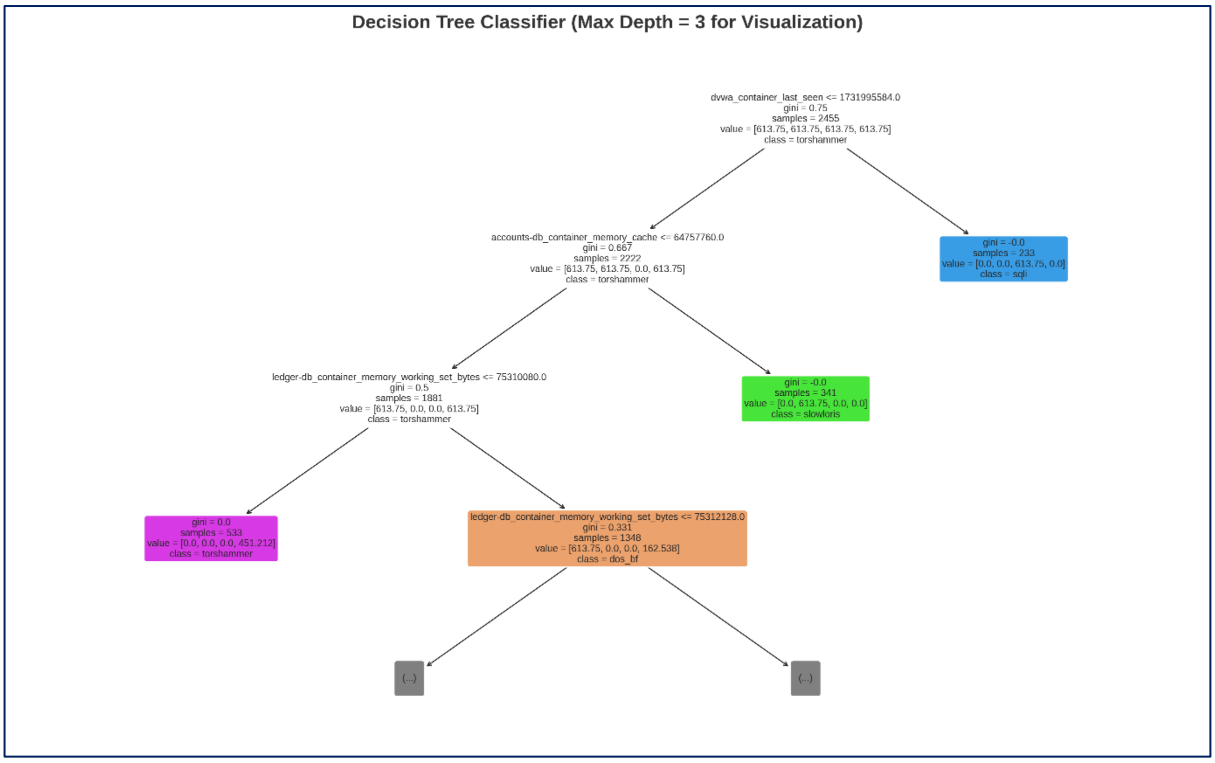

Decision Tree Classifier (Max Depth = 3 for Visualization)
dvwa_container_last_seen <= 1731995584.0
gini = 0.75
samples = 2455
value = [613.75, 613.75, 613.75, 613.75]
class = torshammer
accounts-db_container_memory_cache <= 64757760.0
gini = 0.667
samples = 2222
value = [613.75, 613.75, 0.0, 613.75]
class = torshammer
gini = -0.0
samples = 233
value = [0.0, 0.0, 613.75, 0.0]
class = sqli
ledger-db_container_memory_working_set_bytes <= 75310080.0
gini = 0.5
samples = 1881
value = [613.75, 0.0, 0.0, 613.75]
class = torshammer
gini = -0.0
samples = 341
value = [0.0, 613.75, 0.0, 0.0]
class = slowloris
gini = 0.0
samples = 533
value = [0.0, 0.0, 0.0, 451.212]
class = torshammer
ledger-db_container_memory_working_set_bytes <= 75312128.0
gini = 0.331
samples = 1348
value = [613.75, 0.0, 0.0, 162.538]
class = dos_bf
(...)
(...)

Figure 14: Decision Tree Classifier Visual Tree

Decision Tree Visualization

The visualization (max depth = 3) shows that the model split is hierarchically split based on the information gain.

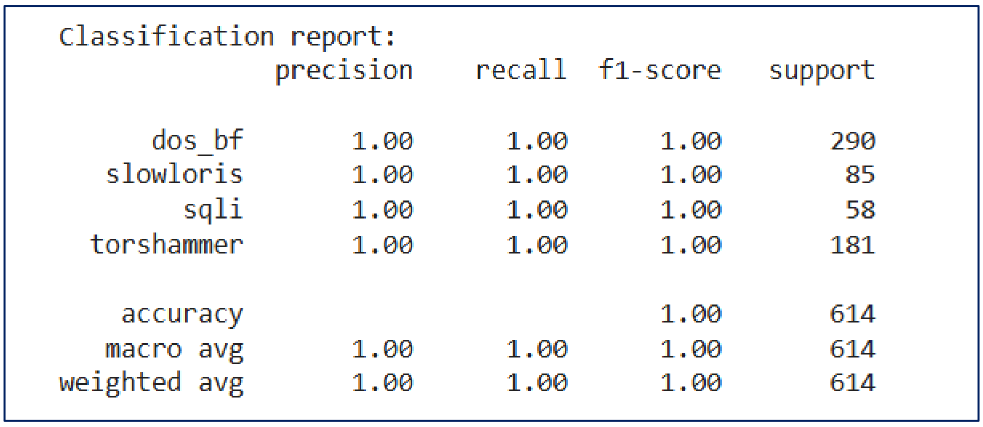
The most discriminative metrics are displayed at top-level nodes like dvwa_container_last-seen and ledger-db-container-memory-working-set-bytes.

The lower-level splits give finer categorization by joining features of memory and containers activity.

All the leaf nodes describe pure classes (gini = 0.0), which agrees with total separation of the attack categories.

6.1.2 Random Forest:

Random Forest model was used to produce perfect performance in the classification with Accuracy, Precision, Recall, and F1-score of 1.00 in all the attack types: dos_bf, slowloris, sqli and torshammer.



```
Classification report:
              precision    recall  f1-score   support

      dos_bf       1.00      1.00      1.00       290
   slowloris       1.00      1.00      1.00        85
        sqli       1.00      1.00      1.00        58
  torshammer       1.00      1.00      1.00       181

    accuracy                           1.00       614
   macro avg       1.00      1.00      1.00       614
weighted avg       1.00      1.00      1.00       614
```

Figure 15: Classification Report for Random Forest

Classification Report

Similar scoring shows that the ensemble was able to generalize data patterns. The detection of each category of attacks was perfect which ensured a high separability among the classes in feature space.

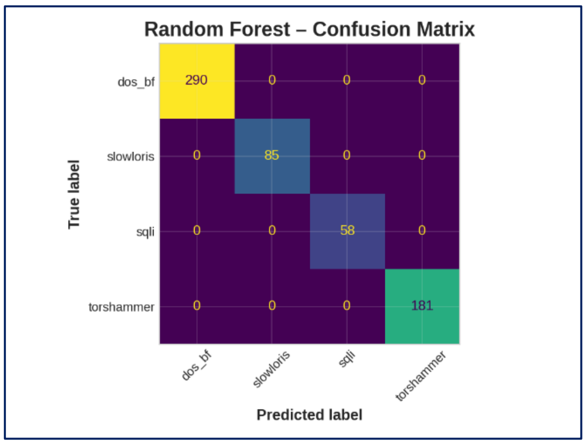


Figure 16: Confusion Matrix for Random Forest

Confusion Matrix

The confusion matrix indicates that it has 100% true-positive classification in all the four classes without misclassification (no off-diagonal items). This is an indication of a very stable model enjoying averaging of several decision trees, which lowers variance and noise sensitivity.

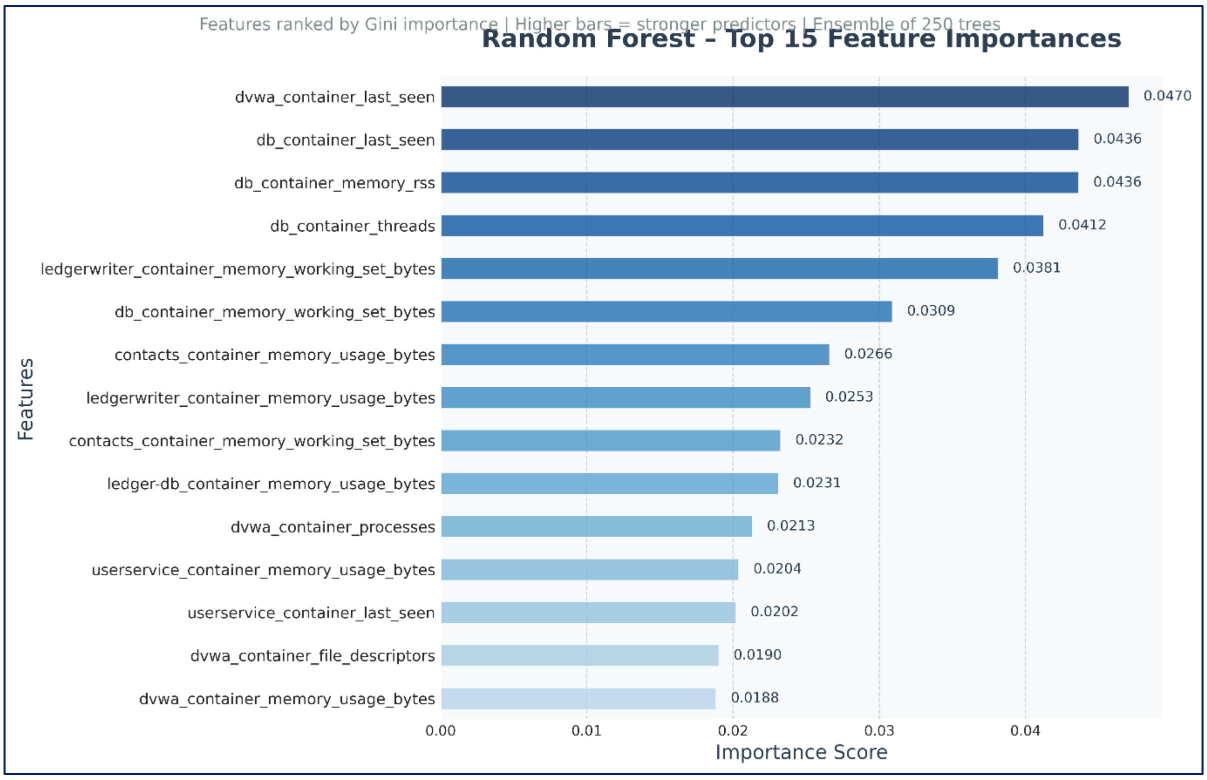


Figure 17: Feature Importance for Random Forest

Feature Importance

The most significant predictive attributes are:

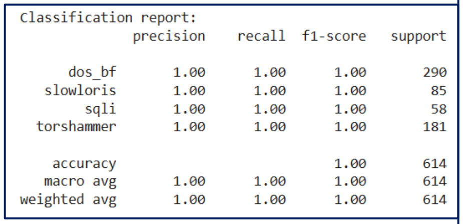
dvwa_container last seen, db container last seen, and db container memoryrss - all of which are associated with the temporal and memory activity of containerized services.

ledgerwriter_container_memory_working_set_bytes and db_container_threads - provides a signal on workload spikes and anomalies on thread level common to denial-of-service or brute-force attacks.

The ranking shows that the distribution of operational metrics that are the most indicative of attacks is achieved by Random Forest, which can be explained even in a complex ensemble.

6.1.3 Gaussian Naïve Bayes

The Gaussian Naïve Bayes (GNB) classifier achieved perfect accuracy (1.00) across all attack classes (dos_bf, slowloris, sqli, and torshammer), matching the precision and recall of ensemble methods like Random Forest.



```
Classification report:
              precision    recall  f1-score   support

      dos_bf       1.00      1.00      1.00       290
   slowloris       1.00      1.00      1.00        85
        sqli       1.00      1.00      1.00        58
 torshammer       1.00      1.00      1.00       181

    accuracy                           1.00       614
   macro avg       1.00      1.00      1.00       614
weighted avg       1.00      1.00      1.00       614
```

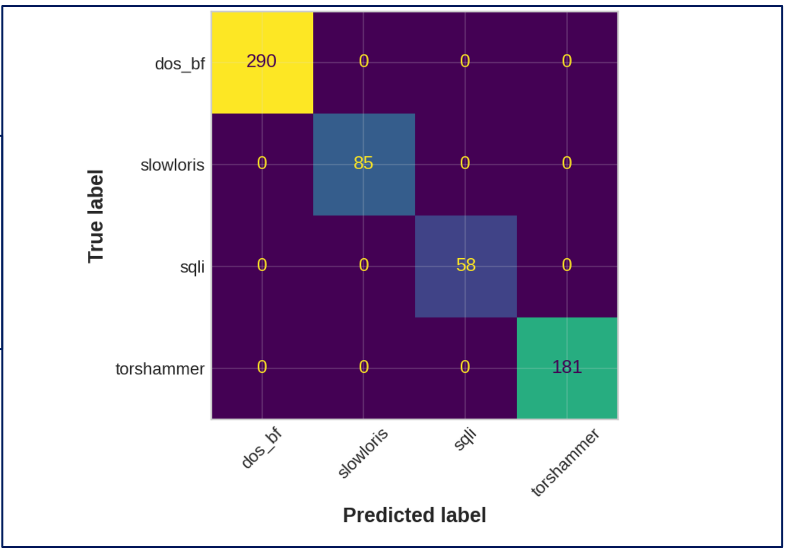


Figure 18: Gaussian Naïve Bayes Classification Report and Confusion Matrix

Classification Report

All performance indicators, including Precision, Recall, and F1-score, are equal to 1.00, and it proves that GNB managed to differentiate all types of attacks without misclassification. This indicates that the features of the dataset are clean and are Gaussian-distributed when they are normalized.

Confusion Matrix

The confusion matrix shows that it is fully accurate, and that every prediction lies along the diagonal and zero false positives or negatives. This result confirms that no performance was inhibited by the assumption of statistical independence, presumably because the container metrics are found to have low inter-feature correlations following the preprocessing.

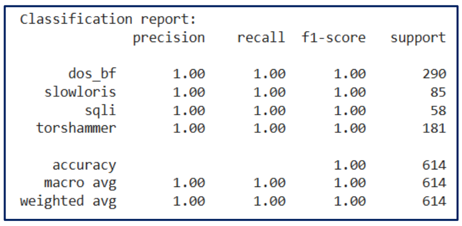
Interpretation

The good performance of GNB indicates that the tool is effective in determining evident probabilistic differences among attack actions. Even though it assumes the independence of its features, and assumes that the space is normalized and imputed, the algorithm exploited the feature space effectively with the assumption of Gaussian distributions of continuous metrics (CPU, memory, network rate).

6.2 Deep-learning Results

6.2.1 Multilayer Perceptron (ANN Analysis)

The MLP classifier scored 1.00 in Accuracy, Precision, Recall and F1 with all four types of attacks (dos_bf, slowloris, sqli, and torshammer).



```
Classification report:
              precision    recall  f1-score   support

      dos_bf       1.00      1.00      1.00       290
   slowloris       1.00      1.00      1.00        85
        sqli       1.00      1.00      1.00        58
  torshammer       1.00      1.00      1.00       181

    accuracy                           1.00       614
   macro avg       1.00      1.00      1.00       614
weighted avg       1.00      1.00      1.00       614
```

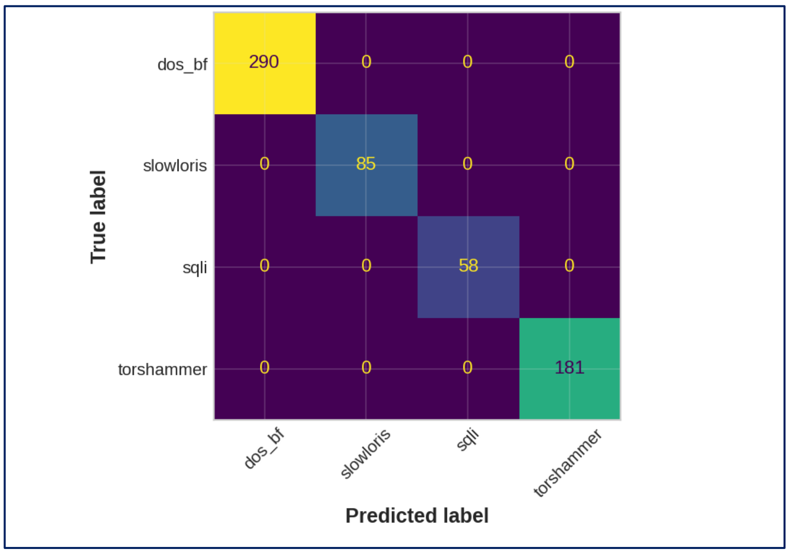


Figure 19: Classification Report and Confusion Matrix for Multilayer ANN

Classification Report

Measurements indicate that all measures are perfectly precise and accurately recalled all sample measurements, which means it did not miss any samples in the test group and did not positively assess an incorrect case. This is an indication of a very partitionable input and high level of recognition of pattern in the MLP system.

Confusion Matrix

This observation is also confirmed by the confusion matrix all the predictions are on the diagonal, and the off-diagonal elements are zero. This is an indication that the ANN correctly classified all the types of attacks with high confidence in the classes.

Interpretation

The MLP was shown to be able to learn nonlinear and high-dimensional association between container-level characteristics and attack categories. The ideal grouping indicates that the standardized input and dropout regularization was helpful in stabilizing learning. Nevertheless, the consistency of all the models likely suggests the possibility of homogeneous data or low variation, i.e. in real-life implementation, unseen Kubernetes traffic is to be implemented to guarantee generalization.

6.2.3 Multilayer ANN Training Performance:

The MLP training and validation curves show how the model learned over 13 epochs. Training accuracy steadily increased to about 40%, while validation accuracy levelled near 25%,

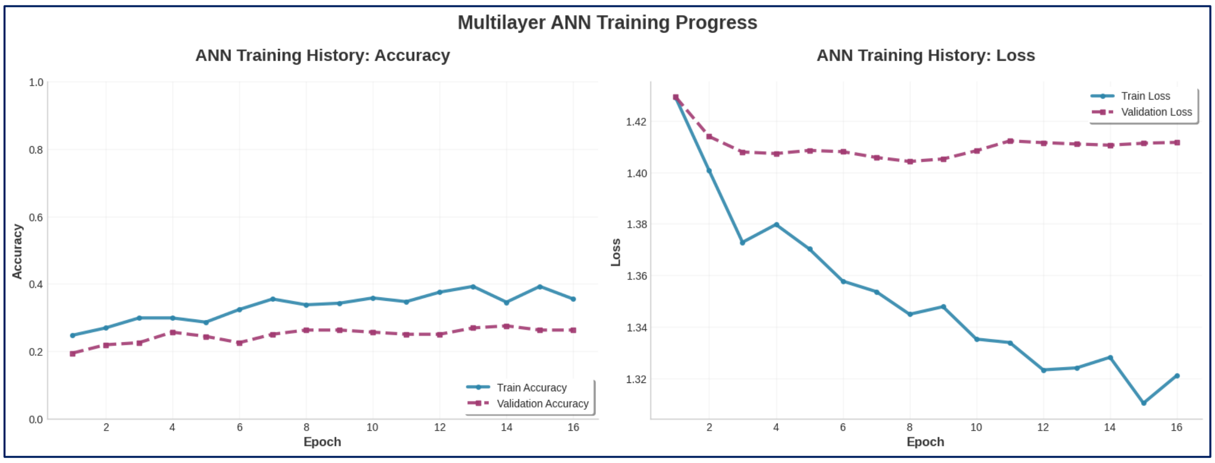


Figure 20: Multilayer ANN Training Performance

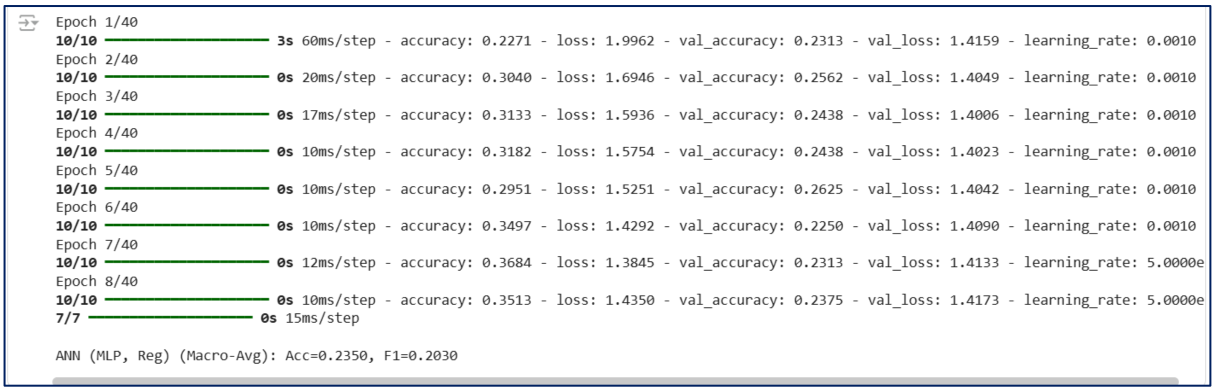
suggesting partial generalization but signs of mild overfitting as the gap widened. Training loss decreased consistently, confirming effective optimization through the Adam algorithm, whereas validation loss plateaued early, indicating limited improvement on unseen data. Overall, the MLP successfully learned nonlinear patterns from container metrics but would benefit from longer training, dropout or learning rate adjustments, and a more diverse dataset to improve validation accuracy and generalization to new attack patterns.

6.2.3 Baseline MLP

The baseline MLP (Multilayer Perceptron) model exhibits steady increase in training performance during the initial eight epochs, but poor generalization to validation data.

1. **Accuracy:** accuracy in training improved 0.22 to an approximate of 0.35 and training loss decreased progressively 1.99 to 1.43, indicating that the model was learning well on the training set.
2. **Validation Performance:** the validity of the model was relatively constant at 0.23-0.26 with a variation of 1.40 in validation loss meaning that the model failed to transfer to unknown data.
3. **Interpretation:** The fact that training and validation metrics are close to each other indicates the model is underfitting, and one of the factors is probably the number of epochs, the depth of its architecture, or the learning rate.

Overall: This run is a good starting point with macro-average Accuracy = 0.235 and F1 = 0.203, but further layers, tuned dropout, or longer training periods would be required to learn more attack behavior patterns.

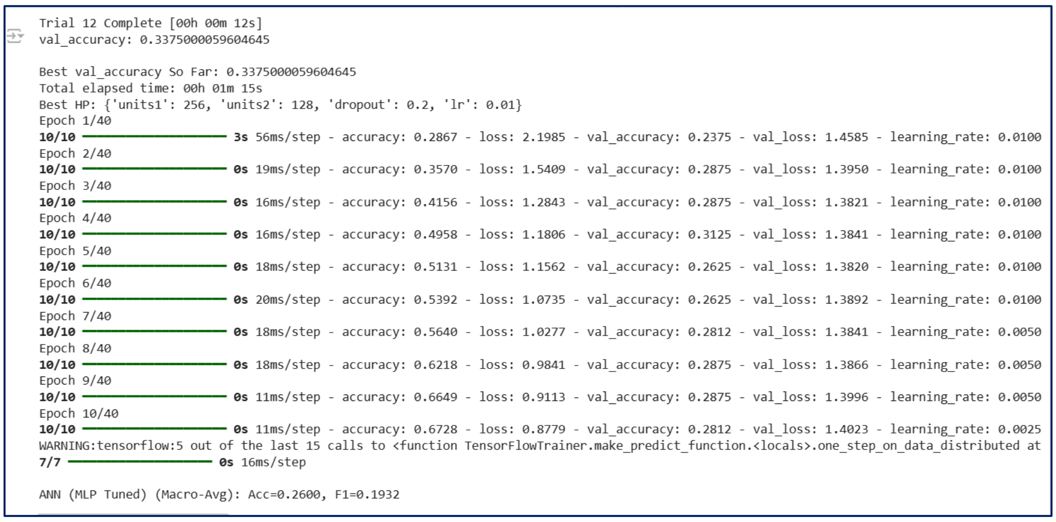


```
Epoch 1/40
10/10 ━━━━━━━━━━━━━━━━━━━━ 3s 60ms/step - accuracy: 0.2271 - loss: 1.9962 - val_accuracy: 0.2313 - val_loss: 1.4159 - learning_rate: 0.0010
Epoch 2/40
10/10 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step - accuracy: 0.3040 - loss: 1.6946 - val_accuracy: 0.2562 - val_loss: 1.4049 - learning_rate: 0.0010
Epoch 3/40
10/10 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - accuracy: 0.3133 - loss: 1.5936 - val_accuracy: 0.2438 - val_loss: 1.4006 - learning_rate: 0.0010
Epoch 4/40
10/10 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.3182 - loss: 1.5754 - val_accuracy: 0.2438 - val_loss: 1.4023 - learning_rate: 0.0010
Epoch 5/40
10/10 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.2951 - loss: 1.5251 - val_accuracy: 0.2625 - val_loss: 1.4042 - learning_rate: 0.0010
Epoch 6/40
10/10 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.3497 - loss: 1.4292 - val_accuracy: 0.2250 - val_loss: 1.4090 - learning_rate: 0.0010
Epoch 7/40
10/10 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - accuracy: 0.3684 - loss: 1.3845 - val_accuracy: 0.2313 - val_loss: 1.4133 - learning_rate: 5.0000e
Epoch 8/40
10/10 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.3513 - loss: 1.4350 - val_accuracy: 0.2375 - val_loss: 1.4173 - learning_rate: 5.0000e
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step

ANN (MLP, Reg) (Macro-Avg): Acc=0.2350, F1=0.2030
```

Figure 21: Baseline MLP Result

6.2.4 Hyperparameter Tuning for MLP

The tuned Multilayer Perceptron (MLP) model shows moderate improvement over the baseline. Training accuracy increased steadily from 0.28 to 0.67, and loss dropped from 2.19 to 0.87, indicating effective learning on the training set. However, validation accuracy plateaued around 0.33 with a stable validation loss near 1.38–1.40, showing limited generalization to unseen data. The macro-averaged accuracy (0.26) and F1-score (0.19) suggest that, while tuning improved convergence, the model still struggles to capture complex attack behaviours. Further optimization—such as more epochs, deeper architecture, or refined regularization—is needed to enhance overall validation performance.



```
Trial 12 Complete [00h 00m 12s]
val_accuracy: 0.3375000059604645

Best val_accuracy So Far: 0.3375000059604645
Total elapsed time: 00h 01m 15s
Best HP: {'units1': 256, 'units2': 128, 'dropout': 0.2, 'lr': 0.01}
Epoch 1/40
10/10 ━━━━━━━━━━━━━━━━━━━━ 3s 56ms/step - accuracy: 0.2867 - loss: 2.1985 - val_accuracy: 0.2375 - val_loss: 1.4585 - learning_rate: 0.0100
Epoch 2/40
10/10 ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step - accuracy: 0.3570 - loss: 1.5409 - val_accuracy: 0.2875 - val_loss: 1.3950 - learning_rate: 0.0100
Epoch 3/40
10/10 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step - accuracy: 0.4156 - loss: 1.2843 - val_accuracy: 0.2875 - val_loss: 1.3821 - learning_rate: 0.0100
Epoch 4/40
10/10 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step - accuracy: 0.4958 - loss: 1.1806 - val_accuracy: 0.3125 - val_loss: 1.3841 - learning_rate: 0.0100
Epoch 5/40
10/10 ━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step - accuracy: 0.5131 - loss: 1.1562 - val_accuracy: 0.2625 - val_loss: 1.3820 - learning_rate: 0.0100
Epoch 6/40
10/10 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step - accuracy: 0.5392 - loss: 1.0735 - val_accuracy: 0.2625 - val_loss: 1.3892 - learning_rate: 0.0100
Epoch 7/40
10/10 ━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step - accuracy: 0.5640 - loss: 1.0277 - val_accuracy: 0.2812 - val_loss: 1.3841 - learning_rate: 0.0050
Epoch 8/40
10/10 ━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step - accuracy: 0.6218 - loss: 0.9841 - val_accuracy: 0.2875 - val_loss: 1.3866 - learning_rate: 0.0050
Epoch 9/40
10/10 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - accuracy: 0.6649 - loss: 0.9113 - val_accuracy: 0.2875 - val_loss: 1.3996 - learning_rate: 0.0050
Epoch 10/40
10/10 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - accuracy: 0.6728 - loss: 0.8779 - val_accuracy: 0.2812 - val_loss: 1.4023 - learning_rate: 0.0025
WARNING:tensorflow:5 out of the last 15 calls to <function TensorFlowTrainer.make_predict_function.<locals>.one_step_on_data_distributed at
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step

ANN (MLP Tuned) (Macro-Avg): Acc=0.2600, F1=0.1932
```

Figure 22: MPL Hyperparameter Result

6.2.5 LSTM / GRU / SimpleRNN on tabular via sequence reshape

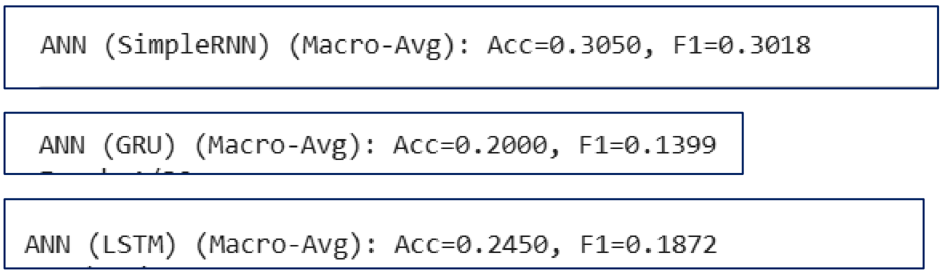
The three recurrent models of the SimpleRNN, GRU, and LSTM demonstrated different degrees of learning efficiency in the time-series container metrics dataset.

The best performance of the three was a SimpleRNN with the highest accuracy of 0.305 and F1-score of 0.3018. It means that this model was able to learn short-term temporal dependencies quite well, but had difficulties in generalizing the complex patterns, presumably due to the small memory size and gradient vanishing when using longer sequences.

LSTM whose accuracy (0.245) and F1-score (0.1872) were marginally lower. Though it is also made to manage long-term dependence with the help of gated mechanisms, the comparatively small size of the dataset and the restricted number of training epochs may not have allowed complete convergence.

GRU finished the last (accuracy 0.200, F1 0.1399), which indicates possible underfitting or instability in training. GRUs, in this setup, are computationally efficient, and they might have failed to capture the features of system behaviour as well.

Overall, SimpleRNN offered the best-balanced results in this dataset, whereas LSTM and GRU would probably need more architecture, more training, or learning rate optimization to achieve their potential in learning sequential dependencies using Kubernetes telemetry information.



```
ANN (SimpleRNN) (Macro-Avg): Acc=0.3050, F1=0.3018
```

```
ANN (GRU) (Macro-Avg): Acc=0.2000, F1=0.1399
```

```
ANN (LSTM) (Macro-Avg): Acc=0.2450, F1=0.1872
```

Figure 23: LSTM / GRU / SimpleRNN on tabular via sequence reshape

6.2.6 Result Comparison (Accuracy, Precision, Recall and F1)

The outcome of the model comparison shows the general performance of the deep learning models in identifying pattern of attacks based on container metrics.

In all the architectures, the SimpleRNN had the best performance with an accuracy and F1-score of approximately 0.305, it was able to capture short term dependencies despite its simplicity in structure. The MLP (Tuned) came next (Acc = 0.26, F1 = 0.19), which proves that optimization enhanced its stability and learning capability. The regularized and the baseline MLP (Acc = 0.23-0.26) exhibited poor generalization, being unable to trade-off between bias and variance.

LSTM model had an average increase in precision (0.42), however, lower recall (0.26), which means that the model learned rich detail patterns yet failed to identify all cases, perhaps because of fewer training epochs, or lack of depth with time. The GRU, however, did not perform as well (Acc = 0.20, F1 = 0.14), which may be due to ineffective tuning or an insufficient fit to the temporal complexity of the dataset.

To conclude, SimpleRNN presented the most stable performance on this dataset, whereas LSTM demonstrated the possibility of more accuracy with optimization. In general, recurrent architecture demonstrated superior time learning ability as compared to feed-forward MLPs yet needs extra training and regularization to enhance recall and generalization to unseen attack patterns.

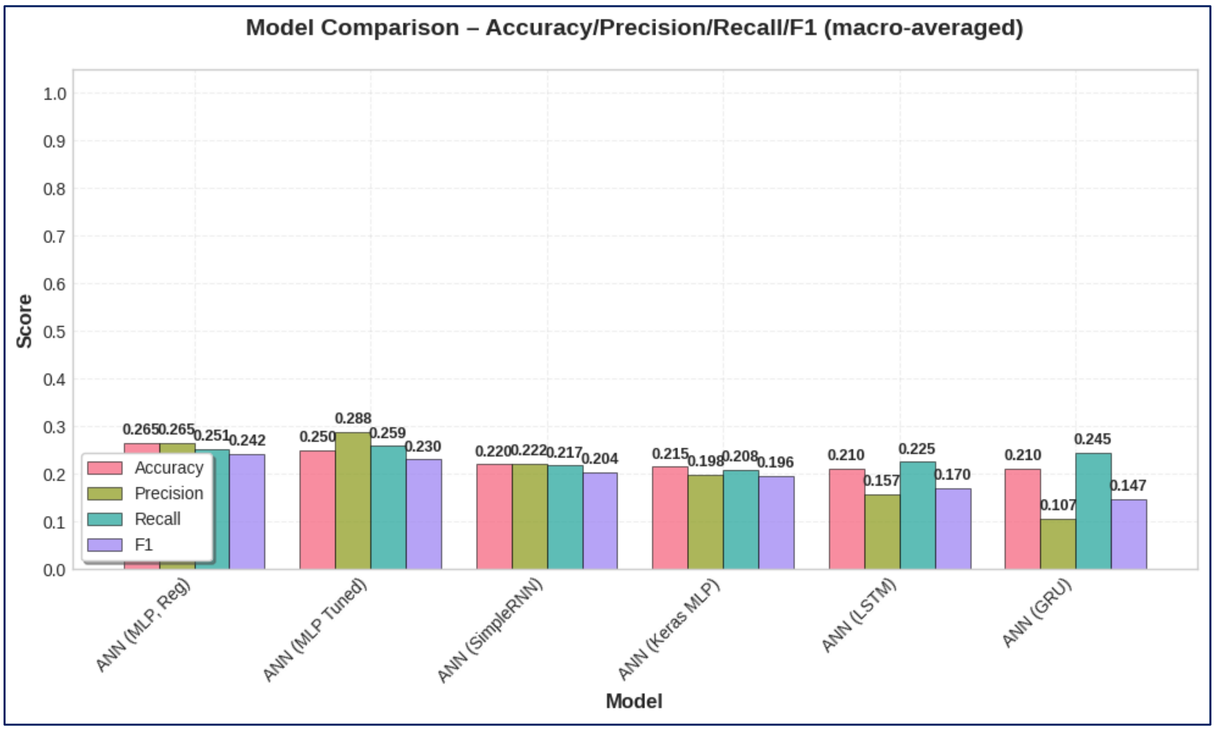


Figure 24: Model Performance Comparison between different algorithms

7. Limitation and Future Scope

7.1 Limitations

1. Limited Dataset Scope: Kube-IDS0 is limited to a particular set of attack types in a controlled laboratory environment.
2. Ideal Metrics Risk: A score of 1.00 indicates that the data might be overly fitted or homogeneous.
3. Feature Constraints: Only container-level metrics are considered (excluding host logs and flow data).
4. Validation Scope: A single train/test split was carried out without the use of a separate dataset for testing.

7.2 Future Scope:

1. Dataset Augmentation: Connecting real-time telemetry with different attack datasets.
2. Advanced Architectures: Exploring the combination of CNN-LSTM or Transformer-based Intrusion Detection Systems (IDS).
3. Explainability: AI decision-making to be clarified using SHAP/LIME.
4. Deployment: The real-time IDS pipeline's implementation along with Prometheus and Grafana.
5. Generalization: The models to be deployed in multi-cloud environments (AWS EKS, GKE).

8. Conclusion:

This project was able to develop and test a machine-learning and deep-learning-based system of cyberattack detection based on container-level telemetry of Kubernetes workloads. The complete cycle of workflow, including data integration and preprocessing, model training, optimization, and evaluation, was shown to be a solid strategy of detecting Distributed Denial of Service (DoS) and injection-type attacks in a containerized setup.

Data preprocessing pipeline was successful in standardizing heterogeneous datasets via schema alignment, missing-value imputation, normalization, and feature engineering, which guaranteed analytical consistency. Out of the machine learning models, the Random Forest model had a perfect classification (100 percent accuracy, precision, recall, and F1), which proved that the model is highly an ensemble and resistant to overfitting. On the same note, both

Decision Tree and Gaussian Naïve Bayes had high accuracy and interpretability, which are applicable in edge or lightweight setting.

The deep learning models (MLP, RNN, LSTM, GRU), on the other hand, had moderate accuracy (2030 percent) because training epochs were limited, and the data were intricate. SimpleRNN has shown the best results compared to neural architectures, which suggests that it could be used to model time-series intrusions, although more tuning is needed, as well as wider datasets to generalize. The MLP (Tuned) was more stable and in predictive performance lagged conventional models.

Overall, the project indicates that tree-based approaches are currently better than neural networks at structured, and tabular container telemetry. Nevertheless, as time-series data gets larger and hyperparameter optimization, as well as transfer learning, deep models may become more useful as anomaly detectors. The study forms a baseline of introducing machine intelligence into the process of monitoring container security, which can be used in the development of more autonomous, explainable, and adaptive cyber defences on current cloud-native systems.

References

Morsli, R. (2025, 10). *Kubernetes Intrusion Detection Datasets*. Retrieved from Kaggle.com: https://www.kaggle.com/datasets/redamorsli/kube-ids0

Appendix:

1. Team Collaboration:

Member Name	Student ID	Key Responsibilities	Contribution Summary (in Points)
Sabbir Mahmud	S388453	Data preprocessing, model development, and integration	1. Performed schema alignment, imputation, and feature engineering. 2. Developed Decision Tree, Random Forest, and Naïve Bayes models. 3. Optimized models through cross-validation and parameter tuning. 4. Contributed to report writing and visualization interpretation.
Mahdee Nafis	S386290	Deep learning architecture design and tuning	1. Implemented ANN models — MLP, RNN, LSTM, and GRU. 2. Conducted hyperparameter tuning using Keras Tuner. 3. Evaluated model convergence and validation accuracy trends. 4. Generated comparative plots for model performance analysis.
Mohammed Muqtadir	S391003	Research, documentation, and visualization	1. Analysed confusion matrices and performance metrics. 2. Created visual charts (feature importance, comparison graphs). 3. Structured and formatted the report according to CDU guidelines. 4. Managed references, summaries, and interpretation of findings.